On floating point approximation of the reciprocal cube root function

Cezary Walczyk[1][0000−0003−2147−7222], Pawel Gepner[2][0000−0003−0004−1729], Hatem Ltaief[3][0000−0002−6897−1095], and Nataliia Gavkalova[2][0000−0003−1208−9607]

[1] University of Bialystok, Bialystok, Poland
Faculty of Physics
c.walczyk@uwb.edu.pl
[2] Warsaw University of Technology, Warsaw, Poland
Faculty of Mechanical and Industrial Engineering
{pawel.gepner, nataliia.gavkalova}@pw.edu.pl
[3] King Abdullah University of Science and Technology, Thuwal, Saudi Arabia
Extreme Computing Research Center
hatem.ltaief@kaust.edu.sa

Abstract. The digital signal processing (DSP) of Internet of Things (IoT) devices using edge-based machine learning (ML) requires fast arithmetic operations and high energy efficiency. Hardware implementations of transcendental functions in ML and deep learning chips are impractical, necessitating effective software algorithms for functions like $1/\sqrt[3]{x}$ across precision levels.
This paper analyzes $1/\sqrt[3]{x}$ approximation, introducing algorithms that use magic numbers and piecewise linear approximation to optimize relative error, precision, and computation speed. A mathematical analysis determines key algorithm parameters, with single-precision implementations in C. These algorithms were tested on various hardware platforms and compared for speed, accuracy, and relative errors.

Keywords: computer arithmetic · approximation algorithm · reciprocal cube root functions · magic numbers.

1 Introduction

In digital signal processing (DSP) for Internet of Things (IoT) edge computing, balancing efficient code execution, silicon resources, and power consumption is critical. System architecture depends on application scenarios, resource availability, and whether operations are handled by hardware or software. The growing use of IoT for artificial intelligence (AI) has renewed interest in efficient methods to compute elementary functions like square roots, cube roots, and their reciprocals [1–3]. While hardware implementations exist for some functions [4], efficient software solutions remain necessary, particularly for infrequently used operations.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

Cube root and reciprocal cube root functions, while less common than square root operations, are essential in thermodynamics, computer graphics, and AI [5–8]. Modern CPUs compute these using functions like *cbrt* from the math.h library, but at a significant clock-cycle cost [9]. Although Intel-specific libraries like SVML [10] and IPP [11] offer fast implementations, support for other platforms such as microcontrollers and FPGAs is lacking, necessitating simple and efficient algorithms for embedded systems [5].

Various methods for computing *cbrt* and *rcbrt* exist [12–21], including powering algorithms [17], and Newton-Raphson (NR) or Householder iterations with initial approximations using magic constants [6, 7, 9, 19, 20]. Among these, magic constant-based methods show particular promise for fast single-precision calculations [9].

Building on this, we propose new algorithms for the reciprocal cube root function that improve accuracy and efficiency over existing methods. The paper is organized as follows: Section 2 introduces the theoretical basis, while Sections 3–5 describe approximation methods using linear functions and magic constants. Section 6 presents a modified NR method, and Section 7 explores further improvements using the Halley method. Experimental results across various hardware platforms are detailed in Section 8, with conclusions and future research directions in the final section.

2 A floating point approximation of the reciprocal cube root function with magic numbers

In this article, we deal with FP numbers in the following form:

$$x = (-1)^{s_x} 2^{e_x} \cdot (1 + m_x), \tag{1}$$

where $s_x \in \{0, 1\}$ denotes the sign of the number, $e_x = \lfloor \log_2 x \rfloor$ is the number's exponent and $m_x = x/2^{e_x} - 1$ is its mantissa. The IEEE-754 standard [22] defines two kinds of FP number representations – 32-bit and 64-bit:

- for s_x, one bit is assigned;
- 8 or 11 bits are assigned to the biased exponent $E_x = e_x + bias$, where $bias$ is equal to 127 or 1023;
- $p = 23$ or $p = 52$ bits are assigned to the mantissa fraction $m_x \in [0, 1)$.

Despite the fact that the theory in the present article refers to both representations, the practical testing of the accuracy and performance achieved by the implemented algorithms was conducted only for 32-bit precision.
The range of available 32-bit floating-point numbers for which we can determine reciprocal cube roots can be divided into disjoint intervals:

$$A_n = \left[2^{3n},\, 2^{3(n+1)}\right) \quad \text{where } n \in \{0,\, \pm 1, \pm 2, \dots\}\,.$$

If $x \in A_n$ then $y = 1/\sqrt[3]{x}$ takes values within the interval $\left(2^{-(n+1)},\, 2^{-n}\right]$. By introducing the new variables

$$\tilde{x} = 2^{-3n} x = 2^{e_{\tilde{x}}} \cdot (1 + m_{\tilde{x}}), \quad \tilde{y} = 2^n y = 2^{e_{\tilde{y}}} \cdot (1 + m_{\tilde{y}}) \tag{2}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

the whole problem may be restricted to computing the $\tilde{y} = 1/\sqrt[3]{\tilde{x}}$, for the $\tilde{x} \in [1, 8)$. Because $e_{\tilde{x}}$ can only take values from the set $\{0, 1, 2\}$, it follows from the equations in (2) that $e_{\tilde{y}} = -1$.
If we calculate the logarithm base 2 of $\tilde{y} = \tilde{x}^{1/3}$, taking into account the equations in (2), we obtain the formula

$$e_{\tilde{y}} + \log_2(1 + m_{\tilde{y}}) = -\frac{1}{3}e_{\tilde{x}} - \frac{1}{3}\log_2(1 + m_{\tilde{x}}). \tag{3}$$

By approximating the base 2 *log* function by a linear function $\log_2 a \simeq a + c_a$, where c_a is a certain undetermined constant, we obtain a linear dependency between $m_{\tilde{y}}$ and $m_{\tilde{x}}$:

$$1 + m_{\tilde{y}} + e_{\tilde{y}} \simeq -\frac{1}{3}e_{\tilde{x}} - \frac{1}{3}(1 + m_{\tilde{x}}) - \underbrace{\left(c_{1+m_{\tilde{y}}} + \frac{c_{1+m_{\tilde{x}}}}{3}\right)}_{-c}, \tag{4}$$

which, in equivalent form

$$\underbrace{2^p\left(E_{\tilde{y}} + m_{\tilde{y}}\right)}_{I_y} \simeq \underbrace{\lfloor 2^p\left(\frac{4}{3}bias - \frac{4}{3} + c\right)\rceil}_{R \leftarrow \text{magic constant}} - \underbrace{\frac{2^p(E_{\tilde{x}} + m_{\tilde{x}})}{3}}_{I_x/3}, \quad I_y,\, I_x,\, R \in \mathbb{N}^+. \tag{5}$$

This formula is analogous to the equation $I_y \simeq R - I_x/2$ from the fast algorithm for computing the reciprocal square root [23–27] and the equation $I_y \simeq R - I_x$ for reciprocal [28, 29]. Assuming from relation (4) that $e_{\tilde{x}}$ and $e_{\tilde{y}}$ are constant integer parameters, we may derive an approximate solution $\tilde{y}_{00} \simeq \tilde{y}$ that is linearly dependent on $\tilde{x}$:

$$\tilde{y}_{00}(\tilde{x}) = 2^{e_{\tilde{y}}}\left(-e_{\tilde{y}} - \frac{1}{3}e_{\tilde{x}} + c - \frac{1}{3}2^{-e_{\tilde{x}}}\tilde{x}\right). \tag{6}$$

Later in this work, we will search for a solution with the optimum relative error, which for the presented approximation scheme is described by the formula

$$\delta_{00}(\tilde{x}) = \sqrt[3]{\tilde{x}}\,y_{00}(\tilde{x}) - 1.$$

We expect that the optimisation will give us the value of the parameter c for which the approximation $\tilde{y}_{00}$, for all analysed argument values $\tilde{x}$, is characterized by the lowest possible difference between maximum and minimum of δ_{00}. However, such condition is not sufficient, in particular if maximal and minimal errors have the same sign. In order to avoid such situation, the second optimization stage involves a modification of the approximation method that makes the error symmetrical. The error function is considered to be symmetrical if the minimal error is negative and its absolute value is equal to the maximal error. In practice it is enough to introduce an additional parameter β, which rescales the previous approximation:

$$\tilde{y}_0(\tilde{x}) = \beta \cdot \tilde{y}_{00}(\tilde{x}), \quad \delta_0(\tilde{x}) = \sqrt[3]{\tilde{x}}\tilde{y}_0(\tilde{x}) - 1 = \beta \cdot \delta_{00}(\tilde{x}) + \beta - 1. \tag{7}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

We have shown that for exact solutions, $\tilde{x} \in [1, 8)$ and $e_{\tilde{y}} = -1$ hold, but in the case of approximation, the error may become so big that $e_{\tilde{y}} = -2$. Taking into account the above possibility, in the proposed model we allow $e_{\tilde{x}} = i \in \{0, 1, 2\}$, as well as $e_{\tilde{y}} = -j \in \{-1, -2\}$, and reformulate equations (7) with a new indexation:

$$\tilde{y}_{0,i,j}(\tilde{x}) = \beta \cdot 2^{-j} \left(j - \frac{1}{3} i + c - \frac{1}{3} 2^{-i} \tilde{x} \right), \quad \delta_{0,i,j}(\tilde{x}) = \sqrt[3]{\tilde{x}} \tilde{y}_{0,i,j}(\tilde{x}) - 1. \tag{8}$$

In computations, two cases must be considered:

– For the given value i, approximation in the whole range $\tilde{x} \in [2^i, 2^{i+1})$ corresponds to a concrete value j.The relative error function (7) always has one local maximum:

$$\delta_{0,i,j,\max} = \frac{\beta \, (3c + 3j - i)^{4/3}}{2^{2+j+(2-i)/3}} - 1, \quad \text{for } \; x = x_{i,j,\max} = \frac{3c + 3j - i}{2^{2-i}}. \tag{9}$$

Minimal values of the error function may appear only at border points (limits):

$$\delta_{0,i,j}(2^k) = \frac{\beta}{3} 2^{-j+k/3} (3c + 3j - k - 1) - 1 \,, \quad k \in \{i, \, i+1\} \tag{10}$$

– For a particular value i, depending on the value $\tilde{x}$, the optimal solution requires the application of formula (8), for $j = 1$ or $j = 2$:

$$\tilde{y}^{(t)}_{0,i}(\tilde{x}) = \begin{cases} \tilde{y}_{0,i,1}(\tilde{x}) & \text{for } x \in [2^i, t_i] \\ \tilde{y}_{0,i,2}(\tilde{x}) & \text{for } x \in [t_i, 2^{i+1}] \end{cases} \qquad \text{where } \; t_i = 2^i (3c - i), \tag{11}$$

is a solution of the equation $\tilde{y}_{0,i,1}(t_i) = \tilde{y}_{0,i,2}(t_i)$. The error function for such a solution,

$$\delta^{(t)}_{0,i}(\tilde{x}) = \sqrt[3]{\tilde{x}} \tilde{y}^{(t)}_{0,i}(\tilde{x}) - 1 \tag{12}$$

always has two local maxima

$$\delta^{(t)}_{0,i,1\,\max} = \delta_{0,i,1,\max} \,, \quad \delta^{(t)}_{0,i,2\,\max} = \delta_{0,i,2,\max} \,, \tag{13}$$

and the minimum for $x = t_i$

$$\delta_{0,i,1}(t_i) = \delta_{0,i,2}(t_i) = \frac{\sqrt[3]{t_i}}{6} \beta (3c + 3 - i - 2^{-i} t) - 1 \,. \tag{14}$$

3 Approximation with a single magic constant and four linear functions

Let us assume that for interpolation we may use a function of the type given in (8) or (11), but parameter c is identical for each $\tilde{x} \in [1, 8)$. The configuration $\tilde{y}^{(c,1)}_0(\tilde{x})$ optimizing the error function

$$\delta^{(c,1)}_0(\tilde{x}) = \sqrt[3]{\tilde{x}} \tilde{y}^{(c,1)}_0(\tilde{x}) - 1 \tag{15}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

is given by the formula

$$\tilde{y}_0^{(c,1)}(\tilde{x}) = \begin{cases} \tilde{y}_{0,i,1}(\tilde{x}) & \text{for } i \in \{0,1\} \\ \tilde{y}_{0,i}^{(t)}(\tilde{x}) & \text{for } i = 2 \end{cases} . \tag{16}$$

The conducted computations show that the optimal value of the error is equal to

$$\delta_{0,\max}^{(c,1)} = \frac{2}{1 + 8\sqrt[3]{5 - 2^{9/4}}(2^{1/4} - 1)} - 1 \simeq 0.0283928\,, \tag{17}$$

which corresponds to:

$$c = c_{(1)} = \frac{1 - 3^{-1}2^{5/4}}{2^{1/4} - 1}, \quad \beta = \beta_{(1)} = \frac{16 \cdot \sqrt[3]{2}(2^{1/4} - 1)^{4/3}}{1 + 8\sqrt[3]{5 - 4 \cdot 2^{1/4}}(2^{1/4} - 1)}\,. \tag{18}$$

The diagram of the relative error $\delta_0^{(c,1)}(\tilde{x})$ is shown in Figure 1. By substituting $c = c_{(1)}$ into equation (5), for single-precision FP numbers (bias = 127, $p = 23$), we derive the corresponding magic number: $R =$ 0x548c2b4b.

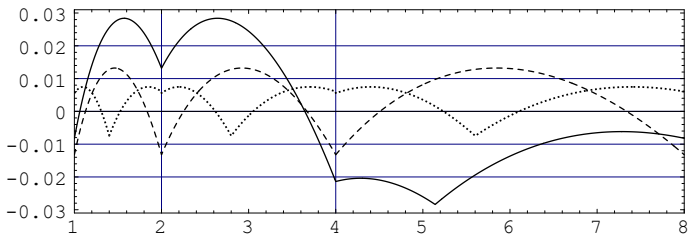

Fig. 1: Error functions $\delta_0^{(c,1)}(\tilde{x})$ (solid line), $\delta_{0,i}^{(c,3)}(\tilde{x})$ (dashed line) and $\delta_{0,i}^{(t,3)}(\tilde{x})$ (dotted line), where $i \in \{0,1,2\}$. For $\delta_{0,i}^{(c,3)}(\tilde{x})$ and $\delta_{0,i}^{(t,3)}(\tilde{x})$ $\tilde{x} \in [2^i,\, 2^{i+1})$.

4 Piecewise approximation with six linear functions and three magic constants

The error of the initial approximation may be significantly reduced if for each range $[2^i, 2^{i+1})$, where $i \in \{0,1,2\}$, the parameters c and β are computed independently. For the approximation using a function of the type given in (11) corresponding to the given i:

$$\tilde{y}_{0,i}^{(t,3)}(\tilde{x}) = \frac{\beta_i^{(t,3)}}{12} \cdot \begin{cases} 6 - 2i + 6c_i^{(t,3)} - 2^{1-i}\tilde{x} & \text{for } x \in [2^i, t_i^{(t,3)}] \\ 6 - i + 3c_i^{(t,3)} - 2^{-i}\tilde{x} & \text{for } x \in [t_i^{(t,3)}, 2^{i+1}] \end{cases} , \tag{19}$$

where $t_i^{(t,3)} = 2^i(3c_i^{(t,3)} - i)$.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

The optimal error function, is obtained for

$$c_i^{(t,3)} = \frac{2-2^{3/4}}{2^{3/4}-1} + \frac{i}{3} \quad \text{and} \quad \beta_i^{(t,3)} = \frac{16 \cdot 2^{(2-i)/3} 3^{-1/3} (2^{3/4}-1)^{4/3}}{4 \cdot 2^{2/3}(2-2^{3/4})^{1/3}(2^{3/4}-1)+3}\,. \tag{20}$$

and three magic constants:

$$R_0^{(t,3)} = 0\text{x}543\text{bbd}85\,, \quad R_1^{(t,3)} = 0\text{x}54666830\,, \quad R_2^{(t,3)} = 0\text{x}549112\text{db} \tag{21}$$

Maximal relative errors for $i \in \{0,1,2\}$, obtaining identical values

$$\delta_{0,\max}^{(t,3)} = \delta_{0,i,1,\max}^{(t,3)} = \frac{6}{4 \cdot 2^{2/3}(2-2^{3/4})^{1/3}(2^{3/4}-1)+3} - 1 \simeq 0.0075\,. \tag{22}$$

This means that the current approximation is better than $\tilde{y}_0^{(c,1)}(\tilde{x})$. Graphs of the function $\delta_{0,i}^{(t,3)}(\tilde{x})$ made with equation (12) is shown in Figure 1.

5 Piecewise approximation with $3N$ linear functions using $3N$ magic constants

In this section, we introduce the partitioning of each range $[2^i, 2^{i+1})$ into N equal subranges and assign a linear approximation function to each subrange. The conducted computations showed that for $N > 1$, approximation using two linear functions and one common magic constant is less accurate. The new model may be described as follows:

$$\tilde{y}_{0,i,n}^{(c,3\cdot N)}(\tilde{x}) = \frac{\beta_{i,n}^{(c,3\cdot N)}}{6} \cdot (3 - i + 3c_{i,n}^{(c,3\cdot N)} - 2^{-i}\tilde{x}) \quad \text{for } x \in [x_{b,i,n}, x_{e,i,n}]\,, \tag{23}$$

where:

$$\frac{x_{b,i,n}}{2^i} = 1 + \frac{n}{N}\,, \quad \frac{x_{e,i,n}}{2^i} = 1 + \frac{n+1}{N}\,, \; i \in \{0,1,2\}\,, \; n \in \{0,1,\ldots N-1\}\,. \tag{24}$$

The relative error function

$$\delta_{0,i,n}^{(c,3\cdot N)}(\tilde{x}) = \frac{\beta_{i,n}^{(c,3\cdot N)}}{6} \cdot \sqrt[3]{\tilde{x}}(3 - i + 3c_{i,n}^{(c,3\cdot N)} - 2^{-i}\tilde{x}) - 1\,, \tag{25}$$

has maxima in each nth subrange

$$\delta_{0,i,n,\max} = \beta\, 2^{-3-(2-i)/3}(3c_{i,n}^{(c,3\cdot N)} + 3 - i)^{4/3} - 1. \tag{26}$$

for x equal to

$$x_{i,n,\max} = 2^{-2+i}(3c_{i,n}^{(c,3\cdot N)} + 3 - i) \tag{27}$$

and minimal border values $\tilde{x}$, i.e. $x_{b,i,n}$ and $x_{e,i,n}$. After optimisation, we obtain: where

$$c_{i,n}^{(c,3N)} = -\frac{2-i}{3} + \frac{n}{3N} + \frac{1}{3N}\frac{\sqrt[3]{N+n+1}}{\sqrt[3]{N+n+1} - \sqrt[3]{N+n}}, \tag{28}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

and

$$\beta_{i,n}^{(c,3N)} = \frac{12 \cdot 2^{-i/3}}{3^{7/3} \cdot 2^{-8/3}(c_{0,n}^{(c,3N)} + 1)^{4/3} + \sqrt[3]{1 + \frac{n}{N}}\left(2 + 3c_{0,n}^{(c,3N)} - \frac{n}{N}\right)}\,. \tag{29}$$

The graph of the error function for $N = 1$ is shown in Figure 1. We again obtain the expression for the maximal error for the given subrange n, which does not depend on i:

$$\delta_{0,i,n,\max}^{(c,3\cdot N)} = \frac{3^{7/3} \cdot 2^{-5/3}(c_{0,n}^{(c,3\cdot N)} + 1)^{4/3}}{3^{7/3} \cdot 2^{-8/3}(c_{0,n}^{(c,3\cdot N)} + 1)^{4/3} + \sqrt[3]{1 + \frac{n}{N}}\left(2 + 3c_{0,n}^{(c,3\cdot N)} - \frac{n}{N}\right)} - 1\,. \tag{30}$$

The maximal relative error for $\tilde{x} \in [2^i, 2^{i+1})$ corresponds to $n = 0$:

$$\delta_{0,\max}^{(c,3\cdot N)} = \delta_{0,0,0,\max}^{(c,3\cdot N)}\,. \tag{31}$$

The values of the maximal relative errors $\delta_{0,\max}^{(c,3\cdot N)}$ and the corresponding accuracies $ac_{\text{bits}}^{(c,3\cdot N)} = \lfloor -\log_2 \delta_{0,\max}^{(c,3\cdot N)} \rfloor$ for the approximation $\tilde{y}_{0,i,n}^{(c,3\cdot N)}(\tilde{x})$ and $N \in \{1, 2, 4, 8\}$ are given in Table 1. Even for $N = 2$, the maximal error obtained is lower than $\delta_{0,\max}^{(t,3)}$.

Table 1: Maximal relative errors and their corresponding numbers of correct bits for the approximation $\tilde{y}_{0,i,n}^{(c,3\cdot N)}(\tilde{x})$.

N	1	2	4	8
$\delta_{0,\max}^{(c,3\cdot N)}$	$1.32 \cdot 10^{-2}$	$4.55 \cdot 10^{-3}$	$1.38 \cdot 10^{-3}$	$3.85 \cdot 10^{-4}$
$ac_{\text{bits}}^{(c,3\cdot N)}$	6	7	9	11

6 The modified Newton-Raphson method

In previous sections, we have proposed reciprocal cube root approximation algorithms that do not guarantee a sufficient accuracy for FP computations. An improvement can be made by using iterative methods. The most popular iterative method is the NR method; the NR method for computing the reciprocal cube root can be described by the following formula:

$$y_k^{(NR)} = \frac{1}{3} y_{k-1}^{(NR)}(4 - x \cdot {y_{k-1}^{(NR)}}^3), \tag{32}$$

which can be transformed into an equation

$$\delta_k^{(NR)}(x) = -\frac{1}{3}{\delta_{k-1}^{(NR)}}^2(x)(6 + 4\delta_{k-1}^{(NR)}(x) + {\delta_{k-1}^{(NR)}}^2(x)) \tag{33}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

that provides nonpositive relative errors. This means that its application is not optimal because the obtained error function is not symmetrical, i.e. its maximal and minimal values $\delta_{k,\max}$ and $\delta_{k,\min}$, respectively, do not satisfy the equation:

$$\delta_{k,\max} = -\delta_{k,\min}. \tag{34}$$

Equation (34) may be satisfied through the modification of equation (33), similarly to what was done in [25–27] in the fast algorithm for computing the reciprocal square root. The modification M1 is based on the introduction of additional coefficients $\kappa_{0,k}$ and $\kappa_{1,k}$ into the iterative formulas:

$$y_k^{(M1)} = \frac{1}{3} y_{k-1}^{(M1)} \kappa_{1,k} (4\kappa_{0,k} - x \cdot {y_{k-1}^{(M1)}}^3), \tag{35}$$

$$\delta_k^{(M1)} = \frac{1}{3} \kappa_{1,k} \left(1 + \delta_{k-1}^{(M1)}\right) \left(4\kappa_{0,k} - \left(1 + \delta_{k-1}^{(M1)}\right)^3\right) - 1, \tag{36}$$

where:

$$\kappa_{0,k} = 1 + {\delta_{k-1,\max}^{(M1)}}^2 \tag{37}$$

and

$$\kappa_{1,k} = \frac{6}{3\kappa_{0,k}^{4/3} + 4\kappa_{0,k} \left(1 + \delta_{k-1,\max}^{(M1)}\right) - \left(1 + \delta_{k-1,\max}^{(M1)}\right)^4}, \tag{38}$$

and the final expression for computing the maximal error of the correction, which depends exclusively on the maximal error of the initial approximation, is as follows:

$$\delta_{k,\max}^{(M1)} = \frac{6 \left(1 + {\delta_{k-1,\max}^{(M1)}}^2\right)^{4/3}}{3 + 3 \left(1 + {\delta_{k-1,\max}^{(M1)}}^2\right)^{4/3} - 2{\delta_{k-1,\max}^{(M1)}}^2 - {\delta_{k-1,\max}^{(M1)}}^4} - 1\,. \tag{39}$$

By applying the above formula to the approximations discussed in the previous sections, it is possible to estimate the accuracy of subsequent iterations and choose the best variant for the implementation for a particular FP number representation.

In Table 2, the maximal relative errors $\delta_{0,\max}^{(c,1)}$, $\delta_{0,\max}^{(c,3)}$, $\delta_{0,\max}^{(t,3)}$ and $\delta_{0,\max}^{(c,3\cdot 2)}$ of the approximations $\tilde{y}_0^{(c,1)}$, $\tilde{y}_0^{(c,3)}$, $\tilde{y}_0^{(t,3)}$ and $\tilde{y}_0^{(c,3\cdot 2)}$ are given in row 2. The corresponding numbers of correct bits $ac_{\text{bits},k}^{(*)} = \lfloor -\log_2 \delta_{k,\max}^{(*)} \rfloor$ for the same approximation schemes are given in row 3. For k iterations of type $M1$, the corresponding errors and numbers of correct bits are shown in rows 4–5 and 6–7.

According to the results in Table 2, for 32-bit numbers of the **float** type (23-bit mantissa), an adequate accuracy for the approximations $\tilde{y}_0^{(c,3)}$, $\tilde{y}_0^{(t,3)}$ and $\tilde{y}_0^{(c,3N)}$ (for $N = 2$) is obtained after applying two iterations of type $M1$.
The $\tilde{y}_0^{(t,3)}$ approximation is the most suitable for practical applications because it is more accurate than $\tilde{y}_0^{(c,3)}$ and has the same structure (an identical number

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

Table 2: The maximal relative errors of the initial approximations $\tilde{y}_0^{(c,1)}$, $\tilde{y}_0^{(c,3)}$, $\tilde{y}_0^{(t,3)}$, $\tilde{y}_0^{(c,3\cdot 2)}$ and the corresponding maximal relative errors $\delta_{k,\max}^{(M1)}$ for k iterations of type $M1$, along with the related accuracies (in bits).

$y_0^{(*)}$	$\tilde{y}_0^{(c,1)}$	$\tilde{y}_0^{(c,3)} \equiv \tilde{y}_0^{(c,3\cdot 1)}$	$\tilde{y}_0^{(t,3)}$	$\tilde{y}_0^{(c,3\cdot 2)}$
$\delta_{0,\max}^{(*)}$	$2.84 \cdot 10^{-2}$	$1.32 \cdot 10^{-2}$	$7.47 \cdot 10^{-3}$	$4.55 \cdot 10^{-3}$
$ac_{\text{bits},0}^{(*)}$	5	6	7	7
$\delta_{1,\max}^{(M1)}$	$8.06 \cdot 10^{-4}$	$1.75 \cdot 10^{-4}$	$5.57 \cdot 10^{-5}$	$2.07 \cdot 10^{-5}$
$ac_{\text{bits},1}^{(M1)}$	10	12	14	15
$\delta_{2,\max}^{(M1)}$	$6.50 \cdot 10^{-7}$	$3.05 \cdot 10^{-8}$	$3.11 \cdot 10^{-9}$	$4.29 \cdot 10^{-10}$
$ac_{\text{bits},2}^{(M1)}$	20	24	28	31

of constants that define the algorithm and its mathematical operations).
The third proposal has to be rejected due to its more complex structure. Moreover, obtaining a better accuracy for 32-bit precision is superfluous. By rewriting the iterative formula (35) with a reduced number of multiplications to obtain

$$y_k^{(M1)} = y_{k-1}^{(M1)}(k_{0,k} - k_{1,k} \cdot x \cdot {y_{k-1}^{(M1)}}^3), \tag{40}$$

where:

$$k_{0,k} = \frac{4}{3}\kappa_{0,k}\kappa_{1,k} \quad k_{1,k} = \frac{1}{3}\kappa_{1,k}\,, \tag{41}$$

for $y_0^{(M1)} = \tilde{y}_0^{(t,3)}$, we obtain a function whose code is presented in Algorithm 1.1.

Algorithm 1.1 *InvCbrtT3*

```
const float beta[3]={1.117795111f, 1.774389135f,
      1.408333590f};
const int r[3]={1418793691, 1413201285, 1415997488};
float InvCbrtT3(float x){
 float y,c;
 int i,j;
 j = *(int*)&x;
 i=j>>23; j=j/3; i=i-3*(j>>23);
 j = r[i]-j; y = *(float*)&j; y*=beta[i];
 y*=1.333382888f-0.3333271391f*x*y*y*y; //I iter
 c=1.f-x*y*y*y; y+=0.333333333f*c*y;    //II iter
 return y;
}
```

7 A modified Halley method

Other approximation schemes that apply magic constants and should be considered in this article are iterative methods of higher order. One of them, the Halley iteration method for computing the function $1/\sqrt[3]{x}$, is expressed by the

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

following formulas:

$$y_k^{(H)} = \frac{1}{9} y_{k-1}^{(H)} (14 - 7xy_{k-1}^{(H)^3} + 2x^2 y_{k-1}^{(H)^6}), \tag{42}$$

$$\delta_k^{(H)} = \frac{1}{9}(1+\delta_{k-1}^{(H)}) \left(14 - 7(1+\delta_{k-1}^{(H)})^3 + 2(1+\delta_{k-1}^{(H)})^6\right) - 1. \tag{43}$$

The relative error function (43) is a nondecreasing error function $\delta_{k-1}^{(H)}$ with an inflection point at zero. Its global maximum and minimum correspond to the global maximum and minimum of the error of the initial approximation. In contrast to the NR method, the application of the Halley method guarantees a more symmetrical error.The maximal relative errors of the approximations $\tilde{y}_0^{(c,1)}$, $\tilde{y}_0^{(t,3)}$, $\tilde{y}_0^{(c,3\cdot2)}$ and $\tilde{y}_0^{(c,3\cdot4)}$ obtained after the first iterations of H are shown in Table 3. Below the error values, the corresponding approximation accuracies in terms of the number of correct bits $ac_{\text{bits},k}^{(*)} = \lfloor -\log_2 \delta_{k,\max}^{(*)} \rfloor$ are given.

Table 3: The maximal relative errors $\delta_{1,\max}^{(*)}$ for the initial approximations $\tilde{y}_0^{(c,1)}$, $\tilde{y}_0^{(t,3)}$, $\tilde{y}_0^{(c,3\cdot2)}$ and $\tilde{y}_0^{(c,3\cdot4)}$ after the first iterations of H and $M2$, along with the related accuracies (in bits).

$y_0^{(*)}$	$\tilde{y}_0^{(c,1)}$	$\tilde{y}_0^{(t,3)}$	$\tilde{y}_0^{(c,3\cdot2)}$	$\tilde{y}_0^{(c,3\cdot4)}$
$\delta_{1,\min}^{(H)}$	$-1.02 \cdot 10^{-4}$	$-1.92 \cdot 10^{-6}$	$-4.37 \cdot 10^{-7}$	$-1.23 \cdot 10^{-8}$
$\delta_{1,\max}^{(H)}$	$1.11 \cdot 10^{-4}$	$1.96 \cdot 10^{-6}$	$4.43 \cdot 10^{-7}$	$1.23 \cdot 10^{-8}$
$ac_{\text{bits},1}^{(H)}$	13	18	21	26
$\delta_{1,\max}^{(M2)}$	$2.67 \cdot 10^{-5}$	$4.86 \cdot 10^{-7}$	$1.10 \cdot 10^{-7}$	$3.08 \cdot 10^{-9}$
$ac_{\text{bits},1}^{(M2)}$	15	20	23	28

The presented iterative method is more successful with input approximations other than that provided by the NR method, but after the first iteration, we obtain a sufficient number of accurate bits for 32-bit numbers only for the approximation $\tilde{y}_0^{(c,3\cdot4)}$. It is known from [9] that the Halley method may be efficiently modified. The function $InvCbrt21$ [9], for the approximation of $\tilde{y}_0^{(c,1)}$, after a single iteration of the optimised Householder iteration of the second order, provides a maximal error of $2.69 \cdot 10^{-5}$, which is approximately four times lower than the maximal error $\tilde{y}_0^{(c,1)}$ after the H iteration (cf. Table 3). Such convergence may be obtained *via* a certain modification $M2$ of the method, similar to the NR iteration, that does not increase the method's complexity:

$$y_k^{(M2)} = \frac{\kappa_{0,k}\, s_k}{9}\, y_{k-1}^{(M2)} (14\kappa_{1,k} - 7x\, s_k^3 y_{k-1}^{(M2)^3} + 2x^2\, s_k^6 y_{k-1}^{(M2)^6}), \tag{44}$$

$$\delta_k^{(M2)} = \frac{\kappa_{0,k}\, s_k}{9}\, (1+\delta_{k-1}^{(M2)}) \left(14\kappa_{1,k} - 7s_k^3(1+\delta_{k-1}^{(M2)})^3 + 2\, s_k^6 (1+\delta_{k-1}^{(M2)})^6\right) - 1. \tag{45}$$

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

The analysis of the error function shows that next to extremal values, related to the extrema of the $(k-1)$th approximation, new maxima appear:

$$\delta^{(M2)}_{k,\max_\kappa} = \kappa_{0,k}\sqrt[3]{1-\sqrt{1-\kappa_{1,k}}}(4\kappa_{1,k}-1+\sqrt{1-\kappa_{1,k}})/3-1, \tag{46}$$

for

$$\delta^{(M2)}_{k-1} = \delta^{(M2)}_{k-1,+} = \sqrt[3]{1-\sqrt{1-\kappa_{1,k}}}/s_k - 1. \tag{47}$$

New minima also appear:

$$\delta^{(M2)}_{k,\min_\kappa} = \kappa_{0,k}\sqrt[3]{1+\sqrt{1-\kappa_{1,k}}}(4\kappa_{1,k}-1-\sqrt{1-\kappa_{1,k}})/3-1, \tag{48}$$

for

$$\delta^{(M2)}_{k-1} = \delta^{(M2)}_{k-1,-} = \sqrt[3]{1+\sqrt{1-\kappa_{1,k}}}/s_k - 1. \tag{49}$$

By solving the equation

$$\delta^{(M2)}_{k,\min_\kappa} + \delta^{(M2)}_{k,\max_\kappa} = 0, \tag{50}$$

we find the relation between $\kappa_{0,k}$ and $\kappa_{1,k}$:

$$\kappa_{0,k} = 6\left(\sqrt[3]{1+\sqrt{1-\kappa_{1,k}}}\,(4\kappa_{1,k}-1-\sqrt{1-\kappa_{1,k}}\,)+ \right.$$
$$\left. +\sqrt[3]{1-\sqrt{1-\kappa_{1,k}}}\,(4\kappa_{1,k}-1+\sqrt{1-\kappa_{1,k}}\,)\right)^{-1}. \tag{51}$$

The remaining parameters s_k and $\kappa_{1,k}$ are found by solving a system of equations for $\kappa_{0,k}=1$:

$$\delta^{(M2)}_{k,\min_\kappa}|_{\kappa_{0,k}=1} = \delta^{(M2)}_k|_{\kappa_{0,k}=1\,,\,\delta^{(M2)}_{k-1}=\delta^{(M2)}_{k-1,\min}=-\delta^{(M2)}_{k-1,\max}},$$
$$\delta^{(M2)}_{k,\max_\kappa}|_{\kappa_{0,k}=1} = \delta^{(M2)}_k|_{\kappa_{0,k}=1\,,\,\delta^{(M2)}_{k-1}=\delta^{(M2)}_{k-1,\max}}.$$

This equalises the new maxima $\delta^{(M2)}_{k,\max_\kappa}$ and minima $\delta^{(M2)}_{k,\min_\kappa}$ with the corresponding extremal values originating from the maxima and minima of the global error of the $(k-1)$th iteration.

The new algorithm $M2$ was applied to the same approximations as the method H. The values of the maximal relative errors for the approximations $\tilde{y}_0^{(c,1)}$, $\tilde{y}_0^{(t,3)}$, $\tilde{y}_0^{(c,3\cdot 2)}$ and $\tilde{y}_0^{(c,3\cdot 4)}$ that were obtained after the first $M2$ iteration are given in Table 3. On the basis of these values, we can conclude that for 32-bit precision, a sufficient accuracy, for a single iteration, is provided by $\tilde{y}_0^{(c,3\cdot 4)}$. From the function $\tilde{y}_0^{(c,3\cdot 4)}$, after the first iteration of type $M2$, we obtain the code given in Algorithm 1.2.

Algorithm 1.2 *InvCbrtC3N4*

```
1  const float beta[12]={1.08226994903f,
2    0.826812502031f, 0.661081551282f, 0.545920576662f,
```

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

```
  1.717996455 61f, 1.31248303551f, 1.04940154994f,
  0.866594897684f, 1.36357469045f, 1.04171847563f,
  0.832910562158f, 0.687816826107f};
const int r[12]={1419038221, 1421840751, 1424641323,
  1427440725,1413445816, 1416248345, 1419048917,
  1421848319,1416242018, 1419044548,1421845120,
  1424644522};
float InvCbrtC3N4(float x){
  float y,c;
  int i,j;
  j=*(int*)&x; i=j>>21; j=j/3;i=i-12*(j>>23);
  j=r[i]-j; y=*(float*)&j; y*=beta[i];
  c=1.f-x*y*y*y; y+=y*c*(0.3333355608f+0.222221851f*c);
  return y;
}
```

Additionally, the accuracy of the approximation that uses $\tilde{y}_0^{(c,1)}$ has been improved. By performing two iterations, with the first iteration of type $M2$ and the second of type $M1$, for the function $\tilde{y}_0^{(c,1)}$, we obtain the code given in Algorithm 1.3.

Algorithm 1.3 *InvCbrtM2M1*

```
float InvCbrtM2M1(float x){
  float k1=1.752319948f;
  float k2=1.250953236f;
  float k3=0.5093824286f;
  int j=*(int*)&x; j=0x548c2b4b-j/3;
  float y=*(float*)&j;
  float c=x*y*y*y; y*=k1-c*(k2-k3*c);
  c = 1.f - x*y*y*y; y+= y*0.3333333333f*c;
  return y;
}
```

8 Comparison of accuracy and speed of algorithms implemented on microcontrollers and single-board computers

In this section, three new approximation algorithms are compared with the library function $1.f/cbrtf()$ and the algorithm $InvCbrt21()$ [9] in terms of the maximal relative errors and average relative computation times. The computational experiments and all necessary measurements of the relative errors and computation times of our algorithms for computing reciprocal cube root functions were performed on two types of processors with floating-point units (FPUs) and one type of processor without an FPU:

- 32-bit microcontrollers with FPUs: STM32F767ZIT6 (ARM Cortex M7 with a clock frequency $f = 216$ MHz; gcc compiler);
- 32-bit microcontrollers without FPUs: RP2040 (ARM Cortex M0+; $f = 133$ MHz; gcc compiler);
- 64-bit processor with four cores used in SBCs and running Linux OS: Broadcom BCM2837B0 (Cortex-A53; $f = 1.4$ GHz; gcc compiler).

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

In Table 4 the columns $\delta_{-,\max}$ and $\delta_{+,\max}$ contain the limits of the relative errors of the particular approximation method used to compute the function $1/\sqrt[3]{x}$. The boundaries of the relative errors were determined for *float*-type numbers from the range $[1, 8)$. The smallest upper bounds and the largest lower bounds of the errors are in bold.
The columns $t \cdot f$ contains the average times, which are expressed by the corresponding number of clock cycles in the given microcontroller/processor architecture. They are computed as the average execution time t of the particular function/algorithm multiplied by the clock frequencies f of the tested units. The lowest time values for all tested algorithms are shown in bold.

Table 4: The limits $\delta_{\pm,\max}$ of the relative errors and average execution times expressed in cycles ($t \cdot f$). The program testing the average execution times of algorithms was compiled with the -O3 option.

Microcontroller	STM32F767ZIT6			RP2040			BCM2837B0		
Method	$\delta_{-,\max}$	$\delta_{+,\max}$	$t \cdot f$	$\delta_{-,\max}$	$\delta_{+,\max}$	$t \cdot f$	$\delta_{-,\max}$	$\delta_{+,\max}$	$t \cdot f$
$1./cbrtf()$	-2.60e-7	2.79e-7	118.4	-2.46e-7	2.73e-7	1661	-1.20e-7	1.19e-7	166.4
$InvCbrt21()$	-1.33e-7	1.33e-7	51.1	-1.43e-7	1.42e-7	1354	-1.33e-7	1.33e-7	69.0
$InvCbrtT3()$	-8.39e-8	7.80e-8	50.1	-9.94e-8	**9.33e-8**	1258	-8.39e-8	7.80e-8	72.3
$InvCbrtC3N4()$	-8.10e-8	8.18e-8	**41.1**	-9.96e-8	9.59e-8	**910**	-8.10e-8	8.18e-8	**56.9**
$InvCbrtM2M1()$	**-7.93e-8**	**7.79e-8**	46.1	**-9.79e-8**	9.55e-8	1350	**-7.93e-8**	**7.79e-8**	64.9

9 Conclusion

This paper deals with the numerical approximation of the reciprocal cube root function using dedicated algorithms that can be implemented in software in DSP platform systems. The article contains theory, algorithms, the testing of these algorithms on various hardware platforms, an analysis of the obtained results and conclusions. The research results presented in this article clearly show the advantages of three new approximate algorithms for computing the reciprocal cube root function within various processor architectures. The basic idea behind the algorithms was the use of magic constants and linear function approximations within subranges of the essential argument range $\tilde{x} \in [1, 8)$.
The introduced algorithms vary in terms of their construction and parameters. Their C implementations were favourable compared with two previous methods for approximately computing the function $1/\sqrt[3]{x}$ on different hardware environments: microcontrollers and SBC-type computer. These embedded processors are examples of potential implementations of CPUs for DSP computing devices. The factors that were taken into account were the relative error limits and the average time measured in clock cycles.
The comparison of the algorithms provided evidence of the wide applicability of the modified Halley method and the algorithm $InvCbrtC3N4()$. This algorithm

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

has an exceptional efficiency in terms of the average execution time, while its relative errors are on the same low level as those of the other new algorithms. The detailed parameters presented in the tables in this paper allow the prospective user to adopt the correct algorithm for a particular processor architecture. An advanced numerical algorithm designer should be able to derive other high-quality algorithms of the same kind for computing numerical functions in critical applications without access to function libraries that are only available commercially.

References

1. L. Parrilla, A. Lloris, E. Castillo, and A. García, "Table-free seed generation for hardware Newton–Raphson square root and inverse square root implementations in IoT devices", *IEEE Internet of Things Journal*, vol. 9, pp. 6985–6995, 2021.
2. Z. Długosz, M. Rajewski, R. Długosz, and T. Talaśka, "A novel, low computational complexity, parallel swarm algorithm for application in low-energy devices", *Sensors*, vol. 21, 2021, Art. no. 8449.
3. W. Langdon and O. Krauss, "Genetic improvement of data for maths functions", *ACM Transactions on Evolutionary Learning and Optimization*, vol. 1, no. 2, pp. 1–30, 2021.
4. P. Gepner, "Machine learning and high-performance computing hybrid systems, a new way of performance acceleration in engineering and scientific applications", in *2021 16th Conference on Computer Science and Intelligence Systems (FedCSIS)*, 2021, pp. 27–36.
5. C. Fäerber, R. Schwemmer, J. Machen, and N. Neufeld, "Particle identification on an FPGA accelerated compute platform for the LHCb upgrade", *IEEE Transactions on Nuclear Science*, vol. 64, no. 7, pp. 1994–1999, 2017.
6. O. Dieterle, T. Greiner, and P. Heidrich, "State reference computation for PMSM implemented with single-precision floating point datatype", in *IEEE International Conference on Power Electronics, Drives and Energy Systems (PEDES)*, 2020, pp. 1–8.
7. S. A. Levin, "Two point raytracing for reflection off a 3D plane", Stanford Exploration Project, 2012.
8. G. Wang, Y. Liu, Y. Sun, J. Yu, and Z. Sun, "Generalized zeroing neural network model for online solving time-varying cube roots with various external disturbances in different domain", Available at SSRN, 2022.
9. L. Moroz, V. Samotyy, C. J. Walczyk, and J. L. Cieśliński, "Fast calculation of cube and inverse cube root using a magic constant and its implementation on microcontrollers", *Energies*, vol. 14, no. 4, 2021, Art. no. 1058, doi: 10.3390/en14041058.
10. "Short Vector Math Library (SVML)". `https://community.intel.com/t5/Intel-C-Compiler/Short-Vector-Math-Library/td-p/857718` (accessed Sept. 25, 2018).
11. "Integrated Performance Primitives (IPP)". `https://software.intel.com/content/www/us/en/develop/tools/integrated-performance-primitives.html` (accessed Feb. 14, 2021).
12. C. M. Guardia and E. Boemo, "FPGA implementation of a binary32 floating point cube root", in *2014 IX Southern Conference on Programmable Logic (SPL)*, 2014, pp. 1–6.

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9

13. Y. N. Zhang, Y. H. Ling, M. Yang, S. Yang, and Z. J. Zhang, "Inverse-free discrete ZNN models solving for future matrix pseudoinverse via combination of extrapolation and ZeaD formulas", *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 6, pp. 2663–2675, 2021.
14. G. Szanto, "64-bit ARM optimization for audio signal processing", Superpowered, Austin, Texas, USA, 2019.
15. "Fast Cube Root Using C33", DSP Related.Com. `https://www.dsprelated.com/showthread/comp.dsp/109311-1.php` (accessed Feb. 26, 2009).
16. A. Pineiro, J. D. Bruguera, F. Lamberti, and P. Montuschi, "A radix-2 digit by-digit architecture for cube root", *IEEE Transactions on Computers*, vol. 57, no. 4, pp. 562–566, 2008.
17. J. Harrison, T. Kubaska, S. Story, and P. T. P. Tang, "The computation of transcendental functions on the IA-64 architecture", *Intel Technology J.*, vol. 4, no. 7, 1999.
18. C. S. Anderson, J. Zhang, and M. Cornea, "Enhanced vector math support on the Intel® AVX-512 architecture", in *2018 IEEE 25th Symposium on Computer Arithmetic (ARITH)*, 2018, pp. 120–124.
19. S. H. Warren, *Hacker's Delight*, 2nd ed. Addison-Wesley, 2013.
20. W. Kahan, "Computing a real cube root", *Lecture Notes*, 1991.
21. S. K. Panda, K. Achyut, S. K. Kulkarni, A. A. Raut, and A. Nayak, "An organized literature review on various cubic root algorithmic practices for developing efficient VLSI computing system—Understanding complexity", *Artificial Intelligence Applications and Reconfigurable Architectures*, pp. 35–62, 2023.
22. IEEE Computer Society. Standards Committee. Working Group of the Microprocessor Standards Subcommittee, and American National Standards Institute, *IEEE Standard for Binary Floating-Point Arithmetic,* vol. 754. IEEE, 1985.
23. J. Blinn, "Floating-point tricks", *IEEE Comput. Graph. Appl. Mag.*, vol. 17, no. 4, pp. 80–84, 1997.
24. L. Moroz, C. J. Walczyk, A. Hrynchyshyn, V. Holimath, and J. L. Cieśliński, "Fast calculation of inverse square root with the use of magic constant – Analytical approach", *Appl. Math. Comput*, vol. 316, pp. 245–255, 2018.
25. C. J. Walczyk, L. Moroz, and J. L. Cieśliński, "A modification of the fast inverse square root algorithm", *Computation*, vol. 7, no. 41, 2019.
26. C. J. Walczyk, L. Moroz, and J. L. Cieśliński, "Improving the accuracy of the fast inverse square root by modifying Newton-Raphson corrections", *Entropy (Basel)*, vol. 23, no. 1, p. 86, 2021.
27. L. Moroz, V. Samotyy, O. Horyachyy, and U. Dzelendzyak, "Algorithms for calculating the square root and inverse square root based on the second-order Householder's method", in *Proc. of the 2019 10th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS)*, France, 2019, pp. 436–442.
28. L. Moroz and V. Samotyy, "Efficient floating-point division for digital signal processing application", *IEEE Signal Processing Magazine*, vol. 36, no. 1, 2019.
29. C. J. Walczyk, L. V. Moroz, V. Samotyy, J. L. Cieśliński, "Optimal approximation of the 1/x function using Chebyshev polynomials and magic constants", *ACM Trans. Math. Softw.*, vol. 51, no. 1, 2025, url = https://doi.org/10.1145/3708472

ICCS Camera Ready Version 2025
To cite this paper please use the final published version:
DOI: 10.1007/978-3-031-97570-7_9