Accelerating Correctly Rounded Floating-Point Division When the Divisor is Known in Advance

Nicolas Brisebarre, Jean-Michel Muller and Saurabh Kumar Raina

Laboratoire LIP, ENSL/CNRS/INRIA Arenaire Project

Ecole Normale Supérieure de Lyon

46 Allée d'Italie, 69364 Lyon Cedex 07, FRANCE

Nicolas.Brisebarre@ens-lyon.fr, Jean-Michel.Muller@ens-lyon.fr, Saurabh-Kumar.Raina@ens-lyon.fr

June 2, 2003

Index terms: Computer arithmetic, Floating-point arithmetic, Division, Compilation optimization.

Abstract

We present techniques for accelerating the floating-point computation of x/y when y is known before x. The proposed algorithms are oriented towards architectures with available fused-MAC operations. The goal is to get exactly the same result as with usual division with rounding to nearest. These techniques can be used by compilers to accelerate some numerical programs without loss of accuracy.

Motivation of this research

We wish to provide methods for accelerating floating-point divisions of the form x/y, when y is known before x, either at compile-time, or at run time. We assume that a fused multiply-accumulator is available, and that division is done in software (this happens for instance on RS6000, PowerPC or Itanium architectures). The computed result must be the correctly rounded result.

A naive approach consists in computing the reciprocal of y (with rounding to nearest), and then, once x is available, multiplying the obtained result by x. It is well known that that "naive method" does not always produce a correctly rounded result. One might then conclude that, since the result should always be correct, there is no interest in investigating that method. And yet, if the probability of getting an incorrect rounding was small enough, one could imagine the following strategy:

- the computations that follow the naive division are performed as if the division was correct;
- in parallel, using holes in the pipeline, a remainder is computed, to check whether the division was correctly rounded;
- if it turns out that the division was not correctly rounded, the result of the division is corrected using the computed remainder, and the computation is started again at that point.

To investigate whether that strategy is worth being applied, it is of theoretical and practical interest to have at least a rough estimation of the probability of getting an incorrect

rounding. Also, one could imagine that there might exist some values of y for which the naive method always work (for any x). These values could be stored. Last but not least, some properties of the naive method are used to design better algorithms. For these reasons, we have decided to dedicate a section to the analysis of the naive method.

Another approach starts as previously: once x is known, it is multiplied by the pre-computed reciprocal of y. Then a remainder is computed, and used to correct the final result. This does not require testing. That approach looks like the final steps of a Newton-Raphson division. It is clear from the literature that the iterative algorithms for division require an initial approximation of the reciprocal of the divisor, and that the number of iterations is reduced by having a more accurate initial approximation. Of course this initial approximation can be computed in advance if the divisor is known. The problem is to always get correctly rounded results, at very low cost.

1 Introduction

We deal with floating-point (FP for short) divisions of the form x/y for which y is known before x, either at compile-time (i.e., y is a constant. In such a case, much pre-computation can be performed), or at run-time. We want to get the result more quickly than by just performing a division, yet with the same accuracy: we wish to get a correctly rounded value, as required by the IEEE 754 Standard for FP arithmetic [1, 6]. In this paper, we focus on rounding to nearest only. Divisions by constants are a clear application of our work. There are other applications, for instance when many divisions by the same y are performed. Consider for instance Gaussian elimination:

```
 for j=1 to n-1 do if a[j,j] = 0 then stop
    else
      for i = j+1 to n do
         c[i,j] = a[i,j] / a[j,j]
            for k = j+1 to n do a[i,k] = a[i,k] - c[i,j]*a[j,k]
            end for
         b[i] = b[i] - l[i,j]*b[j]
      end for
end for
```

Most programmers replace the divisions `a[i,j] / a[j,j]` by multiplications by `p = 1 / a[j,j]` (computed in the `for j...` loop). The major drawback is a loss of accuracy. Our goal is to get the same result as if actual divisions were performed, without the delay penalty they would involve. Presentation of conventional division methods can be found in [4, 10, 14]. To make this paper easier to read, we have put the proofs in appendix.

2 Definitions and notations

Define $\mathbb{M}_n$ as the set of exponent-unbounded, n-bit mantissa, binary FP numbers (with $n \geq 1$), that is: $\mathbb{M}_n = \left\{M \times 2^E, 2^{n-1} \leq M \leq 2^n - 1, M, E \in \mathbb{Z}\right\} \cup \{0\}$. It is an "ideal" system, with no overflows or underflows. We will show results in $\mathbb{M}_n$. These results will remain true in actual systems that implement the IEEE 754 standard, provided that no overflows or underflows do occur. The **mantissa** of a nonzero element $M \times 2^E$

of $\mathbb{M}_n$ is the number $m(x) = M/2^{n-1}$.

The result of an arithmetic operation whose input values belong to $\mathbb{M}_n$ may not belong to $\mathbb{M}_n$ (in general it does not). Hence that result must be *rounded*. The standard defines 4 different rounding modes:

- rounding towards $+\infty$, or upwards: $\circ_u(x)$ is the smallest element of $\mathbb{M}_n$ that is greater than or equal to x;
- rounding towards $-\infty$, or downwards: $\circ_d(x)$ is the largest element of $\mathbb{M}_n$ that is less than or equal to x;
- rounding towards 0: $\circ_z(x)$ is equal to $\circ_u(x)$ if $x < 0$, and to $\circ_d(x)$ otherwise;
- rounding to the nearest even: $\circ_\nu(x)$ is the element of $\mathbb{M}_n$ that is closest to x. If x is exactly halfway between two elements of $\mathbb{M}_n$, $\circ_\nu(x)$ is the one for which M is an even number.

The IEEE 754 standard requires that the user should be able to choose one rounding mode among these ones, called the **active rounding mode**. After that, when performing one of the 4 arithmetic operations, or when computing square roots, the obtained result should be equal to the rounding of the exact result. For $a \in \mathbb{M}_n$, we define a^+ as its **successor** in $\mathbb{M}_n$, that is, $a^+ = \min\{b \in \mathbb{M}_n, b > a\}$, and $ulp(a)$ as $|a|^+ - |a|$. If a is not an element of $\mathbb{M}_n$, we define ulp(a) as $\circ_u(a) - \circ_d(a)$. The name ulp is an acronym for **unit in the last place**. When $x \in \mathbb{M}_n$, $ulp(x)$ is the "weight" of the last mantissa bit of x. We also define a^- as the **predecessor of** a.

We call a **breakpoint** a value z where the rounding changes, that is, if t_1 and t_2 are

real numbers satisfying $t_1 < z < t_2$ and $\circ_t$ is the rounding mode, then $\circ_t(t_1) < \circ_t(t_2)$. For "directed" rounding modes (i.e., towards $+\infty$, $-\infty$ or 0), the breakpoints are the FP numbers. For rounding to the nearest mode, they are the exact middle of two consecutive FP numbers.

3 Preliminary results and previous work

3.1 Preliminary results

We will frequently use the following well-known properties:

Property 1

- *Let $y \in \mathbb{M}_n$. There exists q such that $1/y$ belongs to $\mathbb{M}_q$ if and only if y is a power of 2.*
- *If $m > n$, the exact quotient of two n-bit numbers cannot be an m-bit number.*
- *Let $x, y \in \mathbb{M}_n$. $x \neq y \Rightarrow |x/y - 1| \geq 2^{-n}$.*

The next result gives a lower bound on the distance between a breakpoint (in round-to-nearest mode) and the quotient of two FP numbers.

Property 2 *If $x, y \in \mathbb{M}_n$, $1 \leq x, y < 2$, then the distance between x/y and the middle of two consecutive FP numbers is lower-bounded by $\frac{1}{y \times 2^{2n-1}} > \frac{1}{2^{2n}}$ if $x \geq y$; and $\frac{1}{y \times 2^{2n}} > \frac{1}{2^{2n+1}}$ otherwise. Moreover, if the last mantissa bit of y is a zero, then the lower bounds become twice these ones.*

3.2 The naive method

As said in the introduction, we have to evaluate x/y, and y is known before x. An obvious solution consists in pre-computing $z = 1/y$ (or more precisely z rounded-to-nearest, that is, $z_h = \circ_\nu(1/y)$), and then to multiply x by z_h. We will refer to this as "the naive method". Although this method does not necessarily give the correctly-rounded expected result, we study its properties, because they can be used to derive better algorithms. We assume round-to-nearest mode.

3.2.1 Maximum error of the naive solution

Property 3 *The naive solution returns a result that is at most at distance* 1.5 *ulps from the exact result if* $m(x) < m(y)$ *(reminder:* $m(u)$ *is the mantissa of* u*); and* 1 *ulp from the exact result if* $m(x) \geq m(y)$*.*

If $x < y$ and $1 \leq x, y < 2$, the following property holds. It will allow us to analyze the behavior of another algorithm (Algorithm 1).

Property 4 *If* $x < y$ *and* $1 \leq x, y < 2$*, then the naive solution returns a result* q *such that either* q *is within* 1 *ulp from* x/y*, or* x/y *is at least at a distance* $\frac{2^{-2n+1}}{y} + 2^{-2n+1} - \frac{2^{-3n+2}}{y}$ *from a breakpoint of the round-to-nearest mode.*

Property 3 gives tight bounds: there are values x and y for which the naive solution leads to an error very close to 1.5 ulps. More precisely,

Property 5 *The maximum error of the naive algorithm can be obtained through a reasonably fast algorithm. This maximum error converges to* 1.5 *ulps as* $n \to \infty$*.*

This is illustrated in Table 1. For instance, in the IEEE-754 double precision format ($n = 53$), the division of $x = \frac{268435449}{134217728}$ by $y = \frac{9007199120523265}{4503599627370496}$ by the naive algorithm leads to an error equal to $1.4999999739\cdots$ ulps.

3.2.2 Probability of getting a correctly rounded result using the naive solution

For the first few values of n (up to $n = 13$), we have computed, through exhaustive testing, the proportion of couples (x, y) for which the naive method gives an incorrectly rounded result. These results are given in Table 2. The proportion seems to converge, as n grows, to a constant value that is around 27%. More precisely,

Conjecture 1 *Assuming a uniform distribution of the mantissas of FP numbers, rounding to nearest, and n bits of mantissa, the probability that the naive method return a result different from $\circ_\nu(x/y)$ goes to $13/48 = 0.2708\cdots$ as n goes to $+\infty$.*

This conjecture is an "half-conjecture" only, since we have a rough sketch of a proof, given in the Appendix. The figures given in Table 2 and our conjecture tend to show that for any n, the naive method gives a proportion of incorrectly rounded results around 27%, which is by far too large to be neglected.

3.2.3 Values of y for which the naive method always works

Depending on n, there are a very few values of y (including, of course, the powers of 2) for which the naive method always works (i.e., for all values of x). These values for $n \leq 13$ are given in Table 3. Unfortunately, we are not able to compute them much faster than by exhaustive testing (which does not allow to tackle with the most interesting values of n, namely 24, 53 and 113).

3.3 Division with one multiplication and two fused MACs

On some modern processors (such as the PowerPC, the IBM RISCSystem/6000 [13] and IA64-based architectures [2, 12]), a fused-multiply accumulate instruction (fused-MAC) is available. This makes it possible to evaluate an expression $ax + b$ with one final (correct) rounding only. Let us now investigate how can such an instruction be used to solve our problem. The following result, due to Markstein [13]. was designed in order to get a correctly rounded result from an approximation to a quotient obtained using Newton-Raphson or Goldschmidt iterations.

Theorem 1 (Markstein, 1990 [3, 13]) *Assume $x, y \in \mathbb{M}_n$. If z_h is within $1/2$ ulp of $1/y$ and $q \in \mathbb{M}_n$, q within 1 ulp of x/y then one application of*

$$\begin{cases} r &= \circ_\nu(x - qy) \\ q' &= \circ_\nu(q + rz_h) \end{cases}$$

yields $q' = \circ_\nu(x/y)$.

One would like to use Theorem 1 to get a correctly rounded result from an initial value q obtained by the naive method, that is, by computing $\circ_\nu(xz_h)$, where $z_h = \circ_\nu(1/y)$. Unfortunately, q will not always be within one ulp from x/y (see Property 3), so Theorem 1 cannot be directly applied. One could get a better initial approximation to x/y by performing one step of Newton-Raphson iteration from q. And yet, such an iteration step is not necessary, as shown by the following result (see the work of Markstein [12, 13] for this kind of algorithm).

Theorem 2 (Division with one multiplication and two Macs [12, 13]) *Algorithm 1, given below, always returns the correctly rounded (to nearest) quotient $\circ_\nu(x/y)$.*

Algorithm 1 (Division with one multiplication and two Macs)

- *in advance, evaluate* $z_h = \circ_\nu(1/y)$*;*
- *as soon as* x *is known, compute* $q = \circ_\nu(x \times z_h)$*;*
- *compute* $r = \circ_\nu(x - qy)$*;*
- *compute* $q' = \circ_\nu(q + rz_h)$.

This method requires one division before x is known, and three consecutive (and dependent) MACs once x is known. In the following section, we try to design a faster algorithm. Unfortunately, either there are a few (predictable) values of y for which it does not work, or it requires the availability of an internal precision slightly larger than the target precision.

4 Proposed techniques

4.1 Division with one multiplication and one fused MAC

Using the method presented in Section 3.3, we could compute x/y using one multiplication and two MACs, once x is known. Let us show that in many cases, one multiplication and one MAC (once x is known) do suffice. To do this, we need a double-word approximation to $1/y$. Let us first see how can such an approximation be computed.

4.1.1 Preliminary result: Getting a double-word approximation to $1/y$.

Kahan [8] explains that the fused MAC allows to compute remainders exactly. Let us show how it works.

Property 6 *Let $x, y, q \in \mathbb{M}_n$, such that $q \in \{\circ_d(x/y), \circ_u(x/y)\}$. The remainder $r = x - qy$ is computed exactly with a fused MAC. That is, $\circ_\nu(x - qy) = x - qy$.*

The algorithms we are going to examine require a double-word approximation to $1/y$, that is, 2 FP values z_h and z_ℓ such that $z_h = \circ_\nu(1/y)$ and $z_\ell = \circ_\nu(1/y - z_h)$. The only reasonably fast algorithm we know for getting these values requires a fused MAC. Using Property 6, z_h and z_ℓ can be computed as follows.

Property 7 *Assume $y \in \mathbb{M}_n$, $y \neq 0$. The following sequence of 3 operations computes z_h and z_ℓ such that $z_h = \circ_\nu(1/y)$ and $z_\ell = \circ_\nu(1/y - z_h)$.*

- $z_h = \circ_\nu(1/y)$;
- $\rho = \circ_\nu(1 - yz_h)$;
- $z_\ell = \circ_\nu(\rho/y)$.

4.1.2 The algorithm

We assume that from y, we have computed $z = 1/y$, $z_h = \circ_\nu(z)$ and $z_\ell = \circ_\nu(z - z_h)$ (for instance using Property 7). We suggest the following 2-step method:

Algorithm 2 (Division with one multiplication and one fused MAC) *Compute:*

- $q_1 = \circ_\nu(xz_\ell)$;

- $q_2 = \circ_\nu(xz_h + q_1)$.

This algorithm almost always works. Table 4 shows that for $n \leq 29$, there are more than 98.7% of values of y for which the algorithm returns a correctly rounded quotient for all values of x (these figures have been obtained through exhaustive checking). Moreover, in the other cases (see the proof of Theorem 3), for a given y, there is *at most one value of the mantissa of* x (that can be computed in advance) for which the algorithm may return an incorrectly rounded quotient.

Theorem 3 *Algorithm 2 gives a correct result (that is, $q_2 = \circ_\nu(x/y)$), as soon as at least one of the following conditions is satisfied:*

1. *the last mantissa bit of y is a zero;*
2. *n is less than or equal to 7;*
3. *$|z_\ell| < 2^{-n-2-e}$, where e is the exponent of y (i.e., $2^e \leq |y| < 2^{e+1}$) ;*
4. *for some reason, we know in advance that the mantissa of x will be larger than that of y;*
5. *Algorithm 3, given below, returns **true** when the input value is the integer $Y = y \times 2^{n-1-e_y}$, where e_y is the exponent of y (Y is the mantissa of y, interpreted as an integer).*

Algorithm 3 (Tries to find solutions to Eqn. (9) of the appendix.) *We give the algorithm as a Maple program (to make it more didactic). If it returns "true" then Algorithm 2 always returns a correctly rounded result when dividing by y. It requires the availability of $2n+1$-bit integer arithmetic.*

```
TestY := proc(Y,n)
  local Pminus, Qminus, Xminus, OK, Pplus, Qplus, Xplus;
  Pminus := (1/Y) mod 2^(n+1)
  # requires computation of a modular inverse
  Qminus := (Pminus-1) / 2;
  Xminus := (Pminus * Y - 1) / 2^(n+1);
  if (Qminus >= 2^(n-1)) and (Xminus >= 2^(n-1))
           then OK := false
else
  OK := true
  Pplus := 2^(n+1)-Pminus;
  Qplus := (Pplus-1) / 2;
  Xplus := (Pplus * Y + 1) / 2^(n+1);
  if (Qplus >= 2^(n-1)) and (Xplus >= 2^(n-1))
       then OK := false end if;  end if;
  print(OK)
end proc;
```

Translation of Algorithm 3 into a C or Fortran program is easily done, since computing a modular reciprocal modulo a power of two requires a few operations only, using the extended Euclidean GCD algorithm [9]. Algorithm 3 also computes the only possible mantissa X for which, for the considered value of Y, Algorithm 2 might not work. *Hence, if the algorithm returns* ***false****, it suffices to check this very value of* X (that is, we try to divide this X by Y at compile time) to know if the algorithm will always work, or

if it will work for all X's but this one. (see the proof of Theorem 3).

Let us discuss the consequences of Theorem 3.

- Condition "the last mantissa bit of y is a zero" is very easily checked on most systems. Hence, that condition can be used for accelerating divisions when y is known at run-time, soon enough[1] before x. Assuming that the last bits of the mantissas of the FP numbers appearing in computations are 0 or 1 with probability $1/2$, that condition allows to accelerate half divisions;

- Assuming a uniform distribution of z_ℓ in $(-2^{-n-1-e}, +2^{-n-1-e})$, which is reasonable (see [5]), Condition "$|z_\ell| < 2^{-n-2-e}$" allows to accelerate half remaining cases;

- Our experimental testings up to $n = 24$ show that condition "Algorithm 3 returns **true**" allows to accelerate around 39% of the remaining cases (i.e., the cases for which the last bit of y is a one and $|z_\ell| \geq 2^{-n-2-e}$). If Algorithm 3 returns **false**, then checking the only value of x for which the division algorithm might not work suffices to deal with all remaining cases. And yet, all this requires much more computation: it is probably not interesting if y is not known at compile-time.

4.2 If a larger precision than target precision is available

A larger precision than the target precision is frequently available. A typical example is the double extended precision that is available on Intel microprocessors. We now

[1] The order of magnitude behind this "soon enough" highly depends on the architecture and operating system.

show that if an internal format is available, with at least $n+1$-bit mantissas (which is only one bit more than the target format), then an algorithm very similar to Algorithm 2 always works. In the following, $\circ_{t:+p}(x)$ means x rounded to $n+p$ bits, with rounding mode t. Define $z = 1/y$. We assume that from y, we have computed $z_h = \circ_\nu(z)$ and $z_\ell = \circ_{\nu:+1}(z - z_h)$. They can be computed through:

- $z_h = \circ_\nu(1/y)$;
- $\rho = \circ_\nu(1 - yz_h)$;
- $z_\ell = \circ_{\nu:+1}(\rho/y)$;

We suggest the following 2-step method:

Algorithm 4 (Division with one multiplication and one MAC) *Compute:*

- $q_1 = \circ_{\nu:+1}(xz_\ell)$;
- $q_2 = \circ_\nu(xz_h + q_1)$.

Theorem 4 *Algorithm 4 always returns a correctly rounded quotient.*

Notice that if the first operation returns a result with more than $n+1$ bits, the algorithm still works. We can for instance perform the first operation in double extended precision, if the target precision is double precision.

5 Comparisons

Let us give an example of a division algorithm used on an architecture with an available fused-MAC. In [12], Markstein suggests the following sequence of instructions for

double-precision division on IA-64. The intermediate calculations are performed using the internal double-extended format. The first instruction, `frcpa`, returns a tabulated approximation to the reciprocal of the operand, with at least 8.886 valid bits. When two instructions are put on the same line, they can be performed "in parallel". The returned result is the correctly rounded quotient with rounding mode $\circ_t$.

Algorithm 5 (Double precision division. This is Algorithm 8.10 of [12])

- $z_1 = \texttt{frcpa}(y)$;
- $e = \circ_\nu(1 - yz_1)$;
- $z_2 = \circ_\nu(z_1 + z_1 e)$; $e_1 = \circ_\nu(e \times e)$;
- $z_3 = \circ_\nu(z_2 + z_2 e_1)$; $e_2 = \circ_\nu(e_1 \times e_1)$;
- $z_4 = \circ_\nu(z_3 + z_3 e_2)$;
- $q_1 = \circ_\nu(x z_4)$;
- $r = \circ_\nu(x - y q_1)$;
- $q = \circ_t(q_1 + r z_4)$.

This algorithm requires 8 FP latencies, and uses 10 instructions. The last 3 lines of this algorithm are Algorithm 1 of this paper (with a slightly different context, since Algorithm 5 uses extended precision). Another algorithm also given by Markstein (Algorithm 8.11 of [12]) requires 7 FP latencies only, but uses 11 instructions. The algorithm suggested by Markstein for extended precision is Algorithm 8.18 of [12]. It requires 8 FP latencies and uses 14 FP instructions.

These figures show that replacing conventional division x/y by specific algorithms whenever y is a constant or division by the same y is performed many times in a loop is worth being done. For double-precision calculations, this replaces 7 FP latencies by 3 (using Algorithm 1) or 2 (using Algorithm 2 if y satisfies the conditions of Theorem 3, or Algorithm 4 if a larger internal precision – e.g., double-extended precision – is available). Whenever an even slightly larger precision is available (one more bit suffices), Algorithm 4 is of interest, since it requires 2 FP latencies instead of 3. Algorithm 2 is certainly interesting when the last bit of y is a zero, and, possibly, when $|z_\ell| < 2^{-n-2-e}$. In the other cases, the rather large amount of computation required by checking whether that algorithm can be used (we must run Algorithm 3 at compile-time) limits its use to divisions by constants in applications for which compile time can be large and running time must be as small as possible.

Conclusion

We have presented several ways of accelerating a division x/y, where y is known before x. Our methods could be used in optimizing compilers, to make some numerical programs run faster, without any loss of accuracy. Algorithm 1 always works and does not require much pre-computation (so it can be used even if y is known a few tens of cycles only before x). Algorithm 2 is faster, and yet it requires much pre-computation (for computing z_h and z_ℓ, and making sure that the algorithm works) so it is more suited for division by a constant. Algorithm 4 always works and requires two operations only once x is known, but it requires the availability of a slightly larger precision. The various

programs we have used for this study can be obtained through an email to the authors.

References

[1] American National Standards Institute and Institute of Electrical and Electronic Engineers. IEEE standard for binary floating-point arithmetic. *ANSI/IEEE Standard, Std 754-1985,* New York, 1985.

[2] M. Cornea-Hasegan and B. Norin. IA-64 floating-point operations and the IEEE standard for binary floating-point arithmetic. *Intel Technology Journal*, Q4, 1999.

[3] M. A. Cornea-Hasegan, R. A. Golliver, and P. W. Markstein. Correctness proofs outline for Newton-Raphson based floating-point divide and square root algorithms. *Proc. 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia)*, pages 96–105, April 1999. IEEE Computer Society Press.

[4] M. D. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, Boston, 1994.

[5] A. Feldstein and R. Goodman. *Convergence estimates for the distribution of trailing digits*. *Journal of the ACM*, 23: 287–297, 1976.

[6] D. Goldberg. What every computer scientist should know about floating-point arithmetic. *ACM Computing Surveys*, 23(1):5–47, March 1991.

[7] C. Iordache and D. W. Matula. On infinitely precise rounding for division, square root, reciprocal and square root reciprocal. *Proc. 14th IEEE Symposium on Com-*

puter Arithmetic (Adelaide, Australia), pages 233–240, April 1999. IEEE Computer Society Press.

[8] W. Kahan. Lecture notes on the status of IEEE-754. File accessible at http://http.cs.berkeley.edu/∼wkahan/ieee754status/ieee754.ps, 1996.

[9] D. Knuth. *The Art of Computer Programming*, volume 2. Addison Wesley, Reading, MA, 1973.

[10] I. Koren. *Computer arithmetic algorithms*. Prentice-Hall, Englewood Cliffs, NJ, 1993.

[11] T. Lang and J.-M. Muller. Bound on run of zeros and ones for algebraic functions. *Proc. 15th IEEE Symposium on Computer Arithmetic (Vail, Colorado)*. IEEE Computer Society Press, 2001.

[12] P. W. Markstein. *IA-64 and Elementary Functions : Speed and Precision*. Hewlett-Packard Professional Books. Prentice Hall, 2000. ISBN: 0130183482.

[13] P. W. Markstein. Computation of elementary functions on the IBM Risc System/6000 processor. *IBM Journal of Research and Development*, 34(1):111–119, Jan. 1990.

[14] S. F. Oberman and M. J. Flynn. Division algorithms and implementations. *IEEE Transactions on Computers*, 46(8):833–854, Aug. 1997.

[15] C. Batut, K. Belabas, D. Bernardi, H. Cohen and M. Olivier, *User's Guide to PARI-GP*, available from `ftp://megrez.math.u-bordeaux.fr/pub/pari`.

Appendix: proof of the properties and theorems

Properties 1 and 2: The proofs are straightforward and omitted for reasons of space.

Proof of Property 3. Let $x, y \in \mathbb{M}_n$. Without loss of generality, we can assume that x and y belong to $[1, 2)$. Since the cases $x, y = 1$ or 2 are straightforward, we assume that x and y belong to $(1, 2)$. Thus $1/y \notin \mathbb{M}_n$. Since $z_h = \circ_\nu(z)$ and $z \in (1/2, 1)$, we have, $\left|\frac{1}{y} - z_h\right| < 2^{-n-1}$. Therefore,

$$\left|\frac{x}{y} - xz_h\right| < 2^{-n}. \tag{1}$$

From Property 1 and (1), we cannot have $x/y > 1$ and $xz_h < 1$ or the converse. So xz and xz_h belong to the same "binade" (i.e., $\text{ulp}(xz_h) = \text{ulp}(xz)$). Now, there are two possible cases:

- if $x \geq y$, then $|xz_h - \circ_\nu(xz_h)| \leq 2^{-n}$, so $|x/y - \circ_\nu(xz_h)| < 2^{-n+1} = \text{ulp}(x/y)$.
- if $x < y$, then $|xz_h - \circ_\nu(xz_h)| \leq 2^{-n-1}$, so $|x/y - \circ_\nu(xz_h)| < 3 \times 2^{-n-1} = 1.5 \times \text{ulp}(x/y)$.

□

Proof of Property 4. The proof is similar to that of Property 3. We use the tighter bounds:

- $|1/y - z_h| < 2^{-n-1} - 2^{-2n}/y$ (this comes from Property 2: $1/y$ is at a distance at least $2^{-2n}/y$ from a breakpoint);
- $x \leq 2 - 2^{-n+2}$ (this comes from $x < y < 2$, which implies $x \leq (2^-)^-$).

Combining these bounds gives

$$\left|\frac{x}{y} - xz_h\right| \leq 2^{-n} - \frac{2^{-2n+1}}{y} - 2^{-2n+1} + \frac{2^{-3n+2}}{y}.$$

The final bound ℓ_{min} is obtained by adding the $1/2$ ulp bound on $|xz_h - \circ_\nu(xz_h)|$:

$$\left|\frac{x}{y} - \circ_\nu(xz_h)\right| \leq \ell_{min} = 3 \times 2^{-n-1} - \frac{2^{-2n+1}}{y} - 2^{-2n+1} + \frac{2^{-3n+2}}{y}.$$

If $\circ_\nu(xz_h)$ is not within 1 ulp from x/y, it means that x/y is at a distance at least $1/2$ ulp from the breakpoints that are immediately above or below $q = \circ_\nu(xz_h)$. And since the breakpoints that are immediately above $\circ_\nu(xz_h)^+$ or below $\circ_\nu(xz_h)^-$ are at a distance 1.5 ulps $= 3 \times 2^{-n-1}$ from $\circ_\nu(xz_h)$, x/y is at least at a distance $3 \times 2^{-n-1} - \ell_{min}$ from these breakpoints. □

Proof of Property 5. We look for the couples $(x, y) \in \mathbb{M}_n$ such that $1 \leq x < y < 2$ and $|x/y - \circ_\nu(x \circ_\nu (1/y))|$ is as close as possible to $\frac{1.5}{2^n}$. To hasten the search, we will look for couples such that $\left|\frac{x}{y} - \circ_\nu\left(x \circ_\nu \left(\frac{1}{y}\right)\right)\right| \geq \frac{2K+1}{2^{n+1}}$, where K is a real parameter as close as possible to 1. If we write

$$\frac{x}{y} - \circ_\nu\left(x \circ_\nu \left(\frac{1}{y}\right)\right) = \frac{x}{y} - x \circ_\nu \left(\frac{1}{y}\right) + x \circ_\nu \left(\frac{1}{y}\right) - \circ_\nu\left(x \circ_\nu \left(\frac{1}{y}\right)\right),$$

we see that, as $\left|x \circ_\nu \left(\frac{1}{y}\right) - \circ_\nu\left(x \circ_\nu \left(\frac{1}{y}\right)\right)\right| \leq \frac{1}{2}\frac{1}{2^n}$, we want

$$x\left|\frac{1}{y} - \circ_\nu\left(\frac{1}{y}\right)\right| \geq \frac{K}{2^n} \tag{2}$$

Hence $x > 2K$ since $\left|\frac{1}{y} - \circ_\nu\left(\frac{1}{y}\right)\right| < \frac{1}{2^{n+1}}$ ($1/y \notin \mathbb{M}_n$). Let us write $y = \frac{2^n - s}{2^{n-1}}$ and $x = \frac{2^n - l}{2^{n-1}}$ with $1 \leq s \leq \lfloor 2^n(1-K)\rfloor - 1$ and $s + 1 \leq l \leq \lfloor 2^n(1-K)\rfloor$. We have,

$$\frac{2^n}{y} = 2^{n-1} + \frac{s}{2} + \frac{1}{2}\frac{s^2}{2^n - s}.$$

As $y > x$, (2) implies

$$\left|\frac{1}{y} - \circ_\nu\left(\frac{1}{y}\right)\right| > \frac{K}{2^n y}. \tag{3}$$

The full proof considers two cases: s is odd and s is even. For reasons of space we only deal with the case "s odd" here. The other case is very similar (the full proof can be obtained through an email to one of the authors).

When s is odd, we only keep the $s \in [1, \lfloor 2^n(1-K)\rfloor - 1]$ such that

$$\frac{1}{2}\frac{s^2}{2^n - s} \in \left(0, \frac{1}{2} - K\frac{2^{n-1}}{2^n - s}\right) \cup \bigcup_{k\in\mathbb{N}\setminus\{0\}} \left(k + K\frac{2^{n-1}}{2^n - s} - \frac{1}{2}, k + \frac{1}{2} - K\frac{2^{n-1}}{2^n - s}\right)$$

i.e., $s \in [1, \lfloor 2^n(1-K)\rfloor - 1] \cap \bigcup_{k\in\mathbb{N}}(a_{\text{odd},k}, b_{\text{odd},k})$ with

$$a_{\text{odd},0} = 0 \text{ and } a_{\text{odd},k} = \frac{-2k + 1 + \sqrt{(2k-1)^2 + 2^{n+2}(2k - 1 + K)}}{2} \text{ for all } k \geq 1,$$

$$b_{\text{odd},k} = \frac{-2k - 1 + \sqrt{(2k+1)^2 + 2^{n+2}(2k + 1 - K)}}{2} \text{ for all } k.$$

Let $k_{\text{odd}} = \max\left\{k \in \mathbb{N}, a_{\text{odd},k} < \lfloor 2^n(1-K)\rfloor - 1\right\}$. We have

$$k_{\text{odd}} = \left\lfloor \frac{1}{2}\frac{2^{n+2}(1-K) + 4(\lfloor 2^n(1-K)\rfloor - 1)(\lfloor 2^n(1-K)\rfloor - 2)}{2^{n+2} - 4\lfloor 2^n(1-K)\rfloor + 4} \right\rfloor.$$

Finally, when s is odd, we only keep the

$$s \in \bigcup_{0\leq k\leq k_{\text{odd}}-1} (a_{\text{odd},k}, b_{\text{odd},k}) \cup \left(a_{\text{odd},k_{\text{odd}}}, \min\left(b_{\text{odd},k_{\text{odd}}}, \lfloor 2^n(1-K)\rfloor\right)\right).$$

Let $k \in \mathbb{N}$, $0 \leq k \leq k_{\text{odd}}$ such that $s \in (a_{\text{odd},k}, b_{\text{odd},k})$. We have $2^n \circ_\nu \left(\frac{1}{y}\right) = 2^{n-1} + \frac{s \pm 1}{2} + k$, with $\pm = +$ if $s > -k + \sqrt{k^2 + 2^{n+1}k}$ and $\pm = -$ otherwise. Thus,

$$2^n x \circ_\nu \left(\frac{1}{y}\right) = 2^n - l + s \pm 1 + 2k - \frac{l(s \pm 1 + 2k)}{2^n}. \tag{4}$$

Now, recall that we want

$$\left|x \circ_\nu \left(\frac{1}{y}\right) - \circ_\nu \left(x \circ_\nu \left(\frac{1}{y}\right)\right)\right| \geq \frac{2K+1}{2^{n+1}} - \left|\frac{x}{y} - x \circ_\nu \left(\frac{1}{y}\right)\right|. \tag{5}$$

This can be written as

$$\left| x \circ_\nu \left(\frac{1}{y}\right) - \circ_\nu \left(x \circ_\nu \left(\frac{1}{y}\right)\right)\right| \geq \frac{2K+1}{2^{n+1}} - \frac{2^n - l}{2^{2n-1}} \left| \frac{1}{2} \frac{s^2}{2^n - s} - \frac{(2k \pm 1)}{2} \right| = \varepsilon_{s,l,k,K}.$$

We get from this condition and (4), that

$$\frac{l(s \pm 1 + 2k)}{2^n} \in \bigcup_{m \in \mathbb{N}} (m + 2^n \varepsilon_{s,l,k,K}, m + 1 - 2^n \varepsilon_{s,l,k,K},)$$

i.e., $l \in [s+1, \lfloor 2^n(1-K) \rfloor] \cap \bigcup_{m \in \mathbb{N}} (c_{\text{odd},m}, d_{\text{odd},m})$ where

$$c_{\text{odd},m} = \frac{2^n(m + K + 1/2) - 2^n |s^2/(2^n - s) - (2k \pm 1)|}{s \pm 1 + 2k - |s^2/(2^n - s) - (2k \pm 1)|}$$

and $d_{\text{odd},m} = \dfrac{2^n(m - K + 1/2) + 2^n |s^2/(2^n - s) - (2k \pm 1)|}{s \pm 1 + 2k + |s^2/(2^n - s) - (2k \pm 1)|}$.

Let $m_{\text{odd}} = \min \{m \in \mathbb{N}, s < d_{\text{odd},m}\}$ and $M_{\text{odd}} = \max \{m \in \mathbb{N}, c_{\text{odd},m} < \lfloor 2^n(1-K) \rfloor\}$.

We easily get an exact expression of these integers. Hence, we look for the

$$l \in (\max(c_{\text{odd},m_{\text{odd}}}, s), d_{\text{odd},m_{\text{odd}}}) \cup \bigcup_{m_{\text{odd}}+1 \leq m \leq M_{\text{odd}}-1} (c_{\text{odd},m}, d_{\text{odd},m})$$
$$\cup (c_{\text{odd},M_{\text{odd}}}, \min(d_{\text{odd},M_{\text{odd}}}, \lfloor 2^n(1-K) \rfloor)).$$

Once we have got all these couples (s, l), we end up our research by checking if $\left| \dfrac{x}{y} - \circ_\nu \left(x \circ_\nu \left(\dfrac{1}{y}\right)\right)\right| \geq \frac{2K+1}{2^{n+1}}$ with $x = (2^n - l)/2^{n-1}$ and $y = (2^n - s)/2^{n-1}$.

These remarks lead to an algorithm implemented in GP, the calculator of PARI [15], that gets faster as the parameter K grows. If K is too large, we won't find any couple. But, we know values of K that are close to 1 and associated to a couple (x, y). These values allow us to get the couples $(x, y) \in \mathbb{M}_n$ such that $1 \leq x < y < 2$ and $\left| \dfrac{x}{y} - \circ_\nu \left(x \circ_\nu \left(\dfrac{1}{y}\right)\right)\right|$ is as close as possible to $\dfrac{1.5}{2^n}$. More precisely, we now give a sequence $(x_n, y_n)_{n \in \mathbb{N} \setminus \{0\}}$ such that, for all $n \in \mathbb{N} \setminus \{0\}$, $x_n,\ y_n \in \mathbb{M}_n$, $1 \leq x_n < y_n < 2$

and $2^n \left| \dfrac{x_n}{y_n} - \circ_\nu \left(x_n \circ_\nu \left(\dfrac{1}{y_n} \right) \right) \right| \longrightarrow \dfrac{3}{2}$ as $n \longrightarrow +\infty$. For n even, we choose

$$x_n = \frac{2^n - 2^{n/2} - 2^{n/2-1} + 3}{2^{n-1}}, \quad y_n = \frac{2^{n/2} - 1}{2^{n/2-1}}.$$

For n odd, we choose

$$x_n = \frac{2^{(n+3)/2} - 7}{2^{(n+1)/2}}, \quad y_n = \frac{2^n - 2^{(n+1)/2} + 1}{2^{n-1}}.$$

Let $n = 2p$, $p \in \mathbb{N} \setminus \{0\}$. We have $\dfrac{x_{2p}}{y_{2p}} = \frac{2^{p-1}}{2^p-1} \dfrac{2^{2p} - 2^p - 2^{p-1} + 3}{2^{2p-1}}$. After some calculation, we get, for all $p \geq 2$,

$$2^{2p} \left| \frac{x_{2p}}{y_{2p}} - \circ_\nu \left(x_{2p} \circ_\nu \left(\frac{1}{y_{2p}} \right) \right) \right| = \left| \frac{3}{2} - \frac{5}{2} \frac{2^{-p}}{1 - 2^{-p}} \right| \longrightarrow \frac{3}{2} \text{ as } p \longrightarrow +\infty.$$

Let $n = 2p + 1$, $p \in \mathbb{N}$. We have $\dfrac{x_{2p+1}}{y_{2p+1}} = \frac{2^{p+2}-7}{2^{p+1}} \frac{2^{2p}}{2^{2p+1} - 2^{p+1} + 1}$. After some calculation, we get, for all $p \geq 2$,

$$\begin{aligned} &2^{2p+1} \left| \frac{x_{2p+1}}{y_{2p+1}} - \circ_\nu \left(x_{2p+1} \circ_\nu \left(\frac{1}{y_{2p+1}} \right) \right) \right| \\ &= \left| \frac{3}{2} - 7 \cdot 2^{-p-2} - (2^{-2p-1} - 7 \cdot 2^{-3p-3}) \frac{1}{1 - 2^{-p} + 2^{-2p-1}} \right| \longrightarrow \frac{3}{2} \text{ as } p \longrightarrow +\infty. \end{aligned}$$

Then we use our algorithm with the parameter K obtained from this sequence. We get the values given in Table 1. Note that the couples (x, y) in the table are the couples (x_n, y_n) except for $n = 64$. □

Sketch of a proof for Conjecture 1.

Define $z = 1/y = z_h + z_\rho$, where $z_h = \circ_\nu(z)$, with $1 < y < 2$. When $n \to \infty$, the maximum value of $|z_\rho|$ is asymptotically equal to $1/2\text{ulp}(z)$, and its average value is asymptotically equal to $1/4\text{ulp}(z) = 2^{-n-2}$. Hence, for $1 < x < 2$, we can

write: $xz = xz_h + \epsilon$ where the average value of $|\epsilon|$ is $\dfrac{y+1}{2} \times 2^{-n-2} = (y+1)2^{-n-3}$ for $x < y$ and $\dfrac{2+y}{2} \times 2^{-n-2} = (2+y)2^{-n-3}$ for $x > y$ (to get these figures, we multiply the average value of ϵ by the average value of x, which is $\frac{y+1}{2}$ for $1 < x < y$ and $\frac{2+y}{2}$ for $y < x < 2$). The "breakpoints" of the rounding mode[2], are regularly spaced, at distance 2^{-n} for $x < y$, and 2^{-n+1} for $x > y$. Therefore, the probability that $\circ_\nu(xz) \neq \circ_\nu(xz_h)$ should asymptotically be the probability that there should be a breakpoint between these values. That probability is $(y+1)2^{-n-3}/2^{-n} = \dfrac{y+1}{8}$ for $x < y$, and $(2+y)2^{-n-3}/2^{-n+1} = \dfrac{y+2}{16}$ for $x > y$.

Therefore, for a given y, the probability that the naive method should give a result different from $\circ_\nu(x/y)$ is $\frac{(y+1)(y-1)}{8} + \frac{(y+2)(2-y)}{16} = \frac{y^2}{16} + \frac{1}{8}$. Therefore, assuming now that y is variable, the probability that the naive method give an incorrectly rounded result is

$$\int_1^2 \left(\frac{y^2}{16} + \frac{1}{8}\right) dy = \frac{13}{48} \approx 0.27.$$

□

Proof of Theorem 2. We assume $1 \leq x, y < 2$. First, let us notice that if $x \geq y$, then (from Property 3), q is within one ulp from x/y, therefore Theorem 1 applies, hence $q' = \circ_\nu(x/y)$. Let us now focus on the case $x < y$. Define $\epsilon_1 = x/y - q$ and $\epsilon_2 = 1/y - z_h$. From Property 3 and the definition of rounding to nearest, we have, $|\epsilon_1| < 3 \times 2^{-n-1}$ and $|\epsilon_2| < 2^{-n-1}$. The number $\rho = x - qy = \epsilon_1 y$ is less than 3×2^{-n} and is a multiple of 2^{-2n+1}. It therefore can be represented exactly with $n+1$ bits of mantissa.

[2]Since we assume rounding to nearest mode, the breakpoints are the exact middles of two consecutive machine numbers.

Hence, the difference between that number and $r = \circ_\nu(x - qy)$ (i.e., ρ rounded to n bits of mantissa) is zero or $\pm 2^{-2n+1}$. Therefore, $r = \epsilon_1 y + \epsilon_3,$ with $\epsilon_3 \in \{0, \pm 2^{-2n+1}\}$.

Let us now compute $q + rz_h$. We have $q + rz_h = \dfrac{x}{y} + \dfrac{\epsilon_3}{y} - \epsilon_1\epsilon_2 y - \epsilon_2\epsilon_3$. Hence,

$$\left|\frac{x}{y} - (q + rz_h)\right| \leq \frac{2^{-2n+1}}{y} + 3 \times 2^{-2n-2} y + 2^{-3n}$$

Define $\epsilon = 2^{-2n+1}/y + 3 \times 2^{-2n-2} y + 2^{-3n}$. Now, from Property 4, either q was at a distance less than one ulp from x/y (but in such a case, $q' = \circ_\nu(x/y)$ from Theorem 1), or x/y is at least at a distance

$$\delta = \frac{2^{-2n+1}}{y} + 2^{-2n+1} - \frac{2^{-3n+2}}{y}.$$

from a breakpoint. A straightforward calculation shows that, if $n \geq 4$, then $\epsilon < \delta$. Therefore there is no breakpoint between x/y and $q + rz_h$. Hence $\circ_\nu(q + rz_h) = \circ_\nu(x/y)$. The cases $n < 4$ are easily checked through exhaustive testing. □

Proof of Property 6. Without loss of generality, we assume $1 \leq x, y < 2$. Define $K = n + 1$ if $q < 1$ and $K = n$ otherwise. Since r is a multiple of 2^{-n-K+2} that is less than $2^{-K+1} y$, we have $r \in \mathbb{M}_n$. Hence, it is computed exactly. □

Proof of Property 7. From Property 6, ρ is computed exactly. Therefore it is *exactly* equal to $1 - yz_h$. Hence, ρ/y is equal to $1/y - z_h$. Hence, z_ℓ is equal to $\circ_\nu(1/y - z_h)$.□

Proof of Theorem 3. The cases $n \leq 7$ have been processed through exhaustive searching. Let us deal with the other cases. Without loss of generality, we assume $x \in (1, 2)$

and $y \in (1,2)$ (the cases $x = 1$ or $y = 1$ are straightforward). This gives $z \in (1/2, 1)$, and, from Property 1, the binary representation of z is infinite. Hence, $z_h \in [1/2, 1]$. The case $z_h = 1$ is impossible ($y > 1$ and $y \in \mathbb{M}_n$ imply $y \geq 1 + 2^{-n+1}$, thus $1/y \leq 1 - 2^{-n+1} + 2^{-2n+2} < 1 - 2^{-n} \in \mathbb{M}_n$, thus $\circ_\nu(1/y) \leq 1 - 2^{-n}$). Hence, the binary representation of z_h has the form $0.z_h^1 z_h^2 z_h^3 \cdots z_h^n$. Since z_h is obtained by rounding z to the nearest, we have: $|z - z_h| \leq \frac{1}{2}\text{ulp}(z) = 2^{-n-1}$. Moreover, Property 1 shows that the case $|z - z_h| = 2^{-n-1}$ is impossible. Therefore $|z - z_h| < 2^{-n-1}$. From this, we deduce: $|z_\ell| = |\circ_\nu (z - z_h)| \leq 2^{-n-1}$. Again, the case $|z_\ell| = 2^{-n-1}$ is impossible: if we had $|z - z_h| < 2^{-n-1} = |z_\ell|$, this would imply $|z - (z_h + 2^{-n-1})| < 2^{-2n-1}$ or $|z - (z_h - 2^{-n-1})| < 2^{-2n-1}$ which would contradict the fact that the binary representation of the reciprocal of an n-bit number cannot contain more than $n - 1$ consecutive zeros or ones [7, 11]. Therefore $|z_\ell| < 2^{-n-1}$. Thus, from the definition of z_ℓ, $|(z - z_h) - z_\ell| < 2^{-2n-2}$, thus, $|x(z - z_h) - xz_\ell| < 2^{-2n-1}$, hence, $|x(z - z_h) - \circ_\nu(xz_\ell)| < 2^{-2n-1} + \frac{1}{2}\text{ulp}(xz_\ell) \leq 2^{-2n}$. Therefore,

$$|xz - \circ_\nu[xz_h + \circ_\nu(xz_\ell)]| < 2^{-2n} + \frac{1}{2}\text{ulp}\left(xz_h + \circ_\nu(xz_\ell)\right) \tag{6}$$

Hence, if for a given y there does not exist any x such that $x/y = xz$ is at a distance less than 2^{-2n} from the middle of two consecutive FP numbers, then $\circ_\nu[xz_h + \circ_\nu(xz_\ell)]$ will always be equal to $\circ_\nu(xz)$, i.e., Algorithm 2 will give a correct result. Therefore, from Property 2, if $x \geq y$ then Algorithm 2 will return a correctly rounded quotient. Also, if $|z_\ell| < 2^{-n-2}$ (which corresponds to Condition "$|z_\ell| < 2^{-n-e-2}$" of the theorem we are proving) then we get a sharper bound:

$$|xz - \circ_\nu[xz_h + \circ_\nu(xz_\ell)]| < 2^{-2n-1} + \frac{1}{2}\text{ulp}\left(xz_h + \circ_\nu(xz_\ell)\right) \tag{7}$$

and Property 2 implies that we get a correctly rounded quotient.

Let us now focus on the case $x < y$. Let $q \in \mathbb{M}_n$, $1/2 \leq q < 1$, and define integers X, Y and Q as

$$\begin{cases} X &= x \times 2^{n-1}, \\ Y &= y \times 2^{n-1}, \\ Q &= q \times 2^{n}. \end{cases}$$

If we have $\frac{x}{y} = q + 2^{-n-1} + \epsilon,$ with $|\epsilon| < 2^{-2n}$, then

$$2^{n+1}X = 2QY + Y + 2^{n+1}\epsilon Y, \text{ with } |\epsilon| < 2^{-2n}. \tag{8}$$

But:

- Equation (8) implies that $R' = 2^{n+1}\epsilon Y$ should be an integer.
- The bounds $Y < 2^n$ and $|\epsilon| < 2^{-2n}$ imply $|R'| < 2$.
- Property 1 implies $R' \neq 0$.

Hence, the only possibility is $R' = \pm 1$. Therefore, to find values y for which for any x Algorithm 2 gives a correct result, we have to examine the possible integer solutions to

$$\begin{cases} 2^{n+1}X = (2Q+1)Y \pm 1, \\ 2^{n-1} \leq X \leq 2^n - 1, \\ 2^{n-1} \leq Y \leq 2^n - 1, \\ 2^{n-1} \leq Q \leq 2^n - 1. \end{cases} \tag{9}$$

There are no solutions to (9) for which Y is even. This shows that if the last mantissa bit of y is a zero, then Algorithm 2 always returns a correctly rounded result. Now, if Y is odd then it has a reciprocal modulo 2^{n+1}. Define $P^- = (1/Y) \bmod 2^{n+1}$ and $P^+ = (-1/Y) \bmod 2^{n+1}$, $Q^- = (P^- - 1)/2$ and $Q^+ = (P^+ - 1)/2$. From $0 <$

$P^-, P^+ \leq 2^{n+1} - 1$ and $P^- + P^+ = 0 \bmod 2^{n+1}$, we easily find $P^- + P^+ = 2^{n+1}$. From this, we deduce,

$$\begin{aligned} 0 \leq Q^-, Q^+ \leq 2^n - 1, \\ Q^- + Q^+ = 2^n - 1. \end{aligned} \tag{10}$$

Define $X^- = \frac{P^- Y - 1}{2^{n+1}}$ and $X^+ = \frac{P^+ Y + 1}{2^{n+1}}$. From (10) we easily deduce that either $Q^- \geq 2^{n-1}$ or $Q^+ \geq 2^{n-1}$, but both are impossible. Hence, either (Y, X^+, Q^+) or (Y, X^-, Q^-) can be solution to Eq. (9), but both are impossible. Algorithm 3 checks these two possible solutions. This explains the last condition of the theorem. □

Proof of Theorem 4.

As previously, we can assume $x \in (1/2, 1)$. The proof of Theorem 3 is immediately adapted if $x \geq y$, so that we focus on the case $x < y$. Using exactly the same computations as in the proof of Theorem 3, we can show that

$$|xz - \circ_\nu \left(xz_h + \circ_{\nu:+1}(xz_\ell)\right)| < 2^{-2n-1} + \frac{1}{2}\mathrm{ulp}\left(xz_h + \circ_{\nu:+1}(xz_\ell)\right).$$

and Property 2 implies that we get a correctly rounded quotient.

Table 1: Some maximal errors (in ulps) of the naive solution for various values of n.

n	x	y	Error $>$
32	$\frac{4294868995}{2147483648}$	$\frac{65535}{32768}$	1.4999618524452582589456
53	$\frac{268435449}{134217728}$	$\frac{9007199120523265}{4503599627370496}$	1.4999999739229677997443
64	$\frac{18446744066117050369}{9223372036854775808}$	$\frac{18446744067635550617}{9223372036854775808}$	1.4999999994316597271551
113	$\frac{288230376151711737}{144115188075855872}$	$\frac{10384593717069655112945804582584321}{5192296858534827628530496329220096}$	1.4999999999999999757138

Table 2: Actual probability of an incorrect result for small values of n.

n	probability
7	$0.2485\cdots$
8	$0.2559\cdots$
9	$0.2662\cdots$
10	$0.2711\cdots$
11	$0.2741\cdots$

Table 3: The n-bit numbers y between 1 and 2 for which, for any n-bit number x, $\circ_\nu(x \times \circ_\nu(1/y))$ equals $\circ_\nu(x/y)$.

n				
7	1	$\frac{105}{64}$		
8	1	$\frac{151}{128}$	$\frac{163}{128}$	$\frac{183}{128}$
9	1	$\frac{307}{256}$		
10	1			
11	1	$\frac{1705}{1024}$		
12	1			
13	1	$\frac{4411}{4096}$	$\frac{4551}{4096}$	$\frac{4915}{4096}$

Table 4: Number $\gamma(n)$ and percentage $100\gamma(n)/2^{n-1}$ of values of y for which Algorithm 2 returns a correctly rounded quotient for all values of x. For $n \leq 7$, the algorithm always works.

n	$\gamma(n)$	percentage
7	64	100
8	127	99.218
9	254	99.218
10	510	99.609
11	1011	98.730
12	2022	98.730
13	4045	98.754
14	8097	98.840
15	16175	98.724
16	32360	98.754
17	64686	98.703
18	129419	98.738
19	258953	98.782
20	517591	98.722
21	1035255	98.729
22	2070463	98.727
23	4140543	98.718
24	8281846	98.727
25	16563692	98.727
26	33126395	98.724
27	66254485	98.726
28	132509483	98.727
29	265016794	98.726