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

Mioara Joldes, Valentina Popescu, Jean-Michel Muller

▶ **To cite this version:**

Mioara Joldes, Valentina Popescu, Jean-Michel Muller. Tight and rigourous error bounds for basic building blocks of double-word arithmetic. 2016. hal-01351529v1

HAL Id: hal-01351529

https://hal.science/hal-01351529v1

Preprint submitted on 3 Aug 2016 (v1), last revised 18 Oct 2017 (v3)

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Tight and rigourous 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

July 28, 2016

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 wrong, and we provide a 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

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[6], commonly called "double-precision" (hence the name "double-double"). Double-word arithmetic can be useful when some extra-precision is

needed in a critical part of a numerical program. For example, it has been used with success in BLAS [10]. 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 [7] 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 a "real" FP arithmetic with twice the precision would be a better option), but at least, with rigorously proven and reasonably tight error bounds, expert programmers can rely on double-word arithmetic for extending the precision of calculations in places where the available FP 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 [6, 13], 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 $\mathrm{RN}(cd)$ is the result of the FP multiplication `c*d`, assuming round-to-nearest rounding mode), $\mathrm{ulp}(x)$, 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 [17])**.** *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 [16].

Algorithms 1 and 2, introduced by Moller [12], Dekker [3], and Knuth [8] 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 [3].

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

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 [12, 8].

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

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, Algorithm 1 is to be used only if we have preliminary information on the respective orders of magnitude of a and b.

For computing the exact product of two FP numbers a and b, with exponents e_a and e_b, respectively, we use Algorithm 3, mentioned by Kahan [7], which returns two values π and ρ such that π is the FP number that is closest to ab, and $\pi + \rho = ab$ exactly (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 [14] for a proof.

Algorithm 3 – **Fast2Mult**(a, b)**.** The Fast2Mult algorithm (see for instance [7, 14, 13]). It requires the availability of a fused multiply-add (FMA) instruction for computing $\mathrm{RN}(ab - \pi)$.

$\pi \leftarrow \mathrm{RN}(ab)$
$\rho \leftarrow \mathrm{RN}(ab - \pi)$

Algorithm 3 needs 2 FP basic operations only to compute the result, but it requires the availability of a fused multiply-add (FMA) instruction. Is is worth

being noticed that if no FMA instruction is available, there is an algorithm given by Dekker [3, 1] that computes the same result, but at a much higher cost (17 FP operations). In the following, each time there is a call to Fast2Mult, it can be replaced by Dekker's multiplication algorithm without changing the error bounds (since Fast2Mult—i.e., Algorithm 3—and Dekker's algorithm return the very same result). However, when we count the number of floating-point operations (in Table 1), we assume that Algorithm 3 is used.

Dekker [3] 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, and as said above, they have frequently be called *double-double* numbers in the literature. Throughout this paper we use the term *double-word*, also used in [13]. 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 [11] suggested similar algorithms, assuming that an underlying extended precision format (somehow wider than the format of the two elements of a double-word number) is available. We will not assume that hypothesis here.

Libraries that offer double-word arithmetic (with binary64/double precision as the underlying floating-point format) have been written by Bailey [5] and Briggs [2] (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 [4, 5].

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 [5] 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 of $(x_h, x_\ell)+y$ 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: $(s_h, s_\ell) \leftarrow \text{2Sum}(x_h, y)$
2: $v \leftarrow \text{RN}(x_\ell + s_\ell)$
3: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(s_h, v)$
4: **return** (z_h, z_ℓ)

Algorithm 4, or variants of it, implicitely appears in many "compensated summation" algorithms. Most such algorithms (aimed at accurately computing the sum of several FP numbers) 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 [15].

To prove the correctness and bound the error of Algorithm 4 (and Algorithm 6 below), we will need the following lemma.

Lemma 2.1. *Let a and b be FP numbers, and let $s = \text{RN}(a+b)$. If $s \neq 0$ then*

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

Proof. Without l.o.g., assume $|a| \geq |b|$, so that $\text{ulp}(a) \geq \text{ulp}(b)$. 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 $\frac{1}{2}\text{ulp}(a)$. Therefore $|\text{RN}(a+b)| = \text{RN}(|a+b|) \geq \text{RN}(\frac{1}{2}\text{ulp}(a)) = \frac{1}{2}\text{ulp}(a)$. □

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

Theorem 2.2. *The relative error of Algorithm 4 (DWPlusFP) is bounded by*

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

which is less than $2\cdot 2^{-2p} + 5\cdot 2^{-3p}$ as soon as $p \geq 4$.

Proof. First of all, we can quickly proceed with the case $x_h + y = 0$: in that case $s_h = s_\ell = 0$ and the computation is errorless. Now, without loss of generality, we can assume $|x_h| \geq |y|$ (otherwise, 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), x_h positive (otherwise we change the sign of all the operands), and $1 \le x_h \le 2 - 2u$ (otherwise we scale the operands by a power of 2).

1. If $-x_h \le y \le -x_h/2$, then Sterbenz Lemma implies $s_h = x_h + y$ and $s_\ell = 0$. It follows that $v = x_\ell$. Lemma 2.1 implies $|s_h| \ge \frac{1}{2}\mathrm{ulp}(x_h)$, which implies $|s_h| \ge |x_\ell|$. Hence we can legitimately use Algorithm Fast2Sum at line 3 of the algorithm, so that $z_h + s_\ell = s_h + v = x + y$ exactly.

2. If $-x_h/2 < y \le x_h$, then $\frac{1}{2} \le \frac{x_h}{2} < x_h + y \le 2x_h$, so that $s_h \ge 1/2$. Since $|x_\ell + y_\ell| \le 3u$ (see the two cases considered below), we have $|v| \le 3u$, so that $s_h > |v|$: we can legitimately use Algorithm Fast2Sum at line 3 of the algorithm.

- If $x_h + y \le 2$ then $|s_\ell| \le u$, so that $|x_\ell + s_\ell| \le 2u$, hence,

  $$v = x_\ell + s_\ell + \epsilon,$$

  with $|\epsilon| \le u^2$. Therefore $z_h + z_\ell = s_h + v = x + y + \epsilon$ and the relative error $\epsilon/|x + y|$ of the calculation is bounded by

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

- If $x_h + y > 2$ then $|s_\ell| \le 2u$, so that $|x_\ell + s_\ell| \le 3u$, hence,

  $$v = x_\ell + s_\ell + \epsilon,$$

  with $|\epsilon| \le 2u^2$, and the relative error $\epsilon/|x+y|$ of the calculation is bounded by

  $$\frac{\epsilon}{2 - u} \le \frac{2u^2}{2 - u}.$$

□

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

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

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

$$1.9999999999999933 \cdots \times 2^{-106}.$$

3 Addition of two double-word numbers

Algorithm 5 below was first given by Dekker [3], under the name of *add2* (and in a slightly different presentation: he 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 [5] 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: $(s_h, s_\ell) \leftarrow \text{2Sum}(x_h, y_h)$
2: $v \leftarrow \text{RN}(x_\ell + y_\ell)$
3: $w \leftarrow \text{RN}(s_\ell + v)$
4: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(s_h, w)$
5: **return** (z_h, z_ℓ)

Dekker proved an error bound on the order of $(|x| + |y|)2^{2-2p}$. 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} (i.e., the relative error is equal to 1). Hence, the use of Algorithm 5 should be restricted to special cases such as, for instance, when we know for some reason 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. [9, 10] 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 [9, 10], Li et al. claim that in *binary64/double*-precision arithmetic ($p = 53$) the relative error of Algorithm 6 is upper-bounded by $2 \cdot 2^{-106}$. This is

wrong, as shown by the following example: if

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

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 2.25×2^{-2p}.

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*

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

which is less than $3 \cdot 2^{-2p} + 13 \cdot 2^{-3p}$ as soon as $p \geq 6$.

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

Proof. First of all, we exclude the straightforward case where one of the operands is zero. We can also quickly proceed with the case $x_h + y_h = 0$: in that case one easily sees that 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 $1 \leq x_h < 2$ (i.e., $1 \leq x_h \leq 2 - 2u$, since x_h is a FP number), $x \geq |y|$ (which implies $x_h \geq |y_h|$), and $x_h + y_h$ nonzero.

1. If $-2 + 2u \leq y_h \leq -1$. Notice that $|x_\ell|$ and $|y_\ell|$ are bounded by u, and that x and $|y|$ are bounded by $2 - u$. The sum $x_h + y_h$ is between $-1 + 2u$ and $1 - 2u$ and is a multiple of $2^{-p+1} = 2u$, hence it is a floating-point number. This implies $s_h = x_h + y_h$ and $s_\ell = 0$, hence $c = \mathrm{RN}(t_h) = t_h$. We also have

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

and, since $|x_\ell + y_\ell| \leq 2u$, $|t_h| \leq 2u$ and $|t_\ell| \leq u^2$. Since s_h is a nonzero multiple of $2u$ and $|c| = |t_h| \leq 2u$, we can legitimately use the Fast2Sum algorithm at line 4 of the algorithm. Also,

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

and $|s_h + t_h| \leq (1-2u) + 2u \leq 1$, so that $|v_h| \leq 1$ and $|v_\ell| \leq u/2$. We finally have

$$w = t_\ell + v_\ell + \epsilon_2,$$

with

$$|\epsilon_2| \leq \frac{1}{2}\mathrm{ulp}(t_\ell + v_\ell) \leq \frac{1}{2}\mathrm{ulp}(u/2 + u^2), \tag{3}$$

and

$$|\epsilon_2| \leq \frac{1}{2}\mathrm{ulp}\left[\frac{1}{2}\mathrm{ulp}(x_\ell + y_\ell) + \frac{1}{2}\mathrm{ulp}\left((x+y) + \frac{1}{2}\mathrm{ulp}(x_\ell + y_\ell)\right)\right]. \tag{4}$$

From (3), we have $|\epsilon_2| \leq u^2/2$.

Now, since s_h is a nonzero multiple of $2u$, $|s_h| \geq 2u$, which implies, from Lemma 2.1, that $|s_h + t_h| \geq 2u^2$. Hence $|v_h| = |\mathrm{RN}(s_h + c)| = |\mathrm{RN}(s_h + t_h)| \geq 2u^2$. Hence $|v_h| \geq u^2/(1-u)$, so that $|v_h| \geq u|v_h| + u^2 \geq |v_\ell| + |t_\ell|$. Hence $|w| = |\mathrm{RN}(v_\ell + t_\ell)| \leq |v_h|$, so that Fast2Sum can be used at line 6 of the algorithm. We finally have,

$$z_h + z_\ell = v_h + w = x + y + \epsilon_2. \tag{5}$$

Directly using (5) and the bound $u^2/2$ on $|\epsilon_2|$ to get a relative error bound might result in a rather large bound, because $x + y$ may be small. However, as we are going to see, when $x + y$ is very small, some simplification occurs thanks to Sterbenz Lemma. First, $x_h + y_h$ is a nonzero multiple of $2u$. Hence, since $|x_\ell + y_\ell| \leq 2u$, we have $|x_\ell + y_\ell| \leq x_h + y_h$.

- If $-(x_h + y_h) \leq x_\ell + y_\ell \leq -\frac{1}{2}(x_h + y_h)$, which implies $-s_h \leq t_h \leq -\frac{1}{2}s_h$ then Sterbenz lemma applies to the floating-point addition of s_h and $c = t_h$ at line 4 of the algorithm, so that $v_h = s_h$ 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 $-\frac{1}{2}(x_h + y_h) \leq x_\ell + y_\ell \leq x_h + y_h$, then $2(x_\ell + y_\ell) \leq (x_h + y_h) + (x_\ell + y_\ell)$, so that $x_\ell + y_\ell \leq \frac{1}{2}(x+y)$, and $-\frac{1}{2}(x+y) \leq \frac{1}{2}(x_\ell + y_\ell)$, so that $-(x+y) \leq x_\ell + y_\ell$. Hence $|x_\ell + y_\ell| \leq x + y$, so that

  $$\mathrm{ulp}\,(x_\ell + y_\ell) \leq \mathrm{ulp}(x + y).$$

  Combined with (4), this gives

  $$|\epsilon_2| \leq \frac{1}{2}\mathrm{ulp}\left[\frac{1}{2}\mathrm{ulp}(x+y) + \frac{1}{2}\mathrm{ulp}\left((x+y) + \frac{1}{2}\mathrm{ulp}(x+y)\right)\right].$$

  Hence,

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

2. If $-1 + u \le y_h \le -x_h/2$, which implies $u \le x_h + y_h \le x_h/2$, then Sterbenz Lemma can be applied to the first line of the algorithm: we have $s_h = x_h + y_h$ and $s_\ell = 0$, so that $c = \mathrm{RN}(t_h) = t_h$. Hence, this case is very similar to the previous one. Since s_h is a nonzero multiple of u and $|c| = |t_h| \le 3u/2$, the FP exponent of s_h is at least 2^{-p} and the FP exponent of c is at most 2^{-p}. Hence we can use Algorithm Fast2Sum at line 4 of the algorithm. Now, since s_h is a nonzero multiple of u, $|s_h| \ge u$, which implies, from Lemma 2.1, that $|s_h + t_h| \ge u^2$. Hence $|v_h| = |\mathrm{RN}(s_h + c)| = |\mathrm{RN}(s_h + t_h)| \ge u^2$. If $|v_h| = u^2$ then $|v_\ell + t_\ell| \le u|v_h| + u^2 \le u^2 + u^3$, hence $|w| = |RN(t_\ell + v_\ell)| \le u^2 = |v_h|$. If $|v_h| > u^2$ then (since v_h is a FP number) $|v_h| \ge u^2 + 2u^3$ hence $|v_h| \ge u^2/(1-u)$, so that $|v_h| \ge u|v_h| + u^2 \ge |v_\ell| + |t_\ell|$. Hence $|w| = |\mathrm{RN}(v_\ell + t_\ell)| \le |v_h|$. Hence in all cases we can use Algorithm Fast2Sum at line 6 of the algorithm, and we again have

$$z_h + z_\ell = v_h + w = x + y + \epsilon_2,$$

with

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

From $x_h + y_h \ge u$ and $|x_\ell + y_\ell| \le 2u$, we deduce $-2(x_h + y_h) \le x_\ell + y_\ell \le 2(x_h + y_h)$.

- If $-2(x_h + y_h) \le x_\ell + y_\ell \le -\frac{1}{2}(x_h + y_h)$ then $-2s_h \le t_h = c \le -\frac{1}{2}s_h$, hence Sterbenz Lemma can be applied to line 4 of the algorithm, so that $v_\ell = 0$, hence $w = \mathrm{RN}(t_\ell) = t_\ell$ and $\epsilon_2 = 0$ so that the computation is errorless: $z_h + z_\ell = x + y$;

- If $-\frac{1}{2}(x_h + y_h) < x_\ell + y_\ell \le 2(x_h + y_h)$, then $3(x_\ell + y_\ell) \le 2(x_h + y_h + x_\ell + y_\ell)$ so that $x_\ell + y_\ell \le \frac{2}{3}(x + y)$, and $-\frac{1}{2}(x + y) \le \frac{1}{2}(x_\ell + y_\ell)$, so that $-(x + y) < x_\ell + y_\ell$. Hence, in any case, $|x_\ell + y_\ell| < x + y$, so that $\mathrm{ulp}\,(x_\ell + y_\ell) \le \mathrm{ulp}(x + y)$, and we end up with the same bound as in the previous case:
$$|\epsilon_2| \le 2 \cdot 2^{-2p} \cdot |x + y|.$$

3. If $-x_h/2 < y_h \le x_h$, which implies $x_h/2 < x_h + y_h$, which implies $1/2 < x_h + y_h$.

- If $\frac{1}{2} < x_h + y_h \le 1 - 2u$ then $\frac{1}{2} \le s_h \le 1 - 2u$ and $|s_\ell| \le \frac{u}{2}$. Notice that having $x_h + y_h \le 1 - 2u$ requires y_h to be negative, so that $-1 < y \le 0$, which implies $|y_\ell| \le u/2$. We have

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

  with $|x_\ell + y_\ell| \le 3u/2$, hence $|t_h| \le 3u/2$, and $|t_\ell| \le u^2$. Now,

  $$c = s_\ell + t_h + \epsilon_1,$$

with $|s_\ell + t_h| \leq 2u$, so that $|c| \leq 2u$, and $|\epsilon_1| \leq u^2$. Since $s_h \geq 1/2$ and $|c| \leq 2u$, we can use Algorithm Fast2Sum at line 4 of the algorithm, therefore,

$$v_h + v_\ell = s_h + c \leq 1 - 2u + 2u = 1,$$

so that $v_h \leq 1$ and $|v_\ell| \leq u/2$. Thus,

$$w = t_\ell + v_\ell + \epsilon_2,$$

where $|t_\ell + v_\ell| \leq \frac{u}{2} + u^2$, so that $|\epsilon_2| \leq u^2/2$. Also, $s_h \geq 1/2$ and $|c| \leq 2u$ imply $v_h \geq 1/2 - 2u$. And $|t_\ell + v_\ell| \leq u/2 + u^2$ imply $|w| \leq u/2 + u^2$. Hence, if $p \geq 3$ (i.e., $u \leq 1/8$) we can safely use Algorithm Fast2Sum at line 6 of the algorithm. Therefore,

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

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

Since $x + y \geq (x_h - u) + (y_h - u/2) > 1/2 - 3u/2$, the relative error $|\eta|/(x + y)$ is upper-bounded by

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

- If $1 - 2u < x_h + y_h \leq 2 - 4u$ then $1 - 2u \leq s_h \leq 2 - 4u$ and $|s_\ell| \leq u$. We have,

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

  with $|x_\ell + y_\ell| \leq 2u$, hence $|t_h| \leq 2u$, and $|t_\ell| \leq u^2$. Now,

  $$c = s_\ell + t_h + \epsilon_1,$$

  with $|s_\ell + t_h| \leq 3u$, so that $|c| \leq 3u$, and $|\epsilon_1| \leq 2u^2$. Since $s_h \geq 1 - 2u$ and $|c| \leq 3u$, if $p \geq 3$ we can use Algorithm Fast2Sum at line 4 of the algorithm, therefore,

  $$v_h + v_\ell = s_h + c \leq 2 - 4u + 3u = 2 - u,$$

  so that $v_h \leq 2$ and $|v_\ell| \leq u$. Thus,

  $$w = t_\ell + v_\ell + \epsilon_2,$$

  where $|t_\ell + v_\ell| \leq u + u^2$, so that $|\epsilon_2| \leq u^2$. Also, $s_h \geq 1 - 2u$ and $|c| \leq 3u$ imply $v_h \geq 1 - 5u$. And $|t_\ell + v_\ell| \leq u + u^2$ imply $|w| \leq u$. Hence if $p \geq 3$ we can safely use Algorithm Fast2Sum at line 6 of the algorithm.

  Therefore,

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

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

  Since $x + y \geq (x_h - u) + (y_h - u) > 1 - 4u$, the relative error $|\eta|/(x + y)$ is upper-bounded by

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

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

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

  with $|x_\ell + y_\ell| \leq 2u$, hence $|t_h| \leq 2u$, and $|t_\ell| \leq u^2$. Now,

  $$c = s_\ell + t_h + \epsilon_1,$$

  with $|s_\ell + t_h| \leq 4u$, so that $|c| \leq 4u$, and $|\epsilon_1| \leq 2u^2$. Since $s_h \geq 2 - 4u$ and $|c| \leq 4u$, if $p \geq 3$ we can use Algorithm Fast2Sum at line 4 of the algorithm, therefore,

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

  so that $v_h \leq 4$ and $|v_\ell| \leq 2u$. Thus,

  $$w = t_\ell + v_\ell + \epsilon_2,$$

  where $|t_\ell + v_\ell| \leq 2u + u^2$. Hence, either $|t_\ell + v_\ell| < 2u$ and $|\epsilon_2| \leq \frac{1}{2}\mathrm{ulp}(t_\ell + v_\ell) \leq u^2$, or $2u \leq t_\ell + v_\ell \leq 2u + u^2$, in which case $w = \mathrm{RN}(t_\ell + v_\ell) = 2u$ and $|\epsilon_2| \leq u^2$: in all cases $|\epsilon_2| \leq u^2$, so that $|\epsilon_2| \leq u^2$. Also, $s_h \geq 2 - 4u$ and $|c| \leq 4u$ imply $v_h \geq 2 - 8u$. And $|t_\ell + v_\ell| \leq 2u + u^2$ imply $|w| \leq 2u$. Hence if $p \geq 3$ we can safely use Algorithm Fast2Sum 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 $3u^2 + 13u^3$. □

The bound (2) is probably not optimal. The largest relative error we have obtain through many tests is around 2.25×2^{-2p}.

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

We first consider the following algorithm, suggested by Li et al [9]:

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

1: $(c_h, c_{\ell 1}) \leftarrow \text{Fast2Mult}(x_h, y)$
2: $c_{\ell 2} \leftarrow \text{RN}(x_\ell \cdot y)$
3: $(t_h, t_{\ell 1}) \leftarrow \text{Fast2Sum}(c_h, c_{\ell 2})$
4: $t_{\ell 2} \leftarrow \text{RN}(t_{\ell 1} + c_{\ell 1})$
5: $(z_h, z_\ell) \leftarrow \text{Fast2Sum}(t_h, t_{\ell 2})$
6: **return** (z_h, z_ℓ)

In [9, 10] (with more detail in the technical report [9], which is a preliminary version of the journal paper [10]), Li et Al. explain that they have obtained a relative error bound $4 \cdot 2^{-106}$ (that is, the absolute error is bounded by $4 \cdot 2^{-106}|xy|$) for Algorithm 7 when the underlying floating-point arithmetic is *binary64/double*-precision (i.e., $p = 53$). Below, we show that in the more general context of precision-p arithmetic that relative error bound can be significantly improved. More precisely,

Theorem 4.1. *If $p \geq 3$, the relative error*

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

of Algorithm 7 (DWTimesFP1) is bounded by

$$2 \cdot 2^{-2p} \tag{6}$$

Proof. One easily notices that if $x = 0$ or $y = 0$ the obtained result is exact. 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 than assume $1 + 2u \leq y \leq 2 - 2u$. This gives $1 + 2u \leq x_h y \leq 4 - 8u + 4u^2$, so that $1 + 2u \leq c_h \leq 4 - 8u$. Notice that 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 $|\epsilon_1| \leq u^2$. From $1 + 2u \leq c_h$ and $|c_{\ell 2}| \leq 2u - 2u^2$ we deduce that we can use Algorithm Fast2Sum at line 3 of the algorithm. Also, we deduce that $t_h \geq 1$.

Let us now consider two possible cases:

1. If $x_h y \leq 2$

In that case, $c_h \leq 2$, and $c_h + c_{\ell 1} = x_h y$, with $|c_{\ell 1}| \leq u$. We have $t_h + t_{\ell 1} = c_h + c_{\ell 2}$, and $c_h + c_{\ell 2} < 2 + 2u$, so that $t_h \leq 2$.

- If $t_h < 2$ (i.e., $t_h \leq 2 - 2u$) then $|t_{\ell 1}| \leq u$, and

$$t_{\ell 2} = t_{\ell 1} + c_{\ell 1} + \epsilon_2, \tag{7}$$

with $|t_{\ell 1} + c_{\ell 1}| \leq 2u$, so that $|t_{\ell 2}| \leq 2u$ and $|\epsilon_2| \leq u^2$. Since $t_h \geq 1$ and $|t_{\ell 2}| \leq 2u$, we can use Algorithm Fast2Sum at line 5 of the algorithm. Therefore

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

with $|\epsilon| = |\epsilon_1 - \epsilon_2| \leq 2u^2$, which leads to a relative error $\epsilon/(xy)$ bounded by

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

- If $t_h = 2$, then $2 \leq c_h + c_{\ell 2} < 2 + 2u$. The bound on $|t_{\ell 1}|$ becomes $2u$ so that $|t_{\ell 1} + c_{\ell 1}| \leq 3u$, which gives $|t_{\ell 2}| \leq 3u$ (hence we can use Algorithm Fast2Sum at line 5 of the algorithm), and $t_{\ell 2} = t_{\ell 1} + c_{\ell 1} + \epsilon_2$, with $|\epsilon_2| \leq 2u^2$. Now the absolute error $|\epsilon| = |\epsilon_1 - \epsilon_2|$ becomes bounded by $3u^2$. Also, we have $xy = (x_h + x_\ell) \cdot y \geq c_h(1-u) + c_{\ell 2}(1-u) \geq t_h(1-u)^2 = 2(1-u)^2$, so that the relative error $\epsilon/(xy)$ is bounded by

$$\frac{3}{2} \cdot \frac{u^2}{(1-u)^2},$$

which is less than $2u^2$ as soon as $p \geq 3$ (i.e., $u \leq 1/8$).

1. If $x_h y > 2$

In that case $2 \leq c_h \leq 4 - 8u$, and $c_h + c_{\ell 1} = x_h y$, with $|c_{\ell 1}| \leq 2u$. We have $t_h + t_{\ell 1} = c_h + c_{\ell 2}$, and $c_h + c_{\ell 2} \leq 4 - 8u + 2u - 2u^2 \leq 4 - 6u - 2u^2$, so that $t_h \leq 4 - 4u$, and $|t_{\ell 1}| \leq 2u$. Now,

$$t_{\ell 2} = t_{\ell 1} + c_{\ell 1} + \epsilon_2, \tag{8}$$

with $|t_{\ell 1} + c_{\ell 1}| \leq 4u$, so that $|t_{\ell 2}| \leq 4u$ (hence we can use Algorithm Fast2Sum at line 5 of the algorithm), and $|\epsilon_2| \leq 2u^2$. Therefore

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

with $|\epsilon| = |\epsilon_1 - \epsilon_2| \leq 3u^2$. Also, we have $xy \geq x_h(1-u)y \geq 2 - 2u$, which leads to a relative error $\epsilon/(xy)$ bounded by

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

which is less than $2u^2$ as soon as $p \geq 2$.

Hence in all cases, the relative error is bounded by $2u^2$. □

The bound (6) is probably not optimal. The largest error we have found so far when performing many tests is about $1.5 \times 2^{-2p} = \frac{3}{2}u^2$. For instance, in binary32/single-precision 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 [5] as well as in Briggs' library [2], 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: $(c_h, c_{\ell 1}) \leftarrow \text{Fast2Mult}(x_h, y)$
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: one easily finds values $x = (x_h, x_\ell)$ and y for which the error attained using Algorithm 8 is larger than the bound given by Theorem 4.1. An example with $p = 53$ (which corresponds to the *binary64/double-precision* format of IEEE 754) 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:

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

$$3 \cdot 2^{-2p}. \tag{9}$$

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 can 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 we can use Algorithm Fast2Sum at line 4 of the algorithm.

Hence,

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

and $|-\epsilon_1+\epsilon_2| \le 3u^2$. Since $xy \ge (x_h-u)y \ge (1-u)(1+2u) \ge 1$, 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 – DWTimesFP3(x_h, x_ℓ, y). Algorithm for computing of $(x_h, x_\ell) \times y$ in binary, precision-p, floating-point arithmetic, assuming an FMA instruction is available.

1: $(c_h, c_{\ell 1}) \leftarrow \text{Fast2Mult}(x_h, y)$
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)

By doing this, we obtain a better error bound:

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

$$2 \cdot 2^{-2p}. \tag{10}$$

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/double-precision 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 [3]), with the only difference that Dekker did not use Fast2Mult (Algorithm 3) for getting the result and error of a FP multiplication, but another, more complex algorithm. This algorithm is also the one that has been implemented in the QD library [5] and in Briggs' library [2] 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: $(c_h, c_{\ell 1}) \leftarrow \text{Fast2Mult}(x_h, y_h)$
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 $11 \cdot 2^{-2p}$. 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*

$$\frac{7 \cdot 2^{-2p}}{(1+2^{-p})^2} < 7 \cdot 2^{-2p}. \tag{11}$$

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 $|c_{\ell 3}| \leq 6u$ (hence, since $c_h \geq 1$, we can use Algorithm Fast2Sum at line 6 of the algorithm), and $|\epsilon_4| \leq 4u^2$.

Therefore,

$$\begin{aligned} z_h + z_\ell &= c_h + c_{\ell 3} \\ &= (x_h y_h - c_{\ell 1}) + c_{\ell 1} + c_{\ell 2} + \epsilon_4 \\ &= x_h y_h + t_{\ell 1} + t_{\ell 2} + \epsilon_3 + \epsilon_4 \\ &= x_h y_h + x_h y_\ell + x_\ell y_h + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 \\ &= xy - x_\ell y_\ell + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 \\ &= xy + \eta, \end{aligned} \tag{12}$$

with $|\eta| \leq u^2 + |\epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4| \leq 9u^2$. Now, $x_h \geq 1$ and $y_h \geq 1$ imply $xy \geq (1 - \frac{u}{2})^2$, so that we can first deduce a relative error bound $9u^2/(1 - \frac{u}{2})^2$. That bound can be improved:

- if $x_h y_h > 2$ then, since $x \geq x_h - u$ and $y \geq y_h - u$, we have $xy \geq x_h y_h - u(x_h + y_h) + u^2 > 2 - u(4 - 4u) + u^2 = 2 - 4u + 5u^2$, hence the relative error is bounded by

$$\frac{9u^2}{2 - 4u + 5u^2} = \frac{9}{2}u^2 + 9u^3 + \frac{27}{4}u^4 + \cdots; \tag{13}$$

- if $x_h y_h \leq 2$ then we now obtain $|c_{\ell 1}| \leq u$. Furthermore, Lemma 5.2 implies

$$x_h + y_h \leq 2\sqrt{2}. \tag{14}$$

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 (14),

$$|t_{\ell 1} + t_{\ell 2}| \leq x_h u + y_h u \leq 2\sqrt{2}u. \tag{15}$$

Since $2\sqrt{2}u$ is between $2u$ and $4u$, (15) gives $|\epsilon_3| \leq 2u^2$ (i.e., the same bound on $|\epsilon_3|$ as previously). Hence $c_{\ell 2}$ satisfies

$$|c_{\ell 2}| \leq |t_{\ell 1} + t_{\ell 2}| + |\epsilon_3| \leq 2\sqrt{2}u + 2u^2.$$

We now deduce

$$|c_{\ell 1} + c_{\ell 2}| \leq u \cdot (2\sqrt{2} + 1) + 2u^2.$$

If $p \geq 4$ (i.e., $u \leq 1/16$), this new bound is always less than $4u$. Hence, we now have $|\epsilon_4| \leq 2u^2$. In Eq. (12), this will result in $|\eta| \leq 7u^2$ instead of $9u^2$. Again using $xy \geq (1 - \frac{u}{2})^2$, we can deduce a relative error bound $7u^2/(1 - \frac{u}{2})^2$. However, that error bound can be made slightly smaller by noticing 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

$$\frac{7u^2}{(1 + u)^2} < 7u^2. \tag{16}$$

One easily notices that if $p \geq 4$ the bound (13) is less than the bound (16).

□

The bound $7u^2$ provided by Theorem 5.1 is probably too pessimistic. The largest relative error we have encountered in our tests was $4.9916u^2$, obtained for $p = 53$, $x_h = 4508231565242345$, $x_\ell = -9007199254524053/2^{54}$, $y_h = 4504969740576150$, and $y_\ell = -4503599627273753/2^{53}$. In *binary32/single*-precision arithmetic ($p = 24$), the largest error obtained in our tests was $4.947u$, 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 improve Algorithm 10 and make it more accurate, as follows. Consider Algorithm 11 below.

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

1: $(c_h, c_{\ell 1}) \leftarrow \text{Fast2Mult}(x_h, y_h)$
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: $c_{\ell 2} \leftarrow \text{RN}(t_{\ell 1} + x_\ell y_h)$
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 5.3. *If $p \geq 4$, the relative error of Algorithm 11 (DWTimesDW2) is less than or equal to*

$$\frac{5 \cdot 2^{-2p} + 2^{-3p-1}}{(1+2^{-p})^2} < 5 \cdot 2^{-2p}. \tag{17}$$

Proof. The proof is very similar to the proof of Theorem 5.1, and follows the same structure, so we do not detail it. The major changes are:

- $t_{\ell 1} = x_h y_\ell + x_\ell y_\ell + \epsilon_1$, where ϵ_1, now, is bounded by $u^2 + u^3/2$,
- the term ϵ_2 of the proof of Theorem 5.1 no longer exists;
- instead of (12), we now have

$$z_h + z_\ell = xy + \epsilon_1 + \epsilon_3 + \epsilon_4 = xy + \eta,$$

with $|\eta| \leq |\epsilon_1 + \epsilon_3 + \epsilon_4| \leq 7u^2 + u^3/2$.

□

We do not know if the bound given by Theorem 5.3 is optimal. The largest relative error we have encountered so far in intensive tests was (for $p = 53$) 3.936×2^{-2p}, 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 [9] for dividing a double-word number by a floating-point number is the following (after having replaced a Dekker product

by a call to Fast2Mult: we assume that an FMA instruction is available. This does not change the analysis since Fast2Mult and the Dekker product always return the same result).

Algorithm 12 – **DWDivFP1**(x_h, x_ℓ, y). Calculation of $(x_h, x_\ell) \div y$ in binary, precision-p, floating-point arithmetic.

1: $t_h \leftarrow \mathrm{RN}(x_h/y)$
2: $(\pi_h, \pi_\ell) \leftarrow \mathrm{Fast2Mult}(t_h, y)$
3: $(\delta_h, \delta') \leftarrow 2Sum(x_h - \pi_h)$
4: $\delta'' = \mathrm{RN}(x_\ell - \pi_\ell)$
5: $\delta_\ell = \mathrm{RN}(\delta' + \delta'')$
6: $\delta \leftarrow \mathrm{RN}(\delta_h + \delta_\ell)$
7: $t_\ell \leftarrow \mathrm{RN}(\delta/y)$
8: $(z_h, z_\ell) \leftarrow \mathrm{Fast2Sum}(t_h, t_\ell)$
9: **return** (z_h, z_ℓ)

Let us notice that Algorithm 12 can be simplified. We have $t_h = (x_h/y)(1+\epsilon_0)$ and $\pi_h = t_h y(1+\epsilon_1)$, with $|\epsilon_0|, |\epsilon_1| \leq u$. Hence,

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

Therefore, as soon as $p \geq 2$ (i.e., $u \leq 1/4$), π_h is within a factor 2 from x_h so that, from Sterbenz Lemma (Lemma 1.2), $x_h - \pi_h$ is a floating-point number. As a consequence, we always have $\delta' = 0$, 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 13, below, always returns the same result as Algorithm 12.

Algorithm 13 – **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: $(\pi_h, \pi_\ell) \leftarrow \mathrm{Fast2Mult}(t_h, y)$
3: $\delta_h = \mathrm{RN}(x_h - \pi_h)$
4: $\delta_\ell = \mathrm{RN}(x_\ell - \pi_\ell)$
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 [9] claim that their double-precision (i.e., $p = 53$) implementation of Algorithm 12 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 $p \geq 4$, the relative error of Algorithm 13 (DWDivFP2) is bounded by*

$$\frac{7}{2} \cdot 2^{-2p}.$$

That bound also holds for Algorithm 12 (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 $3 \cdot 2^{-2p}$. For instance, for $p = 53$, relative error $2.95157083\cdots \times 2^{-2p}$ 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*

$$\begin{cases} u - 2u^2/b & \text{if } a/b \geq 1; \\ u/2 - u^2/b & \text{otherwise.} \end{cases} \tag{18}$$

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 $2^{p-1} \leq M_a, M_b \leq 2^p - 1$.

- 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 $(2M_\mu + 1)/2^{p+1}$.

□

Let us now prove Theorem 6.1.

Proof. Without loss of generality, we assume $1 \leq x_h < 2$ (i.e., $1 \leq x_h \leq 2 - 2u$), so that $|x_\ell| \leq u$, and $1 \leq y < 2$ (i.e., $1 \leq y \leq 2 - 2u$). We can quickly notice that the analysis of the case $y = 1$ is straightforward, so we can assume $1 + 2u \leq y \leq 2 - 2u$. Therefore, we have

$$\frac{1}{2 - 2u} \leq \frac{x_h}{y} \leq \frac{2 - 2u}{1 + 2u}. \tag{19}$$

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

$$\frac{1}{2} + u \leq t_h = \mathrm{RN}\left(\frac{x_h}{y}\right) \leq 2 - 6u. \tag{20}$$

1. **If $x \geq y$, which implies $x_h \geq y$ and $t_h \geq 1$,** then Lemma 6.2 implies

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

hence $|t_h y - x_h| \leq uy - 2u^2 \leq 2u - 4u^2$. As a consequence

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

(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 $y \geq 1 + 2u$), and

$$|t_h y| \leq |t_h y - x_h| + |x_h| \leq 2u - 4u^2 + 2 - 2u = 2 - 4u^2, \tag{21}$$

which implies

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

In all cases, $x_h - \pi_h \in \{-2u, 0, 2u\}$, so that $\delta_h = x_h - \pi_h$ exactly. Also, from $|\pi_\ell| \leq u$ and $|x_\ell| \leq u$, we deduce $|\pi_\ell - x_\ell| \leq 2u$, so that $|\delta_\ell| \leq 2u$, and $\epsilon_1 = \delta_\ell - (x_\ell - \pi_\ell)$ satisfies

$$|\epsilon_1| = |\delta_\ell - (x_\ell - \pi_\ell)| \leq u^2. \tag{22}$$

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

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

  $$\pi_\ell = t_h y - \pi_h = (t_h y - x_h) + (x_h - \pi_h)$$

  satisfies

  $$\pi_\ell \leq (2u - 4u^2) + (-2u) \leq -4u^2.$$

  Hence $-u \leq \pi_\ell \leq -4u^2$, so that $-u + 4u^2 \leq \delta_\ell \leq 2u$, which implies $-3u \leq \delta_h + \delta_\ell \leq 0$. Furthermore,

  - if $u \leq \delta_\ell \leq 2u$ then Sterbenz's Lemma implies that $\delta_h + \delta_\ell$ is a FP number, so that $\epsilon_2 = 0$;
  - if $-u + 4u^2 \leq \delta_\ell < u$ then $-3u < \delta_h + \delta_\ell < -u$, hence

    $$|\epsilon_2| = |\delta - (\delta_h + \delta_\ell)| \leq \frac{1}{2}\mathrm{ulp}(3u) = 2u^2.$$

    However, in that case, since $|\delta_\ell| < u$, the bound (22) is improved and becomes $|\epsilon_1| \leq u^2/2$.

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

- Symmetrically, if $\delta_h = 2u$ we also always have $|\epsilon_1 + \epsilon_2| \leq 5u^2/2$.

- If $\delta_h = 0$ then $|\delta_h + \delta_\ell| = |\delta_\ell| \leq 2u$. Since there is no error when adding δ_h and δ_ℓ, we have $\epsilon_2 = 0$.

Hence,

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

with $|\epsilon| = |\epsilon_1 + \epsilon_2| \leq 5u^2/2$. We deduce

$$\frac{\delta}{y} = \frac{x}{y} - t_h + \frac{\epsilon}{y}. \tag{23}$$

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 (tighter than the obvious bound $3u^2/(1+2u)$ we would obtain by dividing the upper bound $3u^2$ on δ by the lower bound $1+2u$ on y). We have

$$\begin{aligned}\left|\tfrac{x}{y} - t_h\right| &\leq \left|\tfrac{x_h}{y} - t_h\right| + \left|\tfrac{x_\ell}{y}\right|\\ &\leq u - \tfrac{2u^2}{y} + \tfrac{u}{y},\end{aligned}$$

and

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

Therefore, using (23),

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

Hence $|t_\ell| \leq 2u$ (which means, since $t_h \geq 1$, that we can use Algorithm Fast2Sum at line 7 of the algorithm). Also,

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

Therefore, using (23),

$$t_\ell = \frac{x}{y} - t_h + \frac{\epsilon}{y} + \epsilon', \tag{24}$$

with $|\epsilon'| \leq u^2$. We finally conclude that

$$\left|(z_h + z_\ell) - \frac{x}{y}\right| = \left|(t_h + t_\ell) - \frac{x}{y}\right| \leq \frac{5u^2}{2y} + u^2, \tag{25}$$

so that the relative error is bounded by

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

since $y/x \leq 1$.

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

We can first notice that the case $x_h = y$ is easily handled. Il 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 $|\eta| \le u|x_\ell|/y \le u^2 x/y$.

We can now focus on the case $x_h < y$. Notice that this case implies $x/y \le (y - 2u)/y$, so that $x/y < 1 - u$, so that $t_h \le 1 - u$.

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 implies

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

so that $|t_h y - x_h| \le (u/2) \cdot y - u^2 \le u - 2u^2$. 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 also have

$$|t_h y| \le |t_h| \cdot |y| \le (1 - u) \cdot (2 - 2u) = 2 - 3u + 2u^2,$$

so that $|\pi_\ell| \le \frac{1}{2}\mathrm{ulp}(t_h y) \le u$.

As previously, define $\epsilon_1 = \delta_\ell - (x_\ell - \pi_\ell)$ and $\epsilon_2 = \delta - (\delta_h + \delta_\ell)$. From $|\pi_\ell| \le u$ and $|x_\ell| \le u$, we deduce $|\delta_\ell| \le 2u$ and $\epsilon_1 \le u^2$.

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

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

  hence,

  $$2u^2 \le \pi_\ell \le \frac{u}{2}. \tag{26}$$

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

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

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

Hence,

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

with $|\epsilon| = |\epsilon_1 + \epsilon_2| \le 3u^2/2$. We deduce

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

so that $|t_\ell| \leq 2u$ (hence, since $t_h \geq 1/2 + u$, we can use Algorithm Fast2Sum at line 7), and

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

Therefore

$$t_\ell = \frac{x}{y} - t_h + \eta,$$

with

$$|\eta| \leq \frac{3u^2}{2y} + u^2,$$

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

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

□

7 Division of two double-word numbers

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

Algorithm 14 – 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.

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

Let us quickly analyze the beginning of Algorithm 14. This will lead us to suggest another, faster yet mathematically equivalent, algorithm (as soon as $p \geq 3$, it always returns the very same result). 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

$$t_h = \frac{x_h}{y_h}(1 + \epsilon_0), \text{ with } |\epsilon_0| \leq u, \tag{27}$$

and, from Theorem 4.1,

$$r_h + r_\ell = t_h y(1+\eta), \text{ with } |\eta| \leq 2u^2. \tag{28}$$

The number r_h can be written $(r_h + r_\ell)(1+\epsilon_1)$, with $|\epsilon_1| \leq u$, so that $r_\ell = -\epsilon_1 t_h y(1+\eta)$. We finally obtain

$$\begin{array}{rcl} r_h & = & t_h y_h(1+\epsilon_y)(1+\epsilon_1)(1+\eta) \\ & = & x_h(1+\epsilon_y)(1+\epsilon_0)(1+\epsilon_1)(1+\eta), \end{array} \tag{29}$$

so that

$$(1-u)^3(1-2u^2)x_h \leq r_h \leq (1+u)^3(1+2u^2)x_h,$$

from which we deduce

$$|x_h - r_h| \leq (3u + 5u^2 + 7u^3 + 6u^4 + 2u^5) \cdot x_h,$$

which implies

$$|x_h - r_h| \leq (3u + 6u^2) \cdot x_h \tag{30}$$

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 14 is always 0 (and that line can be replaced by a simple, errorless, subtraction), $\pi_h = x_h - r_h$ exactly, and $\delta_h = -r_\ell$. Hence, without changing the final result, we can replace Algorithm 14 by the simpler Algorithm 15, below.

Algorithm 15 – 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 14. 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)

Notice: If an FMA instruction is available, Algorithm 9 can be used at line 2 instead of Algorithm 7 without changing the error bound provided by Theorem 7.1 below (the proof of the theorem just uses the fact that Algorithm 7 approximates a double-word-times-floating-point product with relative error bounded by $2u^2$, and Algorithm 9 has the same error bound). We have

Theorem 7.1. *If $p \geq 7$, the relative error of Algorithms 14 (DWDivDW1) and 15 (DWDivDW2) is upper-bounded by*

$$15 \cdot 2^{-2p} + 56 \cdot 2^{-3p}.$$

Proof. We will use the results (27) to (30) obtained when analyzing the beginning of Algorithm 14. We have

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

and we can notice from $|x_\ell| \leq u \cdot x_h$, $|r_\ell| \leq u \cdot r_h$ and (30) that

$$|x_\ell - r_\ell| \leq (2u + 3u^2 + 6u^3) \cdot x_h. \tag{31}$$

We have

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

so that

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

with (using (30) and (31))

$$\begin{array}{rcl} |\alpha| & \leq & (2u + 3u^2 + 6u^3)(2u + u^2) + (3u + 6u^2)u \\ & \leq & 7u^2 + 15u^3 \end{array} \tag{32}$$

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

$$\frac{\delta}{y_h} = \frac{x - t_h y}{y} \cdot \frac{y}{y_h} - \frac{\eta t_h y}{y_h} + \alpha \frac{x_h}{y_h}. \tag{33}$$

The number $x - t_h y$ is equal to $x_h - t_h y_h + x_\ell - t_h y_\ell$; $x_h - t_h y_h$ is equal to $-x_h\epsilon_0$, $|x_\ell|$ is less than or equal to $u x_h$, and

$$|t_h y_\ell| \leq |u t_h y_h| \leq u(1+u)x_h,$$

hence,

$$|x - t_h y| \leq x_h \cdot (u + u + u(1+u)) = x_h \cdot (3u + u^2). \tag{34}$$

From (33), we deduce

$$\begin{array}{rcl} \frac{\delta}{y_h} & = & \frac{x - t_h y}{y} \cdot (1+\epsilon_y) - \eta t_h (1+\epsilon_y) + \alpha \frac{x_h}{y_h}, \\ & = & \frac{x - t_h y}{y} + \beta, \end{array} \tag{35}$$

with

$$\begin{array}{rcl} |\beta| & = & \left| \epsilon_y \cdot \frac{x - t_h y}{y} - t_h(1+\epsilon_y)\eta + \frac{x_h}{y_h} \right| \\ & \leq & u(3u + u^2)\frac{x_h}{y} + (1+u)(2u^2)\frac{x_h}{y_h} + (7u^2 + 15u^3)\frac{x_h}{y_h} \\ & \leq & u(3u + u^2)(1+u)\frac{x}{y} + (1+u)^3(2u^2)\frac{x}{y} + (7u^2 + 15u^3)(1+u)^2\frac{x}{y} \\ & = & \left(12u^2 + 39u^3 + 44u^4 + 17u^5\right) \cdot \frac{x}{y}. \end{array} \tag{36}$$

Hence,

$$\begin{array}{rcl} t_\ell &=& \mathrm{RN}\left(\frac{\delta}{y_h}\right) \\ &=& \frac{\delta}{y_h}(1+\epsilon_4) \text{ with } |\epsilon_4| \le u, \\ &=& \left(\frac{x-t_h y}{y}+\beta\right)(1+\epsilon_4) \\ &=& \frac{x-t_h y}{y}+\gamma, \end{array} \tag{37}$$

with

$$\begin{array}{rcl} |\gamma| &=& \left|\frac{x-t_h y}{y}\epsilon_4+\beta+\epsilon_4\beta\right| \\ &\le& \frac{x_h}{y}(3u+u^2)u+\beta+\beta u \\ &\le& (3u+u^2)u(1+u)\frac{x}{y}+\beta+\beta u \\ &=& \left(15u^2+55u^3+84u^4+61u^5+17u^6\right)\cdot\frac{x}{y}. \end{array} \tag{38}$$

Hence

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

Since we straightforwardly have

$$t_h \ge \frac{x}{y}\cdot(1-u)^3, \tag{39}$$

we deduce

$$|t_\ell| \le \frac{x}{y}\cdot\left((15u^2+55u^3+84u^4+61u^5+17u^6)+(3u-3u^2+u^3)\right). \tag{40}$$

From (39) and (40) we easily deduce that as soon as $p \ge 4$ (i.e., $u \le 1/16$), t_h is larger than $|t_\ell|$, so that we can use Algorithm Fast2Sum 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 15 (and Algorithm 14) 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 $p \ge 7$ (i.e., $u \le 128$), 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 $p = 53$ (which corresponds to the binary64/double precision format of the IEEE 754 Standard), relative error $8.465\cdots\times 2^{-106}$ is attained for $x_h = 4503607118141812$, $x_\ell = 4493737176494969/2^{53}$, $y_h = 4503600552333684$, and $y_\ell = -562937972998161/2^{50}$.

Now, if an FMA instruction is available, it is possible to design a more accurate algorithm. What makes it working 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.*

(as a matter of fact, that property also holds if we replace "RN" by rounding towards $\pm\infty$, but this does not matter here)

Proof. Without l.o.g., assume $1 \leq y \leq 2 - 2u$, 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

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

we deduce

$$-u \leq 1 - yt \leq u.$$

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. □

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

Algorithm 16 – 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

---

1: $t_h \leftarrow \mathrm{RN}(1/y_h)$
2: $r_h \leftarrow (1 - y_h t_h) = 1 - y_h t_h$ (exact operation)
3: $r_\ell \leftarrow -\mathrm{RN}(y_\ell \cdot t_h)$
4: $(e_h, e_\ell) \leftarrow \mathrm{Fast2Sum}(r_h, r_\ell)$
5: $(\delta_h, \delta_\ell) \leftarrow \mathrm{DWTimesFP3}(e_h, e_\ell, t_h)$ {Approximation to $(e_h + e_\ell) \cdot t_h$ with relative error $\leq 2u^2$ using Algorithm 9}
6: $(m_h, m_\ell) \leftarrow \mathrm{DWPlusFP}(\delta_h, \delta_\ell, t_h)$ {Approximation to $\delta_h + \delta_\ell + t_h$ with relative error $\leq 2u^2 + 5u^3$ using Algorithm 4}
7: $(z_h, z_\ell) \leftarrow \mathrm{DWTimesDW2}(x_h, x_\ell, m_h, m_\ell)$ {Approximation to $(x_h + x_\ell)(m_h + m_\ell)$ with relative error $\leq 5u^2$ using Algorithm 11}
8: **return** (z_h, z_ℓ)

We have,

Theorem 7.3. *As soon as $p \geq 14$, and if $y \neq 0$, the relative error of Algorithm 16 (DWDivDW3) is bounded by*

$$9.8u^2.$$

Proof. Roughly speaking, Algorithm 16 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

$$\left(t_h(2 - y t_h) - \frac{1}{y} \right) = -y \cdot \left(t_h - \frac{1}{y} \right)^2. \tag{41}$$

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

$$|r_\ell + y_\ell t_h| \le \frac{u^2}{2}.$$

This gives

$$e_h + e_\ell = r_h + r_\ell = 1 - y_h t_h - y_\ell t_h + \eta, \text{ with } |\eta| \le \frac{u^2}{2}. \tag{42}$$

Also, since $|y_h t_h - 1| = y_h \cdot |t_h - 1/y_h| \le u$, we have $|r_h| \le u$, hence $|r_h + r_\ell| \le 2u$, so that $|e_h| \le 2u$ 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

$$\delta_h + \delta_\ell = e t_h (1 + \omega_1), \text{ with } |\omega_1| \le 2u^2, \tag{43}$$

and from Theorem 2.2, we have

$$m_h + m_\ell = (t_h + \delta_h + \delta_\ell)(1 + \omega_2), \text{ with } |\omega_2| \le 2u^2 + 5u^3. \tag{44}$$

Combining (43) and (44), we obtain

$$\begin{aligned} m_h + m_\ell &= (t_h + e t_h (1 + \omega_1))(1 + \omega_2) \\ &= t_h + e t_h + e t_h \omega_1 + \omega_2 t_h + \omega_2 e t_h + \omega_2 \omega_1 e t_h \\ &= t_h + e t_h + \alpha t_h, \end{aligned} \tag{45}$$

with

$$\begin{aligned} |\alpha| &= |e\omega_1 + \omega_2 + \omega_2 e + \omega_2 \omega_1 e| \\ &\le (2u)(2u^2) + (2u^2 + 5u^3) + (2u^2 + 5u^3)(2u) + (2u^2 + 5u^3)(2u^2)(2u) \\ &= 2u^2 + 13u^3 + 10u^4 + 8u^5 + 20u^6 \\ &\le 2u^2 + 14u^3 \text{ as soon as } p \ge 4. \end{aligned} \tag{46}$$

Therefore,

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

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 (41) and the bounds on η and α,

$$\left|(m_h + m_\ell) - \frac{1}{y}\right| \leq y\left(t_h - \frac{1}{y}\right)^2 + t_h \cdot \left|\frac{5}{2}u^2 + 14u^3\right| \tag{47}$$

Let us now consider

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

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

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

Hence, from (47), we obtain

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

so that

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

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

$$\left|x(m_h + m_\ell) - \frac{x}{y}\right| \leq \frac{x}{y} \cdot \varphi(u), \tag{48}$$

with $\varphi(u) = 2.298u^2 + (1+u)^2\left(\frac{5}{2}u^2 + 14u^3\right)$. Now, from Theorem 17, we have

$$\begin{aligned} |z_h + z_\ell - x(m_h + m_\ell)| &\leq 5u^2|x(m_h + m_\ell)| \\ &\leq 5u^2\tfrac{x}{y} + 5u^2\left|\tfrac{x}{y} - x(m_h + m_\ell)\right| \\ &\leq \tfrac{x}{y}\left(5u^2 + 5u^2\varphi(u)\right). \end{aligned} \tag{49}$$

Combining (48) and (49) we finally obtain

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

□

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 11. In practice, Algorithm 16 is very accurate: the largest relative error found so far in our tests for that algorithm, 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, and 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 wrong. Our results make it possible to have more trust in double-word arithmetic. They also allow us to give some recommendations:

- 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 slightly less accurate, yet slightly faster, than Algorithm 7, hence one cannot say that one of these algorithms 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 11 is to be favoured: 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 12 for dividing a double-word number by a floating-point number: Algorithm 13, presented in this paper, always returns the same result and is faster;
- there is no point in using Algorithm 14 for dividing two double-word numbers: Algorithm 15, 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 16: 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 obtained 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 found in experiments	♯ 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	$2u^2$ (wrong)	$3u^2 + 13u^3$	$2.25u^2$	20
DW × FP	Algorithm 7	$4u^2$	$2u^2$	$1.5u^2$	10
	Algorithm 8	?	$3u^2$	$2.517u^2$	7
	Algorithm 9	N/A	$2u^2$	$1.984u^2$	6
DW × DW	Algorithm 10	$11u^2$	$7u^2$	$4.9916u^2$	9
	Algorithm 11	N/A	$5u^2$	$3.936u^2$	9
DW ÷ FP	Algorithm 12	$4u^2$	$3.5u^2$	$2.95u^2$	16
	Algorithm 13	N/A	$3.5u^2$	$2.95u^2$	10
DW ÷ DW	Algorithm 14	?	$15u^2 + 56u^3$	$8.465u^2$	24
	Algorithm 15	N/A	$15u^2 + 56u^3$	$8.465u^2$	18
	Algorithm 16	N/A	$9.8u^2$	$5.922u^2$	31

References

[1] 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.

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

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

[4] 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.

[5] 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.

[6] 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`.

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

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

[9] 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`.

[10] 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.

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

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

[13] 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.

[14] 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.

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

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

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