Scaling Newton-Raphson division iterations to avoid double rounding

Jean-Michel Muller

Scaling Newton-Raphson division iterations to avoid double rounding

Jean-Michel Muller
CNRS, ENS Lyon, INRIA, UCBL, Université de Lyon
Laboratoire LIP, Ecole Normale Supérieure de Lyon,
46 allée d'Italie, 69364 Lyon Cédex 07
France
Jean-Michel.Muller@ens-lyon.fr

This is LIP research report number RRLIP2010-21
June 2010

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 is in the subnormal range. We show how to avoid this problem.

1 introduction

Throughout the paper, we assume a radix-2, precision-p, floating-point system that is compliant with the IEEE 754-2008 Standard for Floating-Point Arithmetic [4]. We also assume that a fused multiply-add (FMA) instruction is available. That instruction evaluates expressions of the form $xy + z$ with one final rounding only. We also assume that the ambient rounding mode is *round to nearest* (this is the only one for which the problem we are dealing with, namely *double rounding*, occurs).

Many algorithms have been suggested for performing divisions. Here, assuming we wish to evaluate the quotient b/a of two floating-point numbers, we focus on algorithms that first provide an initial approximation q to the quotient and an approximation y to $1/a$, and refine it using a "correcting step" [6].

$$\begin{array}{rcl} r & = & \mathrm{RN}\,(b - aq), \\ q' & = & \mathrm{RN}\,(q + ry), \end{array} \tag{1}$$

where $\mathrm{RN}(u)$ is u rounded to nearest (even). Under some conditions made explicit in Theorem 2—roughly speaking, if q and y are close enough to b/a and $1/a$, respectively, and no underflow occurs—, then $q' = \mathrm{RN}(b/a)$.

In most applications of that property presented in the literature, the approximations y and q are obtained through variants of the Newton-Raphson iteration, but they might as well result from other means. What matters in this paper is that the correcting step (1) is used.

That correcting step method is applicable only under some conditions. More precisely, what makes the method working is the following lemma, which shows that, still under some conditions, $r = b - aq$ *exactly*. That lemma can be traced back at least to Markstein's work [6]. The presentation we give here is close to that of Boldo and Daumas [1, 7].

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 is defined as*

- *b/a if it is exactly representable;*
- *one of the two floating-point numbers that surround b/a otherwise;*

then

$$b - aq$$

is exactly computed using one FMA instruction, with any rounding mode, provided that

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

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 to work (see Theorem 2 below). Also, the intermediate iterations used for computing q and y may require the absence of over/underflow. All this gives somewhat complex conditions on a and b, that 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 (Peter Markstein [6, 2, 5, 3]) *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 mode $\circ$, taken among round to nearest even, round toward zero, round toward $-\infty$, round toward $+\infty$,

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

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

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 $|2^\sigma q'| \geq 2^{e_{\min}}$), then we clearly should return $2^\sigma q'$. The trouble is when $2^\sigma q'$ falls in the subnormal range: we cannot just return $\mathrm{RN}(2^\sigma q')$ because a *double rounding* phenomenon 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 $e_{\max} = 127$). Consider the two floating-point input values:
$$\left\{ \begin{array}{rcl} b & = & 1.00000000001100011001101_2 \times 2^{-113} \\ a & = & 1.00000000000011011001100_2 \times 2^{23} \end{array} \right. \begin{array}{rcl} & = & 8394957_{10} \times 2^{-136}, \\ & = & 8390348_{10}. \end{array}$$

The number b/a is equal to

$$0.10000000000100100000000000001011010011110110011001000000 10 \cdots \times 2^{-135},$$

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

$$0.00000000010000000000101 \times 2^{-126}.$$

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

$$0.10000000000100100000000000001011010011110110011001000000010\cdots \times 2^{-7},$$

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

$$1.00000000001001000000000 \times 2^{-8}.$$

Multiplied by 2^{-128}, this result would be equal equal to

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

which means—since it is in the subnormal range— that, after rounding to the nearest (even) floating-point number, we would get

$$0.00000000010000000000100 \times 2^{-126}.$$

This phenomenon may appear each time the scaled result q', once multiplied by 2^σ, is exactly equal to a (subnormal) midpoint, i.e., a value exactly halfway between two consecutive floating-point numbers. Notice that if we just use this scaled result 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, values computed during the last correction iteration will contain enough information to allow for a correctly rounded final result, as we are now going to see.

2 Avoiding double rounding

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

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

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 $2^\sigma b^*/a^*$. We assume that we are interested in quotients rounded to the nearest ("even", yet with round to nearest "away" the reasoning is not so different). 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.

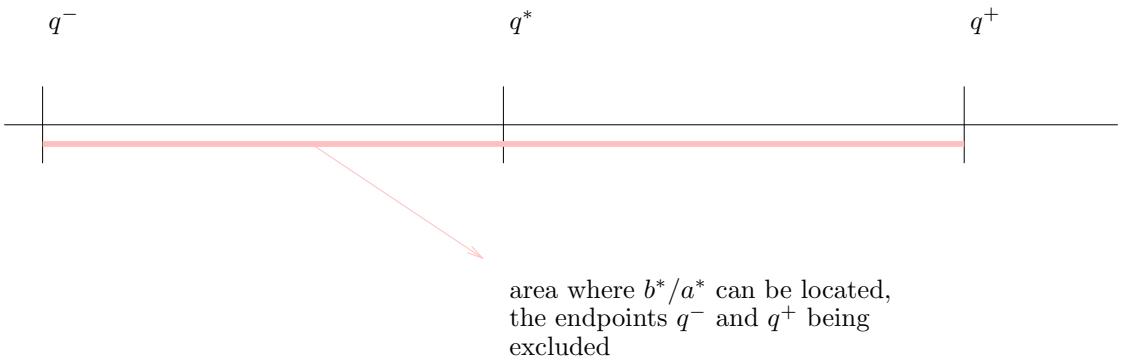

Figure 1: The number q^* is a faithful rounding of b^*/a^*: this means that $q^- < b^*/a^* < q^+$, where q^- and q^+ are the floating-point predecessor and successor of q^*.

As stated before, the "double rounding" problem occurs when $2^\sigma q'$ is a (subnormal) midpoint of the floating-point format. In such a case, 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 $2^\sigma q'$ rounded up;
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;
5. if $q' = q^*$ and $r < 0$, then q' overestimates b^*/a^*. Therefore, one should return $2^\sigma q'$ rounded down.

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

References

[1] S. Boldo and M. Daumas. Representable correcting terms for possibly underflowing floating point operations. In J.-C. Bajard and M. Schulte, editors,

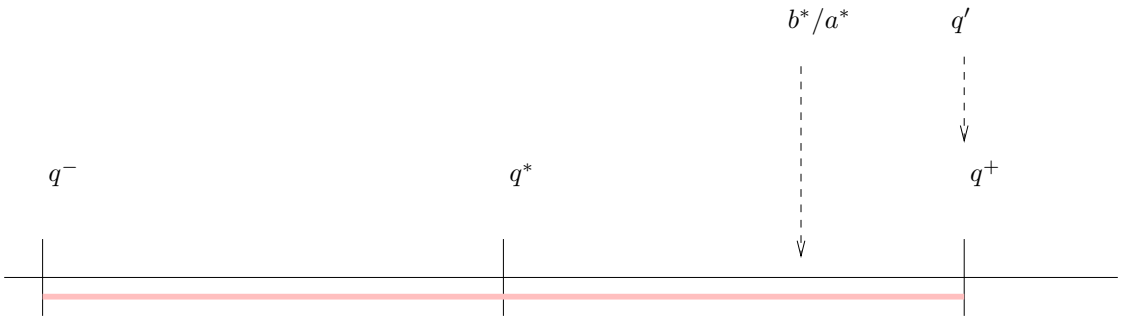

Figure 2: The number q' is equal to q^+. In this case, the "residual" r was positive, and—since $q^- < b^*/a^* < q^+$—, q' is an overestimation of b^*/a^*.

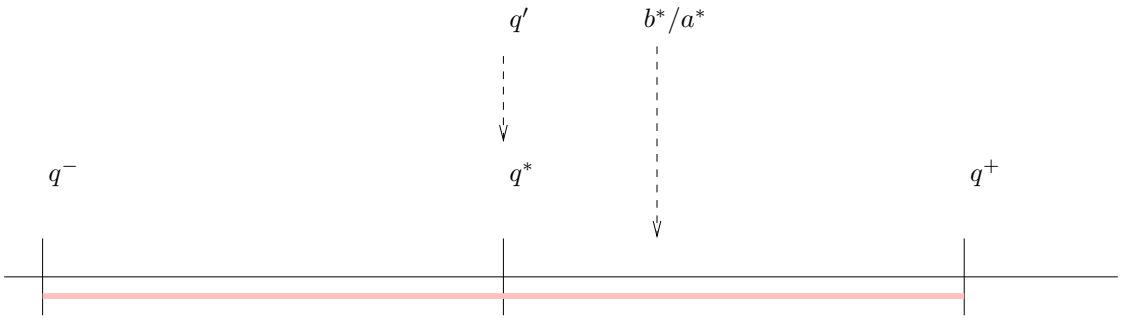

Figure 3: The number q' is equal to q^*. In this case, the "residual" r was positive, and q' is an underestimation of b^*/a^*.

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] 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.

[4] 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`.

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

[6] 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.

[7] 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.