High-level algorithms for correctly-rounded reciprocal square roots

Carlos F. Borges, Claude-Pierre Jeannerod, Jean-Michel Muller

High-level algorithms for correctly-rounded reciprocal square roots

Carlos F. Borges*, Claude-Pierre Jeannerod†, Jean-Michel Muller‡
* Department of Applied Mathematics, Naval Postgraduate School, Monterey CA 93943
† Inria, Laboratoire LIP (UMR 5668, CNRS, ENS de Lyon, Inria, UCBL), F-69342 Lyon, France
‡ CNRS, Laboratoire LIP (UMR 5668, CNRS, ENS de Lyon, Inria, UCBL), F-69342 Lyon, France

***Abstract*—We analyze two fast and accurate algorithms recently presented by Borges for computing $x^{-1/2}$ in binary floating-point arithmetic (assuming that efficient and correctly-rounded FMA and square root are available). The first algorithm is based on the Newton-Raphson iteration, and the second one uses an order-3 iteration. We give attainable relative-error bounds for these two algorithms, build counterexamples showing that in very rare cases they do not provide a correctly-rounded result, and characterize precisely when such failures happen in IEEE 754 binary32 and binary64 arithmetics. We then give a generic (i.e., precision-independent) algorithm that always returns a correctly-rounded result, and show how it can be simplified and made more efficient in the important cases of binary32 and binary64.**

***Index Terms*—reciprocal square root, floating-point arithmetic, correct rounding, accuracy analysis, fused multiply-add, error-free transform**

I. Introduction

The reciprocal square-root function $x \mapsto x^{-1/2}$ is listed (under the name rSqrt) in §9 "Recommended operations" of the IEEE 754-2019 Standard for Floating-Point Arithmetic [1, Table 9.1]. This means that providing a correctly-rounded implementation of that function is recommended, although not mandatory (an environment can be IEEE 754-compliant without offering it). The aim of our work is to give algorithms that compute reciprocal square roots very accurately for any floating-point format, possibly with correct rounding.

In the following, we assume a radix-2, precision-p (with $p \geq 2$) floating-point (FP) arithmetic, with minimum exponent $e_{\min}$. Beyond the usual four arithmetic operations, we also assume that fast, correctly-rounded, fused multiply-add (FMA) and square-root instructions are available. RN will denote the "round-to-nearest" function (with ties "to even"—which is the default in IEEE 754 arithmetic—or "to away"). The set of the FP numbers will be denoted $\mathbb{F}$. We also assume that the input value x to our algorithms is a positive element of $\mathbb{F}$.

Our main contributions are as follows:

- We provide detailed rounding-error analyses of the rsqrtNewton and rsqrtHalley algorithms recently proposed by Borges in [2], [3] for evaluating $x^{-1/2}$ accurately, giving in particular relative error bounds that are attainable.
- We also give counterexamples showing that these algorithms do not always return a correctly-rounded result, and we completely characterize the inputs for which this happens for the binary32 and binary64 formats.
- We then introduce a generic algorithm that is shown to always return a correctly-rounded result for all the FP formats satisfying $e_{\min} \leq -2p$.

The Newton-Raphson iteration for $x^{-1/2}$, of the form

$$y \leftarrow y + y(1 - xy^2)/2, \tag{1}$$

is frequently used in software implementations of reciprocal square roots, but also of the square root (preferably to the well-known Heron iteration), because it is division free; an excellent reference on such implementations is Markstein's book [4]. Kahan [5] gives an interesting discussion of this iteration, and points out that higher-order division-free iterations might be used as well: if $\widehat{y} \approx x^{-1/2}$ and $\epsilon = 1 - x\widehat{y}^2$, then

$$x^{-1/2} = \widehat{y} \cdot \left(1 + \frac{1}{2}\epsilon + \frac{3}{8}\epsilon^2 + \frac{5}{16}\epsilon^3 + \frac{35}{128}\epsilon^4 + O\left(\epsilon^5\right)\right). \tag{2}$$

Truncating the series (2) at order 1 in ϵ gives the Newton-Raphson iteration (1) and Algorithm rsqrtNewton, which we will analyze in Section III; truncating it at order 2 will lead to Algorithm rsqrtHalley, analyzed in Section IV.

Aiming at correctly-rounded results may require to know how close the exact value of $x^{-1/2}$, where x is an FP number, can be to a "midpoint" (that is, the exact middle of two consecutive FP numbers). It was shown in [6] that in binary arithmetic $x^{-1/2}$ cannot be a midpoint. For the binary64 format ($p = 53$) the "hardest-to-round cases" (that is, the values of x where $x^{-1/2}$ is nearest to a midpoint) were found by Lefèvre[1] using [7], [8]; we recall the ones we need in Table II.

Unless otherwise stated, our error analyses assume an unbounded exponent range. Hence the relative error due to rounding $t \in \mathbb{R}_{\neq 0}$ to a nearest element of $\mathbb{F}$ is bounded as

$$\left|\mathrm{RN}(t)/t - 1\right| \leq \tfrac{u}{1+u} \tag{3}$$

with $\frac{u}{1+u} < u := 2^{-p}$; see for example [9, p. 232].

II. Preliminaries

A. Error bounds for the basic algorithm

The most basic ways to evaluate the reciprocal square root using FP division and square root rely on the (mathematically) equivalent expressions $1/\sqrt{x}$, $\sqrt{1/x}$ and $\sqrt{x}/x$. Here, we focus on the second one, because it will be most convenient both for the accurate algorithms rsqrtNewton and rsqrtHalley

[1] See https://www.vinc17.net/research/testlibm/hrcases/hrcases-isq.xz.

and for our generic, correctly-rounded algorithm. Thus, what we actually compute is

$$\widehat{y} := \mathrm{RN}\big(\sqrt{\mathrm{RN}(1/x)}\big). \tag{4}$$

This basic approximation $\widehat{y}$ to $x^{-1/2}$ has three interesting properties which we review now and which will be key for the rest of the paper.

a) Relative error bound: The relative error of $\widehat{y}$ is bounded as

$$|\widehat{y}\sqrt{x} - 1| \le \tfrac{3}{2}u - 2u^2, \qquad p \ge 3. \tag{5}$$

To see this, note that $\widehat{y} = \sqrt{x^{-1}(1+\delta_1)}(1+\delta_2)$, where we know from [10] that $|\delta_1| \le u - 2u^2$ and $|\delta_2| \le 1 - 1/\sqrt{1+2u}$. Hence $\widehat{y} = x^{-1/2}(1+\delta)$ with $\delta = \sqrt{1+\delta_1}(1+\delta_2) - 1$, and it can be checked that the bounds on $|\delta_1|$ and $|\delta_2|$ imply $\delta \le 3u/2 - 2u^2$ for $p \ge 3$ and $\delta \ge -3u/2 + 2u^2$ for $p \ge 2$.

b) Absolute error bound and faithfulness: It turns out that $\widehat{y}$ is always a faithfully-rounded approximation to $x^{-1/2}$, which means that either $\widehat{y} = x^{-1/2}$ or $\widehat{y}$ is one of the two FP numbers surrounding $x^{-1/2}$. To check this property, we can start with the following absolute error bound.

Property 1. *For $x \in \mathbb{F} \cap (1/4, 1]$ and $p \ge 2$, the absolute error of $\widehat{y}$ is bounded as*

$$|\widehat{y} - x^{-1/2}| < \begin{cases} \left(1 + \frac{1}{\sqrt{2}}\right)u & \text{if } x \in (1/4, 1/2), \\ \frac{3}{2}u & \text{if } x \in [1/2, 1]. \end{cases}$$

Proof: Let $\widehat{r} = \mathrm{RN}(x^{-1})$, $A_1 = \widehat{y} - \sqrt{\widehat{r}}$ and $A_2 = \sqrt{\widehat{r}} - x^{-1/2}$, so that $|\widehat{y} - x^{-1/2}| \le |A_1| + |A_2|$. Since $x \in (1/4, 1]$, we have $\widehat{r} \in [1, 4]$ and $\sqrt{\widehat{r}}, \widehat{y} \in [1, 2]$, from which $|A_1| \le u$ follows. Then, in order to bound $|A_2|$, we note that $A_2 = \frac{\widehat{r} - x^{-1}}{\sqrt{\widehat{r}} + x^{-1/2}}$ and consider three sub-cases.

If $x \in (1/4, 1/2)$ then $x^{-1}, \widehat{r} \in [2, 4]$ and thus $|\widehat{r} - x^{-1}| \le 2u$; furthermore, $\sqrt{\widehat{r}} \ge \sqrt{2}$ and $x^{-1/2} > \sqrt{2}$, so that $\sqrt{\widehat{r}} + x^{-1/2} > 2\sqrt{2}$. Hence $|A_2| < 2u/(2\sqrt{2}) = u/\sqrt{2}$ and so $|\widehat{y} - x^{-1/2}| \le |A_1| + |A_2| < u + u/\sqrt{2}$ in this case.

If $x \in [1/2, 1)$ then $x^{-1}, \widehat{r} \in [1, 2]$ and thus $|\widehat{r} - x^{-1}| \le u$; furthermore, $\sqrt{\widehat{r}} \ge 1$ and $x^{-1/2} > 1$, so that $\sqrt{\widehat{r}} + x^{-1/2} > 2$. Hence $|A_2| < u/2$ and $|\widehat{y} - x^{-1/2}| < u + u/2$ in this case.

If $x = 1$ then $\widehat{y} = 1 = x^{-1/2}$ and so $|\widehat{y} - x^{-1/2}| = 0$. ∎

Remark now that when $x \in (1/4, 1]$ both the exact reciprocal square root $x^{-1/2}$ and its approximation $\widehat{y}$ are in the range $[1, 2]$. Since $|\widehat{y} - x^{-1/2}| < 2u$ by Property 1, it follows that $\widehat{y}$ is a faithfully-rounded approximation to $x^{-1/2}$. Although the above property has been given for the input range $(1/4, 1]$, its extension to any range $(4^{e-1}, 4^e]$ is immediate, and so the faithfulness of $\widehat{y}$ applies generally.

c) Consequence for the evaluation of $1 - x\widehat{y}^2$: A third useful feature of the way $\widehat{y}$ is computed is that we can obtain $\mathrm{RN}(1 - x\widehat{y}^2)$, that is, the correctly-rounded value of $1 - x\widehat{y}^2$, using only 3 FMAs. This is detailed in the next subsection.

B. Error-free transforms for division, square root, and FMA

In FP arithmetic, various sequences of operations called error-free transforms (EFT) are known to yield exact quantities. For example, if $a, b \in \mathbb{F}$ with $b \ne 0$ and $q = \mathrm{RN}(a/b)$, then $a - bq \in \mathbb{F}$, which implies that it is exactly computed with one FMA [11, p. 43]. Similarly, if $a \in \mathbb{F}_{\ge 0}$ and $s = \mathrm{RN}(\sqrt{a})$, then $s^2 - a \in \mathbb{F}$ and is obtained exactly as $\mathrm{RN}(s^2 - a)$ [12].

Algorithm ErrFMA below, introduced in [13], returns the error of an FMA instruction as the unevaluated sum of two FP numbers: if $d = \mathrm{RN}(ab + c)$, then $ab + c = d + e_1 + e_2$ and $|e_2| \le \frac{1}{2}\mathrm{ulp}(e_1)$. It uses the well-known Fast2Mult, Fast2Sum, and 2Sum algorithms (see for instance [14, Chap. 4]).

algorithm ErrFMA(a, b, c, d)
  $(v_1, v_2) :=$ Fast2Mult(a, b);
  $(s_1, s_2) :=$ 2Sum(c, v_2);
  $(s_3, s_4) :=$ 2Sum(v_1, s_1);
  $t := \mathrm{RN}\left(\mathrm{RN}(s_3 - d) + s_4\right)$;
  $(e_1, e_2) :=$ Fast2Sum(t, s_2);
  return (e_1, e_2)

In the following, to compute $x^{-1/2}$ accurately we will start from the initial faithfully-rounded approximation $\widehat{y}$ in (4) and then improve it when needed. From (2), we easily infer that computing $\epsilon = 1 - x\widehat{y}^2$ accurately will be crucial in doing so. Interestingly, it was remarked in [2], [3] that one can deduce the correctly-rounded value $\widehat{\epsilon} = \mathrm{RN}(\epsilon)$ with only three FMAs, by exploiting the EFTs for division and square root as follows:

$\widehat{r} := \mathrm{RN}(1/x)$;
$\widehat{y} := \mathrm{RN}(\sqrt{\widehat{r}})$;
$\sigma := \mathrm{RN}(1 - x\widehat{r})$; // EFT for division: $\sigma = 1 - x\widehat{r}$
$\tau := \mathrm{RN}(\widehat{r} - \widehat{y}^2)$; // EFT for square root: $\tau = \widehat{r} - \widehat{y}^2$
$\widehat{\epsilon} := \mathrm{RN}(\sigma + x\tau)$;

We have $\sigma + x\tau = (1 - x\widehat{r}) + x(\widehat{r} - \widehat{y}^2) = 1 - x\widehat{y}^2$, so that

$$\widehat{\epsilon} = \mathrm{RN}(1 - x\widehat{y}^2).$$

This best possible approximation to ϵ will be key for Algorithms rsqrtNewton and rsqrtHalley in sections III and IV; in section V it will be exploited jointly with the EFT for FMA (Algorithm ErrFMA above) in order to design our generic and correctly-rounded algorithm (Algorithm Generic_CR_rsqrt).

III. One iteration of Newton's method

We focus here on improving our initial approximation $\widehat{y}$ to $x^{-1/2}$ by computing y_2 such that

$$y_2 = \widehat{y}(1 + \epsilon/2), \qquad \epsilon = 1 - x\widehat{y}^2.$$

As noted by Kahan [5], this implies

$$y_2\sqrt{x} - 1 = -(\widehat{y}\sqrt{x} - 1)^2(1 + \widehat{y}\sqrt{x}/2),$$

so that (assuming exact arithmetic) the initial relative error

$$\delta = \widehat{y}\sqrt{x} - 1 \tag{6}$$

is improved into $y_2\sqrt{x} - 1 = -\delta^2(3/2 + \delta/2) \approx -\frac{3}{2}\delta^2$.

How should this scheme be implemented in IEEE arithmetic? An approximation $\widehat{\epsilon}$ will be computed first, followed by one exact multiplication by 0.5 and one FMA in order to obtain $\widehat{y}_2 = \mathrm{RN}(\widehat{y} + \widehat{y}\widehat{\epsilon}/2)$. Concerning the computation of $\widehat{\epsilon}$

itself, the availability of an FMA offers several possibilities: $\mathrm{RN}(1 - x\mathrm{RN}(\widehat{y}^2))$ or $\mathrm{RN}(1 - \mathrm{RN}(x\widehat{y})\widehat{y})$ with one multiplication and one FMA, or, as described in the previous section, as $\mathrm{RN}(1 - x\widehat{y}^2)$ via the EFTs for division and square root and using 3 FMAs. In all cases, $\widehat{\epsilon}$ has the form $\mathrm{RN}(1-x\widehat{y}^2(1+\delta_0))$ and, assuming $|\delta| = O(u)$, it is easily checked that an error analysis to first order in u gives

$$|\widehat{y}_2\sqrt{x} - 1| \le u + |\delta_0|/2 + O(u^2).$$

The first two ways shown above for $\widehat{\epsilon}$ have $|\delta_0| \le u$, so $\widehat{y}_2$ has a relative error bounded by about $3u/2$, similarly to the naive scheme used to get $\widehat{y}$. (In practice, the largest relative error for $\widehat{y}_2$ can even be larger than the one for $\widehat{y}$, and this occurs for example when $p = 11$.) In contrast, the EFT-based scheme producing $\widehat{\epsilon} = \mathrm{RN}(1 - x\widehat{y}^2)$ has $\delta_0 = 0$ and the relative error of $\widehat{y}_2$ is now at most $u+O(u^2)$. It is this third scheme that has been chosen in [2], [3] to set up the algorithm rsqrtNewton displayed below and which we analyze further in the rest of this section.

algorithm rsqrtNewton(x)
  $\widehat{r} := \mathrm{RN}(1/x)$;
  $\widehat{y} := \mathrm{RN}\big(\sqrt{\widehat{r}}\big)$;
  $\sigma := \mathrm{RN}(1 - x\widehat{r})$;
  $\tau := \mathrm{RN}(\widehat{r} - \widehat{y}^2)$;
  $\widehat{\epsilon} := \mathrm{RN}(\sigma + x\tau)$;
  $\widehat{y}_2 := \mathrm{RN}\big(\widehat{y} + \widehat{y} \cdot \mathrm{RN}(\widehat{\epsilon}/2)\big)$;
  return $\widehat{y}_2$

A. Rounding error analysis

We present here a detailed accuracy analysis of Algorithm rsqrtNewton, providing bounds on both its relative error and its absolute error.

In particular, we give in Theorem 1 a closed-form expression of the largest relative error and show that it is attained only at special input values of the form $(1 - 2u) \cdot 4^e$. As this theorem shows, the largest possible relative error takes two different values depending on how RN breaks ties: it is

$$1 - \sqrt{1 - 2u} = u + \frac{1}{2}u^2 + O(u^3)$$

if $\mathrm{RN}(1 + u) = 1$, which is the case if RN is *ties-to-even*, and

$$(1 + 2)\sqrt{1 - 2u} - 1 = u - \frac{5}{2}u^2 + O(u^3)$$

if $\mathrm{RN}(1 + u) = 1 + 2u$, which is the case if RN is *ties-to-away*. We also show that outside the special input of the form $x = (1 - 2u) \cdot 4^e$ and whatever the tie-breaking rule, the relative error is always strictly less than u. Third, we provide two absolute error bounds, one for $\widehat{y}_2$ and one for the quantity obtained just before the final rounding, which we will exploit in section III-B in order to design correctly-rounded variants of rsqrtNewton for various floating-point arithmetics.

Theorem 1. *For $x \in \mathbb{F}_{>0}$, let $\widehat{y}_2$ be the approximation to $x^{-1/2}$ computed by Algorithm rsqrtNewton. If $p \ge 5$ then*

- *its relative error is bounded as*

$$\left|\widehat{y}_2\sqrt{x} - 1\right| \le B := \left|\mathrm{RN}(1 + u)\sqrt{1 - 2u} - 1\right|,$$

  where $\mathrm{RN}(1+u)$ is either 1 or $1+2u$ depending on the tie-breaking rule;
- *the bound B is attained if and only if $x = (1 - 2u) \cdot 4^e$, $e \in \mathbb{Z}$, and $|\widehat{y}_2\sqrt{x} - 1| < u$ if $x \ne (1 - 2u) \cdot 4^e$;*
- *the absolute error is bounded by $(\frac{1}{2} + \frac{35}{8}u)\mathrm{ulp}(x^{-1/2})$;*
- *$\widehat{y}_2 = \mathrm{RN}(t)$ with $t \in \mathbb{R}$ such that*

$$|t - x^{-1/2}| \le \frac{35}{8}u\,\mathrm{ulp}(x^{-1/2}). \tag{7}$$

Proof: Assume for simplicity that $x \in (1/4, 1]$ and let $t = \widehat{y}(1 + \widehat{\epsilon}/2)$. Then $\widehat{y}_2 = \mathrm{RN}(t)$ and

$$|\widehat{y}_2\sqrt{x} - 1| \le |\mathrm{RN}(t) - t|\sqrt{x} + |t\sqrt{x} - 1|. \tag{8}$$

Let now $\alpha, \delta \in \mathbb{R}$ be such that $\widehat{\epsilon} = 1 - x\widehat{y}^2 + \alpha$ and $\widehat{y} = x^{-1/2}(1+\delta)$. We have $1-x\widehat{y}^2 = -2\delta-\delta^2$ and, therefore,

$$\begin{aligned} t\sqrt{x} - 1 &= (1 + \delta)\Big(1 - \delta - \frac{\delta^2}{2} + \frac{\alpha}{2}\Big) - 1 \\ &= -\frac{3}{2}\delta^2 + \frac{\alpha}{2}(1 + \delta) - \frac{1}{2}\delta^3. \end{aligned} \tag{9}$$

Recall from (5) that $|\delta| \le \frac{3}{2}u - 2u^2$ for $p \ge 3$, which implies $|1 - x\widehat{y}^2| \le 2|\delta| + \delta^2 < 4u$ and thus $|\alpha| \le 2u^2$. By applying these bounds on $|\alpha|$ and $|\delta|$ to (9) we deduce that

$$\begin{aligned} |t\sqrt{x} - 1| &\le \frac{3}{2}\delta^2 + u^2(1 + |\delta|) + \frac{1}{2}|\delta|^3 \\ &\le \frac{35}{8}u^2 - \frac{93}{16}u^3 - \frac{11}{4}u^4 + 9u^5 - 4u^6 \\ &\le \frac{35}{8}u^2. \end{aligned} \tag{10}$$

Let us now bound $|\mathrm{RN}(t) - t|$. We have $|t| \le |t - x^{-1/2}| + x^{-1/2} = \big(|t\sqrt{x} - 1| + 1\big)x^{-1/2}$ and so, using (10) together with $x > 1/4$, we obtain $|t| \le 2(1 + \frac{35}{8}u^2) \le 2 + u$ for $p \ge 4$. Hence $|\mathrm{RN}(t) - t| \le u$. Together with (8) and (10),

$$|\widehat{y}_2\sqrt{x} - 1| \le u\sqrt{x} + \frac{35}{8}u^2 =: \varphi(x). \tag{11}$$

The claimed bounds on $|t - x^{-1/2}|$ and $|\widehat{y}_2 - x^{-1/2}|$ follow directly from (10) and (11) by multiplying by $x^{-1/2} < 2$ and since $\mathrm{ulp}(x^{-1/2}) = 2u$, and so we are left with determining the largest relative error.

Defining $B_1 = 1-\sqrt{1 - 2u}$ and $B_2 = (1+2u)\sqrt{1 - 2u}-1$, note first that $B_2 < u < B_1$. If $x \le 1 - 14u$ then (11) gives $\varphi(x) < u(1 - 7u) + \frac{35}{8}u^2 = u - \frac{21}{8}u^2$. One can check that $u - \frac{21}{8}u^2 \le B_2$ for $p \ge 4$, so $|\widehat{y}_2\sqrt{x} - 1| < B_2$ in this case.

Assume now $x > 1-14u$. For $p \ge 5$ we have $1-14u > 1/2$ and so x has the form $x = 1 - ju$ with $0 \le j \le 13$. It then suffices to check for each of these values of x that our claims about the relative error of $\widehat{y}_2$ are true. In particular, writing $\delta_2 := \widehat{y}_2\sqrt{x} - 1$, we have the following:

- If $j \in \{0, 1\}$ then $\widehat{y} = \widehat{y}_2 = 1$, and $|\delta_2| < B_2$;
- If $j = 2$ then $\widehat{y} = 1$, $\widehat{y}_2 = \mathrm{RN}(1 + u)$, $|\delta_2| = B$;
- If $j = 3$, then $\widehat{y} = 1 + 2u$, $\widehat{\epsilon} = -u + 8u^2$, $\widehat{y}_2 = 1 + 2u$, $|\delta_2| = (1 + 2u)\sqrt{1 - 3u} - 1 < B_2$ for $p \ge 5$.

Similarly, one can check that $|\delta_2| < B_2$ for the ten remaining values of j, noting in particular that only the case $j = 2$ depends on the tie-breaking rule. ∎

B. Consequences for correct rounding

1) Round to nearest ties-to-even: A first consequence of Theorem 1 is that for rounding to nearest ties-to-even, which is the default in IEEE arithmetic, the relative error of $\widehat{y}_2$ can be (slightly but strictly) larger than u. This already appears in [3], where it has been observed that $x = 1 - 2u$ yields $\widehat{y}_2 = 1$ instead of $\mathrm{RN}(x^{-1/2}) = 1 + 2u$ and, therefore, that rsqrtNewton does not always give the correctly-rounded result.

Going further, one can ask whether the generic input values of the form $x = (1 - 2u) \cdot 4^e$ are the only ones for which $\widehat{y}_2 \neq \mathrm{RN}(x^{-1/2})$. A few exhaustive tests for reasonably small floating-point formats reveal that the answer depends on the precision p and that, at least up to $p = 24$, such extra x's tend to be rare (at most 3 per sub-interval $(4^e, 4^{e+1}]$); see Table I.

In particular, the binary32 format ($p = 24$) has no bad cases other than the above generic ones, so a correctly-rounded scheme for this format could be as follows:

> if $x = 8388607 \cdot 2^{-23+2e}$ then return $8388609 \cdot 2^{23-e}$
> else return rsqrtNewton(x)

TABLE I
LIST OF THE NONGENERIC INPUTS $x \in \mathbb{F}$ FOR WHICH RSQRTNEWTON(x) $\neq$ RN($x^{-1/2}$) FOR $3 \leq p \leq 24$. IF RN IS ROUND-TO-NEAREST *ties-to-even*, THE GENERIC INPUTS $x = (1 - 2u) \cdot 4^e$ MUST BE ADDED TO THIS LIST; IF RN IS ROUND-TO-NEAREST *ties-to-away*, THESE GENERIC CASES ARE CORRECTLY ROUNDED.

p	x
3	$6 \cdot 4^e$
$4-5$	none
6	$13 \cdot 4^e$, $42 \cdot 4^e$
$7-14$	none
15	$31454 \cdot 4^e$
16	$12258 \cdot 4^e$, $22982 \cdot 4^e$
$17-18$	none
19	$401651 \cdot 4^e$
20	$108515 \cdot 4^e$, $883590 \cdot 4^e$, $1107658 \cdot 4^e$
$21-22$	none
23	$11586530 \cdot 4^e$
24	none

2) Round to nearest ties-to-away: In this case, things are somehow better, since we know from Theorem 1 that $\widehat{y}_2$ always has a relative error $< u$ and, as seen in the proof, that the generic values $x = (1 - 2u) \cdot 4^e$ now give the correctly-rounded result.

Furthermore, the same exhaustive tests as before yield the same problematic input values (listed in Table I). Hence, although some incorrectly-rounded results can still be produced for some precisions, we now have in particular

$$\text{rsqrtNewton}(x) = \mathrm{RN}(x^{-1/2}) \text{ for all binary32 } x.$$

3) IEEE binary64 format ($p = 53$): In this case exhaustive testing is not possible anymore, but we will see how to combine the bound in (7) with the hardest-to-round cases from Table II in order to characterize precisely the floating-point numbers x for which rsqrtNewton(x) does not yield a correctly-rounded result.

TABLE II
THE 15 HARDEST-TO-ROUND CASES $x_k \in \mathbb{F} \cap (1/4, 1]$ FOR $x^{-1/2}$ IN ROUNDED-TO-NEAREST BINARY64 ARITHMETIC ($p = 53$), WITH λ_k DEFINING THE DISTANCE BETWEEN $x_k^{-1/2}$ AND ITS NEAREST MIDPOINT m_k AS $x_k^{-1/2} - m_k = \lambda_k u^2$, AND WITH $\lambda_k^{(2)}$ SUCH THAT $t - m_k = \lambda_k^{(2)} u^2$ FOR $t = \widehat{y}(1 + \widehat{\epsilon}/2)$.

k	x_k	λ_k	$\lambda_k^{(2)}$
1	$3717785442934375 \cdot 2^{-53}$	$-0.04\ldots$	$-1.52\ldots$
2	$7976044270474205 \cdot 2^{-53}$	$0.57\ldots$	$-0.31\ldots$
3	$4503599627370495 \cdot 2^{-52}$	$1.50\ldots$	0
4	$2202051755894995 \cdot 2^{-52}$	$2.09\ldots$	$1.32\ldots$
5	$7971447988064653 \cdot 2^{-54}$	$2.24\ldots$	$1.87\ldots$
6	$3399064274801837 \cdot 2^{-52}$	$2.39\ldots$	$0.98\ldots$
7	$566516437981199 \cdot 2^{-51}$	$-4.40\ldots$	$-5.88\ldots$
8	$2647340912692081 \cdot 2^{-53}$	$5.22\ldots$	$4.27\ldots$
9	$2994539392738155 \cdot 2^{-52}$	$-6.19\ldots$	$-6.96\ldots$
10	$3519625113831519 \cdot 2^{-52}$	$-6.33\ldots$	$-7.86\ldots$
11	$7373471117307515 \cdot 2^{-53}$	$-6.73\ldots$	$-8.59\ldots$
12	$4168334449631061 \cdot 2^{-52}$	$-7.52\ldots$	$-9.43\ldots$
13	$7564078810642109 \cdot 2^{-53}$	$7.66\ldots$	$6.29\ldots$
14	$2043522089595771 \cdot 2^{-52}$	$8.59\ldots$	$8.24\ldots$
15	$6287158043890989 \cdot 2^{-54}$	$8.77\ldots$	$8.32\ldots$

First, recall from (7) that $\widehat{y}_2 = \mathrm{RN}(t)$ with $|t - x^{-1/2}| \leq \frac{35}{8} u\,\mathrm{ulp}(x^{-1/2})$. Hence, if $\widehat{y}_2 \neq \mathrm{RN}(x^{-1/2})$ then there must be some midpoint m between t and $x^{-1/2}$, which for $x \in (1/4, 1]$ implies that

$$|m - x^{-1/2}| \leq |t - x^{-1/2}| \leq \frac{35}{8} u \cdot 2u = 8.75u^2.$$

Then, using Table II, we deduce that this requires $|\lambda_k| \leq 8.75$, that is, $x \in \{x_1, \ldots, x_{14}\}$. For $k \leq 14$, if both λ_k and $\lambda_k^{(2)}$ are negative (resp. positive), then both $x^{-1/2}$ and t are just below (resp. above) m_k and thus round down (resp. up) to the same correct value $\widehat{y}_2 = \mathrm{RN}(x^{-1/2})$. This is the case for all $k \notin \{2, 3\}$.

When $k = 2$, we have $\lambda_k^{(2)} < 0 < \lambda_k$, that is, $t < m_k < x^{-1/2}$ and thus $\widehat{y}_2$ is the predecessor of $\mathrm{RN}(x^{-1/2})$.

When $k = 3$, we have $\lambda_k^{(2)} = 0 < \lambda_k$, which means that t is a midpoint and $x^{-1/2}$ is just above it: $t = m_k < x^{-1/2}$. Hence the value of $\widehat{y}_2 = \mathrm{RN}(t)$ will depend on the tie-breaking rule: for *ties-to-away*, t is rounded up to the correctly-rounded value $\mathrm{RN}(x^{-1/2})$; for *ties-to-even*, the value of t in this specific situation is such that it is rounded down to the predecessor of $\mathrm{RN}(x^{-1/2})$. (Alternatively, one could observe that x_3 is in fact the generic case $1 - 2u$ and use Theorem 1 to predict that $\widehat{y}_2$ is either $1 + 2u = \mathrm{RN}(x^{-1/2})$ or 1, depending on the tie-breaking rule.)

To summarize, we have shown that for the binary64 format Algorithm rsqrtNewton returns the correctly-rounded value except for $x_2 \cdot 4^e$ and $x_3 \cdot 4^e$ when rounding is *to nearest ties-to-even*, and except for $x_2 \cdot 4^e$ when rounding is *ties-to-away*. Correctly-rounded schemes then follow immediately: return $\mathrm{RN}(x_2^{-1/2}) \cdot 2^{-e}$ or $\mathrm{RN}(x_3^{-1/2}) \cdot 2^{-e}$ whenever needed, and return rsqrtNewton(x) in all other cases.

IV. Order-3 algorithm

We now consider the algorithm below, recently introduced by Borges in [3] under the name rsqrtHalley and which is based on taking the first three terms of the series in (2): the initial approximation $\widehat{y}$ to $x^{-1/2}$ is refined by one call to the iteration defined as $y_3 = \widehat{y}(1+\epsilon/2+3\epsilon^2/8)$ with $\epsilon = 1-x\widehat{y}^2$.

In exact arithmetic it is easily checked that

$$y_3\sqrt{x} - 1 = (\widehat{y}\sqrt{x} - 1)^3\big(1 + \tfrac{9}{8}\widehat{y}\sqrt{x} + \tfrac{3}{8}x\widehat{y}^2\big).$$

Hence, denoting as before by δ the relative error $\widehat{y}\sqrt{x}-1$ of the initial approximation, we see that the refined approximation y_3 has relative error $\delta^3(1 + \frac{9}{8}(1+\delta) + \frac{3}{8}(1+\delta)^2) \approx \frac{5}{2}\delta^3$. This iteration of order 3 is well known and has been considered for various specific floating- or fixed-point formats [4], [15].

For generic floating-point arithmetic, Borges shows how to exploit the availability of an FMA instruction to produce

- the correctly-rounded value $\widehat{\epsilon}$ as shown in Section II-B;
- an approximation $\widehat{y}_3$ by evaluating y_3 with the scheme $\widehat{y} + \widehat{y}\big(\mu + (\frac{3}{2}\mu)\cdot\mu\big)$, where $\mu = \widehat{\epsilon}/2$ can be computed exactly in base two.

algorithm rsqrtHalley(x)
$\widehat{r} := \mathrm{RN}(1/x)$;
$\widehat{y} := \mathrm{RN}(\sqrt{\widehat{r}})$;
$\sigma := \mathrm{RN}(1 - x\widehat{r})$;
$\tau := \mathrm{RN}(\widehat{r} - \widehat{y}^2)$;
$\widehat{\epsilon} := \mathrm{RN}(\sigma + x\tau)$;
$\mu := \mathrm{RN}(\widehat{\epsilon}/2)$;
$\widehat{\nu} := \mathrm{RN}(\frac{3}{4}\widehat{\epsilon})$;
$\widehat{y}_3 := \mathrm{RN}(\widehat{y} + \widehat{y}\cdot\mathrm{RN}(\mu + \mu\widehat{\nu}))$;
return $\widehat{y}_3$

A. Rounding error analysis

The approach taken in the previous section to analyze the accuracy of the Newton-Raphson method can be extended here in a fairly natural way, by simply introducing and taking into account two extra errors: the one for $\frac{3}{2}\mu = \frac{3}{4}\widehat{\epsilon}$, and the one for $\mu+\mu\widehat{\nu}$. Specifically, assuming again with no loss of generality that $x \in (1/4, 1]$, we let as before α and δ correspond to the absolute error of $\widehat{\epsilon}$ and the relative error of $\widehat{y}$:

$$\alpha = \widehat{\epsilon} - \epsilon, \qquad \delta = \widehat{y}\sqrt{x} - 1.$$

In addition, we now let $\widetilde{\alpha}$ and $\widetilde{\delta}$ be such that

$$\mathrm{RN}(\mu + \mu\widehat{\nu}) = \mu + \mu\widehat{\nu} + \widetilde{\alpha}, \qquad \widehat{\nu} = \frac{3}{2}\mu\big(1+\widetilde{\delta}\big),$$

and s and t be such that

$$s = \mu + \mu\widehat{\nu}, \qquad t = \widehat{y} + \widehat{y}\,\mathrm{RN}(s). \tag{12}$$

From these definitions it follows that

$$\begin{aligned} t\sqrt{x} - 1 &= \widehat{y}\sqrt{x}(1 + s + \widetilde{\alpha}) - 1 \\ &= (1+\delta)\Big(1 + \mu + \frac{3}{2}\mu^2\big(1+\widetilde{\delta}\big) + \widetilde{\alpha}\Big) - 1, \end{aligned}$$

where

$$\mu = -\delta - \frac{\delta^2}{2} + \frac{\alpha}{2}$$

(since $\mu = \widehat{\epsilon}/2$ and, as before, $\widehat{\epsilon} = 1 - x\widehat{y}^2 + \alpha = -2\delta - \delta^2 + \alpha$). This implies that $t\sqrt{x} - 1$ is a polynomial in the four error terms $\alpha, \widetilde{\alpha}, \delta, \widetilde{\delta}$; in particular, it has degree 5 in δ and can thus be expressed as $t\sqrt{x} - 1 = \sum_{i=0}^{5} A_i\delta^i$ with each A_i being a polynomial in $\alpha, \widetilde{\alpha}, \widetilde{\delta}$. More precisely, one can check that

$$\begin{aligned} A_0 &= \frac{\alpha}{2} + \widetilde{\alpha} + \frac{3}{8}\alpha^2(1+\widetilde{\delta}), \\ A_1 &= -\alpha\Big(1 + \frac{3}{2}\widetilde{\delta}\Big) + \widetilde{\alpha} + \frac{3}{8}\alpha^2(1+\widetilde{\delta}), \\ A_2 &= \frac{3}{2}\widetilde{\delta} - \frac{9}{4}\alpha(1+\widetilde{\delta}), \\ A_3 &= \frac{5}{2} + 3\widetilde{\delta} - \frac{3}{4}\alpha(1+\widetilde{\delta}), \\ A_4 &= \frac{15}{8}(1+\widetilde{\delta}), \\ A_5 &= \frac{3}{8}(1+\widetilde{\delta}). \end{aligned}$$

We will now bound $|t\sqrt{x} - 1|$ using bounds on $|\alpha|$, $|\widetilde{\alpha}|$, $|\delta|$, and $|\widetilde{\delta}|$. For $x \in (1/4, 1]$ and from the analysis done for the Newton-Raphson method, we already know that $|\alpha| \le 2u^2$ and, for $p \ge 3$, that $|\delta| \le \frac{3}{2}u - 2u^2$. Furthermore, (3) implies $|\widetilde{\delta}| \le u/(1+u)$. Finally, in order to bound $|\widetilde{\alpha}| = |\mathrm{RN}(s) - s|$, note first that $|1 - x\widehat{y}^2| \le 2|\delta| + \delta^2 < 3u \in \mathbb{F}$, which implies $|\widehat{\epsilon}| \le 3u$ and thus $|\mu| \le \frac{3}{2}u$. Consequently, $|s| \le |\mu| + \frac{3}{2}\mu^2(1 + |\widetilde{\delta}|) \le \frac{3}{2}u + \frac{3}{2}\cdot\frac{9}{4}u^2(1+u) < 2u$ for $p \ge 3$, and it follows that $|\widetilde{\alpha}| = |\mathrm{RN}(s) - s| \le u^2$. To summarize, we can take

$$|\alpha| \le 2u^2, \quad |\widetilde{\alpha}| \le u^2, \quad |\delta| \le \frac{3}{2}u - 2u^2, \quad |\widetilde{\delta}| \le u/(1+u).$$

As an immediate consequence, observe that $|A_0| \le 2u^2 + O(u^4)$, $|A_1| = O(u^2)$, and $|A_2| = O(u)$, from which we deduce that $|t\sqrt{x}-1| \le \sum_{i=0}^{5}|A_i||\delta|^i \le 2u^2 + O(u^3)$. A more careful analysis, where we apply first the triangle inequality to each of the $|A_i|$'s and then the exact expressions of the four bounds above, leads to

$$|t\sqrt{x} - 1| \le 2u^2 + 17u^3.$$

This inequality can be seen as the counterpart of the one in (10) with t as in (12).

Similarly to the error analysis of the Newton-Raphson method seen before, it follows that the relative error of $\widehat{y}_3$ satisfies

$$|\widehat{y}_3\sqrt{x} - 1| \le u\sqrt{x} + 2u^2 + 17u^3 =: \varphi(x).$$

Furthermore, using again $1 \le x^{-1/2} < 2$, we can directly reformulate these two bounds in terms of $\mathrm{ulp}(x^{-1/2}) = 2u$

as, respectively, $|t - x^{-1/2}| \le (2u + 17u^2)\mathrm{ulp}(x^{-1/2})$ and $|\widehat{y}_3 - x^{-1/2}| \le (\frac{1}{2} + 2u + 17u^2)\mathrm{ulp}(x^{-1/2})$.

Let us finally show that the largest relative error of $\widehat{y}_3$ is $B := (1+2u)\sqrt{1-2u} - 1$. If $x \le 1 - 10u$ then $\varphi(x) < u(1 - 5u - \frac{25}{2}u^2) + 2u^2 + 17u^3 = u - 3u^2 + \frac{9}{2}u^3$, which is less than $B = u - \frac{5}{2}u^2 - O(u^3)$ for all $p \ge 4$. If $x > 1 - 10u$ then $x \in [1/2, 1]$ for $p \ge 5$ and it suffices to consider each $x = 1 - ju$ with $0 \le j \le 9$ and to check that the relative error $|\widehat{y}_3\sqrt{x} - 1|$ is B when $j = 2$, and less than B otherwise.

We thus obtain the following result.

Theorem 2. *For $x \in \mathbb{F}_{>0}$, let $\widehat{y}_3$ be the approximation to $x^{-1/2}$ computed by Algorithm rsqrtHalley. If $p \ge 5$ then*

- *its relative error is at most $(1+2u)\sqrt{1-2u} - 1$ and this bound is attained if and only if $x = (1-2u)\cdot 4^e$, $e \in \mathbb{Z}$;*
- *its absolute error is at most $(\frac{1}{2} + 2u + 17u^2)\mathrm{ulp}(x^{-1/2})$;*
- *it has the form $\widehat{y}_3 = \mathrm{RN}(t)$ for some $t \in \mathbb{R}$ such that $|t - x^{-1/2}| \le (2u + 17u^2)\mathrm{ulp}(x^{-1/2})$.*

B. Consequences for correct rounding

The accuracy results given in Theorem 2 have a number of interesting implications. First, the relative error is now always $< u$ whatever the tie-breaking rule. Hence, in contrast to rsqrtNewton, it is not obvious anymore from this error bound that rsqrtHalley could fail to ensure correct rounding.

Exhaustive tests with small precisions show that such failures are still possible, but also that the situation is better than for rsqrtNewton in the sense that they occur for even fewer values of x (see Table III). In particular, for the binary32 format ($p = 24$) and rounding *to nearest ties-to-even*, rsqrtHalley always returns the correctly-rounded value, whereas rsqrtNewton would fail on $x = (1-2u)\cdot 4^e$.

TABLE III
LIST OF ALL $x \in \mathbb{F}$ FOR WHICH RSQRTHALLEY$(x) \ne \mathrm{RN}(x^{-1/2})$ FOR $2 \le p \le 24$. THIS LIST IS THE SAME FOR *ties-to-even* AND *ties-to-away*.

p	x
$2-5$	none
6	$13 \cdot 4^e$
7	none
8	$155 \cdot 4^e$
$9-19$	none
20	$1048576 \cdot 4^e$
$21-24$	none

For the binary64 format ($p = 53$), we can combine our bound on $|t - x^{-1/2}|$ from Theorem 2 with the hardest-to-round cases $x_1, \ldots, x_6$ from Table II to deduce that rsqrtHalley returns the correctly-rounded value unless $x = x_1 \cdot 4^e$, $e \in \mathbb{Z}$. (More precisely, since t is now as in (12), the values $\lambda_k^{(2)}$ in that table must be replaced by some new values, $\lambda_k^{(3)}$, that turn out to be positive for all $k \le 6$, so correct rounding does not occur only when $k = 1$.) For example, for $x = 3717785442934375 \cdot 2^{-53}$ then $\widehat{y}_3 = 3504955453675595 \cdot 2^{-51}$, which is the successor in $\mathbb{F}$ of $\mathrm{RN}(x^{-1/2}) = 7009910907351189 \cdot 2^{-52}$. This shows that rsqrtHalley does not always guarantee correct rounding when $p = 53$, thus answering a question from [3, §3].

V. A GENERIC CORRECTLY-ROUNDED ALGORITHM

We have seen that Algorithms rsqrtNewton and rsqrtHalley almost always return a correctly-rounded result, and that in the binary32 and binary64 formats, all input values for which this is not the case are known. A possible way of implementing a correctly-rounded reciprocal square root in these formats is therefore to check if the input x is one of these few values. This cannot work in binary128 arithmetic: for that format the hardest-to-round cases are not known, and seem out of reach. We now introduce a generic algorithm that always returns a correctly-rounded result. As we will see, it can be simplified in the binary32 and binary64 formats. In very rare cases[2], we will use Algorithm ErrFMA (error of an FMA instruction), presented in Section II-B.

A. If x is between 1 and 4

1) The fully generic algorithm: Let us now introduce our algorithm. We temporarily assume that the input x is between 1 and 4 (we explain later on how to reduce to this case).

algorithm Generic_CR_rsqrt(x)
  $\widehat{r} := \mathrm{RN}(1/x)$
  $\widehat{y} := \mathrm{RN}(\sqrt{\widehat{r}})$;
  $\sigma := \mathrm{RN}\left(1 - x\widehat{r}\right)$;
  $\tau := \mathrm{RN}(\widehat{r} - \widehat{y}^2)$;
  $\widehat{\epsilon} := \mathrm{RN}\left(\sigma + x\tau\right)$;
  if $\widehat{\epsilon} = 0$ **then** $\widehat{y}_{\mathrm{CR}} := \widehat{y}$;
  else
    if $\widehat{\epsilon} > 0$ **then** $s := 1$ **else** $s := -1$ **end if**
    $\widehat{\eta} := \mathrm{RN}\left(xu \cdot \widehat{y} + xsu^2/4\right)$;
    if $\widehat{\eta} > |\widehat{\epsilon}|$ **then** $\widehat{y}_{\mathrm{CR}} := \widehat{y}$;
    else if $\widehat{\eta} < |\widehat{\epsilon}|$ **then** $\widehat{y}_{\mathrm{CR}} := \widehat{y} + su$;
    else
      $(v_\eta, w_\eta) := \mathrm{ErrFMA}\left(xu, \widehat{y}, xsu^2/4, \widehat{\eta}\right)$;
      $(v_\epsilon, w_\epsilon) := \mathrm{ErrFMA}\left(sx, \tau, s\sigma, \widehat{\eta}\right)$;
      if $(v_\eta > v_\epsilon)$ **or** $((v_\eta = v_\epsilon)$ **and** $(w_\eta > 0))$ **then**
        $\widehat{y}_{\mathrm{CR}} := \widehat{y}$;
      else
        $\widehat{y}_{\mathrm{CR}} := \widehat{y} + su$;
      end if
    end if
  end if
  return $\widehat{y}_{\mathrm{CR}}$

Theorem 3. *If the minimum exponent $e_{\min}$ of the FP system satisfies $e_{\min} \le -2p - 2$, then for any $x \in \mathbb{F} \cap [1, 4)$, the number $\widehat{y}_{CR}$ returned by Algorithm Generic_CR_rsqrt satisfies*

$$\widehat{y}_{CR} = \mathrm{RN}\left(x^{-1/2}\right).$$

Proof: We know from Property 1 that $\widehat{y}$ is a faithful rounding of $x^{-1/2}$. We easily find that $\widehat{r} \in \left[\frac{1}{4} + \frac{u}{2}, 1\right]$ and

[2]We will see that this is not needed in binary32 and binary64 arithmetics.

$\widehat{y} \in \left[\frac{1}{2}, 1\right]$. Since the range we have assumed for x implies $x^{-1/2} \in \left(\frac{1}{2}, 1\right]$, when $\widehat{y} = 1/2$, $x^{-1/2}$ is larger than $\widehat{y}$, and when $\widehat{y} = 1$, $x^{-1/2}$ is less than or equal to $\widehat{y}$. From all this, we deduce that $\mathrm{RN}(x^{-1/2})$ is either $\widehat{y}$ or $\widehat{y} + su$, with

$$s = \begin{cases} 1 & \text{if } \widehat{y} \le x^{-1/2}, \text{ that is, } 1 - x\widehat{y}^2 \ge 0, \\ -1 & \text{otherwise.} \end{cases}$$

Therefore, to find which value to return, we need

1) to compute $s = \mathrm{sign}(1 - x\widehat{y}^2) \in \{-1, 1\}$;
2) to determine whether $x^{-1/2}$ is smaller or larger than $\widehat{y} + \frac{su}{2}$. (Equality is impossible, since in base two $x^{-1/2}$ cannot be a midpoint, as shown in [6].)

From section II-B and the proof of Theorem 1, $\epsilon = 1 - x\widehat{y}^2$, $\widehat{\epsilon} = \mathrm{RN}(\epsilon)$, and $|\widehat{\epsilon} - \epsilon| \le 2u^2$. Since x is a multiple of $2u$ and $\widehat{y}$ is a multiple of u, $1 - x\widehat{y}^2$ is a multiple of $2u^3$. Thus, if $1 - x\widehat{y}^2 \ne 0$ then $|1 - x\widehat{y}^2| \ge 2u^3$ and $\widehat{\epsilon}$ is nonzero and of the same sign as soon as $2u^3 > 2^{e_{\min} - p}$ (which holds since $e_{\min} \le -2p - 2$); if $1 - x\widehat{y}^2 = 0$ then $\widehat{\epsilon} = 0$ and $\widehat{y}$ is exactly $x^{-1/2}$. Hence we return $\widehat{y}$ if $\widehat{\epsilon} = 0$, and we set $s = \mathrm{sign}(\widehat{\epsilon})$ otherwise.

Now, to decide whether to return $\widehat{y}$ or $\widehat{y} + su$, we must compute the sign of $(\widehat{y} + su/2) - x^{-1/2}$, that is, the sign of

$$\begin{aligned} x\left(\widehat{y} + su/2\right)^2 - 1 &= x\widehat{y}^2 + x\widehat{y}su + xu^2/4 - 1 \\ &= s\Big(xu \cdot \widehat{y} + xsu^2/4 - |\sigma + x\tau|\Big). \end{aligned} \quad (13)$$

For $e_{\min} \le -2p - 2$ and $s = \pm 1$ and $u = 2^{-p}$, we see that both xu and $xsu^2/4$ are in $\mathbb{F}$ and so the number

$$\widehat{\eta} := \mathrm{RN}\Big(xu \cdot \widehat{y} + xsu^2/4\Big)$$

can be obtained with one FMA after two (exact) multiplications of x by powers of 2. Furthermore, $\mathrm{RN}(|\sigma + x\tau|) = |\widehat{\epsilon}|$.

If $\widehat{\eta}$ and $|\widehat{\epsilon}|$ differ, then comparing them suffices for comparing the exact values $xu \cdot \widehat{y} + xsu^2/4$ and $|\sigma + x\tau|$.

If $\widehat{\eta} = |\widehat{\epsilon}|$, to compare these exact values we use Algorithm ErrFMA. The comparison is simplified by the fact that $|\widehat{\epsilon} - \epsilon| \in \mathbb{F}$ (as a multiple of $2u^3$ at most $2u^2$), which means that when we call $(v_\epsilon, w_\epsilon) := \mathrm{ErrFMA}\,(sx, \tau, s\sigma, \widehat{\eta})$ we have $w_\epsilon = 0$ (since $v_\epsilon = \mathrm{RN}(v_\epsilon + w_\epsilon)$ by definition of Fast2Sum). ■

The approach taken here thus starts from a faithful approximation and subsequently adjusts it thanks to a correction defined by a suitable case analysis. Note that this way of achieving correct rounding appears in other contexts as well, such as FMA-based correctly-rounded decimal FP division (see for example Algorithm 1 in [16]).

Note also that condition $e_{\min} \le -2p - 2$ of Theorem 3 holds for the binary32, binary64, and binary128 formats of the IEEE 754 Standard as well as for the Bfloat16 format [17], [18]. It is, however, not satisfied for the IEEE 754 binary16 format.

We must stress out that the case $\widehat{\eta} = |\widehat{\epsilon}|$, where we need to use Algorithm ErrFMA, almost never happens. Exhaustive tests performed for all input values x between 1 and 4, and for small values of p (from 6 to 24) show that:

- that case never occurs for $p = 7$, 9, 10, 11, 12, 13, 14, 17, 18, 21, 22, 23;
- it occurs for one input value only for $p = 8$, 16, 19, 24;
- it occurs for 2 input values only for $p = 6$, 20;
- it occurs for 4 input values only for $p = 15$.

When $\widehat{\eta} = |\widehat{\epsilon}|$, this means that $x^{-1/2}$ is extremely close to a midpoint. More precisely, we have the following property.

Property 2. *Let $x \in \mathbb{F} \cap [1, 4)$. If, when running Algorithm Generic_CR_rsqrt with input value x we have $\widehat{\eta} = |\widehat{\epsilon}|$, then the distance between $x^{-1/2}$ and its nearest midpoint is less than*

$$\frac{4u^2}{2 - \frac{3u}{2}} \approx 2u \cdot \mathrm{ulp}(x^{-1/2}).$$

Proof: If $\widehat{\eta} = |\widehat{\epsilon}|$ then $xu \cdot \widehat{y} + xsu^2/4$ and $|1 - x\widehat{y}^2|$ are positive reals less than $4u$ that round to the same floating-point number, which means that the absolute value of their difference is at most $4u^2$. Hence, recalling (13),

$$\left|x(\widehat{y} + su/2)^2 - 1\right| \le 4u^2$$

and, therefore,

$$\left|\widehat{y} + su/2 - x^{-1/2}\right| \le \frac{4u^2}{x\left(\widehat{y} + su/2\right) + \sqrt{x}}.$$

Since $x^{-1/2} \in (1/2, 1]$, the faithfulness of $\widehat{y}$ implies that $|\widehat{y} - x^{-1/2}| \le u$ and thus, in particular, $\widehat{y} \ge x^{-1/2} - u \ge u/2$. Together with $su/2 \ge -u/2$ this leads to

$$\left|\widehat{y} + su/2 - x^{-1/2}\right| \le \frac{4u^2}{2\sqrt{x} - 3ux/2} =: \varphi(x).$$

The conclusion follows from $\varphi(x) \le \varphi(1)$ for $x \in [1, 4)$. ■

2) Simplifications for binary32 and binary64 arithmetics: For the two most common cases (binary32 and binary64 arithmetics), one can simplify Algorithm Generic_CR_rsqrt, so that the use of Algorithm ErrFMA becomes unnecessary:

- In binary32 arithmetic, the only value of x in $\mathbb{F} \cap [1, 4)$ for which the case $\widehat{\eta} = |\widehat{\epsilon}|$ occurs is $x = 12196067 \cdot 2^{-22}$. For this value, one easily checks that $\widehat{y} = \mathrm{RN}(x^{-1/2})$. Hence we can replace Lines 10 to 20 of the algorithm by
  if $\widehat{\eta} \ge |\widehat{\epsilon}|$ **then** $\widehat{y}_{\mathrm{CR}} := \widehat{y}$ **else** $\widehat{y}_{\mathrm{CR}} := \widehat{y} + su$
- In binary64 arithmetic, Property 2 implies that it suffices to check Algorithm Generic_CR_rsqrt with $x \in \{4x_1, \ldots, 4x_6\}$ with x_k as in Table II. (We cannot have $\widehat{\eta} = |\widehat{\epsilon}|$ for other values of x in $\mathbb{F} \cap [1, 4)$.) Among these six cases, the only one for which $\widehat{\eta} = |\widehat{\epsilon}|$ is the first one ($x = 4x_1 = 3717785442934375 \cdot 2^{-51}$). For that case, $\mathrm{RN}(x^{-1/2})$ is equal to $\widehat{y} + su$. Hence we can replace Lines 10 to 20 of the algorithm by
  if $\widehat{\eta} > |\widehat{\epsilon}|$ **then** $\widehat{y}_{\mathrm{CR}} := \widehat{y}$ **else** $\widehat{y}_{\mathrm{CR}} := \widehat{y} + su$
  (Note the slight difference with the binary32 case.)

B. General case: x is any positive FP number

Now, for "general" input data, one may use the following two functions specified by the IEEE 754 Standard [1, p. 32]:

- **scaleB**(x, k), which returns (in a binary format, which is the case considered in this paper) $x \cdot 2^k$ for $x \in \mathbb{F}$ and k an integer;

- **logB**(x), which returns (in a binary format) $\lfloor \log_2 |x| \rfloor$ for $x \in \mathbb{F}$.

In the C programming language, these functions are called **scalebn** and **logb** (resp. **scalebnf** and **logbf**) for binary64/double precision operands (resp. binary32/single precision) operands. If $x > 0$ is the input value, we first find the integer k such that $2^{2k} \leq x < 2^{2k+2}$. This can be done by first calling $\ell = \textbf{logB}(x)$ and then defining $k = \lfloor \ell/2 \rfloor$.

Then it suffices to call Algorithm Generic_CR_rsqrt with the input value $x \cdot 2^{-2k} \in [1, 4)$ (obtained by one call to **scaleB**$(x, -2k)$), and to multiply the final result $\widehat{y}_{\text{CR}}$ by 2^k (which too can be done using the **scaleB** function).

Of course, if the available implementations of **logB** and **scaleB** are not efficient enough, one can easily replace these functions using masks and integer operations, but at the risk of obtaining less portable software.

VI. Performance Testing

We have used the *BenchmarkTools* package in Julia to perform a rudimentary analysis of the relative speed of the algorithms presented in this paper. All testing is done in IEEE 754 double precision arithmetic with code written in Julia 1.6.1 running on an Intel(R) Core(TM) i7-7700K CPU @ 4.20GHz, using a call of the form `@benchmark AlgorithmX((3 * rand(1000). + 1.0))`. This specific call structure guarantees that the inputs are in the range $[1, 4)$ so that no range management is necessary for Algorithm Generic_CR_rsqrt. For that algorithm, we have used the simplification introduced in Section V-A2. The mean execution times are summarized in Table IV. Note that these are the costs for applying the operation element by element to a 1000-element vector of random Float64 values. The results are roughly what we might expect. Algorithm Generic_CR_rsqrt appears to run roughly twice as long as either rsqrtNewton or rsqrtHalley. This additional cost is dominated by the cost of branching and conditionals required to guarantee correct rounding. In a fully IEEE-754-compliant implementation, where one has to insert conditionals anyway to check for exceptional input values (infinities, zeros, NaNs), this additional cost will largely vanish.

TABLE IV
Performance Testing Using Julia's BenchmarkTools, for a 1000-element vector.

Algorithm	Mean execution time (μs)
rsqrtNewton	2.9
rsqrtHalley	3.0
Generic_CR_rsqrt	5.8

VII. Conclusion

We have analyzed thoroughly two recent generic and highly-accurate algorithms for computing reciprocal square roots (Algorithms rsqrtNewton and rsqrtHalley from [2], [3]). For binary32 and binary64 arithmetics, we have also determined the (very few) input values for which they do not return the correctly-rounded result $\text{RN}(x^{-1/2})$; for such formats, this makes it possible to transform these algorithms into ones that always return $\text{RN}(x^{-1/2})$ by testing if x is one of these particular input values. Finally, we have introduced a third generic algorithm that has the advantage of always returning a correctly-rounded result, regardless of the format (Algorithm Generic_CR_rsqrt). Our first experiments on a restricted input range suggest that the overhead is reasonable and should become negligible within fully IEEE-754-compliant implementations. Thus, our results pave the way for a more general study, needed for such compliant implementations and which will cover in detail both performance and robustness aspects.

Acknowledgment

We thank Vincent Lefèvre for his help with the computation and verification of the values x_k in Table II.

References

[1] IEEE Computer Society, *IEEE Standard for Floating-Point Arithmetic, IEEE Standard 754-2019*. New York: IEEE Computer Society, 2019.
[2] C. F. Borges, "Fast compensated algorithms for the reciprocal square root, the reciprocal hypotenuse, and Givens rotations," Jun. 2021, arXiv report 2103.08694. [Online]. Available: https://arxiv.org/abs/2103.08694
[3] ——, "A correctly rounded Newton step for the reciprocal square root," Dec. 2021, arXiv report 2112.14321. [Online]. Available: https://arxiv.org/abs/2112.14321
[4] P. Markstein, *IA-64 and Elementary Functions: Speed and Precision*, ser. Hewlett-Packard Professional Books. Prentice Hall, 2000.
[5] W. Kahan, "Square root without division," Feb. 1999. [Online]. Available: http://people.eecs.berkeley.edu/~wkahan/ieee754status/reciprt.pdf
[6] C. Iordache and D. W. Matula, "On infinitely precise rounding for division, square root, reciprocal and square root reciprocal," in *14th IEEE Symposium on Computer Arithmetic*, 1999.
[7] V. Lefèvre, "Moyens arithmétiques pour un calcul fiable," Ph.D. dissertation, École Normale Supérieure de Lyon, Lyon, France, 2000.
[8] ——, "New results on the distance between a segment and $\mathbb{Z}^2$. Application to the exact rounding," in *17th IEEE Symposium on Computer Arithmetic*, 2005.
[9] D. E. Knuth, *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*, 3rd ed. Reading, MA, USA: Addison-Wesley, 1998.
[10] C.-P. Jeannerod and S. M. Rump, "On relative errors of floating-point operations: optimal bounds and applications," *Math. Comp.*, vol. 87, no. 310, pp. 803–819, 2018.
[11] M. Pichat, "Contribution à l'étude des erreurs d'arrondi en arithmétique à virgule flottante," Ph.D. dissertation, Université Scientifique et Médicale de Grenoble & Institut National Polytechnique de Grenoble, Grenoble, France, 1976. [Online]. Available: https://tel.archives-ouvertes.fr/tel-00287209/
[12] G. Bohlender, W. Walter, P. Kornerup, and D. W. Matula, "Semantics for exact floating point operations," in *10th IEEE Symposium on Computer Arithmetic*, 1991.
[13] S. Boldo and J.-M. Muller, "Exact and approximated error of the FMA," *IEEE Trans. Comput.*, vol. 60, no. 2, pp. 157–164, 2011.
[14] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres, *Handbook of Floating-Point Arithmetic*. Birkhäuser Boston, 2018.
[15] M. Iştoan and B. Pasca, "Flexible fixed-point function generation for FPGAs," in *24th IEEE Symposium on Computer Arithmetic*, 2017.
[16] N. Louvet, J.-M. Muller, and A. Panhaleux, "Newton–Raphson algorithms for floating-point division using an FMA," in *Proc. 21st IEEE International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2010, pp. 200–107.
[17] Intel Corporation, "BFLOAT16 – Hardware Numerics Definition," November 2018, White paper. Document number 338302-001US. [Online]. Available: https://www.intel.com/content/dam/develop/external/us/en/documents/bf16-hardware-numerics-definition-white-paper.pdf
[18] G. Henry, P. T. P. Tang, and A. Heinecke, "Leveraging the bfloat16 artificial intelligence datatype for higher-precision computations," in *26th IEEE Symposium on Computer Arithmetic*, 2019.