Avoiding double roundings in scaled Newton-Raphson division

Jean-Michel Muller

Avoiding double roundings in scaled Newton-Raphson division

Jean-Michel Muller
CNRS, Laboratoire LIP
(CNRS, ENS Lyon, INRIA, Univ. Claude Bernard Lyon 1)
Lyon, France
first-name.last-name@ens-lyon.fr

***Abstract*—When performing divisions using Newton-Raphson (or similar) iterations on a processor with a floating-point fused multiply-add instruction, one must sometimes scale the iterations, to avoid over/underflow and/or loss of accuracy. This may lead to double-roundings, resulting in output values that may not be correctly rounded when the quotient falls in the subnormal range. We show how to avoid this problem.**

I. INTRODUCTION

The availability of a fused multiply-add instruction makes it possible to design fast algorithms for correctly-rounded division, that use a variant of the Newton-Raphson iteration. These algorithms will not return an accurate enough result when the exact quotient is below the underflow threshold. The iterations may also overflow if the input operands are too large or too small. A natural solution to overcome this problem is to *scale* the iterations by multiplying one of the input values (or both) by an adequately chosen power of 2. However, doing this may lead to a subtle *double rounding* problem, which sometimes prevents from obtaining a correctly-rounded quotient. After defining some notation, we recall some classical results on the final "correcting step" of Newton-Raphson-based division, then we give an example that illustrates the double rounding problem, and we show how that problem can be solved.

A. Notation

Floating-Point numbers: Throughout the paper, we assume that we use a radix-2, precision-p, floating-point system that is compliant with the IEEE 754-2008 Standard for Floating-Point Arithmetic [6]. We denote $e_{\min}$ and $e_{\max}$ the extremal exponents of that system. In such a system, a floating-point (FP) number is a number x such that

$$x = X \cdot 2^{e-p+1}, \qquad (1)$$

where X and e are integers that satisfy

$$\begin{array}{l} |X| \leq 2^p - 1, \quad \text{and} \\ e_{\min} \leq e \leq e_{\max}. \end{array} \qquad (2)$$

For a given nonzero FP number x, there may be several pairs (X, e) that satisfy (1) with the constraints (2). The one for which $|X|$ is maximum is called the "normalized representation" of x. The corresponding X is the *integral significand* of x, and the corresponding e is the *exponent* of x.

Subnormal numbers: A floating-point number is said *normal* if its magnitude is larger than $2^{e_{\min}}$, and it is said *subnormal* otherwise. The integral significand of a normal number has absolute value larger than or equal to 2^{p-1}.

Midpoints: We will call *midpoint* a number that is exactly halfway between two consecutive floating-point numbers.

Roundings, faithful approximations: In general, the sum, product, quotient, etc., of two FP numbers is not exactly equal to a FP number. It must therefore be *rounded*. The IEEE 754-2008 Standard defines several rounding functions (round towards $-\infty$, towards $+\infty$, towards 0, round to nearest ties "to even", round to nearest ties "to away"), and stipulates that once a rounding function $\circ$ is chosen, each time we perform the arithmetic operation $a \top b$ ($\top \in \{+, -, \times, \div\}$), where a and b are FP numbers, the value $\circ(a \top b)$ is returned. We say that operation $\top$ is *correctly rounded*. In the following, we assume that the rounding function, denoted RN is one of the two round-to-nearest functions defined by the standard:

- *round to nearest ties to even:* if t is not a midpoint, $\text{RN}(t)$ is the FP number nearest t, and if t is a midpoint, $\text{RN}(t)$ is the one of the two FP numbers that surround t whose integral significand is even;
- *round to nearest ties to away:* if t is not a midpoint, $\text{RN}(t)$ is the FP number nearest t, and if t is a midpoint, $\text{RN}(t)$ is the one of the two FP numbers that surround t that has the largest magnitude.

Notice that the problem we are dealing with in this paper, namely double rounding, does not occur with the round towards $\pm\infty$ or round towards 0 rounding functions.

We will say that a FP number X is a *faithful approximation* to a real number x if:

- x is a FP number and $X = x$, or
- X is one of the two FP numbers that surround x.

Inexact results, underflows: We will say that the operation $a \top b$ is *inexact* if $a \top b$ is not a FP number (which is equivalent to saying that the computed result $\text{RN}(a \top b)$ is not equal to $a \top b$). We will say that an arithmetic operation *underflows* if *i*) the returned result is a subnormal number, and *ii*) it is inexact.

Fused Multiply-Add (FMA) instruction: In the following, we assume that a fused multiply-add (FMA) instruction is available. The FMA instruction evaluates expressions of the form

$$\text{RN}(a \pm bc).$$

It is available on processors such at the Intel Itanium, IBM PowerPC, AMD Bulldozer, and Intel Haswell. It allows for faster and, in general, more accurate dot products, matrix multiplications, and polynomial evaluations. It also makes it possible to obtain correctly rounded quotients through a variant of the Newton-Raphson iteration. The FMA instruction is required by the IEEE 754-2008 standard for FP arithmetic, so that within a few years, it will probably be available on most general-purpose platforms.

B. The final correcting step of Newton-Raphson-based division iterations

Many algorithms have been suggested for performing divisions, the most common being digit-recurrence algorithms [4] and variants of the Newton–Raphson iteration [8].

The usual Newton-Raphson iteration for computing $1/a$ is:

$$y_{n+1} = y_n(2 - ay_n). \tag{3}$$

Assuming an FMA instruction is available, that iteration can be implemented as follows:

$$\left\{ \begin{array}{lcl} \varepsilon_n & = & \mathrm{RN}(1 - ay_n) \\ y_{n+1} & = & \mathrm{RN}(y_n + y_n\varepsilon_n) \end{array} \right. . \tag{4}$$

In this paper, we assume that we wish to evaluate the quotient b/a of two floating-point numbers, we focus on algorithms that first provide an approximation y to $1/a$—which can be done using iteration (4)—and an initial approximation q to the quotient b/a, and refine it using the following "correcting step" [2], [3], [8], [9]:

$$\begin{array}{lcl} r & = & \mathrm{RN}\,(b - aq), \\ q' & = & \mathrm{RN}\,(q + ry), \end{array} \tag{5}$$

Under some conditions made explicit in Theorem 2 below, $q' = \mathrm{RN}(b/a)$.

In all applications of that property presented so far in the literature, the approximations y and q are obtained through variants of the Newton-Raphson iteration (indeed, (5) can be viewed as one Newton-Raphson step), but they might as well result from other means.

What makes the method working is the following lemma, which shows that under some conditions, $r = b - aq$ *exactly*. That lemma can be traced back to Kahan [7] or Markstein [9]. The presentation we give here is close to that of Boldo and Daumas [1], [10].

Lemma 1 (Computation of division residuals using an FMA)**.** *Assume a and b are precision-p, radix-2, floating-point numbers, with $a \neq 0$ and $|b/a|$ below the overflow threshold. If q a faithful approximation to b/a then $b - aq$ is exactly computed using one FMA instruction, with any rounding function, provided that*

$$\begin{array}{c} e_a + e_q \geq e_{\min} + p - 1, \\ \textit{and} \\ q \neq \alpha \textit{ or } |b/a| \geq \frac{\alpha}{2}, \end{array} \tag{6}$$

where e_a and e_q are the exponents of a and q and $\alpha = 2^{e_{\min}-p+1}$ is the smallest positive subnormal number.

For this result to be applicable, we need $e_a + e_q \geq e_{\min} + p - 1$. This condition will be satisfied if $e_b \geq e_{\min} + p$. Other conditions will be needed for the correcting iterations (5) to work. They can *very* roughly be summarized as "the quotient and the residual r must be far enough from the underflow and overflow thresholds". More precisely,

Theorem 2 (Markstein [2], [5], [8], [9])**.** *Assume a precision-p binary floating-point arithmetic, and let a and b be normal numbers. If*

- *q is a faithful approximation to b/a, and*
- *q is not in the subnormal range, and*
- *$e_b \geq e_{\min} + p$, and*
- *y approximates $1/a$ with a relative error less than 2^{-p}, and*
- *the calculations*

  $$r = \circ(b - aq), \qquad q' = \circ(q + ry)$$

  are performed using a given rounding function $\circ$, taken among round to nearest even, round toward zero, round toward $-\infty$, round toward $+\infty$,

then $q' = \circ(b/a)$, that is, q' is b/a rounded according to the same rounding function $\circ$.

C. Scaled division iterations

Given arbitrary FP inputs a and b, a natural way to make sure that the conditions of Theorem 2 be satisfied is to *scale* the iterations. This can be done as follows: a quick preliminary checking on the exponents of a and b determines if the conditions of Theorem 2 may not be satisfied, or if there is some risk of over/underflow in the iterations that compute y and q. If this is the case, operand a, or operand b is multiplied by some adequately chosen power of 2, to get new, *scaled*, operands a^* and b^* such that the division b^*/a^* is performed without any problem. An alternate, possibly simpler, solution is to *always* scale: for instance, we chose a^* and b^* equal to the significands of a and b, i.e., we momentarily set their exponents to zero. In any case, we assume that we now perform a division b^*/a^* such that:

- for that "scaled division", the conditions of Theorem 2 are satisfied;
- the *exact* quotient b/a is equal to $2^\sigma b^*/a^*$, where σ is an integer straightforwardly deduced from the scaling.

Assuming now that the scaled iterations return a scaled approximate quotient q^* and a scaled approximate reciprocal y^*, we perform a *scaled correcting step*

$$\begin{array}{lcl} r & = & \mathrm{RN}\,(b^* - a^*q^*), \\ q' & = & \mathrm{RN}\,(q^* + ry^*), \end{array} \tag{7}$$

Notice that q' is in the normal range (i.e., its absolute value is larger than or equal to $2^{e_{\min}}$): the scaling was partly done in order to make this sure. If $2^\sigma q'$ is a floating-point number (e.g., if $2^{e_{\min}} \leq |2^\sigma q'| \leq 2^{e_{\max}+1} - 2^{e_{\max}-p+1}$), then we clearly should return $2^\sigma q'$. If $|2^\sigma q'| > 2^{e_{\max}+1} - 2^{e_{\max}-p+1}$ then we should return $\pm\infty$. The trouble may occur when $2^\sigma q'$ falls in the subnormal range: in that case, if $2^\sigma q'$ is not a FP number, we cannot just return $\mathrm{RN}(2^\sigma q')$ because a *double rounding slip* (see below) might occur and lead to the delivery of a wrong result. Consider the following example. Assume the floating-point format being considered is *binary32* (that format was called *single precision* in the previous version of IEEE 754: precision $p = 24$, extremal exponents $e_{\min} = -126$ and $e_{\max} = 127$), and that RN is round-to-nearest-ties-to-*even*[1]. Consider the two floating-point input values (the significands are represented in binary):

$$\begin{cases} b &= 1.00000000001100011001101_2 \times 2^{-113} \\ &= 8394957_{10} \times 2^{-136}, \\ a &= 1.00000000000011011001100_2 \times 2^{23} \\ &= 8390348_{10}. \end{cases}$$

The number b/a is equal to

$$0.1000000000010010000000000000010110100111 \\ 10110011001000 \cdots \times 2^{-135},$$

so that the correctly-rounded, subnormal value that must be returned when computing b/a should be

$$\mathrm{RN}(b/a) = 0.00000000010000000000101 \times 2^{-126}.$$

Now, if, to be able to use Theorem 2, b was scaled, for instance by multiplying it by 2^{128} to get a value b^*, the exact value of b^*/a would be

$$0.1000000000010010000000000000010110100111 \\ 10110011001000 \cdots \times 2^{-7},$$

which would imply that the computed correctly rounded approximation to b^*/a would be

$$q' = 1.00000000001001000000000 \times 2^{-8}.$$

Multiplied by $2^\sigma = 2^{-128}$, this result would be equal equal to

$$1.00000000001001000000000 \times 2^{-136},$$

which means—since it is in the subnormal range: remember that $e_{\min} = -126$—that, after rounding it to the nearest (even) floating-point number, we would get

$$0.00000000010000000000100 \times 2^{-126} \neq \mathrm{RN}(b/a).$$

This phenomenon—let us call it a *double rounding slip*—will appear each time the scaled result q', once multiplied by 2^σ, is exactly equal to a (subnormal) midpoint, and:

- $b/a > 2^\sigma q'$ and $\mathrm{RN}(2^\sigma q') < 2^\sigma q'$;
- or $b/a < 2^\sigma q'$ and $\mathrm{RN}(2^\sigma q') > 2^\sigma q'$.

Notice that if we are given q' as the output of a "black box" algorithm, i.e., if we just have this scaled result q' without any other information, it is impossible to deduce if the exact, infinitely precise, result is above or below the midpoint, so it is hopeless to try to return a correctly rounded value.

[1]One easily builds a similar example with round-to-nearest-ties-to-*away*. The method presented below works for both round-to-nearest rounding modes

Fortunately, intermediate values computed during the last correction iteration contain enough information to allow for a correctly rounded final result, as we are now going to see.

II. Avoiding double roundings in scaled division iterations

As stated in the previous section, we assume we have performed the correcting step:

$$\begin{aligned} r &= \mathrm{RN}(b^* - a^* q^*), \\ q' &= \mathrm{RN}(q^* + r y^*), \end{aligned}$$

and that the scaled operands a^*, b^*, as well as the approximate scaled quotient q^* and scaled reciprocal y^* satisfy the conditions of Theorem 2. We assume that the scaling was such that the exact quotient b/a is equal to $2^\sigma b^*/a^*$. As said in the introduction, we assume that we are interested in quotients rounded to the nearest (with ties-to-even or ties-to-away): with the other, "directed", rounding functions, there is no double rounding problem. To simplify the presentation, we assume that a and b (and, therefore, a^*, b^*, y^*, q^* and q') are positive (separately handling the signs of the input operands is straightforward). Since q^* is a faithful approximation to b^*/a^*, we deduce that

$$q^- < \frac{b^*}{a^*} < q^+,$$

where q^- and q^+ are the floating-point predecessor and successor of q^*. Also, since $q' = \mathrm{RN}(b^*/a^*)$, we immediately deduce that $q' \in \{q^-, q, q^+\}$. This is illustrated by Figure 1.

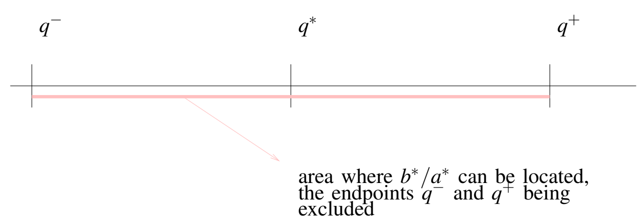

Fig. 1. The number q^* is a faithful rounding of b^*/a^*: this means that $q^- < b^*/a^* < q^+$, where q^- and q^+ are the FP predecessor and successor of q^*.

As stated before, a double rounding slip may occur when $2^\sigma q'$ is a subnormal midpoint of the considered floating-point format. In such a case, in order to return a correctly rounded quotient, one must know if the exact quotient b/a is strictly below, equal to, or strictly above that midpoint. Of, course, this is equivalent to knowing if b^*/a^* is strictly below, equal to, or strictly above q'.

Lemma 1 says that $r = b^* - a^* q^*$ exactly. Therefore, *when $2^\sigma q'$ is a midpoint:*

1) if $r = 0$ then $q' = q^* = b^*/a^*$, hence $b/a = 2^\sigma q'$ exactly. Therefore, one should return $\mathrm{RN}(2^\sigma q')$;
2) if $q' \neq q^*$ and $r > 0$ (which implies $q' = q^+$), then q' overestimates b^*/a^*. Therefore, one should return $2^\sigma q'$ rounded down. This is illustrated by Figure 2;
3) if $q' \neq q^*$ and $r < 0$ (which implies $q' = q^-$), then q' underestimates b^*/a^*. Therefore, one should return $2^\sigma q'$

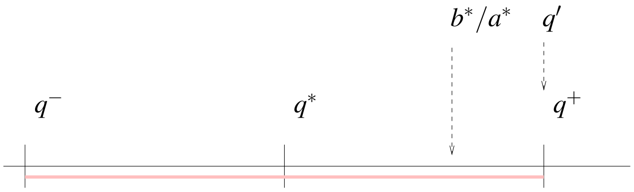

Fig. 2. q' is equal to q^+. In this case, the "residual" r was positive, and since $q^- < b^*/a^* < q^+$, q' is an overestimation of b^*/a^*.

rounded up (this case is symmetrical to the previous one);

4) if $q' = q^*$ and $r > 0$, then q' underestimates b^*/a^*. Therefore, one should return $2^\sigma q'$ rounded up. This is illustrated by Figure 3;

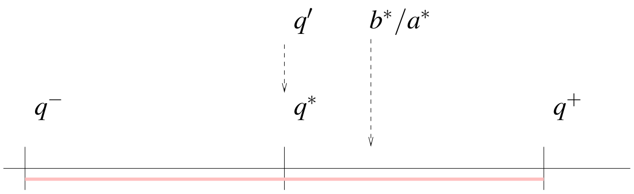

Fig. 3. q' is equal to q^*. In this case, the "residual" r was positive, and q' is an underestimation of b^*/a^*.

5) if $q' = q^*$ and $r < 0$, then q' overestimates b^*/a^*. Therefore, one should return $2^\sigma q'$ rounded down (this case is symmetrical to the previous one).

Of course, when $2^\sigma q'$ is not a midpoint, one should of course return $\mathrm{RN}(2^\sigma q')$.

Therefore, in all cases, we are able to find which value is to be returned.

III. Conclusion

We have proposed a simple and easily implementable way of getting a correctly-rounded result when performing scaled Newton-Raphson divisions.

References

[1] S. Boldo and M. Daumas. Representable correcting terms for possibly underflowing floating point operations. In J.-C. Bajard and M. Schulte, editors, *Proceedings of the 16th Symposium on Computer Arithmetic*, pages 79–86. IEEE Computer Society Press, Los Alamitos, CA, 2003.

[2] M. Cornea, R. A. Golliver, and P. Markstein. Correctness proofs outline for Newton–Raphson-based floating-point divide and square root algorithms. In Koren and Kornerup, editors, *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia)*, pages 96–105. IEEE Computer Society Press, Los Alamitos, CA, April 1999.

[3] M. Cornea, J. Harrison, and P. T. P. Tang. *Scientific Computing on Itanium®-based Systems*. Intel Press, Hillsboro, OR, 2002.

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

[5] J. Harrison. Formal verification of IA-64 division algorithms. In M. Aagaard and J. Harrison, editors, *Proceedings of the 13th International Conference on Theorem Proving in Higher Order Logics, TPHOLs 2000*, volume 1869 of *Lecture Notes in Computer Science*, pages 234–251. Springer-Verlag, 2000.

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

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

[8] P. Markstein. *IA-64 and Elementary Functions: Speed and Precision*. Hewlett-Packard Professional Books. Prentice-Hall, Englewood Cliffs, NJ, 2000.

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

[10] Jean-Michel Muller, Nicolas Brisebarre, Florent de Dinechin, Claude-Pierre Jeannerod, Vincent Lefèvre, Guillaume Melquiond, Nathalie Revol, Damien Stehlé, and Serge Torres. *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2010. ACM G.1.0; G.1.2; G.4; B.2.0; B.2.4; F.2.1., ISBN 978-0-8176-4704-9.