A Parametric Error Analysis of Goldschmidt's Square-Root Algorithm

Peter-Michael Seidel
University of Hawai'i at Manoa
Deptartment of Information and Computer Sciences
Honolulu, HI, 96822
seidel@acm.org

***Abstract*—Back in the 1960s Goldschmidt presented a variation of Newton-Raphson iterations that are the basis for a division and a square root algorithm that are well suited for pipelining. The problem in using Goldschmidt's algorithms is to present an error analysis that enables one to save hardware by using just the right amount of precision for intermediate calculations while still providing correct rounding. Previous implementations relied on combining formal proof methods (that span thousands of lines) with millions of test vectors. These techniques yield correct designs but the analysis is hard to follow and is not quite tight. We have previously presented a simple parametric error analysis of Goldschmidt's division algorithm to allow for improved division implementations and parameter optimizations for the choice of the intermediate precisions. In this work we extend our analysis to Goldschmidt's square root algorithm. This analysis sheds more light on the effect of the different parameters on the error of the square root implementations. In addition, we derive error formulae that help determine optimal parameter choices in practical implementation settings.**

I. Introduction and Summary

Asymptotically optimal division algorithms are based on multiplicative division methods [1], [2], [3]. Square root operations can be implemented similarly with the similar asymptotic convergence. Several commercial processor designs are employing a parallel multiplier for performing multiplicative division and square root operations for the significands of floating-point numbers [4], [5], [6], [7]. In such implementations the parallel multiplier is typically used for additional operations, such as: multiplication and fused multiply-add. Since meeting the precision requirements of division and square root operations requires more precision than other operations, the dimensions of the parallel multiplier need to be determined by precision requirements of the division and square root operations. It follows that tighter analysis of the required multiplier dimensions for division and square root operations can lead to improvements in cost, delay, and even power consumption. The main two methods used for multiplicative division and square root are a variation of Newton's method [8], [9] and a method introduced by Goldschmidt [10] that is based on an approximation of a series expansion. The algorithms based on Newton's methods have a quadratic convergence rate (i.e., the number of accurate bits doubles in each iteration) and are self-correcting (i.e., inaccuracies of intermediate computations do not accumulate).

In this paper we focus on the square root algorithm. A detailed discussion of the division algorithm can be found in [11]. Each iteration of Newton's square root method involves three *dependent* multiplications; namely, the product of the first multiplication determines one of the operands of the second multiplication, and the product of the second multiplication determines one of the operands of the third multiplication.The implication of having to compute three dependent multiplications per iteration is that these multiplications cannot be parallelized or pipelined.

Goldschmidt's square root algorithm also requires three multiplications per iteration and the convergence rate is the same as for Newton's method. However, the most important feature of Goldschmidt's algorithm is that two of the multiplications per iteration are independent and can be pipelined or computed in parallel. On the other hand, Goldschmidt's algorithm is not self-correcting; namely, inaccuracies of intermediate computations accumulate and cause the computed result to drift away from the accurate square root. Goldschmidt's square root algorithm was used in the IBM System/360 model 91 [12], in the IBM S/390 [13] and in the AMD Bobcat and Jaguar microprocessors [14]. However, lack of a general and simple error analysis of Goldschmidt's division and square root algorithms has averted many designers from considering implementing Goldschmidt's algorithms. Thus more implementations of multiplicative methods have been based on Newton's algorithm in spite of the longer latency due to more dependent multiplications in each iteration. [5], [6]

Goldschmidt's methods are not self-correcting. This makes it particularly important and difficult to keep track of accumulated and propagated error terms during intermediate computations. We were not able to locate a general analysis of error bounds of Goldschmidt's square root algorithm in the literature. Goldschmidt's error analysis in [10] is very specific with respect to a design that uses a serial radix-4 Booth multiplier with 61-bits. Other implementations of Goldschmidt's algorithms also rely on an error analysis very specific to the implementation and that provides only rough bounds for the accumulated error [7]. Such overestimates lead to correct designs but waste hardware and cause unnecessary delay (since the multiplier and the initial lookup table are larger). These over-estimations were based on informal arguments that were

978-1-4673-8576-3/15/$31.00 ©2015 IEEE

confirmed by a mechanically verified proof that spans over 250 definitions and 3000 lemmas [17].

We present a version of Goldschmidt's square root algorithm that uses directed rounding implementations. We develop a simple general parametric analysis of tight error bounds for our version of Goldschmidt's square root algorithm. Our analysis is parametric in the sense that it allows arbitrary one-sided errors in each intermediate computation and it allows an arbitrary number of iterations. In addition, we suggest practical simplified settings in which errors in intermediate computations are not arbitrary.

The paper is organized as follows. In Section II we present Newton's method for inverse square-root and then proceed by presenting Goldschmidt's algorithm as a variation of Newton's method to compute the square root. In Section III, a version of Goldschmidt's algorithm with imprecise intermediate computations is presented as well as an error analysis. In Section IV we develop closed form error bounds for Goldschmidt's method with respect to two specific settings.

II. Goldschmidt's square-root algorithm

In this section we present Newton's method for computing the reciprocal square-root of a given number. We then continue by describing a version of Goldschmidt's algorithm for computing the reciprocal square-root and the square-root [10] that use precise intermediate computations. We show how Goldschmidt's algorithm is derived from Newton's method. The error analysis of Newton's method is used to analyze the errors in Goldschmidt's algorithm.

A. Newton's method.

Newton's method can be applied to compute the reciprocal square-root of a given number, which is the reciprocal of what we ultimately want to compute, but we can get $\sqrt{B}$ with a multiplication of the reciprocal square-root by B. To compute the reciprocal square-root of $B > 0$, apply Newton's method to the function $f(x) = B - 1/x^2$. Note that: (a) the root of $f(x)$ is $1/\sqrt{B}$, and (b) the function $f(x)$ has a derivative $f'(x) = 2 \cdot x^{-3}$ in the interval $(0, \infty)$. In particular, the derivative $f'(x)$ is positive in this range.

Newton iterations are defined by the following recurrence: Let x_0 denote an initial estimate $x_0 > 0$ and define x_{i+1} by

$$\begin{aligned} x_{i+1} &= x_i - \frac{f(x_i)}{f'(x_i)} \\ &= x_i - \frac{B - x_i^{-2}}{2 \cdot x_i^{-3}} \\ &= x_i - (B \cdot x_i^3 - x_i)/2 \\ &= x_i \cdot (3 - B \cdot x_i^2)/2. \end{aligned} \tag{1}$$

Consider the *relative error* term e_i defined by

$$\begin{aligned} e_i &\triangleq \frac{1/\sqrt{B} - x_i}{1/\sqrt{B}} \\ &= 1 - \sqrt{B} \cdot x_i. \end{aligned}$$

It follows that

$$\begin{aligned} e_{i+1} &= 1 - \sqrt{B} \cdot x_{i+1} \\ &= 1 - \sqrt{B} \cdot (x_i \cdot (3 - B \cdot x_i^2)/2) \\ &= 1 - 3/2 \cdot \sqrt{B} \cdot x_i + \sqrt{B} \cdot B \cdot x_i^2/2 \\ &= (1 + \frac{x_i \cdot \sqrt{B}}{2}) \cdot (1 - \sqrt{B} \cdot x_i)^2 \\ &= (3/2 - e_i/2) \cdot e_i^2 \\ &\leq 3/2 \cdot e_i^2 \qquad \text{for } e_i \leq 1. \end{aligned} \tag{2}$$

Equation 2 involves two observations:

1) We require positive $B, x_0 > 0$. For $i \geq 1$, the relative error e_i is non-negative, as required for the last line in equation 2.
2) Convergence of x_i to $1/\sqrt{B}$ is guaranteed provided that the absolute value of the initial relative error is less than 1. Considering the positive sign of the error, the convergence is *one-sided*.

The disadvantage of Newton's iterations, with respect to a pipelined multiplier, is that each iteration consists of 3 *dependent* multiplications: $\alpha_i = B \cdot x_i$ and $\beta_i = \alpha_i \cdot x_i$ and $\gamma_i = x_i \cdot (3 - \beta_i)/2$. Namely, the product γ_i cannot be computed before the product β_i is computed, and the product β_i cannot be computed before the product α_i is computed.

B. Goldschmidt's Algorithm

In this section we describe how Goldschmidt's algorithm can be derived from Newton's method. Here, our goal is to compute the square-root $\sqrt{B}$. Goldschmidt's algorithm uses three values N_i, D_i and F_i defined as follows:

$$\begin{aligned} N_i &\triangleq B \cdot x_i \\ D_i &\triangleq B \cdot x_i \cdot x_i \\ F_i &\triangleq (3 - D_i)/2. \end{aligned}$$

Consider Newton's iteration: $x_{i+1} = x_i \cdot (3 - B \cdot x_i^2)/2$. We may rewrite an iteration by

$$x_{i+1} = x_i \cdot F_i,$$

which when multiplied by B, becomes

$$\begin{aligned} N_{i+1} &= N_i \cdot F_i \quad \stackrel{i \to \infty}{\longrightarrow} \sqrt{B} \\ D_{i+1} &= D_i \cdot F_i \cdot F_i \quad \stackrel{i \to \infty}{\longrightarrow} 1. \end{aligned}$$

Since x_i converges to $1/\sqrt{B}$, it follows that N_i converges to $\sqrt{B}$ and D_i converges to 1.

Note that: (a) $N_i/B = x_i = \sqrt{D_i/B}$, for every i; (b) $N_i/\sqrt{D_i} = \sqrt{B}$; (c) N_i converges to $\sqrt{B}$ at the same rate that x_i converges to $1/\sqrt{B}$; and (d) Since the relative error e_i in Newton's algorithm is non-negative, for $i \geq 1$, it follows that $N_i \leq \sqrt{B}$, $D_i \leq 1$ and $F_i \geq 1$ for $i \geq 1$.

As in Newton's iterations, the algorithm converges if $|e_0| < 1$ and the relative error decreases almost quadratically. One could use a fixed initial approximation of the quotient. Usually a more accurate initial approximation of $1/\sqrt{B}$ is computed

Algorithm 1 Goldschmidt-Square-Root(B) - Goldschmidt's iterative algorithm for computing $\sqrt{B}$

Require: $|e_0| < 1$.
1: Initialize: $N_{-1} \leftarrow B$, $D_{-1} \leftarrow B$, $F_{-1} \leftarrow \frac{1-e_0}{\sqrt{B}}$.
2: **for** $i = 0$ to k **do**
3: $N_i \leftarrow N_{i-1} \cdot F_{i-1}$.
4: $D_i \leftarrow D_{i-1} \cdot F_{i-1} \cdot F_{i-1}$.
5: $F_i \leftarrow (3 - D_i)/2$.
6: **end for**
7: Return(N_k)

Algorithm 2 Goldschmidt-Approx-Sqrt(B) - Goldschmidt's square-root algorithm using approximate arithmetic

1: Initialize: $N'_{-1} \leftarrow B$, $D'_{-1} \leftarrow B$, $F'_{-1} \leftarrow \frac{1-e_0}{\sqrt{B}}$.
2: **for** $i = 0$ to k **do**
3: $N'_i \leftarrow (1 - n_i) \cdot N'_{i-1} \cdot F'_{i-1}$.
4: $T'_i \leftarrow (1 + t_i) \cdot D'_{i-1} \cdot F'_{i-1}$.
5: $D'_i \leftarrow (1 + d_i) \cdot T'_{i-1} \cdot F'_{i-1}$.
6: $F'_i \leftarrow (1 - f_i) \cdot (3 - D'_i)/2$.
7: **end for**
8: Return(N'_k)

by a lookup table or a more elaborate functional unit (c.f. [18]).

Algorithm 1 lists Goldschmidt's square-root algorithm. Given B, the algorithm computes the square-root $\sqrt{B}$. The listing uses the same notation used above, and iterates k times. Observe that the multiplication in line 3 of the algorithm is independent from the multiplications in line 4, and therefore, Goldschmidt's square-root algorithm is more amenable to pipelined implementations. The computations for k iterations involve $3k + 1$ multiplications, of which only $2k + 1$ form a chain of dependent multiplications.

An error analysis of Goldschmidt's algorithm with precise arithmetic is based on the following claim.

Theorem 1: The following relative errors are equal $i \geq 0$:

$$e_i = \frac{1/\sqrt{B} - x_i}{1/\sqrt{B}} \tag{3}$$

$$= \frac{\sqrt{B} - N_i}{\sqrt{B}} \tag{4}$$

and bound by

$$0 \leq e_i \leq (3/2)^{2^i - 1} \cdot e_0^{2^i}. \tag{5}$$

Precise computations of the algorithm would require increasingly larger multipliers for each iteration. To keep multiplier dimensions feasible, in practice, intermediate products need to be truncated. This introduces additional error terms in the approximations. The key difficulty in analyzing the error in imprecise implementations of Goldschmidt's algorithm is due to the violation of the invariant $N_i/\sqrt{D_i} = \sqrt{B}$, which might make the N_i drift away from their convergence to $\sqrt{B}$ without any self-correction.

III. Imprecise Intermediate Computations

In this section we present a version of Goldschmidt's algorithm with imprecise intermediate computations. In this algorithm the invariant $N_i/\sqrt{D_i} = \sqrt{B}$ of Goldschmidt's algorithm with precise arithmetic does not hold anymore. We then develop a simple parametric analysis for error bounds in this algorithm. The error analysis is based on relative errors of intermediate computations. The setting is quite general and allows for different relative errors for each computation.

The analysis follows the general steps of the analysis for Goldschmidt's division algorithm from [11]. We also use directed rounding for the computation and assume that the roundings meet the *strict directed* rounding property as defined in [11]. Strict directed rounding intuitively ensures that a rounding argument smaller than 1 is not rounded to a value larger than one and vice versa.

The formulation of the algorithm begins by using the exact values of all the relative errors. The values of the relative errors depend on the actual values of the inputs and on the hardware used for the intermediate computations. However, one can usually easily derive upper bounds on the absolute errors of each intermediate computation. For example, such bounds on the absolute errors are simply derived from the precision of the multipliers. Our analysis continues by translating the upper bounds on the absolute errors to upper bounds on the relative errors. Hence we are able to analyze the accuracy of an implementation of the proposed algorithm based on upper bounds on the absolute errors.

An interesting feature of our analysis is that directed roundings are used for all intermediate calculations. Surprisingly, directed roundings play a crucial role in this analysis and enable a simpler and tighter error analysis than round-to-nearest rounding (c.f. [7]).

A. Goldschmidt's square-root algorithm using approximate arithmetic

A listing of Goldschmidt's square-root algorithm using approximate arithmetic appears in Algorithm 2. The equation for D_i is split into two equations introducing $T_i = D_{i-1} \cdot F_{i-1}$ and calculating D_i from this intermediate value T_i as $D_i = T_i \cdot F_{i-1}$. The values corresponding to N_i, D_i, T_i and F_i using the imprecise computations are then denoted by N'_i, D'_i, T'_i and F'_i, respectively.

Directed roundings are used for the intermediate calculations as follows. For example, N'_i is obtained by rounding down the product $N'_{i-1} \cdot F'_{i-1}$. We denote by n_i the relative error of N'_i with respect to $N'_{i-1} \cdot F'_{i-1}$. Since $N'_{i-1} \cdot F'_{i-1}$ is rounded down, we assume that $n_i \geq 0$. Similarly, rounding down is used for computing F'_i (with the relative error f_i) and rounding up is used for computing T'_i and D'_i (with the relative errors t_i and d_i, respectively).

The initial approximation of the reciprocal square-root $1/\sqrt{B}$ is denoted by F'_{-1}. The relative error of F'_{-1} with respect to $1/\sqrt{B}$ is denoted by e_0. To simplify the discussion,

we assume $e_0 \geq 0$, so that this iteration shares the sign of the error with the other iterations.

Our error analysis is based on the following assumptions: (i) The operand is in the range $B \in [1, 4)$. (ii) We require that $0 \leq e_0 < 1$. (iii) The initial approximation F'_{-1} of $1/\sqrt{B}$ is in the range $[1/2, 1]$.

Observe that, in general, rounding down means that $rnd(x) \leq x$, for all x. Often the absolute error introduced by rounding is bounded by $\varepsilon > 0$, namely $x - \varepsilon \leq rnd(x) \leq x$. Strict rounding down requires that if $x \geq 1$, then $rnd(x) \geq 1$ no matter how close x is to 1. In non-redundant binary representation strict rounding down is easily implemented by truncation.

B. Parametric Error Analysis

Lemma 2 bounds the ranges of N'_i and F'_i in Algorithm 2 under the strict rounding assumption. This lemma is an extension of the properties $D_i \leq 1$ and $F_i \geq 1$ (for $i \geq 1$) of Algorithm 1.

Let $\Delta_{i-1} \triangleq 1 - \sqrt{D'_{i-1}}$. The following equation follows from the definition of F'_{i-1} and the error analysis for Newton's algorithm.

$$D'_{i-1} \cdot F'_{i-1} \cdot F'_{i-1} \;=\; (1 - f_{i-1})^2 \cdot (1 - \frac{3 - \Delta_{i-1}}{2} \cdot \Delta^2_{i-1})^2. \quad (6)$$

Goldschmidt already pointed out that since F_i tends to 1 from above, one could save hardware since the binary representation of F_i begins with the string $1.000\ldots$. An analogous remark holds for D_i. However, Lemma 2 refers to the inaccurate intermediate results (i.e., D'_i and F'_i) rather than the precise intermediate results (i.e., D_i and F_i). Parts 1-2 of lemma 2 show that the same hardware reduction strategy applies to Algorithm 2, even though intermediate calculations are imprecise.

Definition 1: Define δ_i, for $i \geq 0$, as follows:

$$\delta_i \;:=\; \begin{cases} e_0 & \text{for } i = 0 \\ 3/2 \cdot \delta^2_{i-1} + f_{i-1} & \text{otherwise.} \end{cases}$$

Lemma 2: (ranges of D'_i and F'_i)
The following bounds hold:

1) $D'_i \in [(1 - \delta_i)^2, 1]$, for every $i \geq 0$.
2) $F'_i \in [1, 1 + \delta_i]$, for every $i \geq 0$.
3) $D'_i \leq D'_{i+1}$, for every $i \geq 0$.

Proof: Part (1), case $i = 0$: Since $D'_0 = (1 + d_0) \cdot (1 + t_0) \cdot D'_{-1} \cdot F'_{-1} \cdot F'_{-1}$, it follows that $D'_0 = (1 + d_0) \cdot (1 + t_0) \cdot (1 - e_0)^2 = (1 - e_0)^2 + (d_0 + t_0 + d_0 \cdot t_0) \cdot (1 - e_0)^2$. Since $d_0, t_0, e_0 > 0$, and since SD rounding is used, $0 \leq \delta_0 \leq e_0$.

Part (1), case $i > 0$: We prove that if $D_{i-1} \in [(1-\delta_{i-1})^2, 1]$, then $D_i \in [(1 - \delta_i)^2, 1]$. It then follows by induction, that D_i is in the required range, for every $i \geq 0$.

The assumption $D_{i-1} \in [(1 - \delta_{i-1})^2, 1]$ implies that $|\Delta_{i-1}| \leq \delta_{i-1}$. By Eq. 6 it follows that

$$D'_i = (1 + d_i)(1 + t_0)(1 - f_{i-1})^2 \cdot (1 - \frac{3 - \Delta_{i-1}}{2} \cdot \Delta^2_{i-1})^2.$$

Since $(1 - f_{i-1})^2 \cdot (1 - \frac{3-\Delta_{i-1}}{2} \cdot \Delta^2_{i-1})^2 \leq 1$, strict rounding up implies that $D'_i \leq 1$. On the other hand,

$$\begin{aligned} D'_i &= (1 + d_i)(1 + t_0)(1 - f_{i-1})^2 (1 - \frac{3 - \Delta_{i-1}}{2} \cdot \Delta^2_{i-1})^2 \\ &\geq (1 - f_{i-1})^2 \cdot (1 - 3/2 \cdot \Delta^2_{i-1})^2 \\ &\geq (1 - 3/2\delta^2_{i-1} - f_{i-1})^2 \\ &= (1 - \delta_i)^2. \end{aligned}$$

Hence, $D'_i \in [(1 - \delta_i)^2, 1]$, and Part (1) follows.

Part (3): By Part (2), $(1 - \delta_i)^2 \leq D'_i \leq 1$, hence $1 \leq (3 - D'_i)/2 \leq 1 + \delta_i(1 - \delta_i/2)$. Recall that F'_i is obtained by strict rounding down of $(3 - D'_i)/2$, hence $1 \leq F'_i \leq 1 + \delta_i$.

Part (4): From Part (3) it follows that $F'_i \geq 1$. Since D'_{i+1} is obtained by rounding up $D'_i \cdot F'_i \cdot F'_i$ (in T_{i+1} and in D_{i+1}), it follows that $D'_{i+1} \geq D'_i$, as required. □

The following claim summarizes the relative error of Goldschmidt's division algorithm using approximate arithmetic.

Theorem 3: For every $i > 0$, the relative error $\rho(N'_i) = \frac{\sqrt{B} - N'_i}{\sqrt{B}}$ satisfies

$$\pi_i \quad \leq \quad \rho(N'_i) \quad \leq \quad \pi_i + \delta_i, \quad (7)$$

where $\pi_i \triangleq 1 - (1 - n_i) \cdot \prod_{j=0}^{i-1} \frac{1 - n_j}{\sqrt{(1+d_j)(1+t_j)}} \;\geq 0$.

Proof: We decompose $\rho(N'_i)$ as follows.

$$\rho(N'_i) = \frac{\sqrt{B} - \frac{N'_i}{\sqrt{D'_{i-1} F'^2_{i-1}}}}{\sqrt{B}} + \frac{\frac{N'_i}{\sqrt{D'_{i-1} F'^2_{i-1}}} - N'_i}{\sqrt{B}}. \quad (8)$$

We first show that the first term on the right hand side equals π_i. Observe that $\frac{N'_i}{\sqrt{D'_i}}$ can be expanded as follows:

$$\begin{aligned} \frac{N'_i}{\sqrt{D'_i}} &= \frac{N'_{i-1}}{\sqrt{D'_{i-1}}} \cdot \frac{1 - n_i}{\sqrt{(1 + d_i)(1 + t_i)}} \\ &= \sqrt{B} \cdot \prod_{j=0}^{i} \frac{1 - n_j}{\sqrt{(1 + d_i)(1 + t_i)}}. \end{aligned} \quad (9)$$

Equation 9 implies that $N'_i/\sqrt{D'_i}$ is non-increasing.

An expansion of $\frac{N'_i}{\sqrt{D'_{i-1} \cdot F'^2_{i-1}}}$ yields

$$\begin{aligned} \frac{N'_i}{\sqrt{D'_{i-1} \cdot F'^2_{i-1}}} &= \frac{(1 - n_i) N'_{i-1} \cdot F'_{i-1}}{\sqrt{D'_{i-1} \cdot F'^2_{i-1}}} = (1 - n_i) \cdot \frac{N'_{i-1}}{D'_{i-1}} \\ &= (1 - n_i) \cdot \sqrt{B} \cdot \prod_{j=0}^{i-1} \frac{1 - n_j}{\sqrt{(1 + d_j)(1 + t_j)}} \end{aligned}$$

Note that in particular $\frac{N'_i}{\sqrt{D'_{i-1} \cdot F'^2_{i-1}}} \leq \sqrt{B}$, hence $\rho(\frac{N'_i}{\sqrt{D'_{i-1} \cdot F'^2_{i-1}}})$ is non-negative. It follows that $\rho(\frac{N'_i}{\sqrt{D'_{i-1} \cdot F'^2_{i-1}}})$ satisfies

$$\begin{aligned} \rho(\frac{N'_i}{\sqrt{D'_{i-1} \cdot F'^2_{i-1}}}) &= 1 - (1 - n_i) \cdot \prod_{j=0}^{i-1} \frac{1 - n_j}{\sqrt{(1 + d_j)(1 + t_j)}} \\ &= \pi_i. \end{aligned}$$

We now prove that the second term on the right hand side of Eq. 8 is non-negative and bounded by δ_i. Consider the numerator

$$\begin{aligned}\frac{N'_i}{\sqrt{D'_{i-1}\cdot F'^2_{i-1}}} - N'_i &= \frac{N'_i}{\sqrt{D'_{i-1}\cdot F'^2_{i-1}}}\cdot(1-\sqrt{D'_{i-1}\cdot F'^2_{i-1}}) \\ &= (1-\pi_i)\cdot\sqrt{B}\cdot(1-\sqrt{D'_{i-1}\cdot F'^2_{i-1}}) \\ &\geq 0.\end{aligned}$$

To complete the proof we only need to show that $0 \leq 1 - \sqrt{D'_{i-1}\cdot F'^2_{i-1}} \leq \delta_i$. By Lemma 2, it follows that $D'_{i-1} \in [(1-\delta_{i-1})^2, 1]$ and $F'_{i-1} = (1-f_{i-1})\cdot(3-D'_{i-1})/2$. Hence

$$\begin{aligned}1-\delta_i \leq\ & (1-f_{i-1})\cdot(1-3/2\delta^2_{i-1}) \\ & \leq \sqrt{D'_{i-1}\cdot F'^2_{i-1}} \leq D'_{i-1}\cdot(3-D'_{i-1})^2/4 \quad \leq \quad 1,\end{aligned}$$

and the theorem follows. □

IV. Closed Form Error Bounds in Specific Settings

In this section we describe two settings of the relative errors that enable us to derive closed form error bounds. The advantage of having closed form error bounds is that such bounds simplify the task of minimizing an objective function (modeling cost or delay) subject to the required precision. Closed form error bounds also enable one to easily evaluate the effect of design choices (e.g., initial error, precision of intermediate computations, and number of iterations) on the final error.

A. Setting I: $n_i, d_i, t_i \leq \hat{n}$ and $f_i = 0$.

Setting I deals with the situation that all the relative errors n_i, d_i, t_i are bounded by the same value $\hat{n}$. In addition it is assumed in this setting that $f_i = 0$, for every i. The justification for Setting I is that if all internal operands are represented by binary strings of equal length, then it is possible to bound all the relative errors n_i, d_i by the same value. The relative errors f_i can be assumed to be 0, if the computations $F'_i = (3 - D'_i)/2$ are precise.

Using Theorem 3, the relative approximation error in Setting I can be bounded according to the following claim.

Claim 4:

$$0 \ \leq\ \rho(N'_i) = \frac{\sqrt{B} - N'_i}{\sqrt{B}} \ \leq\ (2i+1)\cdot\hat{n} + (3/2)^{2^i-1}\cdot(|e_0|)^{2^i}.$$

B. Setting II: $n_i, d_i \leq \hat{n}$ and f_i/δ_i^2 is constant.

In setting II it is assumed that $f_i/\delta_i^2 \leq C$, where C is an appropriately chosen constant which is independent of the number of iterations k. In addition, it is assumed that $n_i, d_i, t_i \leq \hat{n}$, for every i.

Claim 5:

$$\delta_k \quad \leq \quad (3/2\cdot(1+C))^{2^k-1}\cdot\delta_0^{2^k}.$$

Based on Theorem 3 the error bound in setting II satisfies:

$$\rho(N'_k) < (2k+1)\cdot\hat{n} + \left((3/2\cdot(1+C))^{2^k-1}\cdot e_0^{2^k}\right).$$

We are omitting the discussion of closed form error bounds for two additional settings for the space limitations in this manuscript.

References

[1] K. Mehlhorn and F. Preparata, "Area-time optimal division for $t = \omega((logn)^{1+\epsilon})$," *Information and Computation*, vol. 72, no. 3, pp. 270–282, 1987.

[2] J. Reif and S. Tate, "Optimal size integer division circuits," *SIAM Journal on Computing*, vol. 19, no. 5, pp. 912–924, Oct. 1990.

[3] N. Shankar and V. Ramachandran, "Efficient parallel circuits and algorithms for division," *Information Processing Letters*, vol. 29, no. 6, pp. 307–313, 1988.

[4] R. Agarwal, F. Gustavson, and M. Schmookler, "Series approximation methods for divide and square root in the power3 processor," in *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*, vol. 14. IEEE, 1999, pp. 116–123.

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

[6] P. Markstein, *Ia-64 and Elementary Functions : Speed and Precision*, ser. Hewlett-Packard Professional Books. Prentice Hall, 2000.

[7] S. F. Oberman, "Floating-point division and square root algorithms and implementation in the AMD-K7 microprocessor," in *Proceedings of the 14th IEEE Symposium on Computer Arithmetic (Adelaide, Australia)*, Koren and Kornerup, Eds. Los Alamitos, CA: IEEE Computer Society Press, April 1999, pp. 106–115.

[8] D. Ferrari, "A division method using a parallel multiplier," *IEEE Transactions on Computers*, vol. EC-16, pp. 224–226, Apr. 1967.

[9] M. J. Flynn, "On division by functional iteration," *IEEE Transactions on Computers*, vol. C-19, no. 8, pp. 702–706, Aug. 1970.

[10] R. Goldschmidt, "Applications of division by convergence," Master's thesis, MIT, June 1964.

[11] G. Even, P.-M. Seidel, and W. E. Ferguson, "A parametric error analysis of goldschmidt's division algorithm," *Journal of Computer and System Sciences*, vol. 70, no. 1, pp. 118–139, 2005.

[12] S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, "The IBM 360/370 model 91: floating-point execution unit," *IBM Journal of Research and Development*, Jan. 1967.

[13] E. M. Schwarz, L. Sigal, and T. McPherson, "CMOS floating point unit for the S/390 parallel enterpise server G4," *IBM Journal of Research and Development*, vol. 41, no. 4/5, pp. 475–488, July/Sept 1997.

[14] J. Rupley, J. King, E. Quinnell, F. Galloway, K. Patton, P. Seidel, J. Dinh, H. Bui, and A. Bhowmik, "The floating-point unit of the jaguar x86 core," in *Computer Arithmetic (ARITH), 2013 21st IEEE Symposium on*, April 2013, pp. 7–16.

[15] P. Soderquist and M. Leeser, "Area and performance tradeoffs in floating-point divide and square-root implementations," *ACM Computing Surveys*, vol. 28, no. 3, pp. 518–564, September 1996.

[16] E. V. Krishnamurthy, "On optimal iterative schemes for high-speed division," *IEEE Transactions on Computers*, vol. C-19, no. 3, pp. 227–231, Mar. 1970.

[17] D. Russinoff, "A mechanically checked proof of IEEE compliance of a register-transfer-level specification of the AMD-K7 floating-point multiplication, division, and square root instructions," *LMS Journal of Computation and Mathematics*, vol. 1, pp. 148–200, December 1998.

[18] M. Schulte and J. Stine, "Approximating elementary functions with symmetric bipartite tables," *IEEE Transactions on Computers*, vol. 48, no. 8, p. 842?847, 1999.

[19] W. Paul and P.-M. Seidel, "On the Complexity of Booth Recoding," *Proceedings of the 3rd Conference on Real Numbers and Computers(RNC3)*, pp. 199–218, 1998.

[20] S. M. Mueller and W. J. Paul, *Computer Architecture. Complexity and Correctness*. Springer, 2000.

[21] G. Bewick, "Fast multiplication: Algorithms and implementation," Ph.D. dissertation, Stanford University, March 1994.