Tight and rigourous error bounds for basic building blocks of double-word arithmetic

Mioara Joldes, Jean-Michel Muller, Valentina Popescu

Tight and rigorous error bounds for basic building blocks of double-word arithmetic

Mioara Joldes
LAAS CNRS Toulouse
France
joldes@laas.fr

Jean-Michel Muller
CNRS, ENS Lyon
Université de Lyon, France
jean-michel.muller@ens-lyon.fr

Valentina Popescu
ENS Lyon
Université de Lyon, France
valentina.popescu@ens-lyon.fr

March 24, 2017

Abstract

We analyze several classical basic building blocks of double-word arithmetic (frequently called "double-double arithmetic" in the literature): the addition of a double-word number and a floating-point number, the addition of two double-word numbers, the multiplication of a double-word number by a floating-point number, the multiplication of two double-word numbers, the division of a double-word number by a floating-point number, and the division of two double-word numbers. For multiplication and division we get better relative error bounds than the ones previously published. For addition of two double-word numbers, we show that the previously published bound was incorrect, and we provide a new relative error bound. We introduce new algorithms for division. We also give examples that illustrate the tightness of our bounds.

Keywords. Floating-point arithmetic; double-word arithmetic; double-double arithmetic; error-free transforms.

1 Introduction and notation

Some calculations require a precision significantly higher than the one offered by the binary64 (also known as "double-precision") format. A typical example is the evaluation of transcendental functions in binary64 arithmetic with correct rounding: if all intermediate calculations are done in the target precision, it is very difficult to guarantee last-bit accuracy in the final result. For instance, the CRLibm library of correctly rounded elementary functions uses "double-double" or "triple-double" operations in critical parts [4]. Double-double arithmetic has also been used with success in BLAS [14]. Other examples where higher-precision arithmetic has been useful, mentioned by Briggs [3] or Bailey et al. [1], are studies of dynamical systems, the calculation of two-loop integrals for radiative corrections in muon decay, experimental mathematics, supernova simulations, and studies of the fine structure constant of physics.

There exist very good arbitrary precision libraries, such as GNU-MPFR [6]. However, if one only needs calculations accurate within around 120 bits in a few critical parts of a numerical program, using such libraries will involve a significant penalty in terms of speed and memory consumption.

Although the binary128 format (frequently called "quad-precision") was specified by the IEEE 754-2008 Standard on Floating-Point Arithmetic, it is seldom implemented in hardware. To our knowledge, the only commercially significant platform that has supported binary128 in hardware for the last decade has been the IBM z Systems [15]. Thus, one will be tempted to use "double-double" arithmetic at times. Furthermore, even if hardwired binary128 arithmetic becomes commonplace, there will be a need for "double-quad" operations for carefully implementing very accurate binary128 elementary functions. Hence, designing and analyzing algorithms for double-word arithmetic is of interest.

Double-word arithmetic, called "double-double" in most of the literature, consists in representing a real number as the unevaluated sum of two floating-point numbers. In all existing implementations, the underlying floating-point format is the *binary64* format of the IEEE 754 Standard on Floating-Point Arithmetic [9, 18], commonly called "double-precision" (hence the name "double-double").

Double-word arithmetic is NOT similar to a conventional, IEEE 754-like, floating-point arithmetic with twice the precision. It lacks many nice properties such as Lemma 1.2 below, clearly defined roundings, etc. Furthermore, many algorithms have been published without a proof, or with error bounds that are sometimes loose, sometimes fuzzy (the error is "less than a small integer times u^2"), and sometimes unsure. Kahan qualifies double-double arithmetic as an "attractive nuisance except for the BLAS" and even compares it to an unfenced backyard swimming pool! He also mentions [11] that it "undermines the incentive to provide quadruple precision correctly rounded". The purpose of this paper is to provide a rigorous error analysis of some double-word algorithms, and to introduce a few new algorithms. We cannot suppress all the drawbacks mentioned by Kahan: clearly, having in hardware a "real" floating-point arithmetic with twice the precision would be a better option. And yet, if rigorously proven and reasonably tight error bounds are provided, expert programmers can rely on double-word arithmetic for extending the precision of calculations in critical places where the available floating-point arithmetic does not suffice.

Throughout this paper, we assume a radix-2, precision-p floating-point (FP) arithmetic system, with unlimited exponent range and correct rounding. This means that our results will apply to "real-world" binary floating-point arithmetic, such as the one specified by the IEEE 754-2008 Standard [9, 18], provided that underflow and overflow do not occur.

The notation $\mathrm{RN}(t)$ stands for t rounded to the nearest FP number, ties-to-even. For instance $\mathrm{RN}(c \cdot d)$ is the result of the FP multiplication `c×d`, assuming round-to-nearest rounding mode. The number $\mathrm{ulp}(x)$, for $x \neq 0$ is $2^{\lfloor \log_2 |x| \rfloor - p + 1}$, and $u = 2^{-p} = \frac{1}{2}\mathrm{ulp}(1)$ denotes the roundoff error unit. We will frequently use the three following, classical lemmas.

Lemma 1.1. *Let $t \in \mathbb{R}$. If $|t| \leq 2^k$, where k is an integer, then*

$$|\mathrm{RN}(t) - t| \leq \frac{u}{2} \cdot 2^k.$$

Lemma 1.2 (Sterbenz Lemma [23])**.** *Let x and y be two positive FP numbers. If*

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

then $x - y$ is a floating-point number, so that $\mathrm{RN}(x - y) = x - y$.

Lemma 1.3. *If $t \in \mathbb{R}$, there exist ϵ_1 and ϵ_2, both of absolute value less than or equal to u, such that*

$$\mathrm{RN}(t) = t \cdot (1 + \epsilon_1) = \frac{t}{1 + \epsilon_2}.$$

The algorithms analyzed in this paper use as basic blocks Algorithms 1, 2, and 3 below. They have been coined as "error free transforms" by Rump [21].

Algorithms 1 and 2, introduced by Moller [17], Dekker [5], and Knuth [12] make it possible to compute both the result and the rounding error of a FP addition. We will choose between them depending on the information that we have on the input numbers.

Algorithm 1 – **Fast2Sum**(a, b). The Fast2Sum algorithm [5].

$s \leftarrow \mathrm{RN}(a + b)$
$z \leftarrow \mathrm{RN}(s - a)$
$t \leftarrow \mathrm{RN}(b - z)$

If $a = 0$ or $b = 0$, or if the floating-point exponents e_a and e_b satisfy $e_a \geq e_b$, then $s + t = a + b$. Hence, t is the error of the FP addition $s \leftarrow \mathrm{RN}(a + b)$. In practice, condition "$e_a \geq e_b$" may be hard to check. However, if $|a| \geq |b|$ then that condition is satisfied.

Algorithm 2 – **2Sum**(a, b). The 2Sum algorithm [17, 12].

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

Algorithm 2 gives the same results as Algorithm 1, but without any requirement on the exponents of a and b. It uses 6 FP operations for computing the result (instead of 3 for Algorithm 1), but on modern processors comparing the absolute values of a and b and swapping them if needed before calling Algorithm 1 will in general be more time-consuming than directly calling Algorithm 2. Hence, in general, Algorithm 1 is to be used only if we have preliminary information on the respective orders of magnitude of a and b. However, in all the algorithms presented below, a call to 2Sum can be replaced by a test and a call to Fast2Sum without changing the error bounds.

Let a and b be two FP numbers, with exponents e_a and e_b, respectively. Define $\pi = \mathrm{RN}(ab)$. The number $\rho = ab - \pi$ is a FP number. When the exponent range is not unbounded, this holds provided that $e_a + e_b \geq e_{\min} + p - 1$, where $e_{\min}$ is the minimum exponent of the underlying FP format. See [19] for a proof. The first algorithm introduced for computing π and ρ is due to Dekker [5, 2]. It requires 17 FP operations. When an FMA instruction is available, Algorithm Fast2Mult (Algorithm 3 below), mentioned by Kahan [11], only requires 2 FP operations for computing the same values.

Algorithm 3 – Fast2Mult(a, b)**.** The Fast2Mult algorithm (see for instance [11, 19, 18]). It requires the availability of a fused multiply-add (FMA) instruction for computing $\mathrm{RN}(ab - \pi)$.

$\pi \leftarrow \mathrm{RN}(a \cdot b)$
$\rho \leftarrow \mathrm{RN}(a \cdot b - \pi)$

In the following, we will denote 2Prod an algorithm that computes π and ρ. It can be either Dekker's algorithm or Algorithm 3. However, when we count the number of floating-point operations required by the various algorithms presented in this paper (in Table 1), we assume that Algorithm 3 is used.

Dekker [5] was the first to suggest using algorithms similar to Algorithm 1 and the equivalent (without FMA) of Agorithm 3 in order to manipulate numbers represented as unevaluated sums of two FP numbers. He called such numbers *doublelength* numbers. Dekker presented algorithms for adding, multiplying, and dividing double-word numbers. His addition and multiplication algorithms are very similar (in fact, mathematically equivalent) to Algorithms 5 and 10, analyzed below. His division algorithm was quite different (and less accurate) than the algorithms considered in this paper. Linnainmaa [16] suggested similar algorithms, assuming that an underlying extended precision format is available. We will not assume that hypothesis here.

Libraries that offer double-word arithmetic (with *binary64* as the underlying floating-point format) have been written by Bailey [8] and Briggs [3]. Briggs no longer maintains his library. Fairly recent functions for double-word arithmetic are included in the QD ("quad-double") library by Hida, Li, and Bailey [7, 8].

In Definition 1.4 we formally introduce the concept of *double-word* representation.

Definition 1.4. *A* double-word *number x is the unevaluated sum $x_h + x_\ell$ of two floating-point numbers x_h and x_ℓ such that*

$$x_h = \mathrm{RN}(x).$$

The sequel of the paper is organized as follows: Section 2 deals with the sum of a double-word number and a floating-point number; Section 3 is devoted to the sum of two double-word numbers; in Section 4 we consider the product of a double-word number by a floating-point number; in Section 5 we consider the product of two double-word numbers; Section 6 deals with the division of a double-word number by a floating-point number, and Section 7 is devoted to the division of two double-word numbers. All algorithms considered in this paper return their results as a double-word number. We summarize our results in Table 1, in the Conclusion section.

2 Addition of a double-word number and a floating-point number

The algorithm implemented in the QD library [8] for adding a double-word number and a floating-point number is Algorithm 4 below.

Algorithm 4 – **DWPlusFP**(x_h, x_ℓ, y). Algorithm for computing $(x_h, x_\ell) + y$ 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 Definition 1.4).

1: $(s_h, s_\ell) \leftarrow \mathrm{2Sum}(x_h, y)$
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 4, or variants of it, implicitely appears in many "compensated summation" algorithms. Compensated summation algorithms aim at accurately computing the sum of several FP numbers. Most such algorithms implicitely represent, at intermediate steps of the summation, the sum of all input numbers accumulated so far as a double-word number. For instance the first two lines of Algorithm 4 constitute the internal loop of Rump, Ogita and Oishi's "cascaded summation" algorithm [20].

To prove the correctness and bound the error of Algorithm 4 (and Algorithm 6 below), we will need the following lemma. That lemma is an immediate consequence of Property (2.16) in [22].

Lemma 2.1. *(see Property (2.16) in [22]) Let a and b be FP numbers, and let $s = \mathrm{RN}(a+b)$. If $s \neq 0$ then*

$$|s| \geq \max\left\{\frac{1}{2}\mathrm{ulp}(a), \frac{1}{2}\mathrm{ulp}(b)\right\}.$$

Proof. Without l.o.g., assume $|a| \geq |b|$, so that $\mathrm{ulp}(a) \geq \mathrm{ulp}(b)$. The number $|a+b|$ is the distance between a and $-b$. Hence, since $a \neq -b$ (otherwise s would be 0), $|a+b|$ is larger than or equal to the distance between a and the FP number nearest a, which is larger than or equal to $\frac{1}{2}\mathrm{ulp}(a)$. Therefore $|\mathrm{RN}(a+b)| = \mathrm{RN}(|a+b|) \geq \mathrm{RN}(\frac{1}{2}\mathrm{ulp}(a)) = \frac{1}{2}\mathrm{ulp}(a)$. □

Let us now turn to the analysis of Algorithm 4. We have,

Theorem 2.2. *The relative error*

$$\left|\frac{(z_h + z_\ell) - (x+y)}{x+y}\right|$$

of Algorithm 4 (DWPlusFP) is bounded by

$$\frac{2 \cdot u^2}{1-2u} = 2u^2 + 4u^3 + 8u^4 + \cdots, \tag{1}$$

which is less than $2u^2 + 5u^3$ as soon as $p \geq 4$.

Proof. First of all, the case $x_h + y = 0$ is trivial since $s_h = s_\ell = 0$ and the computation is errorless. Now, without loss of generality, we can assume $|x_h| \geq |y|$. If this is not the case, since x_h and y play a symmetrical role in the algorithm we can exchange them in our proof: we add the double word number (y, x_ℓ) and the floating-point number x_h.[1] We also assume that x_h is positive (otherwise we change the sign of all the operands), and that $1 \leq x_h \leq 2-2u$ (otherwise we scale the operands by a power of 2).

Define ϵ as the error committed at step 2, i.e., $\epsilon = v - (x_\ell + s_\ell)$.

1. If $-x_h < y \leq -x_h/2$, then Sterbenz Lemma implies $s_h = x_h + y$ and $s_\ell = 0$. It follows that $v = x_\ell$. Lemma 2.1 implies $|s_h| \geq \frac{1}{2}\mathrm{ulp}(x_h)$, which implies $|s_h| \geq |x_\ell|$. Hence Algorithm Fast2Sum introduces no error at line 3 of the algorithm, so that $z_h + z_\ell = s_h + v = x + y$ exactly.

2. If $-x_h/2 < y \leq x_h$, then $\frac{1}{2} \leq \frac{x_h}{2} < x_h + y \leq 2x_h$, so that $s_h \geq 1/2$. Since $|x_\ell + s_\ell| \leq 3u$ (see the two cases considered below), we have $|v| \leq 3u$, so that $s_h > |v|$: Algorithm Fast2Sum introduces no error at line 3 of the algorithm. Therefore $z_h + z_\ell = s_h + v = x + y + \epsilon$.

- If $x_h + y \leq 2$ then $|s_\ell| \leq u$, so that $|x_\ell + s_\ell| \leq 2u$, hence $|\epsilon| \leq u^2$, and the relative error $|\epsilon|/|x+y|$ of the calculation is bounded by

$$\frac{|\epsilon|}{\frac{1}{2} - u} \leq \frac{2u^2}{1-2u}.$$

[1] (y, x_ℓ) may not be a double-word number, according to Definition 1.4, in the case $x_\ell = \frac{1}{2}\mathrm{ulp}(y) = \frac{1}{2}\mathrm{ulp}(x_h)$. However, one easily checks that in that case the algorithm returns an exact result.

- If $x_h + y > 2$ then $|s_\ell| \leq 2u$, so that $|x_\ell + s_\ell| \leq 3u$, hence $|\epsilon| \leq 2u^2$, and the relative error $|\epsilon|/|x+y|$ of the calculation is bounded by

$$\frac{|\epsilon|}{2-u} \leq \frac{2u^2}{2-u}.$$

$\square$

Notice that the bound (1) is very sharp. In fact, it is *asymptotically optimal*. This is shown by the following example: $x_h = 1$, $x_\ell = (2^p - 1) \cdot 2^{-2p}$, and $y = -\frac{1}{2}(1 - 2^{-p})$, for which the computed sum is $\frac{1}{2} + 3 \cdot 2^{-p-1}$ and the exact sum is $\frac{1}{2} + 3 \cdot 2^{-p-1} - 2^{-2p}$, resulting in a relative error

$$\frac{2u^2}{1 + 3u - 2u^2} \approx 2u^2 - 6u^3.$$

In the *binary64* format ($p = 53$), this generic example gives an error

$$1.99999999999999933 \cdots \times 2^{-106}.$$

3 Addition of two double-word numbers

Algorithm 5 below was first given by Dekker [5], under the name of *add2*, with a slightly different presentation. Dekker did not use the 2Sum algorithm: instead of Line 1 there was a comparison of $|x_h|$ and $|y_h|$ followed by a possible swap of x and y and a call to Fast2Sum. However, from a mathematical point of view, Dekker's algorithm and Algorithm 5 are equivalent: they always return the same result. This algorithm was then implemented by Bailey in the QD library [8] under the name of "sloppy addition".

Algorithm 5 – SloppyDWPlusDW$(x_h, x_\ell, y_h, y_\ell)$. "Sloppy" calculation of $(x_h, x_\ell) + (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic.

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

Dekker proved an error bound on the order of $(|x| + |y|) \cdot 4u^2$. Notice the absolute values: when x and y do not have the same sign, there is no proof that the relative error is bounded. Indeed, *the relative error can be so large that the obtained result has no significance at all.* Consider for instance the case $x_h = 1 + 2^{-p+3}$, $x_\ell = -2^{-p}$, $y_h = -1 - 6 \cdot 2^{-p}$, and $y_\ell = -2^{-p} + 2^{-2p}$. It leads to a computed value of the sum equal to zero, whereas the exact value is 2^{-2p}: the relative error is equal to 1. This is why the use of Algorithm 5 should be restricted to special cases such as, for instance, when we know in advance that the operands will have the same sign. When accurate computations are required, it is much more advisable to use the following algorithm, presented by Li et al. [13, 14] and implemented in the QD library under the name of "IEEE addition".

Algorithm 6 – **AccurateDWPlusDW**$(x_h, x_\ell, y_h, y_\ell)$. Calculation of $(x_h, x_\ell) + (y_h, y_\ell)$ 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: $c \leftarrow \text{RN}(s_\ell + t_h)$
4: $(v_h, v_\ell) \leftarrow \text{Fast2Sum}(s_h, c)$
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)

In [13, 14], Li et al. claim that in binary64 arithmetic ($p = 53$) the relative error of Algorithm 6 is upper-bounded by $2 \cdot 2^{-106}$. This bound is incorrect, as shown by the following example: if

$$\begin{array}{rcl} x_h &=& 9007199254740991, \\ x_\ell &=& -9007199254740991/2^{54}, \\ y_h &=& -9007199254740987/2, \text{ and} \\ y_\ell &=& -9007199254740991/2^{56}, \end{array} \tag{2}$$

then the relative error of Algorithm 6 is

$$2.24999999999999956\cdots \times 2^{-106}.$$

Note that this example is somehow "generic": in precision-p FP arithmetic, the choice $x_h = 2^p - 1$, $x_\ell = -(2^p - 1) \cdot 2^{-p-1}$, $y_h = -(2^p - 5)/2$, and $y_\ell = -(2^p - 1) \cdot 2^{-p-3}$ leads to a relative error that is asymptotically equivalent (as p goes to infinity) to $2.25u^2$.

Now let us try to find a relative error bound. We are going to show the following result:

Theorem 3.1. *If $p \geq 3$, the relative error of Algorithm 6 (AccurateDW-PlusDW) is bounded by*

$$\frac{3u^2}{1-4u} = 3u^2 + 12u^3 + 48u^4 + \cdots, \tag{3}$$

which is less than $3u^2 + 13u^3$ as soon as $p \geq 6$.

Note that the conditions on p ($p \geq 3$ for the bound (3) to hold, $p \geq 6$ for the simplified bound $3u^2 + 13u^3$) are satisfied in all practical cases.

Proof. First of all, we exclude the straightforward case in which one of the operands is zero. We can also quickly proceed with the case $x_h + y_h = 0$: the returned result is $2\mathrm{Sum}(x_\ell, y_\ell)$, which is equal to $x+y$, i.e., the computation is errorless. Now, without loss of generality, we assume $1 \le x_h < 2$, $x \ge |y|$ (which implies $x_h \ge |y_h|$), and $x_h + y_h$ nonzero. Notice that $1 \le x_h < 2$ implies $1 \le x_h \le 2 - 2u$, since x_h is a FP number.

Define ϵ_1 as the error committed at Line 3 of the algorithm:

$$\epsilon_1 = c - (s_\ell + t_h), \tag{4}$$

and ϵ_2 as the error committed at Line 5:

$$\epsilon_2 = w - (t_\ell + v_\ell). \tag{5}$$

1. If $-x_h < y_h \le -x_h/2$. Sterbenz Lemma, applied to the first line of the algorithm, implies $s_h = x_h + y_h$, $s_\ell = 0$, and $c = \mathrm{RN}(t_h) = t_h$.

Define

$$\sigma = \begin{cases} 2 & \text{if} \quad y_h \le -1, \\ 1 & \text{if} \quad -1 < y_h \le -x_h/2. \end{cases}$$

We have $-x_h < y_h \le (1-\sigma) + \frac{x_h}{2}(\sigma - 2)$, so that $0 \le x_h + y_h \le 1 + \sigma \cdot \left(\frac{x_h}{2} - 1\right) \le 1 - \sigma u$. Also, since x_h is a multiple of $2u$ and y_h is a multiple of σu, $s_h = x_h + y_h$ is a multiple of σu. Since s_h is nonzero, we finally obtain

$$\sigma u \le s_h \le 1 - \sigma u. \tag{6}$$

We have $|x_\ell| \le u$ and $|y_\ell| \le \frac{\sigma}{2} u$, so that

$$|t_h| \le \left(1 + \frac{\sigma}{2}\right) u \quad \text{and} \quad |t_\ell| \le u^2. \tag{7}$$

From (6), we deduce that the floating-point exponent of s_h is at least $-p+\sigma-1$. From (7), the floating-point exponent of $c = t_h$ is at most $-p+\sigma-1$. Therefore, the Fast2Sum algorithm introduces no error at line 4 of the algorithm, which implies

$$v_h + v_\ell = s_h + c = s_h + t_h = x + y - t_\ell.$$

Eq. (6) and (7) imply

$$|s_h + t_h| \le 1 + \left(1 - \frac{\sigma}{2}\right) u \le 1 + \frac{u}{2},$$

so that $|v_h| \le 1$ and $|v_\ell| \le \frac{u}{2}$. From the bounds on $|t_\ell|$ and $|v_\ell|$ we obtain:

$$|\epsilon_2| \le \frac{1}{2}\mathrm{ulp}(t_\ell + v_\ell) \le \frac{1}{2}\mathrm{ulp}\left(u^2 + \frac{u}{2}\right) = \frac{u^2}{2}, \tag{8}$$

and

$$|\epsilon_2| \le \frac{1}{2}\mathrm{ulp}\left[\frac{1}{2}\mathrm{ulp}(x_\ell + y_\ell) + \frac{1}{2}\mathrm{ulp}\left((x+y) + \frac{1}{2}\mathrm{ulp}(x_\ell + y_\ell)\right)\right]. \tag{9}$$

Lemma 2.1 and $|s_h| \ge \sigma u$ imply that either $s_h + t_h = 0$, or $|v_h| = |\mathrm{RN}(s_h + c)| = |\mathrm{RN}(s_h + t_h)| \ge \sigma u^2$. If $s_h + t_h = 0$ then $v_h = v_\ell = 0$ and the sequel of the proof is straightforward. Therefore, in the following, we assume $|v_h| \ge \sigma u^2$.

Now,

- if $|v_h| = \sigma u^2$ then $|v_\ell + t_\ell| \le u|v_h| + u^2 = \sigma u^3 + u^2$, which implies $|w| = |\mathrm{RN}(t_\ell + v_\ell)| \le \sigma u^2 = |v_h|$;
- if $|v_h| > \sigma u^2$ then, since v_h is a FP number, $|v_h|$ is larger than or equal to the FP number immediately above σu^2, which is $\sigma(1+2u)u^2$. Hence $|v_h| \ge \sigma u^2/(1-u)$, so that $|v_h| \ge u \cdot |v_h| + \sigma u^2 \ge |v_\ell| + |t_\ell|$. So, $|w| = |\mathrm{RN}(t_\ell + v_\ell)| \le |v_h|$.

Therefore, in all cases, Fast2Sum introduces no error at line 6 of the algorithm, and we have

$$z_h + z_\ell = v_h + w = x + y + \epsilon_2. \tag{10}$$

Directly using (10) and the bound $u^2/2$ on $|\epsilon_2|$ to get a relative error bound would result in a large bound, because $x + y$ may be small. However, when $x + y$ is very small, some simplification occurs thanks to Sterbenz Lemma. First, $x_h + y_h$ is a nonzero multiple of σu. Hence, since $|x_\ell + y_\ell| \le \left(1 + \frac{\sigma}{2}\right) u$, we have $|x_\ell + y_\ell| \le \frac{3}{2}(x_h + y_h)$. Let us now consider the two possible cases:

- if $-\frac{3}{2}(x_h+y_h) \le x_\ell+y_\ell \le -\frac{1}{2}(x_h+y_h)$, which implies $-\frac{3}{2}s_h \le t_h \le -\frac{1}{2}s_h$, then Sterbenz lemma applies to the floating-point addition of s_h and $c = t_h$. Therefore line 4 of the algorithm results in $v_h = s_h$ and $v_\ell = 0$. An immediate consequence is $\epsilon_2 = 0$, so that $z_h + z_\ell = v_h + w = x + y$: the computation of $x + y$ is errorless;
- if $-\frac{1}{2}(x_h+y_h) < x_\ell+y_\ell \le \frac{3}{2}(x_h+y_h)$ then $\frac{5}{2}(x_\ell+y_\ell) \le \frac{3}{2}(x_h+y_h+x_\ell+y_\ell) = \frac{3}{2}(x+y)$, and $-\frac{1}{2}(x+y) < \frac{1}{2}(x_\ell + y_\ell)$. Hence $|x_\ell + y_\ell| < |x + y|$, so that $\mathrm{ulp}(x_\ell + y_\ell) \le \mathrm{ulp}(x + y)$. Combined with (9), this gives

$$|\epsilon_2| \le \frac{1}{2}\mathrm{ulp}\left(\frac{3}{2}\mathrm{ulp}(x+y)\right) \le 2^{-p}\mathrm{ulp}(x+y) \le 2 \cdot 2^{-2p} \cdot (x+y).$$

2. If $-x_h/2 < y_h \le x_h$

Notice that we have $x_h/2 < x_h + y_h \le 2x_h$, so that $x_h/2 \le s_h \le 2x_h$. Also notice that we have $|x_\ell| \le u$.

- If $\frac{1}{2} < x_h + y_h \le 2 - 4u$. Define

$$\sigma = \begin{cases} 1 & \text{if} \quad x_h + y_h \le 1 - 2u, \\ 2 & \text{if} \quad 1 - 2u < x_h + y_h \le 2 - 4u. \end{cases}$$

We have

$$\frac{\sigma}{2}(1-2u) \le s_h \le \sigma(1-2u) \quad \text{and} \quad |s_\ell| \le \frac{\sigma}{2}u. \tag{11}$$

When $\sigma = 1$, we necessarily have $-x_h/2 < y_h < 0$, so that $|y_\ell| \le u/2$. And when $\sigma = 2$, $|y_h| \le x_h \le 2 - 2u$ implies $|y_\ell| \le u$. Hence we always have $|y_\ell| \le \frac{\sigma}{2}u$. This implies $|x_\ell + y_\ell| \le (1 + \sigma/2)u$, therefore

$$|t_h| \le \left(1 + \frac{\sigma}{2}\right) u \quad \text{and} \quad |t_\ell| \le u^2. \tag{12}$$

Now, $|s_\ell + t_h| \le (1+\sigma)u$, so that

$$|c| \le (1+\sigma)u \quad \text{and} \quad |\epsilon_1| \le \sigma u^2. \tag{13}$$

Since $s_h \ge 1/2$ and $|c| \le 3u$, if $p \ge 3$ then Algorithm Fast2Sum introduces no error at line 4 of the algorithm, i.e.,

$$v_h + v_\ell = s_h + c.$$

Therefore $|v_h + v_\ell| = |s_h + c| \le \sigma(1-2u) + (1+\sigma)u \le \sigma$. This implies

$$|v_h| \le \sigma \quad \text{and} \quad |v_\ell| \le \frac{\sigma}{2}u. \tag{14}$$

Thus $|t_\ell + v_\ell| \le u^2 + \frac{\sigma}{2}u$, so that

$$|w| \le \frac{\sigma}{2}u + u^2 \quad \text{and} \quad |\epsilon_2| \le \frac{\sigma}{2}u^2. \tag{15}$$

From (11) and (13), we deduce $s_h + c \ge \frac{\sigma}{2} - u(2\sigma+1)$, so that $|v_h| \ge \frac{\sigma}{2} - u(2\sigma+1)$. If $p \ge 3$ then $|v_h| \ge |w|$, so that Algorithm Fast2Sum introduces no error at line 6 of the algorithm, i.e., $z_h + z_\ell = v_h + w$.

Therefore,

$$z_h + z_\ell = x + y + \eta,$$

with $|\eta| = |\epsilon_1 + \epsilon_2| \le \frac{3\sigma}{2}u^2$. Since

$$x + y \ge (x_h - u) + (y_h - u/2) > \left\{ \begin{array}{lll} \frac{1}{2} - \frac{3}{2}u & \text{if} & \sigma = 1, \\ 1 - 4u & \text{if} & \sigma = 2, \end{array} \right.$$

the relative error $|\eta|/(x+y)$ is upper-bounded by

$$\frac{3u^2}{1-4u}.$$

- If $2 - 4u < x_h + y_h \le 2x_h$ then $2 - 4u \le s_h \le \mathrm{RN}(2x_h) = 2x_h \le 4 - 4u$ and $|s_\ell| \le 2u$. We have,

$$t_h + t_\ell = x_\ell + y_\ell,$$

with $|x_\ell + y_\ell| \le 2u$, hence $|t_h| \le 2u$, and $|t_\ell| \le u^2$. Now, $|s_\ell + t_h| \le 4u$, so that $|c| \le 4u$, and $|\epsilon_1| \le 2u^2$. Since $s_h \ge 2 - 4u$ and $|c| \le 4u$, if $p \ge 3$ then Algorithm Fast2Sum introduces no error at line 4 of the algorithm. Therefore,

$$v_h + v_\ell = s_h + c \le 4 - 4u + 4u = 4,$$

so that $v_h \le 4$ and $|v_\ell| \le 2u$. Thus, $|t_\ell + v_\ell| \le 2u + u^2$. Hence, either $|t_\ell + v_\ell| < 2u$ and $|\epsilon_2| \le \frac{1}{2}\mathrm{ulp}(t_\ell + v_\ell) \le u^2$; or $2u \le t_\ell + v_\ell \le 2u + u^2$, in which case $w = \mathrm{RN}(t_\ell + v_\ell) = 2u$ and $|\epsilon_2| \le u^2$. In all cases $|\epsilon_2| \le u^2$. Also, $s_h \ge 2 - 4u$ and $|c| \le 4u$ imply $v_h \ge 2 - 8u$. And $|t_\ell + v_\ell| \le 2u + u^2$

implies $|w| \leq 2u$. Hence if $p \geq 3$ then Algorithm Fast2Sum introduces no error at line 6 of the algorithm.

All this gives

$$z_h + z_\ell = v_h + w = x + y + \eta,$$

with $|\eta| = |\epsilon_1 + \epsilon_2| \leq 3u^2$.

Since $x + y \geq (x_h - u) + (y_h - u) > 2 - 6u$, the relative error $|\eta|/(x+y)$ is upper-bounded by

$$\frac{3u^2}{2 - 6u},$$

The largest bound obtained in the various cases we have analyzed is

$$\frac{3u^2}{1 - 4u}.$$

Elementary calculus shows that for $u \in [0, 1/64]$ (i.e., $p \geq 6$) this is always less than $3u^2 + 13u^3$.

$\square$

The bound (3) is probably not optimal. The largest relative error we have obtain through many tests is around $2.25 \times 2^{-2p} = 2.25u^2$. An example is the input values given in Eq. (2), for which, with $p = 53$ (binary64 arithmetic), we obtain a relative error equal to $2.24999999999999956 \cdots \times 2^{-106}$.

4 Multiplication of a double-word number by a floating-point number

We first consider the following algorithm, suggested by Li et al [13]:

Algorithm 7 – DWTimesFP1(x_h, x_ℓ, y). Calculation of $(x_h, x_\ell) \times y$ in binary, precision-p, floating-point arithmetic.

```
(c_h, c_ℓ1) ← 2Prod(x_h, y)
c_ℓ2 ← RN(x_ℓ · y)
(t_h, t_ℓ1) ← Fast2Sum(c_h, c_ℓ2)
t_ℓ2 ← RN(t_ℓ1 + c_ℓ1)
(z_h, z_ℓ) ← Fast2Sum(t_h, t_ℓ2)
return (z_h, z_ℓ)
```

In [13, 14] (with more detail in the technical report [13], which is a preliminary version of the journal paper [14]), Li et Al. give a relative error bound $4 \cdot 2^{-106}$ for Algorithm 7 when the underlying floating-point arithmetic is binary64 (i.e., $p = 53$). Below, we prove an improved sharp relative error bound, even in the more general context of precision-p arithmetic. More precisely, we have:

Theorem 4.1. *If $p \geq 4$, the relative error of Algorithm 7 (DWTimesFP1) is bounded by $\frac{3}{2}u^2 + 4u^3$.*

Proof. One easily notices that if $x = 0$, or $y = 0$, or y is a power of 2, the obtained result is exact. Therefore, without loss of generality, we can assume $1 \leq x_h \leq 2-2u$ and $1+2u \leq y \leq 2-2u$. This gives $1+2u \leq x_h y \leq 4-8u+4u^2$, so that

$$1 + 2u \leq c_h \leq 4 - 8u, \tag{16}$$

and

$$|c_{\ell 1}| \leq \frac{1}{2}\mathrm{ulp}(4 - 8u) = 2u. \tag{17}$$

From $|x_\ell| \leq u$ and $y \leq 2 - 2u$ we deduce

$$|c_{\ell 2}| \leq 2u - 2u^2, \tag{18}$$

so that $\epsilon_1 = x_\ell y - c_{\ell 2}$ satisfies $|\epsilon_1| \leq u^2$. From (16) and (18) we deduce that Algorithm Fast2Sum introduces no error at line 3 of the algorithm, i.e., $t_h + t_{\ell 1} = c_h + c_{\ell 2}$. Also, we deduce that

$$1 = \mathrm{RN}(1 + 2u^2) \leq t_h \leq \mathrm{RN}(4 - 6u - 2u^2) = 4 - 8u, \tag{19}$$

and

$$|t_{\ell 1}| \leq \frac{1}{2}\mathrm{ulp}(4 - 8u) = 2u. \tag{20}$$

From (17) and (20), we obtain

$$|t_{\ell 2}| \leq \mathrm{RN}(4u) = 4u, \tag{21}$$

and we find that $\epsilon_2 = t_{\ell 2} - (t_{\ell 1} + c_{\ell 1})$ satisfies $|\epsilon_2| \leq 2u^2$. Define $\epsilon = \epsilon_2 - \epsilon_1$. Using (19) and (21), we deduce that Algorithm Fast2Sum introduces no error at line 5 of the algorithm. Therefore,

$$\begin{aligned} z_h + z_\ell &= t_h + t_{\ell 2} \\ &= t_h + t_{\ell 1} + c_{\ell 1} + \epsilon_2 \\ &= c_h + c_{\ell 2} + c_{\ell 1} + \epsilon_2 \\ &= x_h y + x_\ell y - \epsilon_1 + \epsilon_2 \\ &= xy + \epsilon. \end{aligned} \tag{22}$$

Hence the absolute error of Algorithm 7 is $|\epsilon| \leq |\epsilon_1| + |\epsilon_2| \leq 3u^2$. Let us now consider two possible cases:

1. If $x_h y \geq 2$, then $xy \geq x_h(1 - u)y \geq 2 - 2u$. This leads to a relative error $|\epsilon/(xy)|$ bounded by

$$\frac{3u^2}{2 - 2u} = \frac{3}{2}u^2 + \frac{3}{2}u^3 + \frac{3}{2}u^4 + \cdots, \tag{23}$$

2. If $x_h y < 2$, which implies $|c_h| \leq 2$, we easily improve on some of the previously obtained bounds. We have, $|c_{\ell 1}| \leq u$, and $t_h \leq \mathrm{RN}(2+2u-2u^2) = 2$.

The case $t_h = 2$ is easily handled: (22) implies $xy = t_h + t_{\ell 2} - \epsilon \geq 2 - 4u - 3u^2$, and the relative error $|\epsilon/(xy)|$ is bounded by

$$\frac{3u^2}{2 - 4u - 3u^2} = \frac{3}{2}u^2 + 3u^3 + \frac{33}{4}u^4 + \cdots. \tag{24}$$

If $t_h < 2$ then $|t_{\ell 1}| \leq u$, $|t_{\ell 2}| \leq 2u$, and $|\epsilon_2| \leq u^2$. Hence, a first upper bound on $|\epsilon|$ is $2u^2$. However, some refinement is possible.

- first, if $|c_{\ell 2}| < u$ then $|\epsilon_1| \leq u^2/2$, which implies $|\epsilon| \leq 3u^2/2$;
- second, if $|c_{\ell 2}| \geq u$, then $c_{\ell 2}$ is a multiple of $\text{ulp}(u) = 2u^2$, so that $t_{\ell 1}$ is a multiple of $2u^2$. Also, since x_h and y are multiple of $2u$, $x_h y$ is a multiple of $4u^2$, so that $c_{\ell 1}$ is a multiple of $4u^2$. Hence, $t_{\ell 1} + c_{\ell 1}$ is a multiple of $2u^2$ of absolute value less than or equal to $2u$. This implies that $t_{\ell 1} + c_{\ell 1}$ is a FP number, hence $\text{RN}(t_{\ell 1} + c_{\ell 1}) = t_{\ell 1} + c_{\ell 1}$ and $\epsilon_2 = 0$.

Therefore, when $t_h < 2$, $|\epsilon|$ is upper-bounded by $3u^2/2$ so that the relative error $|\epsilon/(xy)|$ is bounded by

$$\frac{\frac{3}{2}u^2}{(1-u)(1+2u)} \leq \frac{3}{2}u^2. \tag{25}$$

The largest of the three bounds (23), (24), and (25) is the second one. It is less than $\frac{3}{2}u^2 + 4u^3$ as soon as $u \leq 1/16$. This proves the theorem. $\square$

The bound given by Theorem 4.1 is very sharp. For instance, in binary32 arithmetic ($p = 24$), with $x_h = 8388609$, $x_\ell = 4095/8192$, and $y = 8389633$, the relative error of Algorithm 7 is $1.4993282\cdots \times 2^{-48}$.

In Bailey's QD library [8] as well as in Briggs' library [3], another algorithm (Algorithm 8 below) is suggested for multiplying a double-word number by a floating-point number.

Algorithm 8 – DWTimesFP2(x_h, x_ℓ, y). Algorithm for computing $(x_h, x_\ell) \times y$ in binary, precision-p, floating-point arithmetic, implemented in the QD library.

1: $(c_h, c_{\ell 1}) \leftarrow \text{2Prod}(x_h, y)$
2: $c_{\ell 2} \leftarrow \text{RN}(x_\ell \cdot y)$
3: $c_{\ell 3} \leftarrow \text{RN}(c_{\ell 1} + c_{\ell 2})$
4: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(c_h, c_{\ell 3})$
5: **return** (z_h, z_ℓ)

Algorithm 8 is faster than Algorithm 7 (we save one call to Fast2Sum), but it is less accurate: there are input values for which the error attained using Algorithm 8 is larger than the bound given by Theorem 4.1. An example with $p = 53$ is $x_h = 4525788557405064$, $x_\ell = 8595672275350437/2^{54}$, and $y = 5085664955107621$, for which the relative error is $2.517\cdots \times 2^{-106}$.

Hence, the relative error bound we are going to prove for Algorithm 8 is necessarily larger than the one we had for Algorithm 7. More precisely, we have:

Theorem 4.2. *If $p \geq 3$, the relative error of Algorithm 8 (DWTimesFP2) is less than or equal to $3u^2$.*

Proof. The proof is very similar to (in fact, simpler than) the proof of Theorem 4.1. Without loss of generality, we can assume $1 \leq x_h \leq 2 - 2u$ and $1 \leq y \leq 2 - 2u$. Since the analysis of the case $y = 1$ is straightforward, we even assume $1 + 2u \leq y \leq 2 - 2u$. This gives $1 + 2u \leq x_h y \leq 4 - 8u + 4u^2$, so that $1 + 2u \leq c_h \leq 4 - 8u$ and $|c_{\ell 1}| \leq 2u$. From $|x_\ell| \leq u$ and $y \leq 2 - 2u$ we deduce $|c_{\ell 2}| \leq 2u - 2u^2$, so that $\epsilon_1 = x_\ell y - c_{\ell 2}$ satisfies $|\epsilon_1| \leq u^2$.

Now, $|c_{\ell 1} + c_{\ell 2}| \leq 4u - 2u^2$, hence $|c_{\ell 3}| \leq 4u$, and $c_{\ell 3} = c_{\ell 1} + c_{\ell 2} + \epsilon_2$, with $|\epsilon_2| \leq 2u^2$. From $|c_{\ell 3}| \leq 4u$ and $c_h \geq 1 + 2u$ we deduce that Algorithm Fast2Sum introduces no error at line 4 of the algorithm.

Hence,

$$z_h + z_\ell = c_h + c_{\ell 3} = xy - \epsilon_1 + \epsilon_2,$$

and $| - \epsilon_1 + \epsilon_2| \leq 3u^2$. Since $xy \geq (x_h - u)y \geq (1 - u)(1 + 2u) \geq 1$, we deduce that the relative error of Algorithm 8 is less than $3u^2$. □

If an FMA instruction is available, we can improve Algorithm 8 by merging lines 2 and 3 of the algorithm, and obtain Algorithm 9:

Algorithm 9 – DWTimesFP3(x_h, x_ℓ, y). Algorithm for computing $(x_h, x_\ell) \times y$ in binary, precision-p, floating-point arithmetic, assuming an FMA instruction is available.

1: $(c_h, c_{\ell 1}) \leftarrow \text{2Prod}(x_h, y)$
2: $c_{\ell 3} \leftarrow \text{RN}(c_{\ell 1} + x_\ell y)$
3: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(c_h, c_{\ell 3})$
4: **return** (z_h, z_ℓ)

This results in a better error bound:

Theorem 4.3. *If $p \geq 3$, the relative error of Algorithm 9 (DWTimesFP3) is less than or equal to $2u^2$.*

The proof is very similar to the proof of Theorem 4.2, so we omit it. The bound provided by Theorem 4.3 is sharp. For instance, in binary64 arithmetic ($p = 53$), we attain error $1.984 \cdots \times 2^{-106}$ for $x_h = 4505619370757448$, $x_\ell = -9003265529542491/2^{54}$, and $y = 4511413997183120$.

5 Multiplication of two double-word numbers

Algorithm 10 below was first suggested by Dekker (under the name *mul2* in [5]). It has been implemented in the QD library [8] and in Briggs' library [3] for multiplying two double-word numbers.

Algorithm 10 – **DWTimesDW1**$(x_h, x_\ell, y_h, y_\ell)$. Algorithm for computing $(x_h, x_\ell) \times (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic, implemented in the QD library.

1: $(c_h, c_{\ell 1}) \leftarrow \text{2Prod}(x_h, y_h)$
2: $t_{\ell 1} \leftarrow \text{RN}(x_h \cdot y_\ell)$
3: $t_{\ell 2} \leftarrow \text{RN}(x_\ell \cdot y_h)$
4: $c_{\ell 2} \leftarrow \text{RN}(t_{\ell 1} + t_{\ell 2})$
5: $c_{\ell 3} \leftarrow \text{RN}(c_{\ell 1} + c_{\ell 2})$
6: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(c_h, c_{\ell 3})$
7: **return** (z_h, z_ℓ)

Dekker proved a relative error bound $11u^2$. We are going to show:

Theorem 5.1. *If $p \geq 4$, the relative error of Algorithm 10 (DWTimesDW1) is less than or equal to $7u^2/(1+u)^2 < 7u^2$.*

In the proof of Theorem 5.1, we will use the following lemma:

Lemma 5.2. *Let a and b be two positive real numbers. If $ab \leq 2$ then $a + b \leq 2\sqrt{2}$.*

The proof of the lemma is straightforward calculus. Let us focus on the proof of Theorem 5.1.

Proof. Without loss of generality, we assume that $1 \leq x_h \leq 2 - 2u$ and $1 \leq y_h \leq 2 - 2u$. We have $x_h y_h < 4$, and

$$c_h + c_{\ell 1} = x_h y_h,$$

with $|c_{\ell 1}| \leq 2u$. We also have

$$t_{\ell 1} = x_h y_\ell + \epsilon_1,$$

with $|x_h y_\ell| \leq 2u - 2u^2$, so that $|t_{\ell 1}| \leq 2u - 2u^2$ and $|\epsilon_1| \leq u^2$; and

$$t_{\ell 2} = x_\ell y_h + \epsilon_2,$$

with $|x_\ell y_h| \leq 2u - 2u^2$, so that $|t_{\ell 2}| \leq 2u - 2u^2$ and $|\epsilon_2| \leq u^2$. Now, we have

$$c_{\ell 2} = t_{\ell 1} + t_{\ell 2} + \epsilon_3,$$

with $|t_{\ell 1} + t_{\ell 2}| \leq 4u - 4u^2$, which implies $|c_{\ell 2}| \leq 4u - 4u^2$ and $|\epsilon_3| \leq 2u^2$. We finally obtain

$$c_{\ell 3} = c_{\ell 1} + c_{\ell 2} + \epsilon_4,$$

and, from $|c_{\ell 1} + c_{\ell 2}| \leq 6u - 4u^2$, we deduce $|c_{\ell 3}| \leq 6u$ and $|\epsilon_4| \leq 4u^2$. Since $c_h \geq 1$, Algorithm Fast2Sum introduces no error at line 6 of the algorithm.

Therefore,

$$\begin{array}{rcl} z_h + z_\ell & = & c_h + c_{\ell 3} \\ & = & (x_h y_h - c_{\ell 1}) + c_{\ell 1} + c_{\ell 2} + \epsilon_4 \\ & = & x_h y_h + t_{\ell 1} + t_{\ell 2} + \epsilon_3 + \epsilon_4 \\ & = & x_h y_h + x_h y_\ell + x_\ell y_h + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 \\ & = & xy - x_\ell y_\ell + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 \\ & = & xy + \eta, \end{array} \tag{26}$$

with $|\eta| \leq u^2 + |\epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4| \leq 9u^2$. Let us consider the following cases.

- if $x_h y_h > 2$ then, since $x \geq x_h(1-u)$ and $y \geq y_h(1-u)$, the relative error is bounded by
$$\frac{9u^2}{2(1-u)^2}. \tag{27}$$

- If $x_h y_h \leq 2$ then $|c_{\ell 1}| \leq u$. Furthermore, Lemma 5.2 implies
$$x_h + y_h \leq 2\sqrt{2}. \tag{28}$$
We have,
$$|t_{\ell 1}| = |\mathrm{RN}(x_h y_\ell)| \leq \mathrm{RN}(x_h u) = x_h u,$$
and, similarly, $|t_{\ell 2}| \leq y_h u$, so that, using (28),
$$|t_{\ell 1} + t_{\ell 2}| \leq x_h u + y_h u \leq 2\sqrt{2}u.$$
Therefore, $c_{\ell 2}$ now satisfies
$$|c_{\ell 2}| \leq |t_{\ell 1} + t_{\ell 2}| + |\epsilon_3| \leq 2\sqrt{2}u + 2u^2.$$
We now deduce
$$|c_{\ell 1} + c_{\ell 2}| \leq u \cdot (2\sqrt{2} + 1) + 2u^2 \leq 4u$$
(as soon as $u \leq 1/16$, i.e., $p \geq 4$). Therefore, $|\epsilon_4| \leq 2u^2$. In (26), this results in $|\eta| \leq 7u^2$ instead of $9u^2$. Notice that if $x_h = 1$ or $y_h = 1$ then, either $\epsilon_1 = 0$ or $\epsilon_2 = 0$, which results in a significantly smaller bound for $|\eta|$. So we can assume that $x_h \geq 1+2u$ (hence, $x > 1+u$) and $y_h \geq 1+2u$ (hence, $y > 1+u$). Therefore the relative error is bounded by
$$\frac{7u^2}{(1+u)^2} < 7u^2. \tag{29}$$
If $p \geq 4$ the bound (27) is less than the bound (29). This proves the theorem.

□

The bound $7u^2$ provided by Theorem 5.1 is probably too pessimistic. The largest relative error we have encountered in our tests was 4.9916×2^{-106}, obtained for $p = 53$, $x_h = 4508231565242345$, $x_\ell = -9007199254524053/2^{54}$, $y_h = 4504969740576150$, and $y_\ell = -4503599627273753/2^{53}$. In *binary32* arithmetic ($p = 24$), the largest error obtained in our tests was 4.947×2^{-48}, for $x_h = 8399376$, $x_\ell = 16763823/2^{25}$, $y_h = 8414932$, and $y_\ell = 16756961/2^{25}$.

Now, if a fused multiply-add instruction (FMA) is available, we can slightly improve Algorithm 10, both in terms of speed and accuracy, by merging lines 3 and 4. Consider Algorithm 11 below.

Algorithm 11 – **DWTimesDW2**$(x_h, x_\ell, y_h, y_\ell)$. Algorithm for computing $(x_h, x_\ell) \times (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic, assuming an FMA instruction is available.

1: $(c_h, c_{\ell 1}) \leftarrow \text{2Prod}(x_h, y_h)$
2: $t_\ell \leftarrow \text{RN}(x_h \cdot y_\ell)$
3: $c_{\ell 2} \leftarrow \text{RN}(t_\ell + x_\ell y_h)$
4: $c_{\ell 3} \leftarrow \text{RN}(c_{\ell 1} + c_{\ell 2})$
5: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(c_h, c_{\ell 3})$
6: **return** (z_h, z_ℓ)

We have,

Theorem 5.3. *If $p \geq 5$, the relative error of Algorithm 11 (DWTimesDW2) is less than or equal to*

$$\frac{6u^2 + \frac{1}{2}u^3}{(1+u)^2} < 6u^2.$$

The proof is very similar to (in fact, simpler than) the proof of Theorem 5.1, and follows the same structure, so omit it.

We do not know if the bound given by Theorem 5.3 is sharp. The largest relative error we have encountered during our intensive tests was, for *binary64* ($p = 53$), 4.9433×2^{-106}, obtained for $x_h = 4515802244422058$, $x_\ell = -2189678420952711/2^{52}$, $y_h = 4503988428047019$, and $y_\ell = -2248477851812015/2^{52}$. In *binary32* arithmetic ($p = 24$), the largest error obtained in our tests was 4.936×2^{-48}, for $x_h = 8404039$, $x_\ell = -8284843/2^{24}$, $y_h = 8409182$, and $y_\ell = -4193899/2^{23}$.

The accuracy of the multiplication of two double-word numbers can be improved even more, by also computing the partial product $x_\ell y_\ell$. This gives Algorithm 12 below.

Algorithm 12 – **DWTimesDW3**$(x_h, x_\ell, y_h, y_\ell)$. Algorithm for computing $(x_h, x_\ell) \times (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic, assuming an FMA instruction is available.

1: $(c_h, c_{\ell 1}) \leftarrow \text{2Prod}(x_h, y_h)$
2: $t_{\ell 0} \leftarrow \text{RN}(x_\ell \cdot y_\ell)$
3: $t_{\ell 1} \leftarrow \text{RN}(x_h \cdot y_\ell + t_{\ell 0})$
4: $c_{\ell 2} \leftarrow \text{RN}(t_{\ell 1} + x_\ell \cdot y_h)$
5: $c_{\ell 3} \leftarrow \text{RN}(c_{\ell 1} + c_{\ell 2})$
6: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(c_h, c_{\ell 3})$
7: **return** (z_h, z_ℓ)

We have,

Theorem 5.4. *If $p \geq 4$, the relative error of Algorithm 12 (DWTimesDW3) is less than or equal to*

$$\frac{5u^2 + \frac{1}{2}u^3}{(1+u)^2} < 5u^2.$$

The proof is very similar to the proof of Theorem 5.1, and follows the same structure, so we omit it. We do not know if the bound given by Theorem 5.4 is sharp. The largest relative error we have encountered n intensive tests was (for $p = 53$) 3.936×2^{-106}, obtained for $x_h = 4510026974538724$, $x_\ell = 4232862152422029/2^{53}$, $y_h = 4511576932111935$, and $y_\ell = 2250098448199619/2^{52}$.

6 Division of a double-word number by a floating-point number

The algorithm suggested by Li et al. in [13] for dividing a double-word number by a floating-point number is similar to Algorithm 13 below.

Algorithm 13 – **DWDivFP1**(x_h, x_ℓ, y). Calculation of $(x_h, x_\ell) \div y$ in binary, precision-p, floating-point arithmetic.

1: $t_h \leftarrow \text{RN}(x_h / y)$
2: $(\pi_h, \pi_\ell) \leftarrow \text{2Prod}(t_h, y)$
3: $(\delta_h, \delta') \leftarrow \text{2Sum}(x_h - \pi_h)$
4: $\delta'' \leftarrow \text{RN}(x_\ell - \pi_\ell)$
5: $\delta_\ell \leftarrow \text{RN}(\delta' + \delta'')$
6: $\delta \leftarrow \text{RN}(\delta_h + \delta_\ell)$
7: $t_\ell \leftarrow \text{RN}(\delta / y)$
8: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(t_h, t_\ell)$
9: **return** (z_h, z_ℓ)

Algorithm 13 can be simplified. We have $t_h = (x_h/y)(1 + \epsilon_0)$ and $\pi_h =$

$t_h y(1+\epsilon_1)$, with $|\epsilon_0|, |\epsilon_1| \le u$. Hence,

$$(1-u)^2 x_h \le \pi_h \le (1+u)^2 x_h.$$

Therefore, as soon as $p \ge 2$ (i.e., $u \le 1/4$), π_h is within a factor 2 from x_h. Sterbenz Lemma (Lemma 1.2) therefore implies that $x_h - \pi_h$ is a floating-point number. As a consequence, we always have $\delta_h = x_h - \pi_h$, $\delta' = 0$, line 3 of the algorithm can be replaced by a simple subtraction, and we always have $\delta_\ell = \delta'' = \mathrm{RN}(x_\ell - \pi_\ell)$. Therefore, the significantly simpler Algorithm 14, below, always returns the same result as Algorithm 13.

Algorithm 14 – DWDivFP2(x_h, x_ℓ, y). Calculation of $(x_h, x_\ell) \div y$ in binary, precision-p, floating-point arithmetic.

1: $t_h \leftarrow \mathrm{RN}(x_h/y)$
2: $(\pi_h, \pi_\ell) \leftarrow \mathrm{2Prod}(t_h, y)$
3: $\delta_h \leftarrow \mathrm{RN}(x_h - \pi_h) = x_h - \pi_h$ (exact operation)
4: $\delta_\ell \leftarrow \mathrm{RN}(x_\ell - \pi_\ell)$
5: $\delta \leftarrow \mathrm{RN}(\delta_h + \delta_\ell)$
6: $t_\ell \leftarrow \mathrm{RN}(\delta/y)$
7: $(z_h, z_\ell) \leftarrow \mathrm{Fast2Sum}(t_h, t_\ell)$
8: **return** (z_h, z_ℓ)

The authors of [13] claim that their binary64 (i.e., $p = 53$) implementation of Algorithm 13 has a relative error bounded by $4 \cdot 2^{-106}$. That bound can be slightly improved. We are going to prove:

Theorem 6.1. *If $p \ge 4$, the relative error of Algorithm 14 (DWDivFP2) is bounded by*

$$\frac{7}{2}u^2.$$

That bound also holds for Algorithm 13 (DWDivFP1) since both algorithms return the same result. The bound is reasonably sharp: in practice the largest relative errors we have found in calculations were slightly less than $3u^2$. For instance, for $p = 53$, relative error $2.95157083\cdots \times 2^{-106}$ is attained for $x_h = 4588860379563012$, $x_\ell = -4474949195791253/2^{53}$, and $y = 4578284000230917$.

Before proving Theorem 6.1, let us prove the following Lemma.

Lemma 6.2. *Assume a radix-2, precision-p, FP arithmetic. Let a and b be FP numbers between* 1 *and* 2. *Let* $u = 2^{-p}$. *The distance between* $\mathrm{RN}(a/b)$ *and* a/b *is less than*

$$\begin{cases} u - 2u^2/b & \text{if } a/b \ge 1; \\ u/2 - u^2/b & \text{otherwise.} \end{cases}$$

Proof. It suffices to estimate the smallest possible distance between a/b and a "midpoint" (i.e., a number exactly halfway between two consecutive FP numbers). Let $a = M_a \cdot 2^{-p+1}$, $b = M_b \cdot 2^{-p+1}$, with $2^{p-1} \le M_a, M_b \le 2^p - 1$.

- If $a/b \geq 1$, a midpoint μ between 1 and 2 has the form $(2M_\mu + 1)/2^p$, with $2^{p-1} \leq M_\mu \leq 2^p - 1$. We have

$$\left|\frac{a}{b} - \mu\right| = \left|\frac{2^p M_a - M_b(2M\mu + 1)}{2^p M_b}\right|.$$

  The numerator, $2^p M_a - M_b(2M\mu + 1)$, of that fraction cannot be zero: since $2M\mu + 1$ is odd, having $2^p M_a = M_b(2M\mu + 1)$ would require M_b to be a multiple of 2^p, which is impossible since $M_b \leq 2^p - 1$. Hence that numerator has absolute value at least 1. Hence

$$\left|\frac{a}{b} - \mu\right| \geq \frac{1}{2^p M_b} = \frac{2u^2}{b}.$$

- If $a/b < 1$ the proof is similar. The only change is that a midpoint is of the form $(2M_\mu + 1)/2^{p+1}$.

□

In a recent paper (see [10, Table 1]), a similar bound is given for floating-point division. It could be used instead of Lemma 6.2, but we included the lemma for completeness. Let us now prove Theorem 6.1.

Proof. The case where y is a power of 2 is straightforward, so we omit it. Without loss of generality, we assume $1 \leq x_h \leq 2 - 2u$, so that $|x_\ell| \leq u$; and $1 + 2u \leq y \leq 2 - 2u$. Therefore, we have

$$\frac{1}{2 - 2u} \leq \frac{x_h}{y} \leq \frac{2 - 2u}{1 + 2u}. \tag{30}$$

The quotient $1/(2 - 2u)$ is always larger than $1/2 + u/2$, and, as soon as $p \geq 4$, $(2 - 2u)/(1 + 2u)$ is less than $2 - 5u$. Therefore

$$\frac{1}{2} + u \leq t_h = \mathrm{RN}\left(\frac{x_h}{y}\right) \leq 2 - 6u. \tag{31}$$

We have already proved, when discussing Algorithm 13, that $\delta_h = x_h - \pi_h$.

Define ϵ_1, ϵ_2, and ϵ_3 as the errors committed at steps 4, 5, and 6 of the algorithm. More precisely,

$$\epsilon_1 = \delta_\ell - (x_\ell - \pi_\ell),$$

$$\epsilon_2 = \delta - (\delta_h + \delta_\ell),$$

and

$$\epsilon_3 = t_\ell - \frac{\delta}{y}.$$

Also define $\epsilon = \epsilon_1 + \epsilon_2$. We have,

$$\begin{aligned} \delta &= (x_h - \pi_h) + (x_\ell - \pi_\ell) + \underbrace{\delta_\ell - (x_\ell - \pi_\ell)}_{\epsilon_1} + \underbrace{\delta - (\delta_h + \delta_\ell)}_{\epsilon_2}, \\ &= x - t_h y + \epsilon, \end{aligned} \tag{32}$$

and

$$t_h + t_\ell = \frac{x}{y} + \frac{\epsilon}{y} + \epsilon_3. \tag{33}$$

Lemma 6.2 implies

$$\left| t_h - \frac{x_h}{y} \right| \leq u - \frac{2u^2}{y}, \tag{34}$$

so that

$$|t_h y - x_h| \leq uy - 2u^2 \leq 2u - 4u^2, \tag{35}$$

and

$$t_h y \leq |t_h y - x_h| + |x_h| \leq (2u - 4u^2) + (2 - 2u) = 2 - 4u^2. \tag{36}$$

This implies

$$|\pi_\ell| \leq \frac{1}{2}\mathrm{ulp}(t_h y) = u.$$

From this and $|x_\ell| \leq u$, we deduce $|\delta_\ell| \leq 2u$ and

$$|\epsilon_1| \leq u^2. \tag{37}$$

Also, (35) implies $x_h - 2u \leq t_h y \leq x_h + 2u$, and, since $x_h - 2u$ and $x_h + 2u$ are floating-point numbers,

$$x_h - 2u \leq \pi_h = \mathrm{RN}(t_h y) \leq x_h + 2u. \tag{38}$$

Therefore, $|\delta_h| = |x_h - \pi_h| \leq 2u$. Now, since $|\delta_h + \delta_\ell| \leq 4u$, we obtain

$$|\epsilon_2| \leq 2u^2, \tag{39}$$

and $|\delta| \leq 4u$, which implies $|\delta/y| < 4u$ and $|t_\ell| \leq 4u$. Using this and (31) we deduce that Fast2Sum returns a correct result at Line 7 of the algorithm, i.e., $z_h + z_\ell = t_h + t_\ell$. Let us now consider two possible cases.

1. **If $\boldsymbol{x \geq y}$, which implies $\boldsymbol{x_h \geq y}$ and $\boldsymbol{t_h \geq 1}$,** then (38) implies

$$\pi_h = \mathrm{RN}(t_h y) \in \{x_h - 2u, x_h, x_h + 2u\}.$$

One might think that π_h could be $x_h - u$ in the case $x_h = 1$, but $x_h = 1$ is not compatible with our assumptions, $x \geq y$ and $y \geq 1 + 2u$. In all cases, $\delta_h = x_h - \pi_h \in \{-2u, 0, 2u\}$.

- If $\delta_h = -2u$ then $x_h - \pi_h = -2u$, so that

  $$\pi_\ell = t_h y - \pi_h = (t_h y - x_h) + (x_h - \pi_h) \leq (2u - 4u^2) + (-2u) = -4u^2.$$

  Hence $-u \leq \pi_\ell \leq -4u^2$, so that $-u + 4u^2 \leq \delta_\ell \leq 2u$.

– if $u \le \delta_\ell \le 2u$ then Sterbenz's Lemma implies that $\delta_h + \delta_\ell$ is a FP number, so that $\epsilon_2 = 0$;

– if $-u + 4u^2 \le \delta_\ell < u$ then the bound (37) is improved and becomes $|\epsilon_1| \le u^2/2$.

Hence, if $\delta_h = -2u$, we have $|\epsilon_1 + \epsilon_2| \le 5u^2/2$.

- Symmetrically, if $\delta_h = 2u$ we also have $|\epsilon_1 + \epsilon_2| \le 5u^2/2$.
- If $\delta_h = 0$ then $|\delta_h + \delta_\ell| = |\delta_\ell| \le 2u$. Since there is no error when adding δ_h and δ_ℓ, we have $\epsilon_2 = 0$.

Hence, we always have $|\epsilon| = |\epsilon_1 + \epsilon_2| \le 5u^2/2$. From (32) we deduce

$$\frac{\delta}{y} = \frac{x}{y} - t_h + \frac{\epsilon}{y}. \tag{40}$$

Let us now bound the error committed when rounding δ/y. For that purpose, let us first try to find a reasonably tight bound on δ/y. We have

$$\begin{array}{rcl} \left|\frac{x}{y} - t_h\right| & \le & \left|\frac{x_h}{y} - t_h\right| + \left|\frac{x_\ell}{y}\right| \\ & \le & u - \frac{2u^2}{y} + \frac{u}{y}, \end{array}$$

and

$$\left|\frac{\epsilon}{y}\right| \le \frac{5u^2}{2y}.$$

Therefore, using (40),

$$\left|\frac{\delta}{y}\right| \le u + \frac{u^2}{2y} + \frac{u}{y} \le u + \frac{u^2}{2(1+2u)} + \frac{u}{1+2u} = \frac{4u+5u^2}{2+4u} < 2u.$$

Hence $|t_\ell| \le 2u$, and

$$|\epsilon_3| = \left|t_\ell - \frac{\delta}{y}\right| = \left|\mathrm{RN}\left(\frac{\delta}{y}\right) - \frac{\delta}{y}\right| \le u^2.$$

Using (33), we finally conclude that

$$\left|(z_h + z_\ell) - \frac{x}{y}\right| = \left|(t_h + t_\ell) - \frac{x}{y}\right| \le \frac{5u^2}{2y} + u^2, \tag{41}$$

so that the relative error is bounded by

$$\frac{y}{x}\left(\frac{5u^2}{2y} + u^2\right) \le \frac{5u^2}{2x} + \frac{u^2 y}{x} \le \frac{7u^2}{2} = 3.5u^2,$$

since $y/x \le 1$.

2. If $x < y$, which implies $x_h \leq y$ and $t_h \leq 1$.

The case $x_h = y$ is easily handled. It leads to $t_h = 1$, $\pi_h = x_h$, $\pi_\ell = 0$, $\delta = x_\ell$, and $z_h + z_\ell = t_h + t_\ell = x/y + \eta$, with $|\eta| \leq u|x_\ell|/y \leq u^2 x/y$.

We can now focus on the case $x_h < y$. It implies $x_h \leq y - 2u$, so that $x \leq y - u$. This gives $x/y \leq 1 - u/y \leq 1 - u/(2-2u) < 1 - u/2$, which implies $t_h \leq 1 - u$.

The remainder of the proof is very similar to the proof of the case $x > y$, so we give it with less details. Lemma 6.2 makes it possible to improve on the bound (34): we now have,

$$\left| t_h - \frac{x_h}{y} \right| \leq \frac{u}{2} - \frac{u^2}{y},$$

so that $|t_h y - x_h| \leq (u/2) \cdot y - u^2 \leq u - 2u^2$. This implies $\pi_h = \mathrm{RN}(t_h y) \in \{x_h - u, x_h\}$, so that $\delta_h \in \{0, u\}$. The case $\mathrm{RN}(t_h y) = x_h - u$ (i.e., $\delta_h = u$) being possible only when $x_h = 1$. We again consider two possible cases

- If $\delta_h = 0$ then $\epsilon_2 = 0$.
- If $\delta_h = u$ (which implies $x_h = 1$), then, since $\pi_h = 1 - u$, we have $t_h y < 1$. This implies $|\pi_\ell| \leq \frac{1}{2}\mathrm{ulp}(t_h y) \leq \frac{u}{2}$. We also have

  $$\pi_\ell = t_h y - \pi_h = (t_h y - x_h) + (x_h - \pi_h) \geq (-u + 2u^2) + u = 2u^2,$$

  hence,

  $$2u^2 \leq \pi_\ell \leq \frac{u}{2}. \tag{42}$$

  Also, $x_h = 1$ implies $-\frac{u}{2} \leq x_\ell \leq u$. Therefore $-u \leq x_\ell - \pi_\ell \leq u - 2u^2$, so that $-u \leq \delta_\ell \leq u - 2u^2$, and $|\epsilon_1| \leq u^2/2$. From

  $$0 \leq \delta_\ell + \delta_h \leq 2u - 2u^2,$$

  we deduce $|\epsilon_2| \leq u^2$.

Hence, we always have $|\epsilon| = |\epsilon_1 + \epsilon_2| \leq 3u^2/2$. We deduce from (32) that

$$\left|\frac{\delta}{y}\right| \leq \left|\frac{x_h}{y} - t_h\right| + \left|\frac{x_\ell}{y}\right| + \left|\frac{\epsilon}{y}\right| \leq \frac{u}{2} + \frac{u^2}{2y} + \frac{u}{y} < 2u,$$

so that $|t_\ell| \leq 2u$, and

$$|\epsilon_3| = \left| t_\ell - \frac{\delta}{y} \right| \leq u^2.$$

Using (33), we finally conclude that

$$\left| (z_h + z_\ell) - \frac{x}{y} \right| = \left| (t_h + t_\ell) - \frac{x}{y} \right| \leq \frac{3u^2}{2y} + u^2,$$

hence $z_h + z_\ell = t_h + t_\ell$ approximates x/y with a relative error bounded by

$$\frac{y}{x} \cdot \left(\frac{3u^2}{2y} + u^2 \right) \leq \frac{3u^2}{2x} + u^2 \frac{y}{x} \leq 3.5u^2.$$

$\square$

7 Division of two double-word numbers

The algorithm implemented in the QD library for dividing two double-word numbers is the following.

Algorithm 15 – **DWDivDW1**$(x_h, x_\ell, y_h, y_\ell)$. Calculation of $(x_h, x_\ell) \div (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic.

$t_h \leftarrow \mathrm{RN}(x_h/y_h)$
$(r_h, r_l) \leftarrow \mathrm{DWTimesFP1}(y_h, y_\ell, t_h)$ {approximation to $(y_h + y_\ell) \cdot t_h$ using Alg. 7}
$(\pi_h, \pi_\ell) \leftarrow \mathrm{2Sum}(x_h, -r_h)$
$\delta_h \leftarrow \mathrm{RN}(\pi_\ell - r_\ell)$
$\delta_\ell \leftarrow \mathrm{RN}(\delta_h + x_\ell)$
$\delta \leftarrow \mathrm{RN}(\pi_h + \delta_\ell)$
$t_\ell \leftarrow \mathrm{RN}(\delta/y_h)$
$(z_h, z_\ell) \leftarrow \mathrm{Fast2Sum}(t_h, t_\ell)$
return (z_h, z_ℓ)

Let us quickly analyze the beginning of Algorithm 15. This will lead us to suggest another algorithm, faster yet mathematically equivalent as soon as $p \geq 3$. Without loss of generality, we assume $x_h > 0$ and $y_h > 0$. Define ϵ_x and ϵ_y such that $x_h = x(1+\epsilon_x)$ and $y_h = y/(1+\epsilon_y)$. These two numbers ϵ_x and ϵ_y have an absolute value less than or equal to u. We have

$$t_h = \frac{x_h}{y_h}(1+\epsilon_0), \text{ with } |\epsilon_0| \leq u, \tag{43}$$

and, from Theorem 4.1,

$$r_h + r_\ell = t_h y(1+\eta), \text{ with } |\eta| \leq \frac{3}{2}u^2 + 4u^3 \tag{44}$$

There exists $|\epsilon_1| \leq u$ such that $r_h = (r_h + r_\ell)(1+\epsilon_1)$. This can be rewritten $r_\ell = -\epsilon_1(r_h + r_\ell)$, so that, using (44), $r_\ell = -\epsilon_1 t_h y(1+\eta)$. We finally obtain

$$\begin{array}{rcl} r_h & = & t_h y_h (1+\epsilon_y)(1+\epsilon_1)(1+\eta) \\ & = & x_h (1+\epsilon_y)(1+\epsilon_0)(1+\epsilon_1)(1+\eta), \end{array} \tag{45}$$

so that

$$(1-u)^3(1-2u^2)x_h \leq r_h \leq (1+u)^3\left(\frac{3}{2}u^2 + 4u^3\right)x_h,$$

from which we deduce

$$|x_h - r_h| \leq \left(3u + \frac{9}{2}u^2 + \frac{19}{2}u^3 + \frac{33}{2}u^4 + \frac{27}{2}u^5 + 4u^6\right) \cdot x_h,$$

which implies

$$|x_h - r_h| \leq (3u + 6u^2) \cdot x_h \tag{46}$$

as soon as $p \geq 3$. One easily checks that for $p \geq 3$ (i.e., $u \leq 1/8$), $3u + 6u^2$ is less than $1/2$. Hence, From Sterbenz Lemma (Lemma 1.2), the number $x_h - r_h$ is a floating-point number. Therefore the number π_ℓ obtained at line 3 of Algorithm 15 is always 0 and that line can be replaced by a simple, errorless, subtraction. This gives $\pi_h = x_h - r_h$, and $\delta_h = -r_\ell$. Hence, without changing the final result, we can replace Algorithm 15 by the simpler Algorithm 16, below.

Algorithm 16 – DWDivDW2$(x_h, x_\ell, y_h, y_\ell)$. Calculation of $(x_h, x_\ell) \div (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic: improved version of Algorithm 15. Useless operations have been removed. The result is exactly the same.

1: $t_h \leftarrow \mathrm{RN}(x_h/y_h)$
2: $(r_h, r_l) \leftarrow \mathrm{DWTimesFP1}(y_h, y_\ell, t_h)$ {approximation to $(y_h + y_\ell) \cdot t_h$ using Alg. 7}
3: $\pi_h \leftarrow \mathrm{RN}(x_h - r_h) = x_h - r_h$ (exact operation)
4: $\delta_\ell \leftarrow \mathrm{RN}(x_\ell - r_\ell)$
5: $\delta \leftarrow \mathrm{RN}(\pi_h + \delta_\ell)$
6: $t_\ell \leftarrow \mathrm{RN}(\delta/y_h)$
7: $(z_h, z_\ell) \leftarrow \mathrm{Fast2Sum}(t_h, t_\ell)$
8: **return** (z_h, z_ℓ)

If an FMA instruction is available, Algorithm 9 can be used at line 2 instead of Algorithm 7 without changing much the error bound provided by Theorem 7.1 below. We have

Theorem 7.1. *If $p \geq 7$, the relative error of Algorithms 15 (DWDivDW1) and 16 (DWDivDW2) is upper-bounded by $15u^2 + 56u^3$.*

Proof. For reasons of symmetry, we can assume that x and y are positive. We will use the results (43) to (46) obtained when analyzing the beginning of Algorithm 15. Assume $p \geq 7$. We have

$$\delta_\ell = (x_\ell - r_\ell)(1 + \epsilon_2), \text{ with } |\epsilon_2| \leq u,$$

We have $|x_\ell| \leq u \cdot x_h$ and $|r_\ell| \leq u \cdot r_h$, so that

$$\begin{array}{rcl} |x_\ell - r_\ell| & \leq & |x_\ell| + |r_\ell| \\ & \leq & u \cdot x_h + u \cdot r_h \\ & \leq & u \cdot x_h + u \cdot ((r_h - x_h) + x_h) \\ & \leq & u \cdot x_h + u \cdot (|r_h - x_h| + x_h) \, . \end{array}$$

Therefore, using (46),

$$|x_\ell - r_\ell| \leq u \cdot x_h + u \cdot \big((3u + 6u^2)x_h + x_h\big) \, ,$$

which gives

$$|x_\ell - r_\ell| \leq (2u + 3u^2 + 6u^3) \cdot x_h. \tag{47}$$

We have

$$\delta = (\pi_h + \delta_\ell)(1+\epsilon_3), \text{ with } |\epsilon_3| \le u,$$

so that

$$\begin{aligned} \delta &= x_h - r_h + x_\ell - r_\ell + (x_\ell - r_\ell)(\epsilon_2 + \epsilon_3 + \epsilon_2\epsilon_3) + (x_h - r_h)\cdot\epsilon_3, \\ &= x - (r_h + r_\ell) + \alpha\cdot x_h, \end{aligned}$$

with (using (46) and (47))

$$\begin{aligned} |\alpha| &\le (2u+3u^2+6u^3)(2u+u^2) + (3u+6u^2)u \\ &\le 7u^2 + 15u^3 \end{aligned} \tag{48}$$

as soon as $p \ge 4$. Hence $\delta = x - t_h y(1+\eta) + \alpha x_h$, so that

$$\frac{\delta}{y_h} = \frac{x - t_h y}{y}\cdot\frac{y}{y_h} - \frac{\eta t_h y}{y_h} + \alpha\frac{x_h}{y_h}. \tag{49}$$

The number $x - t_h y$ is equal to $x_h - t_h y_h + x_\ell - t_h y_\ell$. From (43), $x_h - t_h y_h$ is equal to $-x_h\epsilon_0$. Also, $|x_\ell|$ is less than or equal to $u x_h$, and

$$|t_h y_\ell| \le |u t_h y_h| \le u(1+u)x_h.$$

Hence,

$$|x - t_h y| \le x_h\cdot(u + u + u(1+u)) = x_h\cdot(3u+u^2). \tag{50}$$

From (49), we deduce

$$\begin{aligned} \tfrac{\delta}{y_h} &= \tfrac{x-t_h y}{y}\cdot(1+\epsilon_y) - \eta t_h(1+\epsilon_y) + \alpha\tfrac{x_h}{y_h}, \\ &= \tfrac{x-t_h y}{y} + \beta, \end{aligned} \tag{51}$$

with

$$\begin{aligned} |\beta| &= \left|\epsilon_y\cdot\tfrac{x-t_h y}{y} - t_h(1+\epsilon_y)\eta + \tfrac{x_h}{y_h}\right| \\ &\le u(3u+u^2)\tfrac{x_h}{y} + (1+u)(2u^2)\tfrac{x_h}{y_h} + (7u^2+15u^3)\tfrac{x_h}{y_h} \\ &\le u(3u+u^2)(1+u)\tfrac{x}{y} + (1+u)^3(2u^2)\tfrac{x}{y} + (7u^2+15u^3)(1+u)^2\tfrac{x}{y} \\ &= \left(12u^2 + 39u^3 + 44u^4 + 17u^5\right)\cdot\tfrac{x}{y}. \end{aligned} \tag{52}$$

Hence,

$$\begin{aligned} t_\ell &= \mathrm{RN}\left(\tfrac{\delta}{y_h}\right) \\ &= \tfrac{\delta}{y_h}(1+\epsilon_4) \text{ with } |\epsilon_4| \le u, \\ &= \left(\tfrac{x-t_h y}{y} + \beta\right)(1+\epsilon_4) \\ &= \tfrac{x-t_h y}{y} + \gamma, \end{aligned} \tag{53}$$

with

$$
\begin{array}{rcl}
|\gamma| & = & \left|\frac{x-t_h y}{y}\epsilon_4 + \beta + \epsilon_4\beta\right| \\
& \leq & \frac{x_h}{y}(3u+u^2)u + \beta + \beta u \\
& \leq & (3u+u^2)u(1+u)\frac{x}{y} + \beta + \beta u \\
& = & \left(15u^2 + 55u^3 + 84u^4 + 61u^5 + 17u^6\right) \cdot \frac{x}{y}.
\end{array}
\tag{54}
$$

Hence

$$t_h + t_\ell = \frac{x}{y} + \gamma.$$

Since we straightforwardly have

$$t_h \geq \frac{x}{y} \cdot (1-u)^3, \tag{55}$$

we deduce

$$|t_\ell| \leq \frac{x}{y} \cdot \left((15u^2 + 55u^3 + 84u^4 + 61u^5 + 17u^6) + (3u - 3u^2 + u^3)\right). \tag{56}$$

From (55) and (56) we easily deduce that as soon as $p \geq 4$ (i.e., $u \leq 1/16$), t_h is larger than $|t_\ell|$, so that Algorithm Fast2Sum introduces no error at line 7 of the algorithm. Therefore,

$$z_h + z_\ell = t_h + t_\ell = \frac{x}{y} + \gamma,$$

so that the relative error of Algorithm 16 (and Algorithm 15) is upper-bounded by

$$15u^2 + 55u^3 + 84u^4 + 61u^5 + 17u^6,$$

which is less than $15u^2 + 56u^3$ as soon as $p \geq 7$ (i.e., $u \leq 128$), which always holds in practice. □

The bound provided by Theorem 7.1 is almost certainly not optimal. However, during our intensive tests, we have encountered cases for which the relative error, although significantly less than the bound $15u^2 + 56u^3$ of Theorem 7.1, remains of a similar order of magnitude—i.e., more than half the bound. For instance, for $p = 53$, relative error $8.465 \cdots \times 2^{-106}$ is attained for $x_h = 4503607118141812$, $x_\ell = 4493737176494969/2^{53}$, $y_h = 4503600552333684$, and $y_\ell = -562937972998161/2^{50}$.

If an FMA instruction is available, one can design a more accurate algorithm. What makes it work is the following property (easy to prove, and common knowledge among the designers of Newton-Raphson-based division algorithms):

Property 7.2. *If y is a nonzero FP number, and if $t = \mathrm{RN}(1/y)$, then $yt - 1$ is a FP number.*

Proof. Without l.o.g., assume $1 \le y \le 2-2u$, which implies that y is a multiple of $2^{-p+1} = 2u$. The number $1/y$ is between $1/(2-2u) = 1/2 + u/2 + u^2/2 + \cdots$ and 1, so that t is between $1/2$ and 1, so that t is a multiple of $2^{-p} = u$. From

$$\frac{1-u}{y} \le t \le \frac{1+u}{y}$$

we deduce

$$-u \le 1 - yt \le u.$$

Hence, $1-yt$ is a multiple of 2^{-2p+1} of absolute value less than or equal to 2^{-p}, which implies that it is a FP number. $\square$

Now, let us suggest a new division algorithm, that makes use of that property.

Algorithm 17 – DWDivDW3$(x_h, x_\ell, y_h, y_\ell)$. Calculation of $(x_h, x_\ell) \div (y_h, y_\ell)$ in binary, precision-p, floating-point arithmetic: more accurate algorithm that requires the availability of an FMA instruction

$t_h \leftarrow \mathrm{RN}(1/y_h)$
$r_h \leftarrow (1 - y_h t_h) = 1 - y_h t_h$ (exact operation)
$r_\ell \leftarrow -\mathrm{RN}(y_\ell \cdot t_h)$
$(e_h, e_\ell) \leftarrow \mathrm{Fast2Sum}(r_h, r_\ell)$
$(\delta_h, \delta_\ell) \leftarrow \mathrm{DWTimesFP3}(e_h, e_\ell, t_h)$ {Approximation to $(e_h + e_\ell) \cdot t_h$ with relative error $\le 2u^2$ using Algorithm 9}
$(m_h, m_\ell) \leftarrow \mathrm{DWPlusFP}(\delta_h, \delta_\ell, t_h)$ {Approximation to $\delta_h + \delta_\ell + t_h$ with relative error $\le 2u^2 + 5u^3$ using Algorithm 4}
$(z_h, z_\ell) \leftarrow \mathrm{DWTimesDW2}(x_h, x_\ell, m_h, m_\ell)$ {Approximation to $(x_h + x_\ell)(m_h + m_\ell)$ with relative error $\le 5u^2$ using Algorithm 12}
return (z_h, z_ℓ)

We have,

Theorem 7.3. *As soon as* $p \ge 14$, *and if* $y \ne 0$, *the relative error of Algorithm 17 (DWDivDW3) is bounded by* $9.8u^2$.

Proof. Roughly speaking, Algorithm 17 first approximates $1/y$ by $t_h = \mathrm{RN}(1/y_h)$, then improves that approximation to $1/y$ by performing one step of Newton-Raphson iteration, and then multiplies the obtained approximation (m_h, m_ℓ) by x.

Without loss of generality, we assume $1 \le y_h \le 2 - 2u$, so that $1/2 \le t_h \le 1$. We have

$$\left| t_h - \frac{1}{y_h} \right| \le \frac{u}{2},$$

and (from Property 7.2)

$$r_h = 1 - y_h t_h.$$

We also easily check that

$$\left(t_h(2-yt_h)-\frac{1}{y}\right)=-y\cdot\left(t_h-\frac{1}{y}\right)^2. \tag{57}$$

Now, from $|y_\ell|\le u$ and $|t_h|\le 1$, we deduce $|y_\ell t_h|\le u$, so that $|r_\ell|\le u$, and $|r_\ell+y_\ell t_h|\le u^2/2$. This gives

$$e_h+e_\ell=r_h+r_\ell=1-y_ht_h-y_\ell t_h+\eta,\ \text{with } |\eta|\le\frac{u^2}{2}. \tag{58}$$

Also, since $|y_ht_h-1|=y_h\cdot|t_h-1/y_h|\le u$, we have $|r_h|\le u$, hence $|r_h+r_\ell|\le 2u$. This implies $|e_h|\le 2u$ and $|e_\ell|\le u^2$. Define $e=e_h+e_\ell=r_h+r_\ell$, we have $|e|\le 2u$.

Now, from Theorem 4.3, we have

$$\delta_h+\delta_\ell=et_h(1+\omega_1),\ \text{with } |\omega_1|\le 2u^2, \tag{59}$$

and from Theorem 2.2, we have

$$m_h+m_\ell=(t_h+\delta_h+\delta_\ell)(1+\omega_2),\ \text{with } |\omega_2|\le 2u^2+5u^3. \tag{60}$$

Combining (59) and (60), we obtain

$$\begin{array}{rcl} m_h+m_\ell &=& (t_h+et_h(1+\omega_1))(1+\omega_2)\\ &=& t_h+et_h+et_h\omega_1+\omega_2t_h+\omega_2et_h+\omega_2\omega_1et_h\\ &=& t_h+et_h+\alpha t_h, \end{array} \tag{61}$$

with

$$\begin{array}{rcl} |\alpha| &=& |e\omega_1+\omega_2+\omega_2e+\omega_2\omega_1e|\\ &\le& (2u)(2u^2)+(2u^2+5u^3)+(2u^2+5u^3)(2u)+(2u^2+5u^3)(2u^2)(2u)\\ &=& 2u^2+13u^3+10u^4+8u^5+20u^6\\ &\le& 2u^2+14u^3 \text{ as soon as } p\ge 4. \end{array} \tag{62}$$

Therefore,

$$\begin{array}{rcl} m_h+m_\ell &=& t_h+et_h+\alpha t_h\\ &=& t_h+t_h(1-yt_h+\eta)+\alpha t_h\\ &=& t_h(2-yt_h)+t_h(\eta+\alpha), \end{array}$$

which implies

$$\left|(m_h+m_\ell)-\frac{1}{y}\right|=\left|t_h(2-yt_h)-\frac{1}{y}+t_h(\eta+\alpha)\right|,$$

so that, using (57) and the bounds on η and α,

$$\left|(m_h+m_\ell)-\frac{1}{y}\right|\le y\left(t_h-\frac{1}{y}\right)^2+t_h\cdot\left|\frac{5}{2}u^2+14u^3\right|. \tag{63}$$

Let us now consider $y^2\left(t_h - 1/y\right)$. That term is less than

$$y^2\left(\left(t_h - \frac{1}{y_h}\right) + \frac{y - y_h}{yy_h}\right)^2,$$

which is less than

$$y^2u^2\left(\frac{1}{2} + \frac{1}{y(y-u)}\right)^2.$$

The largest value of

$$y^2\left(\frac{1}{2} + \frac{1}{y(y-u)}\right)^2$$

for $1 \le y < 2$ is always attained for $y = 1$, so that as soon as $p \ge 6$ (i.e., $u \le 1/64$), we have

$$y^2\left(t_h - \frac{1}{y}\right) \le \left(\frac{1}{2} + \frac{1}{1 - \frac{1}{64}}\right)^2 u^2 = \frac{36481}{15876}u^2 \le 2.298u^2.$$

Hence, from (63), we obtain

$$\left|(m_h + m_\ell) - \frac{1}{y}\right| \le \frac{1}{y} \cdot 2.298u^2 + t_h\left(\frac{5}{2}u^2 + 14u^3\right),$$

which implies

$$\left|x(m_h + m_\ell) - \frac{x}{y}\right| \le \frac{x}{y} \cdot 2.298u^2 + xt_h\left(\frac{5}{2}u^2 + 14u^3\right).$$

Notice that $|t_h| \le (1+u)/y_h \le (1+u)^2/y$, so that

$$\left|x(m_h + m_\ell) - \frac{x}{y}\right| \le \frac{x}{y} \cdot \varphi(u), \tag{64}$$

with $\varphi(u) = 2.298u^2 + (1+u)^2\left(\frac{5}{2}u^2 + 14u^3\right)$. Now, from Theorem 5.4, we have

$$\begin{array}{rcl} |z_h + z_\ell - x(m_h + m_\ell)| & \le & 5u^2|x(m_h + m_\ell)| \\ & \le & 5u^2\frac{x}{y} + 5u^2\left|\frac{x}{y} - x(m_h + m_\ell)\right| \\ & \le & \frac{x}{y}\left(5u^2 + 5u^2\varphi(u)\right). \end{array} \tag{65}$$

Combining (64) and (65) we finally obtain

$$\begin{array}{rcl} \left|z_h + z_\ell - \frac{x}{y}\right| & \le & \frac{x}{y}(5u^2 + \varphi(u) + 5u^2\varphi(u)) \\ & \le & \frac{x}{y}\left(9.798u^2 + 19u^3 + 54.49u^4 + 109u^5 + 152.5u^6 + 70u^7\right) \\ & \le & 9.8u^2\frac{x}{y} \text{ as soon as } p \ge 14. \end{array}$$

□

This relative error bound is certainly a large overestimate, since we cumulate in its calculation the overestimates of the errors of Algorithms 9, 4, and 12. In practice, Algorithm 17 is rather accurate: the largest relative error found so far in our tests for $p = 53$, is $5.922\cdots \times 2^{-106}$, obtained for $x_h = 4528288502329187$, $x_\ell = 1125391118633487/2^{51}$, $y_h = 4522593432466394$, and $y_\ell = -9006008290016505/2^{54}$.

Conclusion

We have proven relative error bounds for several basic building blocks of double-word arithmetic, suggested a new algorithm for multiplying two double-word numbers, suggested an improvement of the algorithms used in the QD library for dividing a double-word number by a floating-point number, and for dividing two double-word numbers. We have also suggested a new algorithm for dividing two double-word numbers when an FMA instruction is available. Table 1 summarizes the obtained results. For the functions for which an error bound was already published, we always obtain a significantly smaller bound, except in one case, for which the previously known bound turned out to be slightly incorrect. Our results make it possible to have more trust in double-word arithmetic. They also allow us to give some recommendations in what follows.

- For adding two double-word numbers, *never* use Algorithm 5, unless you are certain that both operands have the same sign. Double-word numbers can be added very accurately using the (unfortunately more expensive) Algorithm 6.

- For multiplying a double-word number by a floating-point number, Algorithm 8 is less accurate, yet slightly faster, than Algorithm 7. Hence one cannot say that one is really better than the other one. Choose between them depending on whether you mainly need speed or accuracy. If an FMA instruction is available, Algorithm 9 is a good candidate.

- For multiplying two double-word numbers, if an FMA instruction is available, Algorithm 12 is to be favored. It is more accurate both from a theoretical (better error bound) and from a practical (smaller observed errors in our intensive testings) points of view.

- There is no point in using Algorithm 13 for dividing a double-word number by a floating-point number: Algorithm 14, presented in this paper, always returns the same result and is faster.

- There is no point in using Algorithm 15 for dividing two double-word numbers: Algorithm 16, presented in this paper, always returns the same result and is faster. If an FMA instruction is available, depending whether the priority is speed or accuracy, one might prefer Algorithm 17. It is almost certainly significantly more accurate (although we have no full proof of that: we can just say that our bounds are smaller, as well as the observed errors), however, it is slower.

Table 1: Summary of the results presented in this paper. For each algorithm, we give the previously known bound (when we are aware of it, and when the algorithm already existed), the bound we have proved, the largest relative error observed in our fairly intensive tests, and the number of floating-point operations required by the algorithm.

Operation	Algorithm	Previously known bound	Our bound	Largest relative error observed in experiments	$\sharp$ of FP ops
DW + FP	Algorithm 4	?	$2u^2 + 5u^3$	$2u^2 - 6u^3$	10
DW + DW	Algorithm 5	N/A	N/A	1	11
	Algorithm 6	$2u^2$ (incorrect)	$3u^2 + 13u^3$	$2.25u^2$	20
DW $\times$ FP	Algorithm 7	$4u^2$	$\frac{3}{2}u^2 + 4u^3$	$1.5u^2$	10
	Algorithm 8	?	$3u^2$	$2.517u^2$	7
	Algorithm 9	N/A	$2u^2$	$1.984u^2$	6
DW $\times$ DW	Algorithm 10	$11u^2$	$7u^2$	$4.9916u^2$	9
	Algorithm 12	N/A	$5u^2$	$3.936u^2$	9
DW $\div$ FP	Algorithm 13	$4u^2$	$3.5u^2$	$2.95u^2$	16
	Algorithm 14	N/A	$3.5u^2$	$2.95u^2$	10
DW $\div$ DW	Algorithm 15	?	$15u^2 + 56u^3$	$8.465u^2$	24
	Algorithm 16	N/A	$15u^2 + 56u^3$	$8.465u^2$	18
	Algorithm 17	N/A	$9.8u^2$	$5.922u^2$	31

References

[1] D.H. Bailey, R. Barrio, and J.M. Borwein. High-precision computation: Mathematical physics and dynamics. *Applied Mathematics and Computation*, 218(20):10106 – 10121, 2012.

[2] S. Boldo. Pitfalls of a full floating-point proof: example on the formal proof of the Veltkamp/Dekker algorithms. In U. Furbach and N. Shankar, editors, *Proceedings of the 3rd International Joint Conference on Automated Reasoning*, volume 4130 of *Lecture Notes in Computer Science*, pages 52–66, 2006.

[3] K. Briggs. The `doubledouble` library, 1998. Available at `http://www.boutell.com/fracster-src/doubledouble/doubledouble.html`.

[4] Florent de Dinechin, Alexey V. Ershov, and Nicolas Gast. Towards the post-ultimate libm. In *Proceedings of the 17th IEEE Symposium on Computer Arithmetic*, ARITH '05, pages 288–295, Washington, DC, USA, 2005. IEEE Computer Society.

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

[6] L. Fousse, G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. MPFR: A Multiple-Precision Binary Floating-Point Library with Correct Rounding. *ACM Transactions on Mathematical Software*, 33(2), 2007. available at `http://www.mpfr.org/`.

[7] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating-point arithmetic. In *ARITH-16*, pages 155–162, June 2001.

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

[9] IEEE Computer Society. *IEEE Standard for Floating-Point Arithmetic*. IEEE Standard 754-2008, August 2008. available at `http://ieeexplore.ieee.org/servlet/opac?punumber=4610933`.

[10] Claude-Pierre Jeannerod and Siegfried M. Rump. On relative errors of floating-point operations: optimal bounds and applications. https://hal.inria.fr/hal-00934443, 2016.

[11] W. Kahan. Lecture notes on the status of IEEE-754. PDF file accessible at `http://www.cs.berkeley.edu/~wkahan/ieee754status/IEEE754.PDF`, 1996.

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

[13] 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. Technical Report 45991, Lawrence Berkeley National Laboratory, 2000. `http://crd.lbl.gov/~xiaoye/XBLAS`.

[14] 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.

[15] C. Lichtenau, S. Carlough, and S. M. Mueller. Quad precision floating point on the ibm z13. In *2016 IEEE 23nd Symposium on Computer Arithmetic (ARITH)*, pages 87–94, July 2016.

[16] S. Linnainmaa. Software for doubled-precision floating-point computations. *ACM Transactions on Mathematical Software*, 7(3):272–283, 1981.

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

[18] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010.

[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] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, 2005.

[21] S. M. Rump. Transformations and ill-conditioned problems. In *Proceedings of the International workshop on verified computations and related topics*, March 2009.

[22] S. M. Rump, T. Ogita, and S. Oishi. Accurate floating-point summation part I: Faithful rounding. *SIAM Journal on Scientific Computing*, 31(1):189–224, 2008.

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