Algorithms for triple-word arithmetic

Nicolas Fabiano, Jean-Michel Muller

Algorithms for triple-word arithmetic

Nicolas Fabiano and Jean-Michel Muller

Abstract—Triple-word arithmetic consists in representing high-precision numbers as the unevaluated sum of three floating-point numbers. We introduce and analyze various algorithms for manipulating triple-word numbers. Our new algorithms are faster than what one would obtain by just using the usual floating-point expansion algorithms in the special case of expansions of length 3, for a comparable accuracy.

1 Introduction and notation

Numerical computations sometimes require a precision significantly higher than the one offered by the basic floating-point formats. This occurs for instance when evaluating transcendental functions with correct rounding: it is almost impossible to guarantee last-bit accuracy in the final result if all intermediate calculations are done in the target precision. For instance, the CRLibm library [3] of correctly rounded elementary functions uses "double-double" or "triple-double" [12], [13] operations in the last steps of the evaluation of approximating polynomials.

Double-word and Triple-word arithmetics consist in representing a real number as the unevaluated sum of two and three floating-point numbers, respectively. They are frequently called "double double" and "triple double", because in practice the underlying format being used is the binary64/double precision format of the IEEE 754 Standard on Floating-Point Arithmetic [6], [17].

Double-word arithmetic had been useful for implementing BLAS [15]. Bailey, Barrio and Borwein [1] give several timely examples in mathematical physics and dynamics where higher precisions are needed.

A generalization of these arithmetics is the notion of floating-point expansion [21], [23], [9], where a high-precision number is represented as the sum of n floating-point numbers.

Arbitrary precision libraries such as GNU-MPFR [5] have the advantage of being versatile, but may involve a significant penalty in terms of speed and memory consumption if one only requires computations accurate within around 150 bits in a few critical parts of a numerical program.

Algorithms for double word arithmetic have been presented in [14], [8]. The purpose of this paper is to introduce and analyze efficient algorithms for performing the arithmetic operations in triple word arithmetic. Our goal is to obtain algorithms that are faster to the ones we could obtain simply by using floating-point expansion algorithms in the particular case $n = 3$, for a comparable accuracy.

In the following, we assume a radix-2, precision-p floating-point (FP) arithmetic system, with unlimited exponent range and correct rounding. As a consequence, our results will apply to "real-world" binary floating-point arithmetic, such as the one specified by the IEEE 754-2008 Standard provided that underflow and overflow do not occur. We also assume the availability of an FMA (*fused multiply-add*) instruction. Such an instruction evaluates expressions of the form $ab + c$ with one final rounding only.

The notation $a|b$ means "a divides b". The notation $\mathrm{RN}(t)$ stands for t rounded to the nearest FP number, ties-to-even, and $\mathrm{RU}(t)$ (resp. $\mathrm{RD}(t)$) stands for t rounded towards $+\infty$ (resp. $-\infty$). If $x \neq 0$ is a real number, then we define three functions $\mathrm{ufp}(x)$, $\mathrm{ulp}(x)$, and $\mathrm{uls}(x)$ as follows:

- $\mathrm{ufp}(x) = 2^{\lfloor \log_2 |x| \rfloor}$;
- $\mathrm{ulp}(x) = \mathrm{ufp}(x) \cdot 2^{-p+1}$;
- $\mathrm{uls}(x)$ is the largest power of 2 that divides x.

When x is a FP number, $\mathrm{ufp}(x)$ is the weight of its most significant bit, $\mathrm{ulp}(x)$ is the weight of its least significant bit, and $\mathrm{uls}(x)$ is the weight of its rightmost nonzero bit.

The number $u = 2^{-p} = \frac{1}{2}\mathrm{ulp}(1)$ denotes the roundoff error unit. When an arithmetic operation $a \nabla b$ is performed, where a and b are FP numbers, what is effectively computed is $\mathrm{RN}(a \nabla b)$, and if t is a real number and $T = \mathrm{RN}(t)$, then

$$|t - T| \leq \frac{u}{1+u} \cdot |t| \leq u \cdot |t| \text{ and } |t - T| \leq u \cdot |T|,$$

and more precisely

$$|t - T| \leq \frac{1}{2}\mathrm{ulp}(t) \leq \frac{1}{2}\mathrm{ulp}(T).$$

The algorithms presented in this paper (as well as the usual algorithms that manipulate double words or general expansions) use as basic blocks Algorithms 1, 2, and 3 below.

- *N. Fabiano is with École Normale Supérieure, 45 Rue d'Ulm, 75005 Paris, France. E-mail: nicolabiano22@yahoo.fr*
- *J.-M. Muller is with CNRS, Laboratoire LIP, ENS Lyon, INRIA, Université Claude Bernard Lyon 1, Lyon, France. E-mail: jean-michel.muller@ens-lyon.fr*

Algorithm 1 – Fast2Sum(a, b). (3 operations) [4].

Require: $\exists$ integers $k_a \geq k_b, M_a, M_b$ (with $|M_a|, |M_b| \leq 2^p - 1$), s.t. $a = M_a \cdot 2^{k_a}$ and $b = M_b \cdot 2^{k_b}$.
Ensure: $(s, e) = a + b$
$s \leftarrow \text{RN}(a + b)$
$z \leftarrow \text{RN}(s - a)$
$e \leftarrow \text{RN}(b - z)$

If there exist integers $k_a \geq k_b, M_a, M_b$ (with $|M_a|, |M_b| \leq 2^p - 1$), s.t. $a = M_a \cdot 2^{k_a}$ and $b = M_b \cdot 2^{k_b}$ (which holds if $|a| \geq |b|$) then values s and e computed by Algorithm 1 satisfy $s + e = a + b$. Hence, e is the error of the FP addition $s \leftarrow \text{RN}(a + b)$.

Algorithm 2 – 2Sum(a, b). (6 operations) [16], [11].

Ensure: $(s, e) = a + b$
$s \leftarrow \text{RN}(a + b)$
$a' \leftarrow \text{RN}(s - b)$
$b' \leftarrow \text{RN}(s - a')$
$\delta_a \leftarrow \text{RN}(a - a')$
$\delta_b \leftarrow \text{RN}(b - b')$
$e \leftarrow \text{RN}(\delta_a + \delta_b)$

Algorithm 2 requires twice as many operations as Algorithm 1, but its output variables always satisfy $s + e = a + b$: no knowedge of the respective magnitudes of $|a|$ and $|b|$ is needed.

Algorithm 3 – 2Prod(a, b)**.** (2 operations) [10], [18], [17])

Ensure: $(\pi, e) = a \cdot b$
$\pi \leftarrow \text{RN}(a \cdot b)$
$e \leftarrow \text{RN}(a \cdot b - \pi)$ (FMA)

The values π and e computed by Algorithm 3 satisfy $\pi + e = ab$. Algorithm 3 uses an FMA instruction (for computing $\text{RN}(a \cdot b - \pi)$).

Sometimes, we know in advance the value of s of π. In that case, e can be computed saving the first operation, with algorithms denoted by for instance $\text{2Sum}_2(s)(x, y)$.

When one defines a number as the unevaluated sum of two, three or more FP numbers, one has to explain to which extent they can "overlap": after all the sum of the three double-precision/binary64 numbers 1, 2, and 4 is just a three-bit number, expressing it as the sum of three FP numbers does not make it more precise. Several definitions appear in the literature. The first needed in this paper is Priest's definition:

Definition 1. *The sequence (x_i) is P-nonoverlapping (with Priest's definition [22]) when $\forall i, |x_{i+1}| < \text{ulp}(x_i)$.*

We also introduce the following definition, more restrictive than Shewchuk's definition [23]:

Definition 2. *The sequence (x_i) is F-nonoverlapping (with Fabiano's definition) when $\forall i, |x_{i+1}| \leq \frac{1}{2}\text{uls}(x_i)$.*

Definition 3. *For any definition of nonoverlapping, a sequence is said nonoverlapping wIZ (with possible interleaving zeros) when we have a set I_0 such that $\forall i \in I_0, x_i = 0$, and $(x_i)_{i \notin I_0}$ nonoverlapping.*

We can now formally define the double word and triple word numbers:

Definition 4. *We call Double Word number (DW) a pair (x_0, x_1) of FP numbers such that $x_0 = RN(x_0 + x_1)$.*

Definition 5. *We call Triple Word number (TW) a triplet (x_0, x_1, x_2) of FP numbers that is P-nonoverlapping.*

The usual definition of general expansions [20] is based on ulp-nonoverlapping ($\forall i, |x_{i+1}| \leq \text{ulp}(x_i)$), which is slightly less restritive than the one we chose for TW. Therefore algorithms proven for general expansions may not be correct for TW.

The rest of the paper is organized as follows. Section 2 presents some other basic blocks that will be used in the rest of the paper, and some orignal results related to them. Section 3 proves that the classical algorithm to turn a arbitrary sequence into an expansion works for TW. Section 4 presents an algorithm to correctly round a TW, and proves it correctness. Section 5 does the same as section 3 for the sum of two TW. Section 6 presents two versions of an original algorithm for the product of two TW, and proves their correctness and tight error bounds. Sections 7 to 10 provide a similar analysis, in the case of the product of a DW by a TW, the reciprocal of a TW, the quotient of two TW, and the square root of a TW, respectively. Section 11 compares our results to the ones known for general n-word expansions, with $n = 3$.

2 Other basic blocks

The Algorithms on TW presented in this paper use as basic blocks the 2Sum, Fast2Sum and 2Prod algorithms presented in the previous section, as well as the following, less classical, VecSum and VecSumErrBranch algorithms. Properties of these algorithms have been proven elsewhere [21], [19], [2], but in this paper, we will need specific properties, proven below.

2.1 VecSum

The VecSum algorithm first appears as a part of Priest's normalization algorithm [21]. The name "VecSum" was coined by Ogita et al [19]. The aim of this algorithm is to turn a sequence that is "slightly" nonoverlapping into one that is "more" nonoverlapping, with no error.

Algorithm 4 – VecSum$(x_0, \ldots, x_{n-1})$**.** ($6n-6$ operations)

Ensure: $e_0 + \cdots + e_{n-1} = x_0 + \cdots + x_{n-1}$
$s_{n-1} \leftarrow x_{n-1}$
for $i = n - 2$ **to** 0 **do**
$s_i, e_{i+1} \leftarrow \text{2Sum}(x_i, s_{i+1})$
end for
$e_0 \leftarrow s_0$

There are several theorems related to this algorithm, that use different definitions of nonoverlapping. In what follows, we will use the following original result:

Theorem 1. *Assume, after removing possible interleaving zeros, that we can write in a non-necessarily canonical way* $x_i = M_i 2^{k_i - p + 1}$, $|M_i| < 2^p$, *such that* $\forall i \leq n-2, k_{i-1} \geq k_i + 1$, *and* $k_{n-2} \geq k_{n-1}$. *Then VecSum*$(x_0, \ldots, x_{n-1})$ *is F-nonoverlapping wIZ with the same sum.*

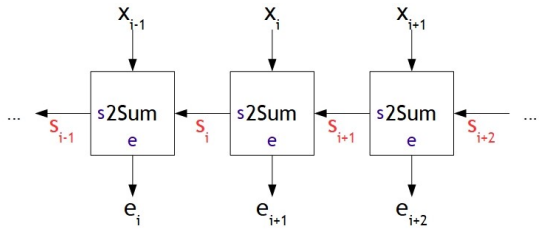
In this case, Fast2Sum can be used instead of 2Sum, so that VecSum only costs $3n-3$ *operations.*

Proof: Interleaving zeros in the input simply give some interleaving zeros in the output without changing the non-zero terms, so that we can suppose that we have removed them.

Firstly, $\forall i, |s_i| \leq (2-2u)2^{k_{i-1}}$.

Indeed, if by induction $|s_{i+1}| \leq (2-2u)2^{k_i}$, given $|x_i| \leq (2-2u)2^{k_i}$ we get $|s_{i+1}| + |x_i| \leq (4-4u)2^{k_i} \leq (2-2u)2^{k_{i-1}}$ so $|s_i| \leq (2-2u)2^{k_{i-1}}$.

This gives $|e_i| \leq 2u2^{k_{i-1}}$, and justifies Fast2Sum being used.



We suppose that $|e_i| > \frac{1}{2}\text{uls}(e_{i'})$ with $i' < i$. We also suppose WLOG that $\text{uls}(e_{i'}) = u$. We easily get by induction that for all i, if $2^k | s_i, x_{i-1}, \ldots, x_0$, then $2^k | e_i, \ldots, e_0$. Yet $|e_i| \leq \frac{1}{2}\text{ulp}(s_{i-1})$ so $|s_{i-1}| \geq 1$ so s_{i-1} is a multiple of $2u$. Given we want a $e_{i'}$ non-multiple of $2u$, that must be the case for one of the $x_j, j \leq i-2$. In particular, we have $2^{k_j} \leq \frac{1}{2}$ so by isotony $2^{k_{i-2}} \leq \frac{1}{2}$ so $2^{k_{i-1}} \leq \frac{1}{4}$, which contradicts $|e_i| \leq 2u2^{k_{i-1}}$. □

The conditions on the input of this theorem are complex, so we will use the following corollary:

Corollary 2. *Assume that we have* $I \subset [\![1, n-2]\!]$ *with no 2 consecutive indices such that*

$$\forall i \in [\![0, n-2]\!], i \notin I, \text{ufp}(x_{i+1}) \leq \frac{1}{2}\text{ufp}(x_i)$$

$$\forall i \in I, \text{ufp}(x_{i+1}) \leq 2^{p-2}\text{uls}(x_i) \wedge \text{ufp}(x_{i+1}) \leq \frac{1}{4}\text{ufp}(x_{i-1})$$

Then VecSum$(x_0, \ldots, x_{n-1})$ *is F-nonoverlapping with the same sum. In this case, Fast2Sum can be used, so that it only costs* $3n-3$ *operations.*

Proof: For $i \notin I$, we take $k_i = e_{x_i}$ the canonical exponent, and for $i \in I$, we take $k_i = \max(k_{i+1}+1, e_{x_i})$. This is possible because: $2^{k_{i+1}-p+2} | x_i$ and $2^{e_{x_i}-p+1} | x_i$, which imply $2^{k_i - p + 1} | x_i$, and $|x_i| \leq 2 \cdot 2^{e_{x_i}}$, which imply $|x_i| \leq 2 \cdot 2^{k_i}$.

For $i, i+1 \notin I$, we have $k_{i+1} \leq k_i - 1$. For $i \in I$, we have on one hand $k_{i+1} \leq k_i - 1$, and on the other hand $e_{x_i} \leq k_{i-1} - 1$ and $k_{i+1} \leq k_{i-1} - 2$ so $k_i \leq k_{i-1} - 1$.

□

2.2 VecSumErrBranch

Algorithm 5 below is similar to the previous one, but sums are computed starting from the larger terms, and some branching helps avoiding to return too many zero terms

Algorithm 5 – VSEB$(e_0, \ldots, e_{n-1})$**.** ($6n-6$ operations & $n-2$ tests)

Ensure: $y_0 + \cdots + y_{n-1} = e_0 + \cdots + e_{n-1}$
  $j \leftarrow 0$
  $\epsilon_0 \leftarrow e_0$
  for $i = 0$ **to** $n-3$ **do**
    $r_i, \epsilon_{i+1} \leftarrow \text{2Sum}(\epsilon_i + e_{i+1})$
    if $\epsilon_{i+1} \neq 0$ **then**
      $y_j \leftarrow r_i$
      incr j
    else
      $\epsilon_{i+1} \leftarrow r_i$
    end if
  end for
  $y_j, y_{j+1} \leftarrow \text{2Sum}(\epsilon_{n-2} + e_{n-1})$
  $y_{j+2}, \ldots, y_{n-1} \leftarrow 0$

We prove the following property, of Algorithm 5, that will be useful later on.

Theorem 3. *If* (e_i) *is F-nonoverlapping wIZ, then VSEB*$(x_0, \ldots, x_{n-1})$ *is P-nonoverlapping with the same sum, provided that* $p \geq n-1$. *In this case, Fast2Sum can be used, so that it only costs* $3n-3$ *operations.*

Proof: Again, interleaving zeros in the input can be ignored without changing anything, so we suppose that we have removed them. We write $e_i = M_i \cdot 2^{k_i}$ with $|M_i| < 2^p$ odd so that $|e_{i+1}| \leq \frac{1}{2}2^{k_i}$. By an easy induction, for all i, r_{i-1} and ϵ_i are multiples of 2^{k_i}.

$\star$ Let i_0 be such that $|\epsilon_{i_0}| = 2^{k_{i_0}}$. Let us show by induction that for all $i_0 \leq i \leq n-2$, we have $|r_i| \leq 2^{k_{i_0}}(2 - 2^{i_0 - i - 1})$.

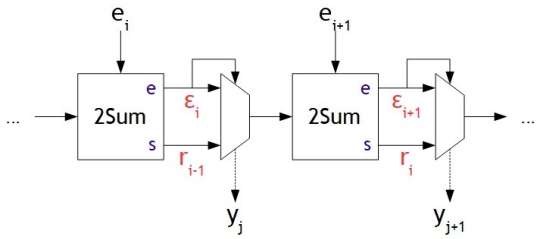
- We can initialize with $i = i_0 - 1$, because what is transmitted to next step (playing the role of r_{i_0-1}) is ϵ_{i_0}, which exactly satisfies the condition.

- We suppose that $|r_{i-1}| \leq 2^{k_{i_0}}(2 - 2^{i_0 - i})$.

We have $|e_{i+1}| \leq \frac{1}{2}2^{k_i} \leq \frac{1}{4}2^{k_{i-1}} \leq \cdots \leq 2^{i_0 - i - 1}2^{k_{i_0}}$.

Thus $|r_{i-1}| + |e_{i+1}| \leq 2^{k_{i_0}}(2 - 2^{i_0 - i - 1})$, and this is a FP number because $i_0 \geq 1$ and $i \leq n-2$ give $i_0 - i - 1 \geq -n + 2 \geq -p + 1$, so that $|r_i| \leq 2^{k_{i_0}}(2 - 2^{i_0 - i - 1})$. This gives the result by induction.

In particular, with j such that $y_j = r_{i_0 - 1}$, we have $\text{ulp}(y_j) \geq 2|\epsilon_{i_0}| = 2 \cdot 2^{k_{i_0}}$ so $|y_{j+1}| < \text{ulp}(y_j)$.



$\star$ For i_0 such that $|\epsilon_{i_0}| > 2^{k_{i_0}}$, we similarly get that for all $i \geq i_0$ such that $\epsilon_{i_0+1}, \ldots, \epsilon_i = 0$, we have $|r_i| \leq |\epsilon_{i_0}| + 2^{k_{i_0}}$.

Indeed, the same proof replacing $2^{k_{i_0}}(2 - \cdots)$ by $|\epsilon_{i_0}| + 2^{k_{i_0}}(1 - \cdots)$ works, except that the equality case can be reached in case of errors. In this case, this is sufficient given $\mathrm{ulp}(y_j) \geq 2|\epsilon_{i_0}| > |\epsilon_{i_0}| + 2^{k_{i_0}} \geq y_{j+1}$.

$\star$ Finally, we can use Fast2Sum. Indeed, we have $2^{k_i} | e_0, \ldots, e_i$ so $2^{k_i} | \epsilon_i$, and $|e_{i+1}| \leq 2^{k_i}$. □

Usually, we do not want to keep the complete output, but only a fixed number of terms. The resulting algorithm is denoted by $\mathrm{VSEB}(k)(e_0, \ldots, e_{n-1})$.

It costs as many operations, but the number of tests depends on how this is performed in practice. In what follows, we will assume in our complexity analysis that everything is unrolled (which is reasonable because typically $n = 4$), so that only the $n - 2$ tests are required.

Theorem 4. *If the total output would be P-nonoverlapping, then the relative error caused by keeping only the first k terms is bounded by $2u^k - 4.2u^{k+1}$, provided that $p \geq 6$.*

Proof: We have by P-nonoverlapping:

$$\mathrm{ufp}(y_k) \leq u\mathrm{ufp}(y_{k-1}) \leq \cdots \leq u^k \mathrm{ufp}(y_0)$$

$$|y_k| + \cdots + |y_{n-1}| \leq (2-2u)(u^k + u^{k+1} + \cdots + u^n)\mathrm{ufp}(y_0)$$

$$|y_k + \cdots + y_{n-1}| \leq 2u^k \mathrm{ufp}(y_0)$$

$$|y_0 + \cdots + y_{n-1}| \geq |y_0| - |y_1 + \cdots + y_{n-1}| \geq (1-2u)\mathrm{ufp}(y_0)$$

$$|y_k + \cdots + y_{n-1}| \leq \frac{2u^k}{1-2u}|y_0 + \cdots + y_{n-1}|$$

Thus the relative error is bounded by $2u^k + 4.2u^{k+1}$, provided that $p \geq 5$. □

2.3 Composed algorithm

Algorithms VecSum and VSEB were designed to be composed. In that case, we note that the first 2Sum in VSEB can be skipped because (e_0, e_1) is already a DW. When Fast2Sum can be used everywhere, we get a total of $6n - 9$ operations and $n - 2$ tests.

3 Arbitrary three FP numbers to TW

Before manipulating Triple-Word numbers, we want to be able to turn any unevaluated sum of three FP numbers into a TW, with no error (this is the equivalent of the 2Sum algorithm for 3 FP numbers). We can use Algorithm 6, which can be found in [20, page 87] for general expansions:

Algorithm 6 – ToTW(a, b, c)**.** (21 operations & 1 test)

Ensure: $\bar{r}$ TW **and** $\bar{r} = a + b + c$
  $d_0, d_1 \leftarrow \mathrm{2Sum}(a, b)$
  $e_0, e_1, e_2 \leftarrow \mathrm{VecSum}(d_0, d_1, c)$
  $r_0, r_1, r_2 \leftarrow \mathrm{VSEB}(e_0, e_1, e_2)$

We have,

Theorem 5. *If a, b, c are FP numbers, then $toTW(a, b, c)$ is a TW, provided that $p \geq 4$.*

Proof: If (e_0, e_1, e_2) is F-nonoverlapping, then Theorem 3 concludes.

First, $|e_1| \leq \frac{1}{2}\mathrm{ulp}(e_0)$ gives (e_0, e_1) F-nonoverlapping. We denote $s := \mathrm{RN}(d_1 + z)$ the intermediate value in $\mathrm{VecSum}(d_0, d_1, c)$.

- If $e_1 \neq 0$, we suppose WLOG that $\mathrm{uls}(e_1) = u$, and $e_2 > \frac{1}{2}u$. Then $|e_2| \leq \frac{1}{2}\mathrm{ulp}(s)$ gives $s \geq 1$ so $2u | s$ but e_1 is not divisible by $2u$ so d_0 neither, hence $d_0 < 1$, so $|d_1| \leq \frac{1}{2}\mathrm{ulp}(d_0) \leq \frac{1}{2}u$.
  Furthermore, $|c + d_1| \geq 1 + \frac{1}{2}u$. Thus $|c| \geq (1 + \frac{1}{2}u) - \frac{1}{2}u = 1$ so $\mathrm{ulp}(c) \geq 2u > 2|d_1|$ so $s = c$ and $e_2 = d_1$, which is impossible since $|e_2| > \frac{1}{2}u \geq |d_1|$.
  So (e_1, e_2) is F-nonoverlapping too.
- If $e_1 = 0$, then the same reasoning works with e_0 instead of e_1.

□

Another typical way of forming a TW consists in using any of the following algorithms, but with inputs that are DW instead of TW (with an implicit third term equal to zero, and simplifications obtained by removing useless operations).

4 Rounding of TW to FP number

Once our calculations are done, we may want to obtain the FP number closest to our TW result. This is for instance the case when we are using TW numbers in intermediate calculations for implementing correctly rounded elementary functions. This can be done using Algorithm 7.

Algorithm 7 – RoundTW(x_0, x_1, x_2)**.** (3 operations & 4 tests)

Require: $\bar{x}$ TW
Ensure: $y = \mathrm{RN}(\bar{x})$
  if $\mathrm{RN}(x_0 + 2x_1)$ inexact operation **or** $(\star)$ $\mathrm{RN}(-(\frac{3}{2}u - 2u^2) \cdot x_0) \neq x_1$ **then**
    $y \leftarrow \mathrm{RN}(x_0 + x_1)$
  else if $x_2 > 0$ **then**
    $y \leftarrow \mathrm{RU}(x_0 + x_1)$
  else if $x_2 < 0$ **then**
    $y \leftarrow \mathrm{RD}(x_0 + x_1)$
  else
    $y \leftarrow \mathrm{RN}(x_0 + x_1)$
  end if

Remark 1. *Depending on the underlying architecture, the fact that the operation $x_0 + 2x_1$ is exact can be detected by checking a flag, but if this is too expensive to do, we can instead start computing Fast2Sum$(x_0, 2x_1)$ and test whether $z = y$ to know if e is zero. This costs 2 more operations.*

Remark 2. *Instead of testing $x_2 > 0$ (resp. $x_2 < 0$), it may be better to test something like $x_2 > 64u|x_1|$ (resp. $x_2 < -64u|x_1|$) and to raise a warning if none of them is true, because it means that we are in the "grey zone" of rounding errors.*

Theorem 6. *If $\bar{x}$ is a TW, then $FromTW(\bar{x}) = RN(\bar{x})$, provided that $p \geq 4$.*

Proof: First, if x_0+x_1 is a FP number, then $y = x_0+x_1$ anyway, and it is easy to check that $x_0 + x_1 = \mathrm{RN}(\bar{x})$. We suppose for the rest of the proof that this is not the case.

Given $|x_1| < \mathrm{ulp}(x_0)$, the first condition is false iff $x_0 + x_1$ is halfway between two adjacent FP numbers, or in a special case that can WLOG be reduced to $x_0 = 1+2u$ and $x_1 = -\frac{3}{2}u$. When that first condition is false, the condition $(\star)$ is designed to be true in the special case, but false elsewise (because of the magnitude of $|x_1|$).

- If x_0+x_1 is halfway between two adjacent FP numbers, then the rounding is decided by the sign of x_2.

- Otherwise, one easily checks that $\mathrm{RN}(x_0 + x_1 + x_2) = \mathrm{RN}(x_0 + x_1)$, given $|x_2| < \mathrm{ulp}(x_1)$. □

Remark 3. *When we implement correctly rounded elementary functions, we may know that when TW are needed, this is because the final value $\bar{x}$ is close to the midpoint of two adjacent FP numbers, so that Condition $(\star)$ can be skipped. This saves 1 operation and 1 test.*

5 Sum

To compute the sum of two TW, we simply use the composition of VecSum and VSEB after a preliminary sort of the input. This gives Algorithm 8 below. That algorithm would of course also work to compute the sum of a DW and a TW, or of two DW, with same correctness theorem and same error bound but with less operations and tests.

Algorithm 8 – 3Sum$(x_0, x_1, x_2, y_0, y_1, y_2)$**.** (42 operations & 8 tests)

Require: $\bar{x}$ and $\bar{y}$ TW
Ensure: $\bar{r}$ TW **and** $\left|\frac{\bar{r}-(\bar{x}+\bar{y})}{\bar{x}+\bar{y}}\right| \leq 2u^3 + 4.2u^4$
$z_0, \ldots, z_5 \leftarrow Merge((x_0, x_1, x_2), (y_0, y_1, y_2))$
$e_0, \ldots, e_5 \leftarrow \mathrm{VecSum}(z_0, \ldots, z_5)$
$r_0, r_1, r_2 \leftarrow \mathrm{VSEB}(3)(e_0, \ldots, e_5)$

5.1 Correctness of Algorithm 8

We have,

Theorem 7. *Let $x_0, \ldots, x_5$ be FP numbers such that*

$$\forall i, |x_{i+1}| \leq |x_i| \text{ and } \forall i, |x_{i+2}| < \mathrm{ulp}(x_i).$$

Then VSEB (VecSum$(x_0, \ldots, x_5)$) is P-nonoverlapping, provided that $p \geq 4$.

Remark 4. *This theorem may not hold for more than 6 floating-point inputs. Indeed, for 7 inputs, we can consider*

$$(x_i) = 1-u, -1+2u, -u+u^2, u-u^2, u^2-u^3, u^2-u^3, u^3-u^4$$

which gives $(e_i) = u, u^2, u^2, -u^3, -u^4$, and finally

$$(y_i) = u, 2u^2, -u^3, -u^4$$

with $2u^2 = \mathrm{ulp}(u)$.

This is why it is reasonable to use the notion of P-nonoverlapping for TW only, but not for general expansions, for which Algorithm 8 preserves ulp*-nonoverlapping only [20, page 90].*

Sketch of the proof: For space constraints, the proof is not detailed. The main steps are:

- prove by induction that $|s_i| \leq 2\mathrm{ufp}(x_{i-1})$ and $|s_i| \leq 4\mathrm{ufp}(x_i)$;
- if $e_i > \frac{1}{2}\mathrm{uls}(e_j)$ for some $j < i$, deduce some conditions on i and the nearby terms in various cases;
- conclude with a case study: $i \leq 3$ and $e_i > \frac{1}{2}\mathrm{uls}(e_j)$; $i \geq 4$ and $e_i > \frac{1}{2}\mathrm{uls}(e_j)$; or $0 < e_i \leq \frac{1}{2}\mathrm{uls}(e_j)$.

□

5.2 Number of operations

In the $Merge$, if the last two numbers to sort are x_2 and y_2, there is no need to do it because they play symmetrical roles in 2Sum. Thus this part costs only 4 tests. In VecSum, there are for each block examples where Fast2Sum cannot be used, so it costs 30 operations. One easily checks that Fast2Sum can be used in VSEB, so it costs 12 operations and 4 tests.

6 Product of two TW

To compute the product of two TW, we simply distribute the sub-products and aggregate the terms ensuring P-nonoverlapping, with an error as small as possible. The algorithms presented below guarantee commutativity, even if it is rarely useful in practice.

Algorithm 9 – 3Prod$^{acc}_{3,3}(x_0, x_1, x_2, y_0, y_1, y_2)$**.** (46 operations & 2 tests)

Require: $\bar{x}$ and $\bar{y}$ TW ; $p \geq 6$
Ensure: $\bar{r}$ TW **and** $\left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| \leq 28u^3 + 107u^4$
$z_{00}^+, z_{00}^- \leftarrow \mathrm{2Prod}(x_0, y_0)$
$z_{01}^+, z_{01}^- \leftarrow \mathrm{2Prod}(x_0, y_1)$
$z_{10}^+, z_{10}^- \leftarrow \mathrm{2Prod}(x_1, y_0)$
$b_0, b_1, b_2 \leftarrow \mathrm{VecSum}(z_{00}^-, z_{01}^+, z_{10}^+)$
$c \leftarrow \mathrm{RN}(b_2 + x_1y_1)$ (FMA)
$z_{3,1} \leftarrow \mathrm{RN}(z_{10}^- + x_0y_2)$ (FMA)
$z_{3,2} \leftarrow \mathrm{RN}(z_{01}^- + x_2y_0)$ (FMA)
$z_3 \leftarrow \mathrm{RN}(z_{3,1} + z_{3,2})$
$e_0, e_1, e_2, e_3, e_4 \leftarrow \mathrm{VecSum}(z_{00}^+, b_0, b_1, c, z_3)$
$r_0 \leftarrow e_0$
$r_1, r_2 \leftarrow \mathrm{VSEB}(2)(e_1, e_2, e_3, e_4)$

6.1 Estimates of the different terms

We suppose WLOG that $1 \le x_0, y_0 < 2$, so that $|x_1|, |y_1| < 2u$ and $|x_2|, |y_2| < 2u^2$. Then, we have:

$$1 \le |z_{00}^+| < 4$$
$$|z_{00}^-| \le 2u \ ; \ \text{uls}(z_{00}^-) \ge 4u^2$$

$$|z_{01}^+|, |z_{10}^+| < 4u \qquad |b_2| \le 4u^2$$
$$|x_1 y_1| < 4u^2 - 4u^3 \qquad |c| < 8u^2$$

$$|z_{01}^-|, |z_{10}^-| \le 2u^2 \qquad |x_0 y_2|, |x_2 y_0| < 4u^2$$
$$|z_{3,1}|, |z_{3,2}| \le 6u^2 \qquad |z_3| \le 12u^2$$

$$|s_3| \le 20u^2$$

6.2 Correctness of Algorithm 9

We have,

Theorem 8. *If $\bar{x}$, $\bar{y}$ are TW, then $3Prod_{3,3}^{acc}(\bar{x}, \bar{y})$ is a TW, provided that $p \ge 6$.*

Lemma 9. *For all FP numbers x, y, $\frac{1}{2}\text{ulp}(x) \big| RN(x+y)$.*

Proof of the lemma: If $\text{ufp}(y) \ge \frac{1}{2}\text{ufp}(x)$, then $\frac{1}{2}\text{ulp}(x) \big| x + y$ so $\frac{1}{2}\text{ulp}(x) \big| \text{RN}(x+y)$.

If $\text{ufp}(y) \le \frac{1}{4}\text{ufp}(x)$, then $|y| < \frac{1}{2}|x|$ so $|x+y| \ge \frac{1}{2}\text{ufp}(x)$ so $\frac{1}{2}\text{ulp}(x) \big| \text{RN}(x+y)$. □

Proof of the theorem:

⋆ First, let us prove that the last 2 lines are equivalent to computing $r_0, r_1, r_2 = \text{VSEB}(3)(e_0, e_1, e_2, e_3, e_4)$

- If $e_1 \neq 0$, then the fact that $e_0 = \text{RN}(e_0 + e_1)$ concludes immediately.
- If $e_1 = 0$, one easily checks that $|s_1|, |s_2|, |s_3| < 16u \le \frac{1}{2}\text{ufp}(z_{00}^+)$ so that the next nonzero $|e_i|$ is strictly less than $\frac{1}{2}\text{ulp}(e_0)$, which concludes.

⋆ Then, let us prove that, with this equivalent version, (r_0, r_1, r_2) is P-nonoverlapping.

We denote $a := \text{RN}(z_{01}^+ + z_{10}^+)$ and $s_3 := \text{RN}(c + z_3)$ intermediate sums in VecSum.

We want to show that $\text{VecSum}(z_{00}^+, b_0, b_1, s_3)$ is F-nonovelapping, with e_4 F-nonovelapping them too.

Thanks to Theorem 3, this would give (r_0, r_1, r_2) P-nonoverlapping.

- First, let us show that $(z_{00}^+, b_0, b_1, s_3)$ satisfies the conditions of Theorem 2.

First, we always have $\text{ufp}(z_{00}^+) \ge 1$ much larger than four times any other number computed ; and in case on they are non-zero $\text{ufp}(b_1) \le \frac{1}{2}\text{ulp}(b_0) < \frac{1}{2}\text{ufp}(b_0)$.

For the rest of the sequence, we suppose WLOG that $|x_1| \ge |y_1|$.

On one hand, we easily obtain $|s_3| \le 10\text{ulp}(x_1)$.

On the other hand lemma 9 gives $\frac{1}{2}\text{ulp}(x_1) \big| a$ with $\frac{1}{2}\text{ulp}(x_1) \le u^2 < \text{uls}(z_{00}^-)$ so $\frac{1}{2}\text{ulp}(x_1) \big| b_0, b_1$.

- Case $s_3 = 0$: we use $I = \emptyset$. (nothing more to show)
- Case $s_3 \neq 0$ and $b_0 = 0$ (so $b_1 = 0$): we use $I = \emptyset$. (idem)
- Case $s_3 \neq 0$, $b_0 \neq 0$ and $b_1 = 0$: we use $I = \{1\}$.
  Indeed, we have $\text{ufp}(s_3) \le \frac{1}{4}\text{ufp}(z_{00}^+)$, and $\text{ufp}(s_3) \le 2^{p-2}\text{uls}(b_0)$. (using $p \ge 6$)
- Case $s_3 \neq 0$, $b_0 \neq 0$ and $b_1 \neq 0$: we use $I = \{2\}$.
  Indeed, we have $\text{ufp}(s_3) \le 16\text{ufp}(b_1) \le \frac{1}{4}\text{ufp}(b_0)$ and $\text{ufp}(s_3) \le 2^{p-2}\text{uls}(b_1)$. (using $p \ge 6$)

It also allows us to call Fast2Sum instead of the three 2Sum in this part of the algorithm.

- Then, let us show that e_4 is F-nonoverlapping with the other e_i.

Firstly, $\text{ulp}(s_3) \ge 2|e_4|$. Secondly, we have seen that $\text{uls}(b_0), \text{uls}(b_1) \ge \frac{1}{2}\text{ulp}(x_1) \ge \frac{1}{20}|s_3| \ge \text{ulp}(s_3)$. (using $p \ge 6$). Thirdly, $\text{ulp}(z_{00}^+) \ge 2u > \text{ulp}(s_3)$.

Thus e_0, e_1 and e_2 are divisible by $\text{ulp}(s_3) \ge 2|e_4|$. □

6.3 Number of operations

Before the last 3 lines, we count 22 operations. There are 3 Fast2Sum in $\text{VecSum}(z_{00}^+, b_0, b_1, c, z_3)$, so that it costs 15 operations. Finally, the call to VSEB costs 9 operations and 2 tests. Thus the total is 46 operations and 2 tests.

Remark 5. *The optimization that directly uses $e_0 = r_0$ does not save any operation, but it saves the first branching, which is very interesting.*

6.4 Bound on the error

Theorem 10. *If $\bar{x}$, $\bar{y}$ are TW, then the relative error committed by $3Prod_{3,3}^{acc}(\bar{x}, \bar{y})$ is bounded by $28u^3 + 107u^4$, provided that $p \ge 6$.*

Proof: There are three sources of error: the terms that are ignored, the roundings in the computation of z_3 and c and the terms not kept in VSEB. A naive analysis gives:

$$\begin{aligned} |\epsilon_0| &:= |x_1 y_2 + x_2 y_1 + x_2 y_2| \\ &\le 2(2u - 2u^2)(2u^2 - 2u^3) + (2u^2 - 2u^3)^2 \\ &\le 8u^3 - 11.9u^4 \end{aligned}$$

$$\begin{aligned} |\epsilon_1| &:= |(z_{10}^- + x_0 y_2) - z_{3,1}| \\ &\le u\text{ufp}(z_{10}^- + x_0 y_2) \\ &\le u\text{ufp}(2u^2 + 4u^2) \\ &\le 4u^3 \end{aligned}$$

$$\begin{aligned} |\epsilon_2| &:= |(z_{01}^- + x_2 y_0) - z_{3,2}| \\ &\le 4u^3 \end{aligned}$$

$$\begin{aligned} |\epsilon_3| &:= |(z_{3,1} + z_{3,2}) - z_3| \\ &\le 8u^3 \end{aligned}$$

$$\begin{aligned} |\epsilon_4| &:= |(b_2 + x_1 y_1) - c| \\ &\le 4u^3 \end{aligned}$$

$$\begin{aligned} |\epsilon_5| &:= |(z_{00}^+ + b_0 + b_1 + c + z_3) - (r_0 + r_1 + r_2)| \\ &\le (2u^3 + 4.2u^4)|z_{00}^+ + b_0 + b_1 + c + z_3| \end{aligned}$$

Yet, $\bar{x}, \bar{y} \ge 1 - (2u - 2u^2) - (2u^2 - 2u^3) \ge 1 - 2u$ so $\bar{x}\bar{y} \ge 1 - 4u$. We eventually obtain that the error cannot

be too large if $\epsilon_5 \neq 0$ (not detailed here, but similar to the proof of Theorem 11), and finally:

$$\begin{array}{rcl}\left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| & \leq & \frac{28u^3-11.9u^4}{1-4u} \\ & \leq & 28u^3+107u^4\end{array}$$

□

This bound is very tight. Indeed, for instance for binary64, if we take:

$$\begin{array}{c}x_0, x_1, x_2 = 1+(13\cdot 2^{26}+28)u, 2u-2^{27}u^2, 2u^2-4u^3 \\ y_0, y_1, y_2 = 1+7\cdot 2^{27}u, 2u-(2^{28}-8)u^2, 2u^2-4u^3\end{array}$$

we obtain a relative error around $(28-10^{-5})u^3$.

6.5 Faster version

In this version, e_4 is not computed.

Algorithm 10 – 3Prod$^{fast}_{3,3}(x_0,x_1,x_2,y_0,y_1,y_2)$. (38 operations & 1 test)

Require: $\bar{x}$ and $\bar{y}$ TW ; $p \geq 6$
Ensure: $\bar{r}$ TW **and** $\left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| \leq 44u^3+176u^4$
[same 5 first lines as 9]
$z_{3,1} \leftarrow \text{RN}(z_{10}^- + x_0y_2)$ (FMA)
$z_{3,2} \leftarrow \text{RN}(z_{01}^- + x_2y_0)$ (FMA)
$z_3 \leftarrow \text{RN}(z_{3,1}+z_{3,2})$
$s_3 \leftarrow \text{RN}(c+z_3)$
$e_0,e_1,e_2,e_3 \leftarrow \text{VecSum}(z_{00}^+, b_0, b_1, s_3)$
$r_0 \leftarrow e_0$
$r_1, r_2 \leftarrow \text{VSEB}(2)(e_1,e_2,e_3)$

This saves 8 operations and 1 test. Correctness is still ensured for the same reasons, provided that $p \geq 6$. Concerning the error bound, the proof is similar (in fact, simpler) than for the previous algorithm, so we omit it, and we obtain

$$\begin{array}{rcl}\left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| & \leq & \frac{44u^3-11.9u^4}{1-4u} \\ & \leq & 44u^3+176u^4\end{array}$$

This bound is very tight, because the same input as for the previous version gives a relative error around $(44-10^{-5})u^3$.

7 Product of a DW with a TW

To compute the product of a DW with a TW, we use the same algorithms as previously, slightly simplified using $x_2 = 0$. Additionally to being interesting in itself, this operation will be used to compute reciprocals and quotients, which justifies a detailed analysis.

Algorithm 11 – 3Prod$^{acc}_{2,3}(x_0,x_1,y_0,y_1,y_2)$. (45 operations & 2 tests)

Require: $\bar{x}$ DW and $\bar{y}$ TW ; $p \geq 6$
Ensure: $\bar{r}$ TW **and** $\left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| \leq 10.5u^3+39u^4$
[same 5 first lines as 9]
$z_{3,1} \leftarrow \text{RN}(z_{10}^- + x_0y_2)$ (FMA)
$z_3 \leftarrow \text{RN}(z_{3,1}+z_{01}^-)$
$e_0,e_1,e_2,e_3,e_4 \leftarrow \text{VecSum}(z_{00}^+, b_0, b_1, c, z_3)$
$r_0 \leftarrow e_0$
$r_1, r_2 \leftarrow \text{VSEB}(2)(e_1,e_2,e_3,e_4)$

7.1 Estimates of the different terms

We suppose WLOG that $1 \leq x_0, y_0 < 2$, so that $|x_1| \leq u$, $|y_1| < 2u$ and $|y_2| < 2u^2$. Then:

$$\begin{array}{c}1 \leq |z_{00}^+| < 4 \\ |z_{00}^-| \leq 2u \ ; \ \text{uls}(z_{00}^-) \geq 4u^2\end{array}$$

$$\begin{array}{cc}|z_{01}^+| < 4u \ ; \ |z_{10}^+| < 2u & |b_2| \leq 4u^2 \\ |x_1y_1| < 2u^2 & |c| \leq 6u^2\end{array}$$

$$\begin{array}{cc}|z_{01}^-| \leq 2u^2 \ ; \ |z_{10}^-| \leq u^2 & |x_0y_2| < 4u^2 \\ |z_{3,1}| \leq 5u^2 & |z_3| \leq 7u^2\end{array}$$

$$|s_3| \leq 13u^2$$

7.2 Correctness and number of operations

Given this is a particular case of the previous algorithm, correctness is directly ensured, provided that $p \geq 6$.

Compared to the previous algorithm, we saved 1 operation, so that the total is 41.

7.3 Bound on the error

Theorem 11. *If $\bar{x}$ DW and $\bar{y}$ TW, then the relative error committed by $3Prod^{acc}_{2,3}(\bar{x},\bar{y})$ is bounded by $10.5u^3+39u^4$, provided that $p \geq 6$.*

Proof: The new naive error analysis gives:

$$\begin{array}{rcccl}|\epsilon_0| & := & |x_1y_2| & \leq & 2u^3-2u^4 \\ |\epsilon_1| & := & |(z_{10}^- + x_0y_2) - z_{3,1}| & \leq & 4u^3 \\ |\epsilon_3| & := & |(z_{3,1}+z_{01}^-) - z_3| & \leq & 4u^3 \\ |\epsilon_4| & := & |(b_2+x_1y_1)-c| & \leq & 4u^3\end{array}$$

$$\begin{array}{rcl}|\epsilon_5| & := & |(z_{00}^+ + b_0 + b_1 + c + z_3) - (r_0+r_1+r_2)| \\ & \leq & (2u^3+4.2u^4)|z_{00}^+ + b_0 + b_1 + c + z_3|\end{array}$$

We get $|\epsilon_0 + \cdots + \epsilon_4| \leq 14u^3 - 2u^4$, and finally

$$\begin{array}{rcl}\left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| & \leq & (2u^3+4.2u^4) + \frac{(14u^3-2u^4)(1+2u^3+4.2u^4)}{1-4u} \\ & \leq & 16u^3+62u^4\end{array}$$

⋆ First, we distinguish cases depending on whether $\bar{x}\bar{y}$ is significantly larger than 1 or not.

- We suppose that $|\epsilon_1| > 2u^2$.
Then, given $|z_{10}^-| \leq u^2$, we must have $|x_0y_2| > 3u^2$ so $|x_0| > 1.5$, so that $\bar{x}\bar{y} \geq 1.5 - 5u$.

- We suppose that $|\epsilon_4| > 2u^2$.

Then, given $|x_1y_1| \leq 2u^2$, we must have $|b_2| > 2u^2$ so $|z_{01}^+ + z_{10}^+| > 4u$. This gives $x_0(2u - 2u^2) + y_0u \geq 4u$, so that $x_0y_0 \geq 1.5$ and $\bar{x}\bar{y} \geq 1.5 - 5u$.

Thus $\bar{x}\bar{y} \geq 1.5 - 5u$ (instead of $1 - 4u$) or $|\epsilon_1|, |\epsilon_4| \leq 2u^3$ (instead of $4u^3$ each).

$\star$ Then, we suppose $\epsilon_5 \neq 0$. By P-nonoverlapping, $\bar{r}$ covers at least $3p$ bits, so this implies that one of the e_i is not divisible by $2u^3\text{ufp}(r_0)$.

- We suppose that $\text{ufp}(r_0) \geq 2$. Then by P-nonoverlapping we must have $\bar{r} \geq 2 - 4u$ so by the naive error bound $\bar{x}\bar{y} \geq 2 - 5u$.

- We suppose that $\text{ufp}(r_0) \leq 1$.

Then we must have a number among z_{00}^+, z_{00}^-, z_{01}^+, z_{10}^+, c and z_3 that is not divisible by $2u^3$, so in particular whose absolute value is strictly smaller than u^2.

- We have seen that z_{00}^+ and z_{00}^- are always divisible by $4u^2$.
- If $|z_{01}^+| < u^2$, then $|x_1| < u^2$ so $|\epsilon_0| \leq 2u^4$ (instead of $2u^3 - 2u^4$)
- If $|z_{10}^+| < u^2$, then $|y_1| < u^2$ so $|y_2| < u^3$ so $|\epsilon_0| \leq u^4$ (instead of $2u^3 - 2u^4$)
- If $|z_3| < u^2$, then $|\epsilon_3| \leq \frac{1}{2}u^3$ (instead of $4u^3$).
- If $|c| < u^2$, then $|\epsilon_4| \leq \frac{1}{2}u^3$ (instead of $4u^3$).

$\star$ Finally, we are left with several cases:

- Case $\bar{x}\bar{y} \geq 2 - 5u$. Then

$$\begin{array}{rcl} \left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| & \leq & (2u^3 + 4.2u^4) + \frac{(14u^3-2u^4)(1+2u^3+4.2u^4)}{2-5u} \\ & \leq & 10u^3 \end{array}$$

- Case $2 - 5u > \bar{x}\bar{y} \geq 1.5 - 5u$ and $\epsilon_5 = 0$. Then

$$\begin{array}{rcl} \left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| & \leq & \frac{14u^3-2u^4}{1.5-5u} \\ & \leq & 10u^3 \end{array}$$

- Case $2 - 5u > \bar{x}\bar{y} \geq 1.5 - 5u$ and $\epsilon_5 \neq 0$. Then

$$\begin{array}{rcl} \left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| & \leq & (2u^3 + 4.2u^4) + \frac{(12u^3+2u^4)(1+2u^3+4.2u^4)}{1.5-5u} \\ & \leq & 10u^3 + 34u^4 \end{array}$$

- Case $1.5 - 6u > \bar{x}\bar{y}$ and $\epsilon_5 = 0$. Then

$$\begin{array}{rcl} \left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| & \leq & \frac{10u^3-2u^4}{1-4u} \\ & \leq & 10u^3 + 41u^4 \end{array}$$

- Case $1.5 - 6u > \bar{x}\bar{y}$ and $\epsilon_5 \neq 0$. Then (the worst case being the one where only $|c| < u^2$, because it is partially redundant with what $1.5 - 6u > \bar{x}\bar{y}$ gives)

$$\begin{array}{rcl} \left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| & \leq & (2u^3 + 4.2u^4) + \frac{(8.5u^3-2u^4)(1+2u^3+4.2u^4)}{1-4u} \\ & \leq & 10.5u^3 + 39u^4 \end{array}$$

□

It is unclear whether this bound is very tight, but it is not so far from optimality. Indeed, for instance for binary64, if we take:

$$x_0, x_1 = 1 + 3 \cdot 2^{27}u, u - 2^{27}u^2$$
$$y_0, y_1, y_2 = 1 + (3 \cdot 2^{26} + 6)u, 2u - 5 \cdot 2^{27}u^2, 2u^2 - 26u^3$$

we get a relative error around $(10 - 2 \cdot 10^{-6})u^3$.

7.4 Faster version

Algorithm 12 – 3Prod$_{2,3}^{fast}(x_0, x_1, y_0, y_1, y_2)$. (37 operations & 1 test)

Require: $\bar{x}$ DW and $\bar{y}$ TW ; $p \geq 6$
Ensure: $\bar{r}$ TW **and** $\left|\frac{\bar{r}-\bar{x}\bar{y}}{\bar{x}\bar{y}}\right| \leq 18u^3 + 75u^4$
[same 5 first lines as 9]
$z_{3,1} \leftarrow \text{RN}(z_{10}^- + x_0y_2)$ (FMA)
$z_3 \leftarrow \text{RN}(z_{3,1} + z_{01}^-)$
$s_3 \leftarrow \text{RN}(c, z_3)$
$e_0, e_1, e_2, e_3 \leftarrow \text{VecSum}(z_{00}^+, b_0, b_1, s_3)$
$r_0 \leftarrow e_0$
$r_1, r_2 \leftarrow \text{VSEB}(2)(e_1, e_2, e_3)$

Similarly to what was done before, we obtain, provided that $p \geq 6$,

$$\left|\frac{\bar{r} - \bar{x}\bar{y}}{\bar{x}\bar{y}}\right| \leq 18u^3 + 75u^4.$$

This bound is very tight, because the same input as for the previous version gives a relative error around $(18 - 2.4 \cdot 10^{-6})u^3$.

8 Reciprocal of a TW

To compute the reciprocal of a TW, we use Algorithm 13 below, which is based on Newton-Raphson iteration, following the idea of [7]. This algorithm requires the stronger constraint $p \geq 10$. For computing $1/x$, the Newton-Raphson iteration is

$$r_{n+1} = r_n(2 - r_nx)$$

which ensures a quadratic convergence towards $\frac{1}{x}$ as soon as r_0 is close enough to $1/x$ because

$$\left|r_{n+1} - \frac{1}{x}\right| = |x| \cdot \left|r_n - \frac{1}{x}\right|^2.$$

Algorithm 13 – 3Reci(x_0, x_1, x_2). (73 operations & 2 tests, resp. 65 operations & 1 test)

Require: $\bar{x}$ TW ; $p \geq 10$
Ensure: $\bar{y}$ TW **and** $\left|\frac{\bar{y}-\bar{x}^{-1}}{\bar{x}^{-1}}\right| \leq 11.5u^3 + 1465u^4$, resp. $19u^3 + 1502u^4$
$a \leftarrow \text{RN}\left((1 + \text{ulp}(1))/x_0\right)$
$h_{1,1} \leftarrow \text{2Prod}_2(1 + \text{ulp}(1))(a, x_0)$
$h_1 \leftarrow \text{RN}(-h_{1,1} - ax_1)$ (FMA)
$b_{0,1}, b_{1,1} \leftarrow \text{2Prod}(a, 1 - \text{ulp}(1))$
$b_{1,2} \leftarrow \text{RN}(b_{1,1} + ah_1)$ (FMA)
$\bar{b} \leftarrow \text{Fast2Sum}(b_{0,1}, b_{1,2})$
$\bar{i} \leftarrow 2 - 3Prod_{2,3}(\bar{b}, \bar{x})$
$\bar{y} \leftarrow 3Prod_{2,3}(\bar{b}, \bar{i})$

8.1 Computation of $\bar{h}$

Algorithm 13 uses $1 + \text{ulp}(1)$ instead of 1 to start the calculations in order to take profit from the following lemma :

Lemma 12. *For any FP number x, $RN(x \times RN(\frac{1+2u}{x})) = 1 + 2u$.*

Proof: Denote $a := \text{RN}(\frac{1+2u}{x})$, which can WLOG be assumed to be in $1, 2[$. If $a \in [1+4u, 2[$, then we conclude using

$$\begin{aligned} &|\text{RN}(xa) - (1+2u)| \\ &\leq (1+2u)\left(\tfrac{u}{1+u} + \tfrac{u}{1+5u} + \tfrac{u^2}{(1+u)(1+5u)}\right) \\ &< 2u \end{aligned}$$

Thus, we are left with only $x \in \{1-2u, 1-u, 1, 1+2u\}$, which are easily treated separately. □

Thus in Algorithm 13, one has to imagine a "virtual" $h_0 = 2 - (1 + \text{ulp}(1)) = 1 - \text{ulp}(1)$.

8.2 Bound on the error

Momentarily, the computation of $\bar{i}$ is seen as a black box (see Section 8.3). We have

$$\begin{aligned} \left|a - \tfrac{1}{x_0}\right| &\leq \left|a - \tfrac{1+2u}{x_0}\right| + \left|\tfrac{1+2u}{x_0} - \tfrac{1}{x_0}\right| \\ &\leq u\left|\tfrac{1+2u}{x_0}\right| + \left|\tfrac{2u}{x_0}\right| \\ &\leq (3u+2u^2)\cdot\left|\tfrac{1}{x_0}\right|, \end{aligned}$$

which implies $\left|\frac{1-3.1u}{x_0}\right| \leq |a| \leq \left|\frac{1+3.1u}{x_0}\right|$, so that $|h_1| \leq (1+u)(u|ax_0| + |a|\cdot 2u|x_0|) \leq 3u + 13u^2$. Now,

$$\begin{aligned} \left|\tfrac{1}{x_0} - \tfrac{1}{x_0+x_1}\right| &= \left|\tfrac{1}{x_0+x_1}\right|\left|1 - \tfrac{x_0+x_1}{x_0}\right| \\ &\leq (2u-2u^2)\left|\tfrac{1}{x_0+x_1}\right| \\ \left|a - \tfrac{1}{x_0+x_1}\right| &\leq (5u+6u^2)\left|\tfrac{1}{x_0+x_1}\right| \\ \left|a(2-a(x_0+x_1)) - \tfrac{1}{x_0+x_1}\right| &= |x_0+x_1|\left(a - \tfrac{1}{x_0+x_1}\right)^2 \\ &\leq (25u^2+61u^3)\left|\tfrac{1}{x_0+x_1}\right| \end{aligned}$$

$$\rightarrow |a| \leq \left|\tfrac{1+6u}{x_0+x_1}\right|$$

$$\begin{aligned} \left|\bar{h} - (2-a(x_0+x_1))\right| &= |h_1 - (-h_{1,1} - ax_1)| \\ &\leq u|h_1| \\ &\leq 3u^2 + 13u^3 \\ \left|\bar{b} - a\bar{h}\right| &= |b_{1,2} - (b_{1,1} + ah_1)| \\ &\leq u|b_{1,1} + ah_1| \\ &\leq u\left(u|a|(1-2u) + |ah_1|\right) \\ &\leq (4u^2+11u^3)|a| \end{aligned}$$

$$\begin{aligned} \left|\bar{b} - \tfrac{1}{x_0+x_1}\right| &\leq (7u^2+24u^3)|a| + (25u^2+61u^3)\left|\tfrac{1}{x_0+x_1}\right| \\ &\leq (32u^2+121u^3)\left|\tfrac{1}{x_0+x_1}\right| \end{aligned}$$

$$\begin{aligned} \left|\tfrac{1}{x_0+x_1} - \tfrac{1}{\bar{x}}\right| &= \left|\tfrac{1}{\bar{x}}\right|\left|1 - \tfrac{\bar{x}}{x_0+x_1}\right| \\ &\leq \tfrac{2u^2}{1-2u}\left|\tfrac{1}{\bar{x}}\right| \\ &\leq (2u^2+5u^3)\left|\tfrac{1}{\bar{x}}\right| \\ \left|\bar{b} - \tfrac{1}{\bar{x}}\right| &\leq (34u^2+126u^3)\left|\tfrac{1}{\bar{x}}\right| \\ \left|\bar{b}(2-\bar{b}\bar{x}) - \tfrac{1}{\bar{x}}\right| &= |\bar{x}|(\bar{b} - \tfrac{1}{\bar{x}})^2 \\ &\leq 1165u^4\left|\tfrac{1}{\bar{x}}\right| \end{aligned}$$

$$\rightarrow \left|\tfrac{1-35u^2}{\bar{x}}\right| \leq |\bar{b}| \leq \left|\tfrac{1+35u^2}{\bar{x}}\right|$$

Remark 6. *This term is ultimately negligible if p is large, because the accuracy is doubled at each step so it jumps from roughly 2 words to roughly 4 instead of just 3. Thus it is not a problem that computations were not performed precisely, for instance starting with $1 + \text{ulp}(1)$ instead of 1.*

We denote δ_1 the relative error committed when computing $\bar{i}$ (taken relatively to $\bar{x}\bar{b}$) and δ_2 the one for $\bar{y}$. They are supposed less than $20u^3$ (see later).

$$\begin{aligned} \left|\bar{i} - (2-\bar{x}\bar{b})\right| &\leq \delta_1|\bar{x}\bar{b}| \\ &\leq \delta_1(1+35u^2) \\ |\bar{i}| &\leq |2-\bar{x}\bar{b}| + \delta_1(1+35u^2) \\ &\leq 1+35u^2 \\ \left|\bar{y} - \bar{b}\bar{i}\right| &\leq \delta_2|\bar{b}\bar{i}| \\ &\leq \delta_2(1+35u^2)|\bar{b}| \end{aligned}$$

$$\begin{aligned} \left|\bar{y} - \tfrac{1}{\bar{x}}\right| &\leq (\delta_1+\delta_2)(1+35u^2)|\bar{b}| + 1165u^4\left|\tfrac{1}{\bar{x}}\right| \\ &\leq \left((\delta_1+\delta_2)(1+70u^2) + 1165u^4\right)\left|\tfrac{1}{\bar{x}}\right|. \end{aligned}$$

To conclude, we only have to bound δ_1 and δ_2 as accurately as possible depending on the algorithms used.

8.3 Computation of $\bar{i}$

We have seen that $|\bar{b}\bar{x} - 1| \leq 35u^2$. Inside the computation of $3Prod_{2,3}(\bar{b}, \bar{x})$, we have seen that $|\bar{e} - \bar{b}\bar{x}| \leq 20u^3$. Thus $|\bar{e} - 1| \leq 36u^2$. Given that $\bar{e}$ is F-nonoverlapping, we have $|e_0 - \bar{e}| \leq (1-2^{-4})\text{uls}(e_0)$. If $|e_0| < 1$, then $|\bar{e}| \geq (1-u) + (1-2^{-4})u = 1 - 2^{-4}u < 1 - 36u^2$, which is excluded, and we can exclude similarly $|e_0| > 1$. Thus $e_0 = 1$.

Remark 7. *This property is the one that necessitates $p \geq 10$. It probably works for some smaller values of p, but we have no proof of that.*

Therefore, in order to compute $2 - 3Prod_{2,3}(\bar{b}, \bar{x})$, we can simply replace $e_1, \ldots$ by their opposites by turning some $+$ into $-$ operations and conversely in order not to waste any operation. Correctness and the error bound are still ensured for the same reasons. For instance, if we choose to use the accurate version, we obtain algorithm 14. One operation (the computation of e_0) is saved compared to the general version.

Algorithm 14 – 2 - 3Prod$^{acc}_{2,3}(b_0, b_1, x_0, x_1, x_2)$. (44 operations & 2 tests)

$z^+_{00}, z^-_{00} \leftarrow$ 2Prod(b_0, x_0)
$z^+_{01}, z^-_{01} \leftarrow$ 2Prod(b_0, x_1)
$z^+_{10}, z^-_{10} \leftarrow$ 2Prod(b_1, x_0)
$b'_0, b'_1, b'_2 \leftarrow$ VecSum$(z^-_{00}, z^+_{01}, z^+_{10})$
$c \leftarrow$ RN$(b'_2 + b_1 x_1)$ (FMA)
$z_{3,1} \leftarrow$ RN$(z^-_{10} + b_0 x_1)$ (FMA)
$z_3 \leftarrow$ RN$(z_{3,2} + z^-_{01})$
$s_1, e_2, e_3, e_4 \leftarrow$ VecSum$(-b'_0, -b'_1, -c, -z_3)$
$(i_0 = 1)$
$e_1 \leftarrow$ Fast2Sum$_2(1)(-z^+_{00}, s_1)$
$i_1, i_2 \leftarrow$ VSEB$(2)(e_1, e_2, e_3, e_4)$

8.4 Computation of $\bar{y}$

We have very precise information about $\bar{i}$, so we use a modified version of the fast algorithm.

Algorithm 15 – 3Prod$^{fast}_{2,3}(b_0, b_1, (1), i_1, i_2)$. (20 operations)

$z^+_{01}, z^-_{01} \leftarrow$ 2Prod(b_0, i_1)
$b'_0, b'_1 \leftarrow$ Fast2Sum(b_1, z^+_{01})
$z_{3,1} \leftarrow$ RN$(z^-_{01} + b_1 i_1)$ (FMA)
$z_3 \leftarrow$ RN$(z_{3,1} + b_0 i_2)$ (FMA)
$s_3 \leftarrow$ RN$(b'_1 + z_3)$
$e_0, e_1, e_2 \leftarrow$ VecSum(b_0, b'_0, s_3)
$y_0 \leftarrow e_0$
$y_1, y_2 \leftarrow$ Fast2Sum(e_1, e_2)

Remark 8. *In VecSum(b_0, b'_0, s_3), the sum of b'_0 and s_3 is performed with a 2Sum in order to ensure correctness, but we can still use a Fast2Sum for the second one.*

A Fast2Sum can be used for the sum of b_1 and z^+_{01} because if the condition for Fast2Sum to be errorless is not satisfied, this means that $|b_1|$ is very small, so that the global error will be small anyway.

Given the estimates on $\bar{i}$, basically all errors are negligible, except the one when computing s_3. We get:

$$\left|\frac{\bar{y} - \bar{b}\bar{i}}{\bar{b}\bar{i}}\right| \leq \frac{u^3 + 256u^4}{(1-2u)(1-35u^2)}$$

Thus we can take $\delta_2 = u^3 + 260u^4$.

8.5 Final error bound and number of operations

If we use the accurate version for the first $3Prod_{2,3}$, then we can use $\delta_1 = 10.5u^3 + 39u^4$ and finally obtain

Theorem 13. *If $\bar{x}$ is a TW, then the relative error committed by $3Reci^{acc}(\bar{x})$ is bounded by $11.5u^3 + 1465u^4$, provided that $p \geq 10$.*

The total cost is 73 operations and 2 tests. If we use the fast version, then we can use $\delta_1 = 18u^3 + 75u^4$ and finally obtain

Theorem 14. *If $\bar{x}$ is a TW, then the relative error committed by $3Reci^{fast}(\bar{x})$ is bounded by $19u^3 + 1502u^4$, provided that $p \geq 10$.*

The total cost is 65 operations and 1 test.

9 Quotient of two TW

In this section, we also assume $p \geq 10$. In order to compute $\bar{z}/\bar{x}$, we could simply compute the product of $\bar{z}$ and the reciprocal of $\bar{x}$. However, this would mean that we compute something like $\bar{z} \times (\bar{b} \times (2 - \bar{b}\bar{x}))$, while it is significantly better to compute $(\bar{z}\bar{b}) \times (2 - \bar{b}\bar{x})$.

Indeed, this allows to parallelize the computations of $\bar{z}\bar{b}$ and $2 - \bar{b}\bar{x}$, and the optimizations allowed by the constraints on $\bar{i}$ are more efficiently used when multiplying by a TW rather than by a DW.

Algorithm 16 – 3Div$(z_0, z_1, z_2, x_0, x_1, x_2)$. (119 operations & 4 tests, resp. 103 operations & 2 test)

Require: $\bar{z}$,$\bar{x}$ TW ; $p \geq 10$
Ensure: $\bar{y}$ TW **and** $\left|\frac{\bar{y} - \bar{z}/\bar{x}}{\bar{z}/\bar{x}}\right| \leq 24u^3 + 1509u^4$, resp. $39u^3 + 1582u^4$
[same 5 first lines as 13]
$\bar{b} \leftarrow$ Fast2Sum$(b_{0,1}, b_{1,2})$
$\bar{i} \leftarrow 2 - 3Prod_{2,3}(\bar{b}, \bar{x})$
$\bar{a} \leftarrow 3Prod_{2,3}(\bar{b}, \bar{z})$
$\bar{y} \leftarrow 3Prod_{3,3}(\bar{a}, \bar{i})$

For the computation of $2 - \bar{b}\bar{x}$, the same algorithm and error bound as previoulsy are used. For the computation of $\bar{z}\bar{b}$, $3Prod_{2,3}$ is used (the relative error is denoted by δ_3). For the computation of the final product, algorithm 17 is used.

Algorithm 17 – 3Prod$^{fast}_{3,3}(a_0, a_1, a_2, (1), i_1, i_2)$. (21 operations)

$z^+_{01}, z^-_{01} \leftarrow$ 2Prod(a_0, i_1)
$b'_0, b'_1 \leftarrow$ Fast2Sum(a_1, z^+_{01})
$z_{3,1} \leftarrow$ RN$(z^-_{01} + a_1 i_1)$ (FMA)
$z_{3,2} \leftarrow$ RN$(z_{3,1} + a_0 i_2)$ (FMA)
$z_3 \leftarrow$ RN$(z_{3,2} + b'_1)$
$s_3 \leftarrow$ RN$(z_3 + a_2)$
$e_0, e_1, e_2 \leftarrow$ VecSum(b_0, b'_0, s_3)
$y_0 \leftarrow e_0$
$y_1, y_2 \leftarrow$ Fast2Sum(e_1, e_2)

The error analysis is similar to the one of Algorithm 15 with an additional $2u^3$, and we get

$$\left|\frac{\bar{y} - \bar{a}\bar{i}}{\bar{a}\bar{i}}\right| \leq \frac{3u^3 + 256u^4}{(1-2u)(1-35u^2)} \leq 3u^3 + 264u^4.$$

Globally, we have,

$$\left|\bar{y} - \frac{\bar{z}}{\bar{x}}\right| \leq \left((\delta_1 + \delta_2 + \delta_3)(1 + 70u^2) + 1165u^4\right)\left|\frac{\bar{z}}{\bar{x}}\right|$$

If we use the accurate versions, then we can use $\delta_1 = 10.5u^3 + 39u^4 = \delta_3$, and we finally obtain

Theorem 15. *If $\bar{x}$,$\bar{z}$ are TW, then the relative error committed by $3Div^{acc}(\bar{z},\bar{x})$ is bounded by $24u^3+1509u^4$, provided that $p \geq 10$.*

The total cost is 119 operations and 4 tests. If we use the fast versions, then we can use $\delta_1 = 18u^3+75u^4 = \delta_3$ and finally obtain

Theorem 16. *If $\bar{x}$,$\bar{z}$ are TW, then the relative error committed by $3Div^{fast}(\bar{z},\bar{x})$ is bounded by $39u^3+1582u^4$, provided that $p \geq 10$.*

The total cost is 103 operations and 2 tests.

10 Square root of a TW

In this section, we assume $p \geq 11$. To compute the square root of a TW, we use Algoirithm 18 below, based again on the Newton-Raphson iteration. Its analysis and the optimizing tricks are very similar to the ones used before for division. The underlying iteration is now

$$r_{n+1} = r_n(1.5 - \frac{1}{2}r_n^2 x),$$

which ensures a quadratic convergence towards $\frac{1}{\sqrt{x}}$ as soon as r_0 is close enough to $\sqrt{x}$, because

$$\left| r_{n+1} - \frac{1}{\sqrt{x}} \right| = \frac{1}{2}\sqrt{x}(r_n\sqrt{x}+2)\left| r_n - \frac{1}{\sqrt{x}} \right|^2$$

Algorithm 18 – 3SqRt(x_0,x_1,x_2)**.** (127 operations & 4 tests, resp. 111 operations & 2 tests)

Require: $\bar{x}$ TW ; $p \geq 11$
Ensure: $\bar{y}$ TW **and** $\left|\frac{\bar{y}-\sqrt{\bar{x}}}{\sqrt{\bar{x}}}\right| \leq 24u^3+10260u^4$, resp. $39u^3+10333u^4$
$a \leftarrow$ RN$\left((1+2\text{ulp}(1))/\text{RN}(\sqrt{x_0})\right)$
$a' = \frac{1}{2}a$ (exact)
$h_0^{(1)}, h_{1,1}^{(1)} \leftarrow$ 2Prod(a, x_0)
$h_1^{(1)} \leftarrow$ RN$(-h_{1,1}^{(1)} - ax_1)$ (FMA)
$h_{0,1}^{(2)}, h_{1,1}^{(2)} \leftarrow$ 2Prod$(a', h_0^{(1)})$
$h_0^{(2)} \leftarrow 1.5 - h_{0,1}^{(2)}$ (exact)
$h_1^{(2)} \leftarrow$ RN$(-h_{1,1}^{(2)} - a'h_1^{(1)})$ (FMA)
$b_{0,1}, b_{1,1} \leftarrow$ 2Prod$(a, h_0^{(2)})$
$b_{1,2} \leftarrow$ RN$(b_{1,1} + ah_1^{(2)})$ (FMA)
$\bar{b} \leftarrow$ Fast2Sum$(b_{0,1}, b_{1,2})$
$\bar{b}' = \frac{1}{2}\bar{b}$ (exact)
$\overline{i^{(1)}} \leftarrow 3Prod_{2,3}(\bar{b}, \bar{x})$
$\overline{i^{(2)}} \leftarrow 1.5 - 3\overline{Prod_{2,3}}(\bar{b}', \overline{i^{(1)}})$
$\bar{y} \leftarrow 3Prod_{3,3}(\overline{i^{(1)}}, \overline{i^{(2)}})$

The computation of $h_0^{(2)}$ is exact because $h_{0,1}^{(2)} \geq 0.5$ (this is why we started with $1+2\text{ulp}(1)$ instead of 1).

The computation of $\overline{i^{(1)}}$ is performed with a $3Prod_{2,3}$ algorithm (relative error bounded by δ_1).

The computation of $\overline{i^{(2)}}$ is performed using Algorithm 14 (or its fast version), where the penultimate line is replaced by $e_1 \leftarrow$ Fast2Sum$(.5)(-z_{00}^+, s_1)$ (relative error bounded by δ_2). This works provided that $p \geq 11$.

The computation of $\bar{y}$ is performed using Algorithm 17 (relative error bounded by $\delta_3 = 3u^3+263u^4$).

We obtain

$$\left|\bar{b} - \frac{1}{\sqrt{\bar{x}}}\right| \leq (81u^2+622u^3)\left|\frac{1}{\sqrt{\bar{x}}}\right|, \text{and}$$

$$\left|\bar{b}\bar{x}(1.5 - \frac{1}{2}\bar{b}^2\bar{x}) - \sqrt{\bar{x}}\right| \leq 9916u^4\sqrt{\bar{x}}$$

Globally, we have

$$\left|\bar{y} - \sqrt{\bar{x}}\right| \leq \begin{pmatrix} \delta_1(1.5+287u^2) \\ +\delta_2(0.5+123u^2) \\ +\delta_3(1+162u^2) \\ +9916u^4 \end{pmatrix}\sqrt{\bar{x}}$$

so that,

Theorem 17. *If $\bar{x}$ is a TW, then the relative error committed by $3SqRt^{acc}(\bar{x})$ (resp. $3SqRt^{fast}(\bar{x})$) is bounded by $24u^3+10260u^4$ (resp. $39u^3+10333u^4$).*

The total cost is 127 operations and 4 tests (resp. 111 operations and 2 tests).

11 Comparison with general expansions ($n = 3$)

The algorithms presented in this paper can be compared to the "general" floating-point expansion algorithms (in the special case $n = 3$), and to the algorithms related presented in [20].

11.1 Sum

The algorithm used is the same, but using P-nonoverlapping instead of ulp-nonoverlapping roughly divides the error in the worst case by 4.

11.2 Product

Many more terms are sought in the general algorithms, so that the relative error is bounded by only around $8u^3$. Still, if we have in mind the implementation of correctly rounded elementary functions, our bounds will, in general, be far enough.

The cost of the general algorithms is around twice larger than ours: 88 operations and 2 tests for the "quick-and-dirty" algorithm, assuming that "Renormalize" can be used.

11.3 Reciprocal, division and square root

Our algorithms cannot be fairly compared with Newton-Raphson based algorithms, because they are designed for a number of terms being a power of 2.

Concerning the "paper-and-pencil" method for division, using it with our algorithms (for instance the accurate versions) for sum and product costs around 150 operations and gives a relative error around 2^9u^3. It is

clearly possible to improve these figures, but probably not enough to compete with the algorithm we proposed.

It is anyway significantly better than what Priest proposes in [22, page 116], where he provides 7265 as a bound for the number of operations, and $\approx 2^{13}u^3$ for the relative error.

12 Conclusion

We have shown that usual floating-point expansion algorithms adapted for building and adding triple word numbers are correct in the context of triple words, and we have introduced algorithms for rounding, multiplying, reciprocating, dividing and computing square roots of triple word numbers, along with correctness proofs and error bounds.

Some bounds have been shown to be tight, but this is not the case for $2Prod^{acc}_{2,3}$, and for the reciprocal, division and square root algorithms, so the error bounds on these algorithms may be improved.

The natural continuation of this paper would be to suggest additional algorithms that would make TW arithmetic efficiently compatible with general expansions: for instance, adding a TW with a quad-word number to get a TW output cannot be done with the usual algorithm, because of the counter-example we presented.

References

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

[2] Sylvie Boldo, Mioara Joldeş, Jean-Michel Muller, and Valentina Popescu, *Formal verification of a floating-point expansion renormalization algorithm*, 8th International Conference on Interactive Theorem Proving (ITP) (Brasilia, Brazil), 2017.

[3] Florent de Dinechin, Alexey V. Ershov, and Nicolas Gast, *Towards the post-ultimate libm*, 17th IEEE Symposium on Computer Arithmetic (ARITH-17), 2005, pp. 288–295.

[4] T. J. Dekker, *A floating-point technique for extending the available precision*, Numerische Mathematik **18** (1971), no. 3, 224–242.

[5] 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** (2007), no. 2, 15 pages. Available at http://www.mpfr.org/.

[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] M. Joldeş, J.-M. Muller, and V. Popescu, *On the computation of the reciprocal of floating point expansions using an adapted newton-raphson iteration*, 25th IEEE International Conference on Application-Specific Systems, Architectures and Processors (ASAP'2014), June 2014, pp. 63–67.

[8] Mioara Joldeş, Jean-Michel Muller, and Valentina Popescu, *Tight and rigourous error bounds for basic building blocks of double-word arithmetic*, ACM Transactions on Mathematical Software **44** (2017), no. 2.

[9] Mioara Joldeş, Jean-Michel Muller, Valentina Popescu, and Warwick Tucker, *CAMPARY: Cuda multiple precision arithmetic library and applications*, 5th International Congress on Mathematical Software (ICMS), July 2016.

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

[11] D. E. Knuth, *The art of computer programming*, 3rd ed., vol. 2, Addison-Wesley, Reading, MA, 1998.

[12] C. Q. Lauter, *Basic building blocks for a triple-double intermediate format*, Tech. Report 2005-38, LIP, École Normale Supérieure de Lyon, September 2005.

[13] C. Q. Lauter, *Arrondi correct de fonctions mathématiques*, Ph.D. thesis, École Normale Supérieure de Lyon, Lyon, France, October 2008, In French, available at http://www.ens-lyon.fr/LIP/Pub/Rapports/PhD/PhD2008/PhD2008-07.pdf.

[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*, Tech. Report 45991, Lawrence Berkeley National Laboratory, 2000, https://publications.lbl.gov/islandora/object/ir\%3A115848.

[15] ———, *Design, implementation and testing of extended and mixed precision BLAS*, ACM Transactions on Mathematical Software **28** (2002), no. 2, 152–205.

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

[17] Jean-Michel Muller, Nicolas Brunie, Florent de Dinechin, Claude-Pierre Jeannerod, Mioara Joldes, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, and Serge Torres, *Handbook of floating-point arithmetic, 2nd edition*, Birkhäuser Boston, 2018, ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-3-319-76525-9.

[18] Y. Nievergelt, *Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit*, ACM Transactions on Mathematical Software **29** (2003), no. 1, 27–48.

[19] T. Ogita, S. M. Rump, and S. Oishi, *Accurate sum and dot product*, SIAM Journal on Scientific Computing **26** (2005), no. 6, 1955–1988.

[20] V. Popescu, *Towards fast and certified multiple-precision libraries*, Ph.D. thesis, Université de Lyon, 2017, Available at https://hal.archives-ouvertes.fr/tel-01534090.

[21] D. M. Priest, *Algorithms for arbitrary precision floating point arithmetic*, 10th IEEE Symposium on Computer Arithmetic (ARITH-10), June 1991, pp. 132–143.

[22] D. M. Priest, *On properties of floating-point arithmetics: Numerical stability and the cost of accurate computations*, Ph.D. thesis, University of California at Berkeley, 1992.

[23] J. R. Shewchuk, *Adaptive precision floating-point arithmetic and fast robust geometric predicates*, Discrete Computational Geometry **18** (1997), 305–363.

Nicolas Fabiano was born in Paris, France, in 1998. He is currently studying Computer Science at the ENS Paris. He is involved in different math associations, especially the scientific part of the Tournoi Français des Jeunes Mathématiciennnes et Mathématiciens (TFJM^2) and its international version.

Jean-Michel Muller was born in Grenoble, France, in 1961. He received his Ph.D. degree in 1985 from the Institut National Polytechnique de Grenoble. He is Directeur de Recherches (senior researcher) at CNRS, France, and he is the co-head of GDR-IM. His research interests are in Computer Arithmetic. Dr. Muller was co-program chair of the 13th IEEE Symposium on Computer Arithmetic (Asilomar, USA, June 1997), general chair of SCAN'97 (Lyon, France, sept. 1997), general chair of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia, April 1999), general chair of the 22nd IEEE Symposium on Computer Arithmetic (Lyon, France, June 2015). He is the author of several books, including "Elementary Functions, Algorithms and Implementation" (3rd edition, Birkhauser, 2016), and he coordinated the writing of the "Handbook of Floating-Point Arithmetic" (2nd edition Birkhäuser, 2018). He is an associate editor of the IEEE Transactions on Computers, and a fellow of the IEEE.