



Article

Pseudo-Normalization via Integer Fast Inverse Square Root and Its Application to Fast Computation without Division

Takashi Kusaka ^{*,†}  and Takayuki Tanaka [†] 

Graduate School of Information Science and Technology, Hokkaido University, Sapporo 060-0814, Japan

* Correspondence: kusaka@ssi.ist.hokudai.ac.jp; Tel.: +81-11-706-6761

† These authors contributed equally to this work.

Abstract: Vector normalization is an important process in several algorithms. It is used in classical physical calculations, mathematical techniques, and machine learning, which has witnessed significant advancements in recent years. Normalization and regularization ensure the stability of solutions and play an important role in algorithm convergence. Normalization typically refers to the division of elements by their norm. Division should not be used in algorithmic implementations because its computational cost is considerably higher than that of multiply–add operations. Based on this, there is a well-known method referred to as the fast inverse square root (FISR) algorithm in floating-point calculations (IEEE754). In deeper-level embedded systems that require fast responses or power efficiency, integer instead of real number arithmetic (floating-point number arithmetic) should be used to increase speed. Conversely, in deeper-level embedded systems that require fast responses or power efficiency, integer arithmetic should be used instead of real number arithmetic (floating-point number arithmetic) to increase speed. Therefore, embedded engineers encounter problems in instances in which they use integer arithmetic for implementation, but real number arithmetic is required to compute vectors and other higher-dimensional algebra. There is no conventional normalization algorithm similar to the FISR algorithm for integer arithmetic; however, the proposed pseudo-normalization achieves vector normalization within a restricted domain using only multiply–add operations and bit shifts. This allows for fast and robust operations, even for low-performance MCUs that do not have power-efficient FPUs. As an example, this study demonstrates the computation of the arctangent (Arctan2 function; $\text{atan2}(y, x)$) with high precision using only integer multiply–add operations. In this study, we proposed a method of vector normalization using only integer arithmetic for embedded systems and confirmed its effectiveness by simulation using Verilog. The research results can contribute to various fields such as signal processing of IMU sensor data, faster artificial intelligence training, and efficient rendering of computer graphics.

Keywords: approximate computing; low-power arithmetic; integer calculation; fast inverse square root; embedded system



Citation: Kusaka, T.; Tanaka, T. Pseudo-Normalization via Integer Fast Inverse Square Root and Its Application to Fast Computation without Division. *Electronics* **2024**, *13*, 2955. <https://doi.org/10.3390/electronics13152955>

Academic Editor: Luis Gomes

Received: 8 June 2024

Revised: 18 July 2024

Accepted: 24 July 2024

Published: 26 July 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

In recent years, the rapid development of the Internet-of-Things (IoT) and artificial intelligence (AI) has led to a rise in the demand for advanced computational processing for all device types; for example, communication devices, such as mobile terminals and wearable devices; mobile vehicles, such as self-driving cars and public transportation systems; data centers for learning AI; all large and small central processing units (CPUs), graphics processing units, and microcomputers. It is important to reduce the power consumption of these devices [1–3]. A reduction in the power consumption of mobile terminals and IoT devices improves their battery life. Furthermore, the power consumption of data centers has considerably increased owing to the increased use of AI; this has become a bottleneck in AI development [4–7]. Fast computation with low power consumption is required in all fields of computation [8–12].

We previously proposed a fast computation algorithm for embedded systems, which approximates nonlinear elementary functions, such as trigonometric and inverse trigonometric, with high accuracy using multiply–add operations [13,14]. This fast computation is important for the attitude computation of wearable IoT sensors and the attitude control of drones, which require fast processing and low-power consumption [15]. Division is crucial for the embedded implementation of algorithms. Although division is one of the four basic arithmetic operations, its computational cost is substantially higher than that of a multiply–add operation. Therefore, various methods have been proposed to achieve efficient division. Imani et al. propose a configurable approximate divider CADE that performs floating-point division with controllable precision at runtime [16]. Liu et al. [17] and Saadat et al. [18] proposed a method based on logarithmic divisors and applied it to image processing and image compression. Vahdat et al. [19] and Zendegani et al. [20] also proposed a method to convert division into the product of the reciprocal of the divisor and applied it to image processing on DSPs and elsewhere.

One method for accelerating the calculations in embedded systems is to use integers (or fixed-point numbers) instead of floating-point numbers, but this method is incompatible with conventional division and normalization algorithms. Therefore, embedded engineers encounter problems in instances in which they want to use integer arithmetic for implementation, but real number arithmetic is required to compute vectors and other higher-dimensional algebra. As a solution to this problem, this study proposes a fixed-point vector normalization method using only multiply–add and bit-shift operations for high-speed and power-saving processing in embedded systems. The proposed method can be easily implemented in hardware, because it operates based only on simple sum-of-products operations.

In this study, we confirm that the proposed method can be implemented in FPGA, can only be composed of combinational circuits, and can be executed using hardware with an electrical delay of at most one clock operation. As will be shown later, there is a trade-off between the accuracy of the conventional method over the entire definition region and the efficiency of the proposed method. When the domain is known, the efficiency of our proposed method is very effective in reducing power consumption.

A limitation of the proposed method is that instead of being very fast and efficient, it limits the domain over which it can be computed accurately. The proposed method is referred to as pseudo-normalization, because it is not available for all definition domains. A domain can be controlled by the number of iterations of the algorithm. As the range of vectors used in embedded systems is typically known, the algorithm can be used in a broad range of fields by appropriately designing systems. The proposed algorithm consists of the fast approximate computation of the inverse function ($1/x$) and square root function (\sqrt{x}), both of which can be used independently.

In addition, for the approximate calculations of trigonometric and inverse trigonometric functions designed for floating-point numbers, integer transformations are performed using the fast inverse square root (FISR) algorithm to convert them to fixed-point numbers. The acceleration effect of the floating-point computation is limited because of the optimization of the instruction set and compiler of the CPU and microcontroller. However, the implementation of fixed-point computations is approximately three times faster than that of “math.h” on a CPU with an FPU. The proposed algorithm can be used in various vector normalization applications. An example of accelerating integer transformation using inverse trigonometric functions is described herein.

2. Method

First, we overview the conventional and proposed normalization algorithms. Subsequently, we design an approximate computation of one of the components, the reciprocal, using the residual correction method. We then design an approximate computation method for the square root computation, which is another component of the normalization algo-

rithm. Finally, we combine them to propose pseudo-normalization as a vector normalization for integer arithmetic (proposed method).

2.1. Conventional and Proposed Methods

It is important to optimize algorithms in IoT systems wherein power efficiency is required and the amount of computation should be reduced (even by one clock operation). The low-power Cortex-M0 omits the DIV instruction [21], and in low-spec microcontrollers, 45 divisions are implemented in software rather than provided in the instruction set [22]. Based on this, the FISIR algorithm has been developed for IEEE754 floating-point numbers [23–26]. This algorithm uses the bit structure of IEEE754 to approximate rapidly $1/\sqrt{x}$. Additionally, various extension algorithms have been proposed in recent years [27–29]. The FISIR algorithm can be used for floating-point arithmetic in the IEEE754 format and has been implemented in various programming languages. It takes advantage of the fact that bit shifting the bit structure of a floating-point number results in a value close to the inverse square root of the original number. It calculates a practical value by converging the approximate value using Newton's method.

We have used the FISIR algorithm to develop fast computation algorithms for embedded systems; however, it is desirable to develop algorithms with fixed-point numbers rather than floating-point numbers to increase computational speed [30]. Therefore, we propose a new vector normalization algorithm for fixed-point numbers that does not require floating-point number inputs.

In the proposed method, we normalize vectors without using division. If the norm of the vector (x, y) is $r = \sqrt{x^2 + y^2}$, normalization is performed to find $(x', y') = (\frac{x}{r}, \frac{y}{r})$. Herein, we can consider normalization if we can compute the function $I(r)$, i.e., the inverse of the norm $\frac{1}{r} \approx I(r)$ within a domain of a function by applying a multiply-add operation on r . As division is required to create $I(r)$ in any domain, we consider pseudo-normalization as an approximation restricted to the domain of the function. The next section describes the design of the algorithm. First, we develop a low-cost approximation algorithm for each real number using the proposed residual correction method [31]. This method can be used to obtain a highly accurate approximate function within a defined region using only multiply-add operations. We then modify the algorithm for use to fixed-point numbers.

2.2. Optimal Reciprocal Approximation

The first formulation of the normalization problem is to search for $I(r)$ such that $rI(r) \approx 1$ in a specific domain of the function, $r_{min} < r < r_{max}$. For approximate computations, $I(r)$ should be an approximate function such that the error is globally smaller within the domain of definition rather than being strictly accurate. We propose the residual correction method to search for such approximate functions and realize highly accurate approximate trigonometric and arctangent functions. In a previously proposed residual correction method, a simple approximation was considered and its residuals were evaluated to create a residual correction function. A highly accurate approximate function was then obtained by removing the residuals. However, owing to the strong nonlinearity of the normalization targeted here, multiplicative residual correction is preferred over additive residual correction.

We consider a Taylor expansion at approximately $r = 1$, which is suitable as a simple approximate function. $rI_1(r) \approx 1$ has a wider domain compared with r without normalization.

$$I_1(r) = 2 - r \quad (1)$$

We also consider a function that corrects this residual. As additive residual correction results in a constant equation, we use multiplicative residual correction to search for $f_{rc}(r)$ that minimizes the residual of $rI_2(r) = rI_1(r)f_{rc}(r) \rightarrow 1$. For function shaping, a downward

convex quadratic function centered at $r = 1$ is a candidate for $f_{rc}(r)$, and it must satisfy $f_{rc}(1) = 1$. The following are possible candidates for this function,

$$f_{rc}(r) = (1 - r)^2 + 1 \quad (2)$$

Therefore, $I_2(r)$ has the following form,

$$I_2(r) = I_1(r)f_{rc}(r) = (2 - r)\left((1 - r)^2 + 1\right) \quad (3)$$

The result corrected by this function is shown in Figure 1 as the second expansion of the domain of definition by the residual correction method. Without normalization ($I(r) = 1$), it is evident that only $rI(r) \approx 1$ satisfies $r \approx 1$. However, with the first expansion of the domain of definition, the domain that satisfies $rI_1(r) \approx 1$ expands to approximately $0.9 < r < 1.1$. After the second expansion, the domain that satisfies $rI_2(r) \approx 1$ expands to approximately $0.7 < r < 1.3$. This second approximation can have a broader definition domain compared with the approximate formula obtained using a Taylor expansion of the same order.

Further expansion of the domain using the residual correction method increases the computational cost owing to the strong nonlinearity of the original function; however, the error reduction effect is negligible. Therefore, we consider expanding the domain of definition by iteratively applying this approximate function. The application of $I_2(r)$ once and its modification to $r \rightarrow rI_2(r)$ is a more ideal state from the norm's perspective with a reduced error. As shown in Figure 1b, this improvement can be obtained at all points except the endpoints of the $0 < r < 2$ region. Therefore, we consider the repeated application of $I_2(r)$ as follows:

$$r' = rI_2(r) \quad (4)$$

$$r'' = r'I_2(r') = r\left[I_2(r) \cdot I_2(rI_2(r))\right] = rI_2'(r) \quad (5)$$

$$\Rightarrow I_2'(r) = I_2(r) \cdot I_2(r')$$

$$r^{(3)} = r''I_2(r'') = r\left[I_2(r) \cdot I_2(rI_2(r)) \cdot I_2(rI_2(r)I_2(rI_2(r)))\right] = rI_2''(r) \quad (6)$$

$$\Rightarrow I_2''(r) = I_2(r) \cdot I_2(r') \cdot I_2(r'')$$

$$\therefore I_2^{(n)}(r) = I_2(r) \cdot I_2(r') \cdot I_2(r'') \cdot I_2(r^{(3)}) \cdots I_2(r^{(n)}) = \prod_{i=1}^n I_2(r^{(i)}) \quad (7)$$

The number of primes indicates the number of iterations. Although the formula for the expansion looks complicated, it is simply an iterative process associated with the application of $I_2(r)$, which can ultimately be considered as a function of r . The effect of this reiteration is shown in Figure 2. Two iterations of r'' expand the domain of definition to $0.25 < r < 1.75$, and three iterations of r''' expand the domain to $0.07 < r < 1.93$, thus covering almost the entire area. With a good approximate function and convergence by iteration, we obtain a highly accurate estimation formula for $1/r$. The increase in computational cost owing to iterative calculations is described in Section 4.

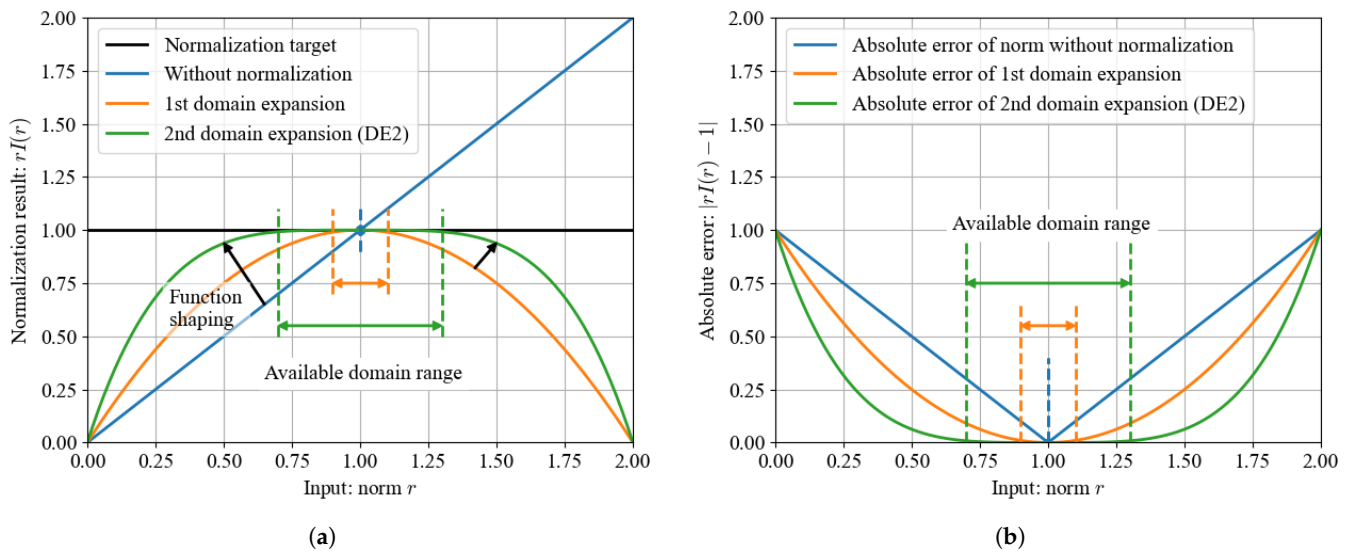


Figure 1. Inverse norm approximation using residual correction method. (a) Domain expansion effect of function shaping. (b) Error reduction using function shaping.

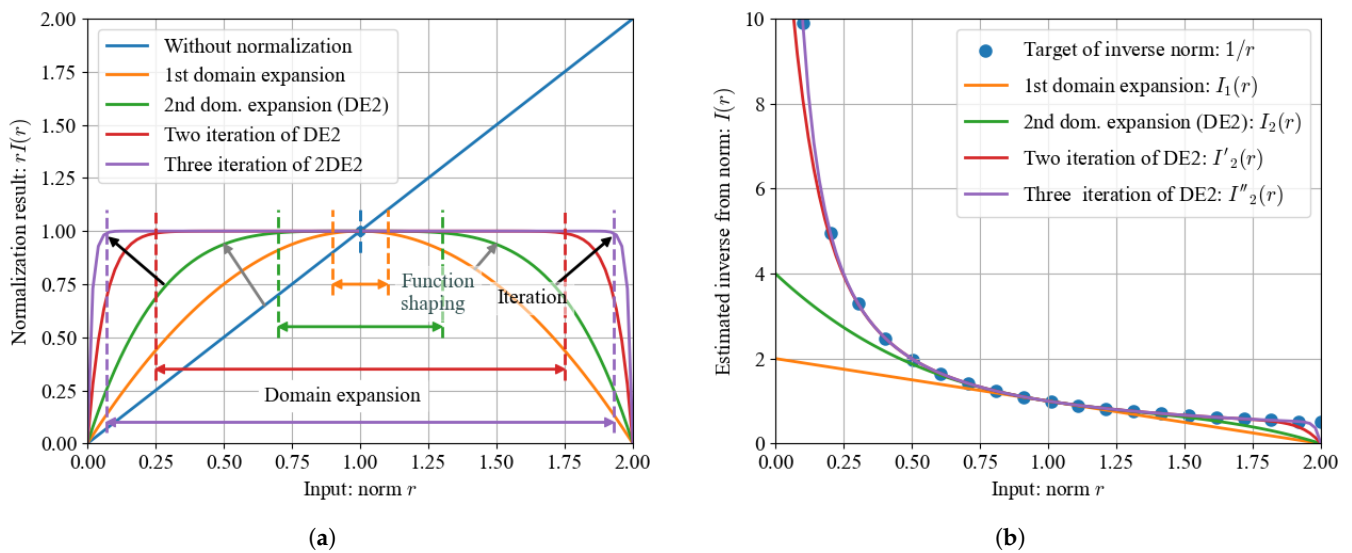


Figure 2. Pseudo-normalization using inverse norm approximation and its iterations. (a) Normalization effects of domain expansion and iteration. (b) Estimation results of inverse norm.

2.3. Square Root Approximation

To find the norm of a vector, the sum of squares ($r^2 = x^2 + y^2$) of each element can be directly calculated using a multiply-add operation, but the square root is required for normalization. Therefore, we consider the approximate function of the square root, which is $r^2 \rightarrow r$.

To determine the approximate function using the residual correction method, we first consider a simple candidate approximate function. The function design uses 2^N multiples for all but the constant terms, based on an embedded implementation. This can be obtained as the following linear expression using the Taylor expansion around $r = 1$.

$$r \approx f_{rt}(r^2) = \frac{1}{2}(r^2 + 1) \quad (8)$$

We obtain the blue dots and solid green line in Figure 3a by evaluating the residuals of the simple approximate function $f_{rt}(r^2)$. The search for residual correction functions leads to the following candidate functions,

$$f_{rt}(r^2) - r = e_{rt} \approx f_{rcf}(r^2) = \frac{1}{8}(0.5r^2)^2 - \frac{1}{32} \quad (9)$$

Thus, the second approximation is as follows:

$$r \approx f_{rt2}(r^2) = f_{rt}(r^2) - e_{rt} = f_{rt}(r^2) - \frac{1}{8}(0.5r^2)^2 + \frac{1}{32}. \quad (10)$$

This is a more reduced residual in the $r > 1$ region compared with the fourth-order Taylor expansion. Although higher-order Taylor expansions are required to reduce errors over the entire region, the residuals are minimized by restricting the region. The coefficients are adjusted to powers of 2 for the subsequent calculations.

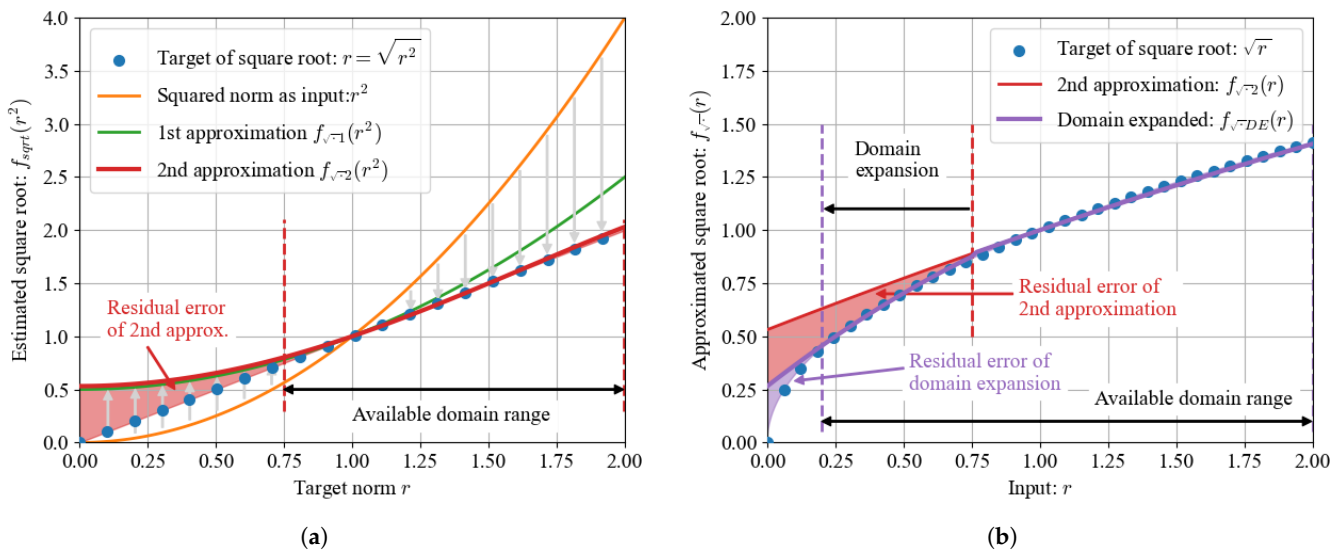


Figure 3. Pseudo-normalization using inverse norm approximation and its iterations. (a) Design of approximate function for square root. (b) Approximated result of square root and its domain expansion.

2.4. Pseudo-Normalization for Integer

In the previous sections, we showed that the inverse and square roots can be approximated using only a finite number of multiply-add operations. We convert these to fixed-point calculations, and consider fast normalization using an integer. We convert floating-point r to fixed-point z . For the fixed-point representation, $M = 2^N$ is defined as $r = 1$, where $N = 7$ and $M = 128$ is used as the base. First, the inverse is computed as follows:

$$\begin{aligned}
 \frac{1}{r} &\approx I_2(r) = (2 - r)((1 - r)^2 + 1) \\
 &= \left(2 - \frac{z}{M}\right) \left(\left(1 - \frac{z}{M}\right)^2 + 1 \right) \\
 &= \frac{1}{M} (2^{N+1} - z) \left\{ \left[(2^N - z) \gg N \right]^2 + 1 \right\} \\
 \therefore \frac{1}{z} \propto \frac{M^2}{z} &\approx I_2(z) = (256 - z) \left\{ [(128 - z) \gg 7]^2 + 1 \right\} \\
 &= (256 - z) \left\{ [(128 - z)^2 \gg 7] + 128 \right\} \gg 7
 \end{aligned} \quad (11)$$

where the operation \gg denotes a right arithmetic shift. The reciprocal is likely to be small. In particular, integer expressions are not possible in this domain. Therefore, the value is expressed in the upper bits of a fixed-point number; hence, it is defined as $I_2(z) = M^2/z (=M/r)$. For normalization, the bit width is adjusted by bit shifting after multiplying the number to be normalized by the reciprocal estimate. For example, if (x_i, y_i) is an original vector, it can be normalized to $x_o = (\frac{x_i}{z})M = x_i(\frac{M^2}{z} \gg N) = ([x_i I_2(z)] \gg N)$. Here, this equation and Equation (11) adjust the position of the bit shift. Although the computation of real numbers does not cause a drop-of-digit problem, integer multiplication and bit shifting change the bit width. Thus, the product is adjusted such that it is placed before bit shifting.

The square root can be computed using a multiply-add operation and bit-shift operator, as shown by the following equation:

$$\begin{aligned} r &\approx f_{rt2}(r^2) = \frac{1}{2}(r^2 + 1) - \frac{1}{8}(0.5r^2)^2 + \frac{1}{32} \\ &= \frac{r^2}{2} - \frac{r^4}{2^5} + \left(\frac{1}{2} + \frac{1}{32}\right) \\ &\rightarrow \frac{z^2}{2^{N+1}} + \frac{z^4}{2^{3N+5}} + \frac{2^4 + 1}{2^{5-N}} \\ \therefore z &\approx f_{rt2}(z^2) = (z^2 \gg 8) - (z^4 \gg 26) + 68 \\ &= (z^2 \gg 8) - (z^2 \gg 8)^2 \gg 10 + 68 \end{aligned} \quad (12)$$

These are used to construct a pseudo-normalization algorithm. For any input vector of integers, the square of its norm is computed (Figure 4) using a simple multiply-add operation. The norm is then obtained by the approximate function $f_{rt2}(z^2)$ of the square root. Finally, the reciprocal approximate function $I_2^{(n)}(z)$ provides the final normalized coefficients.

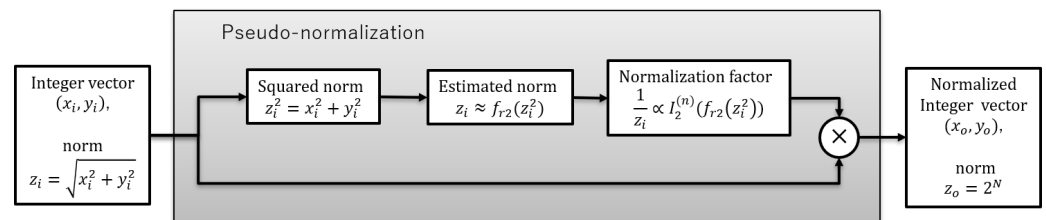


Figure 4. Pseudo-normalization algorithm for integer vector normalization using only multiply-add and bit-shift operations.

3. Results and Discussion

First, we show how the proposed algorithm works on an actual FPGA device, based on simulations and experiments. Considerations regarding computation time and efficiency are then discussed. Finally, as an example of the proposed method, an angle measurement by the IMU is demonstrated. The IMU angle measurement is one of the applications suitable for the proposed method, because it uses acceleration vectors and geomagnetic vectors, and the approximate length of the vectors is known and nonzero.

3.1. Results

The behavior of the proposed algorithm was examined based on simulations. First, the effect of the number of normalization iterations n was investigated experimentally; the results are shown in Figure 5. For practical purposes, one or two iterations are sufficient for expanding the domain of definition. However, if the range of the input norm is broad, more iterations can be used to extend the definition domain.

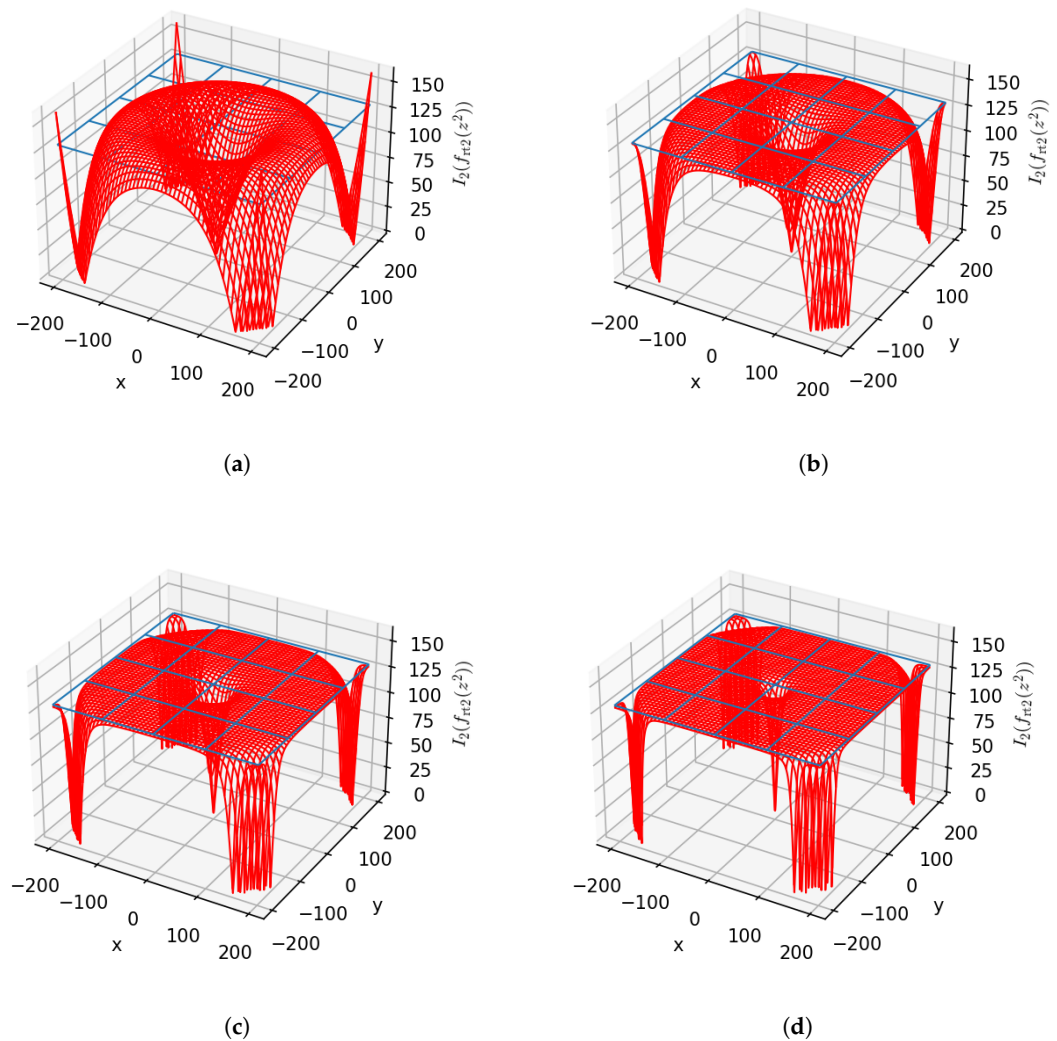


Figure 5. Effect of the number of iterations on normalization (blue plane represents the target value of normalization). (a) No iterations. (b) One iteration. (c) Two iterations. (d) Three iterations.

Subsequently, we describe the proposed normalization algorithm in Verilog, which is a hardware description language, and examine its computability for embedded systems. The basic circuit of the proposed algorithm in Verilog is shown in Figure 6. In addition, the inverse and root functions are shown in Figures 7 and 8 as the basic modules. These bit-widths and constants are slightly adjusted to simplify the Verilog's diagram, but do not affect the calculated results. The simulated operation is shown in Figure 9. In the simulation, (x_i, y_i) is generated by assuming $\theta = \pi/4$, and the magnitude of the inputs is changed from 0 to 200. The simulation results show that the output vector (x_o, y_o) is almost constant in the range of the input magnitude [24, 166]. Normalization is performed to cancel the variation in z_i .

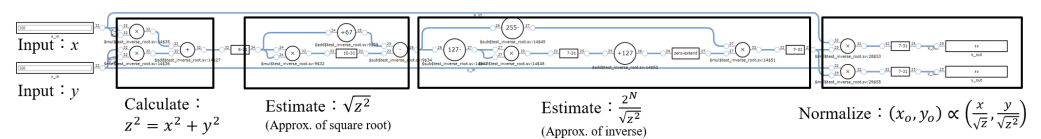


Figure 6. Overview of circuit example of proposed algorithm (no iteration) using Verilog (visualization created using <https://digitaljs.tilk.eu> (accessed on 14 May 2024)) [32].

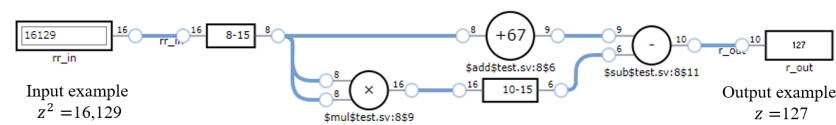


Figure 7. Basic function for approximation of square root.

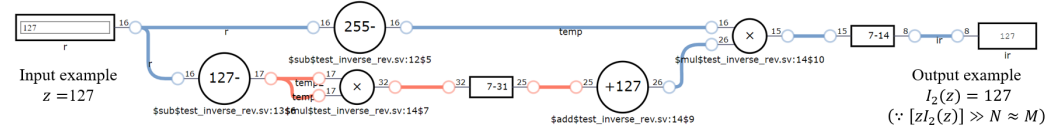


Figure 8. Basic function for approximation of inverse.

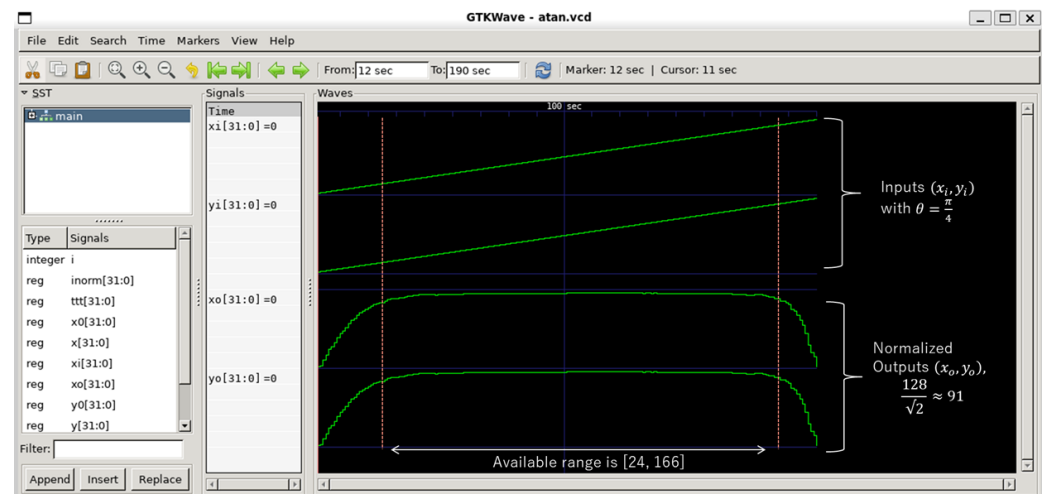


Figure 9. Simulation outcome for integer vector normalization based on the use of the proposed algorithm (three iterations).

Finally, the proposed algorithm was implemented on an actual FPGA device to evaluate its computational feasibility and speed. The FPGA used was the DE0-nano (manufactured by Terasic Inc., Hsinchu, Taiwan), an evaluation board for INTEL's Cyclone IV. The Verilog HDL source was the same as that used in the aforementioned simulation. Figure 10a shows the results. The experimental results show that the expression $128/\sqrt{2} \approx 91 = (01011101)_2$ can be calculated. Figure 10b also shows the timing of the instant when a specific bit on the input becomes "1" and the instant at which a specific bit on the output becomes "1". The FPGA used in this study is 50 MHz (sampling time 20 ns); this means that our algorithm can compute with only the electrical delay is less than one cycle. The proposed algorithm can be implemented using only combinational circuits, and does not require pipeline processing like sequential circuits, so it can be processed within one cycle in embedded circuits (such as FPGAs). Efficient computation with combinational circuits has been the subject of active research in recent years, including work on binary tree-based dividers, such as those proposed in [33]. A comparison with other similar vector normalization algorithms is shown in Table 1. Note that other similar algorithms are implemented as sequential circuits rather than combinational circuits; therefore, the evaluation methods are different. As the conventional algorithms require approximately 20 cycles, implementation in the same FPGA requires a computation time which is approximately equal to 400 ns. Our algorithm, which can be implemented using only combinational circuits for embedded circuits such as FPGAs, is very fast.

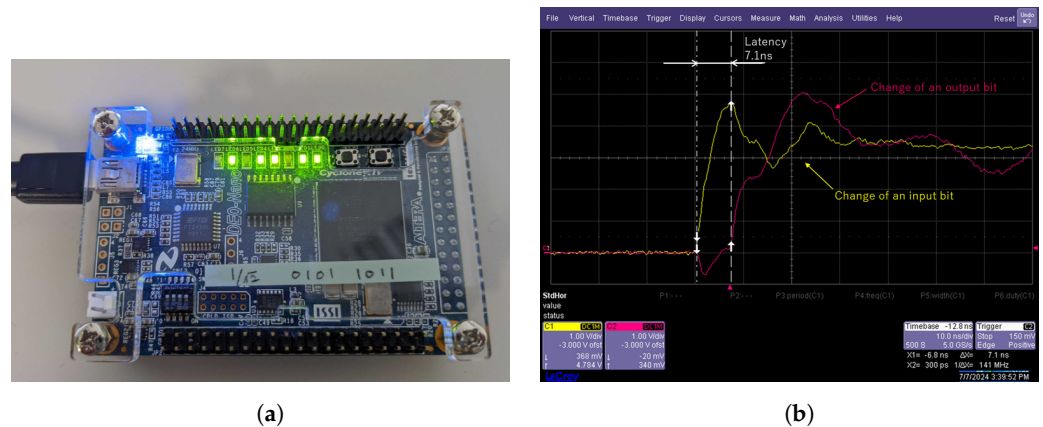


Figure 10. Implementation to FPGA and calculation speed evaluation. (a) Experimental implementation on FPGA. (b) Latency evaluation between input and output.

Table 1. Comparison of calculated speeds of similar algorithms.

	Calculation Speed	Circuit Type	Value Type
Proposed method	7.1 ns (Less than 1 cycle)	Combinational	Fixed point
INTEL's IP (INV_SQRT) [34]	26 cycles	Sequential	Floating point
PROVEN [35]	36 cycles	Sequential	Fixed point
POL [36]	13 cycles	Sequential	Floating point

Based on these results, accuracy and computation speed of the proposed algorithm and the proposed algorithm application will be discussed in subsequent sections.

3.2. Calculation Speed and Accuracy

As shown in the design phase, the approximation algorithm for the root computation is accurate for real numbers close to $r = 1$. However, the error increases as the numbers deviate from $r = 1$. This is because the highly nonlinear root function is approximated by a linear polynomial with residual correction, and further residual correction to reduce the error increases the computational cost. Therefore, it is desirable to utilize the system in fields where a limit of application can be ensured.

The accuracy of the inverse approximate function $I_2(\cdot)$ increases with the number of iterations. Although shown in the design phase of product representation, this can be rewritten as follows: the repeated application of I_2 is a recursive structure, and a recurrence relation can be computed.

$$\begin{aligned}
 I_1(x) &= (2 - x) \\
 I_2(x) &= (2 - x)((1 - x)^2 + 1) = I_1(x)(2 - xI_1(x)) = I_1(x)I_1(I_1(x)) \\
 I_{n+1}(x) &= \prod_{k=0}^n (2 - xI_k(x)), I_0 = 1
 \end{aligned} \tag{13}$$

This recurrence relation is equivalent to Newton's method for $1/x$ and is ensured to converge. In addition, compared with the Taylor expansion of $1/x$, the following equation holds.

$$I_{n+1}(x) = \prod_{k=0}^n (2 - xI_k(x)) = \sum_{k=0}^{2^n-1} (-1)^k (x-1)^k \tag{14}$$

In other words, $(n + 1)$ iterations in the approximate function coincide with the Taylor expansion up to $k = 2^n - 1$ odd orders. This implies that the iterative computation of the proposed method is analytically equal to the original function in $n \rightarrow \infty$. Additionally, the proposed multiplicative expansion can be computed with $O(n)$, whereas $O(2^n)$ is required to achieve the same accuracy in the Taylor expansion, which significantly reduces the computational cost. Although the proposed method cannot yield Taylor expansion approximations up to even orders because $1/x$ is an odd function, the form of the proposed method that does not deal with even orders is the preferred expansion basis. The amount of computations and the corresponding error order are summarized in Table 2.

Table 2. Comparison of calculation cost for Taylor expansion and product approximation.

Error	Taylor Expansion			Product Approximation		
	$2^n - 1$	Addition	Multiplication	n	Addition	Multiplication
$O(x^2)$	1	2	0	1	1	0
$O(x^3)$	2	3	1	-	-	-
$O(x^4)$	3	4	2	2	2	2
\vdots	\vdots	\vdots	\vdots	-	-	-
$O(x^8)$	7	8	6	3	3	4
\vdots	\vdots	\vdots	\vdots	\vdots	\vdots	\vdots
$O(x^{2^n})$	$2^n - 1$	2^n	$2^n - 2$	n	n	$2n - 2$

Using higher-order Taylor expansions as candidates for the residual correction function is one approach, but this approach will likely violate the concept of the residual correction method, which is the search of a simple correction function for efficient computation. However, the fact that the search results agree with the Taylor expansion results is a basis for asserting the validity of the correction function. Currently, the search for the correction function is conducted both empirically and heuristically, but the analytical agreement implies a degree of similarity with the original function, which could be one of the indicators of the search.

3.3. Further Extension of Domain of Definition

Certain applications require broad definition domains. The domain of definition can be expanded by increasing the number of iterations; however, a large number of iterations reduces computational efficiency. Therefore, we expand the domain using another method. First, accuracy decreases when the norm is small. Therefore, when the norm is smaller than a certain value (z_{th}), the domain expansion as an analogy to the domain expansion based on the double-angle formulas of trigonometric functions [13] is as follows:

$$\frac{1}{z} \propto I'_{sqrt}(z^2) = \begin{cases} I_2(f_{rt2}(z^2)) & \text{if } z^2 \geq z_{th}^2 \\ 2I_2(f_{rt2}((2z)^2)) & \text{otherwise} \end{cases} \quad (15)$$

For $N = 7$, $z_{th} = 97$ is experimentally confirmed to be appropriate in terms of continuity and domain expansion. This effect is shown in Figure 11a. Although $I_2(z)$ is used here, the domain of definition can be further expanded close to $z = 0$ by repeating $I_2(z)$, as shown in Figure 11b. Furthermore, we note that this is also true for large norms. This is because the norm is estimated to be smaller in the first iteration, even when it is large; as a result, the iterative computation is effective when the norm is large or small. This process extends the available input range to approximately [4, 173].

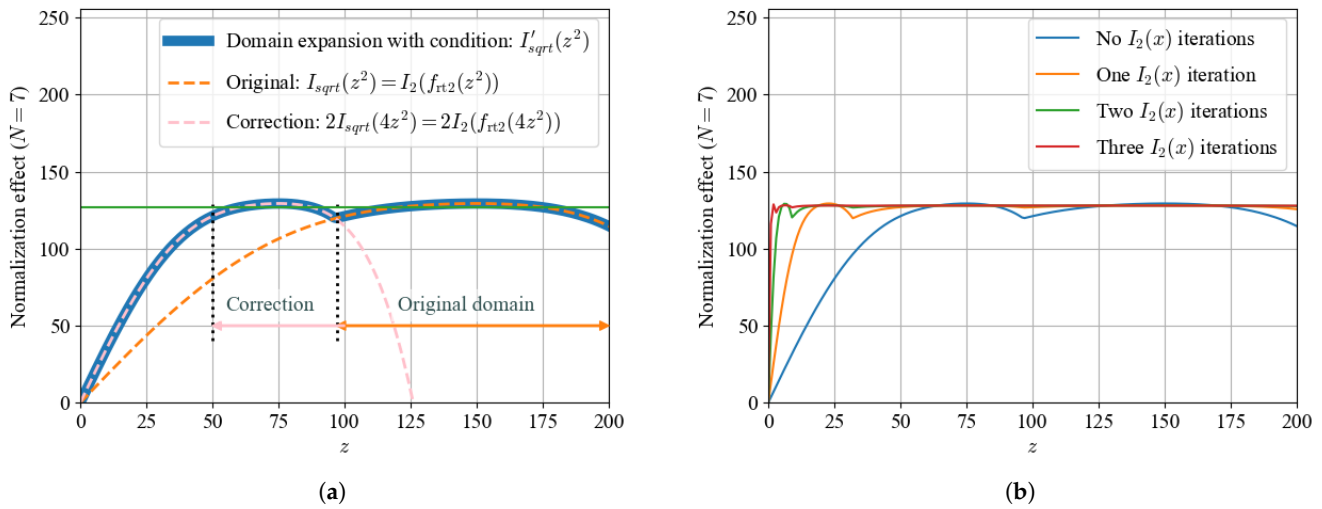


Figure 11. Pseudo-normalization using inverse norm approximation and its iteration. (a) Domain expansion with input condition. (b) Effects of the number of iterations on condition.

3.4. Application: Computing atan2 Function Using Integer Arithmetic

We have proposed the low-cost and fast approximate computation of trigonometric and inverse trigonometric functions for small embedded devices for the motion measurement of robots and humans. The algorithm is designed using a multiply-add operation. However, the prerequisite is that the input vector (sensor output) must be normalized using the FISIR algorithm. The FISIR algorithm is used for floating-point numbers, which limits its ability to perform processing in integer calculation cases that are ideal for embedded devices. The proposed method applies the FISIR algorithm to integers to solve this problem. For this purpose, integer transformations are performed on previously proposed trigonometric and inverse trigonometric functions (atan2). As shown in the results, a limitation of the proposed method is that the error becomes larger at the edge of the definition region. However, in the IMU sensor signal processing application considered here, such as the gravity acceleration and geomagnetic vectors, the norm as the basis of the frame matrix is not zero, but has a certain magnitude, and the objective is to normalize it. Therefore, it is a suitable target as an application example for the proposed method. The structure of the equation remains the same, and an integer conversion with $N = 7$ yields the following results:

$$\begin{aligned}
 \text{First approx. of sine function} &: s_i(\phi) = \phi(180 - |\phi|) \gg 6 \\
 \text{Second approx. of sine function} &: s_{i2}(\phi) = 29(440 - |s_i(\phi)|)s_i(\phi) \gg 14 \\
 \text{Second approx. of cosine function} &: c_{i2}(\phi) = s_{i2}(90 - |\phi|) \\
 \text{First approx. of atan2 function} &: \phi_1 = at2_i(y, x) = 41(283(127 - x)\text{sgn}(y) + xy) \gg 14 \\
 \text{Second approx. of atan2 function} &: \phi_2 = at2_{i2}(y, x) = \phi_1 + ((x(s_{i2}(\phi_1) + 1) - yc_{i2}(\phi_1)))
 \end{aligned}$$

This encodes the angle as an intuitively understandable integer from -180 to 180 degrees. The process flow of this calculation is shown in Figure 12. The details of the algorithm are the same as those provided in [13], except that the input and output are integers. Even if all approximations are made using integers (Figure 13a), the estimation error is within the quantization error, because it is less than or equal to 1° , although the resolution is increased for evaluation. The results with floating-point numbers, which have been reported in a previous publication [13], are shown in Figure 13b. The maximum error in the floating-point calculation was about 0.001 rad ($=0.05^\circ$), while the integer conversion resulted in an error approximately equal to 1 degree. This calculation is approximately three times faster than that of “math.h”, even when it is performed on a CPU with an FPU. Therefore, fast

calculations can be expected even if a microcontroller without an FPU or compiler with low optimization capability is used.

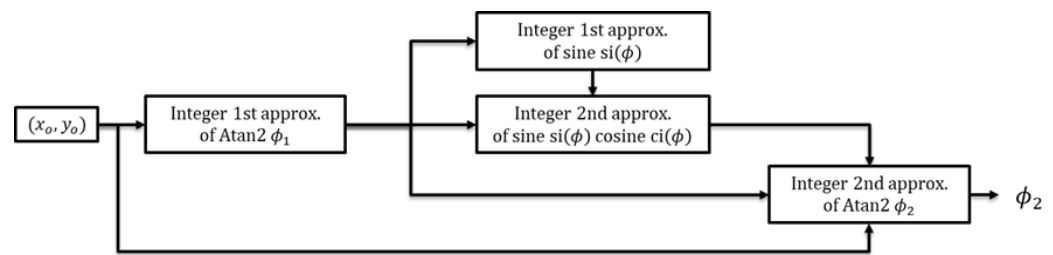


Figure 12. Fast calculation of atan2 using integer inputs.

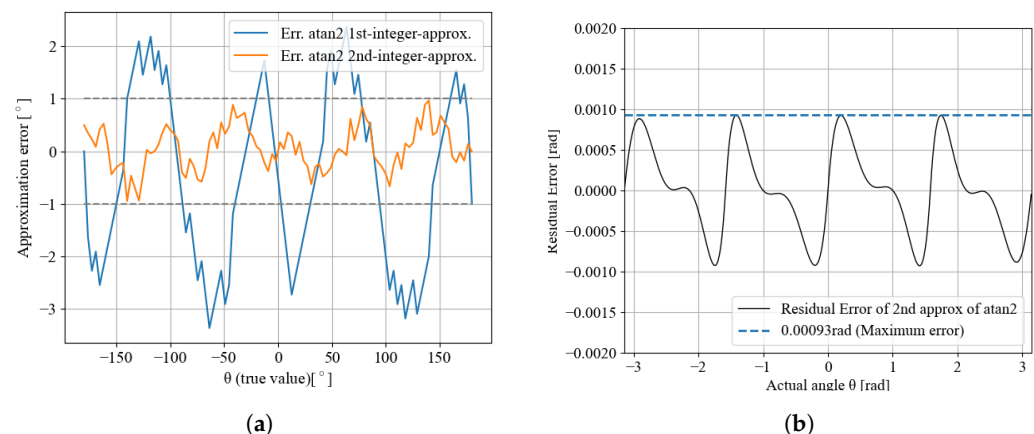


Figure 13. Accuracy comparison between integer and floating point approximations. (a) Accuracy responses of integer atan2 approximations. (b) Accuracy responses of floating-point atan2 approximations [13].

We applied integer pseudo-normalization as a preprocessor for a previously developed algorithm. However, it can be applied to various preprocessors that require fast normalization. For example, the proposed algorithm may be effective when performing a large number of calculations on GPUs, such as computing normal vectors for computer graphics and normalization for machine learning. Especially in the field of machine learning, implementation techniques on FPGAs are important for efficient learning and edge AI [37]. Normalization, division, and square roots are basic elements used to calibrate various calculations. We are currently discussing their application to the field of machine learning. In future work, we will work on the improvement of the efficiency of machine learning implementation by applying the proposed method.

4. Conclusions

We propose pseudo-normalization as a vector normalization method for fixed-point numbers that used only multiply–add operations and bit shifts. Pseudo-normalization consists of the approximate fast computation of $1/x$ and \sqrt{x} , which can also be used independently.

The approximate fast computation of $1/x$ can be expanded by n iterations, and it is analytically equivalent to the original function at $n \rightarrow \infty$. The proposed multiplicative approximation method converges more efficiently than the series approximation method. Thus, only one or more iterations are required to create a sufficiently broad domain of definition for embedded systems. To verify the feasibility of implementing the algorithm in a FPGA, the algorithm is verified using Verilog and simulations. As an example of applying the proposed algorithm, we confirm that the previously developed fast computation of inverse trigonometric functions can be combined with the proposed algorithm to perform fast and accurate approximate computation by integer transformation.

Future studies will include the construction of a high-speed sensor signal processing system using an actual FPGA device and its application to high-precision human motion measurement with high-temporal resolution. In addition, we will develop a fast computation method for nonlinear elementary functions, such as exponential and logarithmic functions, using the residual correction method. By developing an environment in which these basic operations can be efficiently processed in hardware, we will work to realize large-scale advanced technologies such as machine learning on edge devices.

Author Contributions: Conceptualization, T.K.; investigation, T.K.; methodology, T.K.; supervision, T.T.; validation, T.K. and T.T.; writing—original draft, T.K.; writing—review and editing, T.T. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data are contained within the article.

Conflicts of Interest: The authors declare no conflicts of interest.

References

- Leon, V.; Hanif, M.A.; Armeniakos, G.; Jiao, X.; Shafique, M.; Pekmestzi, K.; Soudris, D. Approximate Computing Survey, Part I: Terminology and Software & Hardware Approximation Techniques. *arXiv* **2023**, arXiv:2307.11124.
- Leon, V.; Hanif, M.A.; Armeniakos, G.; Jiao, X.; Shafique, M.; Pekmestzi, K.; Soudris, D. Approximate Computing Survey, Part II: Application-Specific & Architectural Approximation Techniques and Applications. *arXiv* **2023**, arXiv:2307.11128.
- Mittal, S. A Survey of Techniques for Approximate Computing. *ACM Comput. Surv.* **2016**, *48*, 62. [CrossRef]
- Booming AI Demand Threatens Global Electricity Supply. Available online: <https://www.ft.com/content/b7570359-f809-49ce-8cd5-9166d36a057b> (accessed on 22 April 2024).
- The Hidden Trap of GPU-Based AI: The Von Neumann Bottleneck. Available online: <https://www.linkedin.com/pulse/hidden-trap-gpu-based-ai-von-neumann-bottleneck-dr-eric-woodell-n9m9e> (accessed on 13 May 2024).
- Lu, C.H.; Lin, C.S.; Chao, H.L.; Shen, J.S.; Hsiung, P.A. Reconfigurable Multi-core Architecture—A Plausible Solution to the Von Neumann Performance Bottleneck. In Proceedings of the 2013 IEEE 7th International Symposium on Embedded Multicore Socs, Tokyo, Japan, 26–28 September 2013; pp. 159–164. [CrossRef]
- Montuschi, P.; Chang, Y.H.; Piuri, V. In-Memory Computing: The Emerging Computing Topic in the Post-von Neumann Era. *Computer* **2023**, *56*, 4–6. [CrossRef]
- Cococcioni, M.; Rossi, F.; Ruffaldi, E.; Saponara, S. Fast Approximations of Activation Functions in Deep Neural Networks when using Posit Arithmetic. *Sensors* **2020**, *20*, 1515. [CrossRef]
- Romeric. fastapprox: Approximate and Vectorized Versions of Common Mathematical Functions. Available online: <https://github.com/romeric/fastapprox> (accessed on 22 April 2022).
- Tsmots, I.; Skorokhoda, O.; Rabyk, V. Hardware Implementation of Sigmoid Activation Functions using FPGA. In Proceedings of the 2019 IEEE 15th International Conference on the Experience of Designing and Application of CAD Systems (CADSM), Polyana, Ukraine, 26 February–2 March 2019; pp. 34–38. [CrossRef]
- Bouguezzi, S.; Faiedh, H.; Souani, C. Hardware Implementation of Tanh Exponential Activation Function using FPGA. In Proceedings of the 2021 18th International Multi-Conference on Systems, Signals Devices (SSD), Monastir, Tunisia, 22–25 March 2021; pp. 1020–1025. [CrossRef]
- Zhang, H.; Putic, M.; Lach, J. Low power GPGPU computation with imprecise hardware. In Proceedings of the 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC), San Francisco, CA, USA, 1–5 June 2014; pp. 1–6.
- Kusaka, T.; Tanaka, T. Fast and Accurate Approximation Methods for Trigonometric and Arctangent Calculations for Low-Performance Computers. *Electronics* **2022**, *11*, 2285. [CrossRef]
- Kusaka, T.; Tanaka, T. Stateful Rotor for Continuity of Quaternion and Fast Sensor Fusion Algorithm Using 9-Axis Sensors. *Sensors* **2022**, *22*, 7989. [CrossRef]
- Tagliabue, A.; How, J.P. Efficient Deep Learning of Robust Policies from MPC using Imitation and Tube-Guided Data Augmentation. *arXiv* **2023**, arXiv:2306.00286.
- Imani, M.; Garcia, R.; Huang, A.; Rosing, T. CADE: Configurable Approximate Divider for Energy Efficiency. In Proceedings of the 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE), Florence, Italy, 25–29 March 2019; pp. 586–589. [CrossRef]
- Liu, W.; Li, J.; Xu, T.; Wang, C.; Montuschi, P.; Lombardi, F. Combining Restoring Array and Logarithmic Dividers into an Approximate Hybrid Design. In Proceedings of the 2018 IEEE 25th Symposium on Computer Arithmetic (ARITH), Amherst, MA, USA, 25–27 June 2018; pp. 92–98. [CrossRef]
- Saadat, H.; Javaid, H.; Parameswaran, S. Approximate Integer and Floating-Point Dividers with Near-Zero Error Bias. In Proceedings of the 2019 56th ACM/IEEE Design Automation Conference (DAC), Las Vegas, NV, USA, 2–6 June 2019; pp. 1–6.

19. Vahdat, S.; Kamal, M.; Afzali-Kusha, A.; Pedram, M.; Navabi, Z. TruncApp: A truncation-based approximate divider for energy efficient DSP applications. In Proceedings of the Design, Automation & Test in Europe Conference & Exhibition (DATE), Swisstech, Lausanne, Switzerland, 27–31 March 2017; pp. 1635–1638. [\[CrossRef\]](#)
20. Zendegani, R.; Kamal, M.; Fayyazi, A.; Afzali-Kusha, A.; Safari, S.; Pedram, M. SEERAD: A high speed yet energy-efficient rounding-based approximate divider. In Proceedings of the 2016 Design, Automation & Test in Europe Conference & Exhibition (DATE), Dresden, Germany, 14–18 March 2016; pp. 1481–1484.
21. Divide and Conquer: Arm Cores and Division—Architectures and Processors Blog—Arm Community Blogs—Arm Community. 2014. Available online: <https://community.arm.com/arm-community-blogs/b/architectures-and-processors-blog/posts/divide-and-conquer> (accessed on 22 April 2024).
22. How to Perform Faster Mathematical Calculation in Cortex-M0+ Microcontrollers 2017. Available online: <https://ww1.microchip.com/downloads/aemDocuments/documents/OTH/ProductDocuments/SupportingCollateral/90003178A.pdf> (accessed on 22 April 2024).
23. Lomont, C. Fast Inverse Square Root. Tech-315 Nical Rep. 2003. Volume 32. Available online: <http://www.matrix67.com/data/InvSqrt.pdf> (accessed on 5 July 2022).
24. Robertson, M. A Brief History of InvSqrt. Bachelor's Thesis, University of New Brunswick, Fredericton, NB, Canada, 2012.
25. id-Software/Quake-III-Arena, 2022. Original-Date: 2012-01-31T19:39:13Z. Available online: https://www.pcgamingwiki.com/wiki/Quake_III_Arena (accessed on 24 April 2022).
26. Eberly, D.H. *GPGPU Programming for Games and Science*; A K Peters: Natick, MA, USA; CRC Press: New York, NY, USA, 2014. [\[CrossRef\]](#)
27. Moroz, L.V.; Samotyy, V.V.; Horyachyy, O.Y. Modified Fast Inverse Square Root and Square Root Approximation Algorithms: The Method of Switching Magic Constants. *Computation* **2021**, *9*, 21 [\[CrossRef\]](#)
28. Walczyk, C.J.; Moroz, L.V.; Cieśliński, J.L. Improving the Accuracy of the Fast Inverse Square Root by Modifying Newton–Raphson Corrections. *Entropy* **2021**, *23*, 86. [\[CrossRef\]](#)
29. Hasnat, A.; Bhattacharyya, T.; Dey, A.; Halder, S.; Bhattacharjee, D. A fast FPGA based architecture for computation of square root and Inverse Square Root. In Proceedings of the 2017 Devices for Integrated Circuit (DevIC), Kalyani, India, 23–24 March 2017; pp. 383–387. [\[CrossRef\]](#)
30. Warren, H.S. *Hacker's Delight*, 2nd ed.; Addison-Wesley Professional: Boston, MA, USA, 2012; ISBN 9780133084993.
31. Kusaka, T.; Tanaka, T.; Kajiwar, H. Residual Correction Method for Fast Calculation of Arctangent in Embedded Systems. In Proceedings of the IEEE/ASME International Conference on Advanced Intelligent Mechatronics, Busan, Republic of Korea, 7–11 July 2015; p. 6.
32. Materzok, M. DigitalJS: A Visual Verilog Simulator for Teaching. In Proceedings of the 8th Computer Science Education Research Conference, CSERC'19, New York, NY, USA, 19–20 October 2020; pp. 110–115. [\[CrossRef\]](#)
33. Ugurdag, H.F.; Bayram, A.; Levent, V.E.; Goren, S. Efficient Combinational Circuits for Division by Small Integer Constants. In Proceedings of the 2016 IEEE 23rd Symposium on Computer Arithmetic (ARITH), Silicon Valley, CA, USA, 10–13 July 2016.
34. 9.4. ALTFP_INV_SQRT Resource Utilization and Performance. Available online: https://cdrdv2-public.intel.com/666430/ug_altfp_mfug-683750-666430.pdf (accessed on 8 July 2024).
35. Knittel, G. Proven-prompt vector normalizer. In Proceedings of the Sixth Annual IEEE International ASIC Conference and Exhibit, Rochester, NY, USA, 27 September–1 October 1993; pp. 112–115. [\[CrossRef\]](#)
36. Huang, Z.; Ercegovac, M. FPGA Implementation of Pipelined On-Line Scheme for 3-D Vector Normalization. In Proceedings of the 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines (FCCM'01), Rohnert Park, CA, USA, 29 April–2 May 2001; pp. 61–70.
37. Kashiwagi, R.; Mukaeda, T.; Shima, K. FPGA Implementation of Approximate Gaussian Mixture Model for Open-Set Recognition in Interface Control. In Proceedings of the 10th International Conference on Control Decision and Information Technologies (CoDiT2024), Valletta, Malta, Italy, 1–4 July 2024.

Disclaimer/Publisher's Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.