Avoiding double roundings in scaled Newton-Raphson division
Jean-Michel Muller

To cite this version:

HAL Id: ensl-00875366
https://hal-ens-lyon.archives-ouvertes.fr/ensl-00875366
Submitted on 21 Oct 2013

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers. L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
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 Arithmetics. We denote $\epsilon_{\text{min}}$ and $\epsilon_{\text{max}}$ the extremal
exponents of that system. In such a system, a floating-point
(FP) number is a number $x$ such that

$$x = X \cdot 2^{\epsilon-p+1},$$

where $X$ and $\epsilon$ are integers that satisfy

$$|X| \leq 2^p - 1, \quad \text{and} \quad \epsilon_{\text{min}} \leq \epsilon \leq \epsilon_{\text{max}}.$$ (1)

For a given nonzero FP number $x$, there may be several pairs
$(X, \epsilon)$ 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 $\epsilon$ is the exponent of $x$.

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

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 $a \circ b \ (T \in \{+,-,\times,\div\})$, where $a$ and
$b$ are FP numbers, the value $\circ(a \circ b)$ is returned. We say that
operation $T$ is correctly rounded. In the following, we assume
that the rounding function, denoted $\text{RN}$ 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
$a \circ b$ is inexact if $a \circ b$ is not a FP number (which is equivalent
to saying that the computed result $\text{RN}(a \circ b)$ is not equal to
$a \circ b$). 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 as 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 and variants of the Newton–Raphson iteration. The usual Newton-Raphson iteration for computing \( \frac{1}{a} \) is:

\[
y_{n+1} = y_n(2 - ay_n).
\]

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

\[
\begin{align*}
\epsilon_n &= \text{RN}(1 - ay_n) \\
y_{n+1} &= \text{RN}(y_n + y_n\epsilon_n)
\end{align*}
\]

In this paper, we assume that we wish to evaluate the quotient \( \frac{b}{a} \) of two floating-point numbers, we focus on algorithms that first provide an approximation \( y \) to \( 1/a \)—which can be done using iteration—and an initial approximation \( q \) to the quotient \( b/a \), and refine it using the following “correcting step”:

\[
r = \circ(b - aq),
q' = \circ(q + ry)
\]

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 may 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 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 are satisfied;
- the exact quotient \( b/a \) is equal to \( 2^\sigma b^*/a^* \), where \( \sigma \) 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:

\[
r = \text{RN}(b^* - a^*q^*),
q' = \text{RN}(q^* + ry^*),
\]

Notice that \( q' \) is in the normal range (i.e., its absolute value is larger than or equal to \( 2^\sigma \)). The scaling was partly done in order to make this sure. If \( 2^\sigma q' \) is a floating-point number (e.g., if \( 2^\sigma \leq |2^\sigma q'| \leq 2^{\max+1} - 2^{\max-p+1} \)), then we clearly
would imply that the computed correctly rounded result may overestimate or underestimate the true result. This phenomenon—let us call it a double rounding slip (see below)—might occur and lead to the delivery of a wrong result. Consider the following example. Assume that 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 \( e_{\text{min}} = -126 \) and \( e_{\text{max}} = 127 \), and that RN is round-to-nearest-ties-to-even.)

Consider the two floating-point input values (the significands are represented in binary):

\[
\begin{align*}
  b &= 1.00000000001100101100110110110111_2 \times 2^{-113} \\
  a &= 1.00000000000110101100110011011000_2 \times 2^{23}
\end{align*}
\]

The number \( b/a \) is equal to

\[
0.10000000000101000000000000000110110111101110110010000 \cdots \times 2^{-135}
\]

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

\[
\text{RN}(b/a) = 0.00000000010000000000000000101 \times 2^{-126}.
\]

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

\[
0.10000000000101000000000000000110110111101110110010000 \cdots \times 2^{-7},
\]

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

\[
q' = 1.0000000000100000000000000010 \times 2^{-8}.
\]

Multiplied by \( 2^2 = 2^{128} \), this result would be equal equal to

\[
1.0000000000100000000000000010 \times 2^{-136},
\]

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

\[
0.000000000010000000000000010 \times 2^{-126} \neq \text{RN}(b/a).
\]

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

- \( b/a > 2^2 q' \) and RN\((2^2 q') < 2^2 q'\);
- \( b/a < 2^2 q' \) and RN\((2^2 q') > 2^2 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.

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

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

\[
\begin{align*}
  r &= \text{RN}(b^* - a^* q^*), \\
  q' &= \text{RN}(q^* + ry^*),
\end{align*}
\]

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 \( 2^2 b^*/a^* \). 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, \( a^* \), \( b^* \), \( y^* \) 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 \( q^* = \text{RN}(b^*/a^*) \), we immediately deduce that \( q' \in \{q^- , q , q^+ \} \).

As stated before, a double rounding slip may occur when \( 2^2 q' \) 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 \( 2^2 q' \) is a midpoint:

1) if \( r = 0 \) then \( q' = b^*/a^* \), hence \( b/a = 2^2 q' \) exactly. Therefore, one should return RN\((2^2 q')\); 2) if \( q' \neq q^* \) and \( r > 0 \) (which implies \( q' = q^+ \)), then \( q' \) overestimates \( b^*/a^* \). Therefore, one should return \( 2^2 q' \) 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 \( 2^2 q' \);

1One easily builds a similar example with round-to-nearest-ties-to-away.
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 \( b^*/a^* \).

\[ q^- \quad q^* \quad q^+ \]

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.

\[ q^+ \quad b^*/a^* \]

\[ q^- \quad q^* \quad q^+ \]

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 \( RN(2\sigma q') \).

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


