mathematics

MDPI

Article

Accurate Sum and Dot Product with New Instruction for High-Precision Computing on ARMv8 Processor

Kaisen Xie [1,2], Qingfeng Lu [1], Hao Jiang [1,*] and Hongxia Wang [2]

[1] College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China; xiekaisen@nudt.edu.com (K.X.); 15211061615@163.com (Q.L.)
[2] College of Science, National University of Defense Technology, Changsha 410073, China; wanghongxia@nudt.edu.cn
* Correspondence: haojiang@nudt.edu.com

Abstract: The accumulation of rounding errors can lead to unreliable results. Therefore, accurate and efficient algorithms are required. A processor from the ARMv8 architecture has introduced new instructions for high-precision computation. We have redesigned and implemented accurate summation and the accurate dot product. The number of floating-point operations has been reduced from $7n-5$ and $10n-5$ to $4n-2$ and $7n-2$, compared with the classic compensated precision algorithms. It has been proven that our accurate summation and dot algorithms' error bounds are $\gamma_{n-1}\gamma_n\text{cond}+u$ and $\gamma_n\gamma_{n+1}\text{cond}+u$, where 'cond' denotes the condition number, $\gamma_n = n\cdot u/(1-n\cdot u)$, and u denotes the relative rounding error unit. Our accurate summation and dot product achieved a 1.69× speedup and a 1.14× speedup, respectively, on a simulation platform. Numerical experiments also illustrate that, under round-towards-zero mode, our algorithms are as accurate as the classic compensated precision algorithms.

Keywords: compensated precision; accurate summation; accurate dot product; error analysis; error-free transformation

MSC: 65G50



Academic Editor: Andrea Scozzari

Received: 12 December 2024
Revised: 9 January 2025
Accepted: 10 January 2025
Published: 15 January 2025

Citation: Xie, K.; Lu, Q.; Jiang, H.; Wang, H. Accurate Sum and Dot Product with New Instruction for High-Precision Computing on ARMv8 Processor. *Mathematics* **2025**, *13*, 270. https://doi.org/10.3390/math13020270

Copyright: © 2025 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

When computations have reached the maximum precision limit of the hardware or programming language and demand even greater accuracy, it is imperative to consider alternative methods, such as compensated summation and compensated dot. However, these algorithms will take more FLOPs.

After rounding, computers round away the significant bits that exceed the precision limit, often referred to as the 'rounding error'. Compensated summation and compensated dot can reduce the rounding error but increase the FLOPs. The capacity to directly extract the rounding error has the potential to significantly reduce the FLOPs of accurate algorithms. This has led to the introduction of several high-precision instructions into the processors of the ARMv8 architecture.

Compensated summation and compensated dot, which traditionally utilize add, subtract, and multiplication instructions to compute the rounding error, are able to leverage new instructions for acceleration. Hence, there is a clear need for the development of novel algorithms that harness advanced instructions for accurate summation and dot product operations on the ARMv8 processor.

1.1. Previous Work

The inaugural compensated algorithm, FastTwoSum, was introduced by Kahan in 1965 [1]. This pioneering method effectively computes the sum of two floating-point numbers, concurrently accounting for their rounding errors. However, FastTwoSum is subject to constraints regarding the relative magnitudes of the operands, which necessitates the introduction of branching conditions. In 1970, Knuth proposed the TwoSum algorithm [2] to circumvent the impact of the branching conditions and enhance the computational efficiency of compensated summation on real platforms. Unlike its predecessor, TwoSum can accurately ascertain the rounding error without regard to the absolute values of the operands, a and b. This algorithm eschews branching and comprises six floating-point operations. When considering the computation of absolute values and comparisons as a single floating-point operation, the TwoSum algorithm demonstrates a 50% increase in speed over FastTwoSum when the number of floating-point operations is held constant, attributed to the elimination of branching conditions. In 2017, Boldo and Graillat revealed that these algorithms possess greater robustness than is commonly acknowledged [3].

The TwoProduct algorithm [4], proposed by Dekker, is designed to accurately calculate the rounding error associated with the product of two floating-point numbers. This method exhibited a marked enhancement in efficiency following acceleration with the fused multiply–add (fma) instruction. In 2020, Graillat introduced algorithms capable of computing "exact" products through arithmetic operations, all rounded either toward $-\infty$ or $+\infty$ [5]. This approach underscores the significance of employing specific instructions and alternative rounding models in computational practice.

Subsequent research in precision control has been dedicated to enhancing the velocity and parallelism of compensation summation algorithms. In 2005, Oishi, Ogita, and Rump first introduced the concept of error-free transformation [6]. Building upon the foundational work of Dekker and Knuth, Oishi developed a method for the accurate computation of the sum and dot product (inner product) of a sequence of floating-point numbers in 2008 [7,8]. Numerical results demonstrated that the efficiency in achieving the same level of precision was significantly improved compared to algorithms reliant on double–double improvements. In the same year, Yamanaka crafted a parallel algorithm for the accurate computation of the dot product (inner product) [9]. In 2011, K. Ozaki proposed an error-free transformation approach for matrix multiplication [10]. The following year, Graillat employed similar techniques to design and implement compensated precision summation, dot product, and polynomial algorithms within the complex domain [11]. In 2013, Kadric demonstrated the utilization of tree-reduce parallelism to compute correctly rounded floating-point sums in O(log(n)) depth [12], while Demmel proposed a technique for floating-point summation that ensured reproducibility, independent of the order of summation [13]. In 2014, Abdalla leveraged the single instruction multiple data (SIMD) paradigm found in modern CPUs to accelerate matrix multiplication calculations [14]. In 2015, Jankovic presented a solution for the accurate summation problem using a fixed-point accumulator [15]. The year 2016 saw Goodrich provide several efficient parallel algorithms for the summing of floating-point numbers, aimed at producing a faithfully rounded floating-point representation of the sum [16]. In 2022, Evstigneev improved parallel algorithms based on the work of Rump, Ogita, and Oishi [6–8], achieving higher precision in floating-point reduction-type operations for graphics processing units (GPUs) [17]. In the same year, Lange proposed the aaaSum algorithm [18], which exploits ideas from Zhu and Hayes' OnlineExactSum [19]. Some research has been anchored in specific high-performance computer platforms, such as the ARMv8 platform. In 2017, Jiang designed, implemented, and optimized quad-precision (double-double) dense matrix multiplication (QGEMM) [20] based on the domestic ARMv8 64-bit multi-core processor, applying it to large-scale computations.

As reliability is paramount in compensation algorithms, a rigorous analysis of the error bounds is indispensable. Between 2017 and 2019, Lange, drawing on the study of standard Wilkinson-type error estimates for floating-point algorithms, developed novel a priori estimates for the summation of real numbers, where each sum is subject to perturbations [21,22]. In 2021, Muller demonstrated the implementation of a new "rounding direction" using current operations [23]. The following year, he utilized formal proofs to significantly refine the error bounds and generalize certain results, showing their validity under slight modifications to the rounding mode [24]. In 2023, they explored functions for which correctly rounded implementations are available, enabling the use of these functions for floating-point multiplication or division with tight error bounds measured in units in the last place (ulps) [25]. Recently, Hubrecht revealed that the use of fused arithmetic operators can enable more accurate calculations with smaller costs than traditional operations alone [26].

In this paper, we design more efficient algorithms for accurate summation and dot product operations on an ARMv8 architecture processor at the FPGA simulation stage. This processor is being designed and developed by the National University of Defense Technology. The architecture of this processor is similar to those of other ARMv8 processors, except that some instructions (such as the ones mentioned in this article) are added due to requirements. We provide the error bounds for our algorithms, along with their proofs, and describe numerical experiments on the FPGA simulation platform to assess the accuracy and performance of our algorithms.

1.2. Background

1.2.1. Notation

The working precision of our algorithms is double, and the rounding mode is round-towards-zero. All floating-point representations and operations follow the IEEE 754 floating-point standard [27]. It is assumed that overflow will not occur, but underflow is permitted. We denote the set of all floating-point numbers at this precision as $\mathbb{F}$. The relative rounding error unit is denoted as u, and the unit of underflow is denoted as $elow$. Double precision has 53 significant bits and an 11-bit exponent; thus, $elow = 2^{-1074}$. When using the round-to-nearest mode, $u = 2^{-53}$, and with the round-towards-zero mode, $u = 2^{-52}$. Since the additional instructions only support round-towards-zero, in subsequent proofs, the value of u is 2^{-52}.

$\mathrm{fl}(\circ)$ represents the result of floating-point computation, where $\circ$ denotes all operations performed in the working precision. This paper will employ parentheses to uniquely determine the order of computation. According to the IEEE 754 standard, the backward error for a single floating-point operation satisfies [27]

$$\begin{aligned} \mathrm{fl}(a \circ b) &= (a \circ b)(1+\epsilon)^{\pm 1}, \\ \circ &\in \{+,-\}, \text{ and } |\epsilon| \le u; \end{aligned} \tag{1}$$

and

$$\begin{aligned} \mathrm{fl}(a \circ b) &= (a \circ b)(1+\epsilon)^{\pm 1} + \eta, \\ \circ &\in \{\cdot, /\}, \text{ and } |\epsilon| \le u, |\eta| \le elow \end{aligned} \tag{2}$$

This means that, for $\circ \in \{+,-\}$, the forward error satisfies

$$\begin{aligned} |a \circ b - \mathrm{fl}(a \circ b)| &\le u \cdot |a \circ b| \\ |a \circ b - \mathrm{fl}(a \circ b)| &\le u \cdot |\mathrm{fl}(a \circ b)| \end{aligned} \tag{3}$$

Meanwhile, for $\circ \in \{\cdot, /\}$, the forward error satisfies

$$\begin{aligned} |a \circ b - \mathrm{fl}(a \circ b)| &\leq u \cdot |a \circ b| \\ |a \circ b - \mathrm{fl}(a \circ b)| &\leq u \cdot |\mathrm{fl}(a \circ b)| \end{aligned} \tag{4}$$

The quantities γ are defined by

$$\gamma_n := n \cdot u/(1 - n \cdot u), n \cdot u < 1 \tag{5}$$

The following lemma is proven in [28].

Lemma 1 ([28]). *For floating-point numbers $a_i \in \mathbb{F}, 1 \leq i \leq n$, the following inequality is true:*

$$|\mathrm{fl}(\sum_{i=1}^{n} a_i) - \sum_{i=1}^{n} a_i| \leq \gamma_{n-1} \sum_{i=1}^{n} |a_i| \tag{6}$$

When using γ_n, we assume that $n \cdot u < 1$ by default, and it will not be explicitly mentioned in subsequent expressions.

1.2.2. Error-Free Transformation

The compensated algorithm needs to extend the error-free transformation of two floating-point numbers to vector summation and dot product. Therefore, the following content will help in understanding this work. For the error-free transformation of addition, Knuth [2] proposed the TwoSum algorithm (Algorithm 1).

Algorithm 1 TwoSum [2]

function $[x, y] = \mathrm{TwoSum}(a, b)$
$x = \mathrm{fl}(a + b)$
$z = \mathrm{fl}(x - a)$
$y = \mathrm{fl}((a - (x - z)) + (b - z))$

Algorithm 1 converts two inputs $a, b \in \mathbb{F}$ into two outputs $x, y \in \mathbb{F}$, satisfying

$$a + b = x + y \text{ and } x = \mathrm{fl}(a + b). \tag{7}$$

Since addition and subtraction are exact in this case, the proof in [2] also applies to cases of underflow.

For the error-free transformation of multiplication, the algorithm TwoProductFMA (Algorithm 2) is currently the best solution, because a Fused-Multiply-and-Add operation is available on the ARMv8 processor.

Algorithm 2 TwoProductFMA [6]

function $[x, y] = \mathrm{TwoProductFMA}(a, b)$
$x = \mathrm{fl}(a \cdot b)$
$y = \mathrm{fl}(a \cdot b - x)$

Algorithm 2 has no branches, with only a basic and optimizable sequence of floating-point operations. In the absence of underflow, the following equation is confirmed:

$$a \cdot b = x + y \text{ and } x = \mathrm{fl}(a \cdot b). \tag{8}$$

Therefore, the properties of the TwoSum and TwoProductFMA algorithms are as follows.

Theorem 1 ([2]). *If $a, b \in \mathbb{F}$, the result of Algorithm 1 (TwoSum) is x, y, even in the case of underflow:*

$$a + b = x + y, \quad x = \mathrm{fl}(a + b),$$
$$|y| \leq u \cdot |x|, \quad |y| \leq u \cdot |a + b|.$$

Algorithm 1 requires 6 floating-point operations. If $a, b \in \mathbb{F}$, the result of Algorithm 2 (TwoProductFMA) is x, y. If underflow does not occur, then

$$a \cdot b = x + y, \quad x = \mathrm{fl}(a \cdot b),$$
$$|y| \leq u \cdot |x|, \quad |y| \leq u \cdot |a \cdot b|,$$

In the case of underflow, the equation $a \cdot b = x + y + 5\eta, x = \mathrm{fl}(a \cdot b), |y| \leq u \cdot |x| + 5$ elow, $|y| \leq u \cdot |a \cdot b| + 5$ elow holds. Here, η satisfies $|\eta| \leq$ elow. Algorithm 2 requires 2 floating-point operations.

1.2.3. Compensated Summation and Dot

The compensated accurate sum (Algorithm 3) can be viewed as the vector form of TwoSum and also satisfies the condition of error-free transformation.

Algorithm 3 CompensatedSum [6]

function Res = CompensatedSum($\mathbf{a}$)
$\pi_1 = a_1, r_1 = 0$
for $i = 2$ to n **do**
  $[\pi_i, y_i] = \mathrm{TwoSum}(\pi_{i-1}, a_i)$
  $r_i = \mathrm{fl}(r_{i-1} + y_i)$
end for
Res $= \pi_n + r_n$

Ref. [6] showed the relative error bound of Algorithm 3 and proposed the following theorem.

Theorem 2 ([6]). *The variables in Algorithm 3 algorithm are all double-precision floating-point, and the rounding mode is round-towards-zero. The forward error of the result is $|E_n| = |\mathrm{Res} - s_n|$, where $s_n = \sum_{i=1}^{n} a_i$, $S_n = \sum_{i=1}^{n} |a_i|$, and the following inequality holds:*

$$|E_n| / |s_n| \leq \gamma_{n-1}^2 \mathrm{cond}(\sum_{i=1}^{n} a_i) + u. \tag{9}$$

where $\mathrm{cond}(\sum_{i=1}^{n} a_i) = S_n / |s_n|$.

The compensated accurate dot product (Algorithm 4) can be seen as a combination of Algorithms 2 and 3.

Algorithm 4 CompensatedDot [6]

function Res = CompensatedDot($\mathbf{a}, \mathbf{b}$)
$[\pi_1, p_1] = \mathrm{TwoProductFMA}(a_1, b_1)$
for $i = 2$ to n **do**
  $[h_i, l_i] = \mathrm{TwoProductFMA}(a_i, b_i)$
  $[\pi_i, r_i] = \mathrm{TwoSum}(\pi_{i-1}, h_i)$
  $p_i = \mathrm{fl}(p_{i-1} + (r_i + l_i))$
end for
Res $= \pi_n + p_n$

To show the relative error bound of Algorithm 4, ref. [6] also proposed the following theorem.

Theorem 3 ([6]). *The variables in Algorithm 3 algorithm are all double-precision floating-point, and the rounding mode is round-towards-zero. The forward error of the result is* $|E_n| = |\mathrm{Res} - \mathbf{a}^T \cdot \mathbf{b}|$, *where* $s_n = \mathbf{a}^T \cdot \mathbf{b}$, $S_n = |\mathbf{a}^T||\mathbf{b}|$, *and the following inequality holds:*

$$|E_n|/\mathbf{a}^T \cdot \mathbf{b} \leq \gamma_{n-1}^2 \mathrm{cond}(\mathbf{a}^T \cdot \mathbf{b}) + u. \tag{10}$$

where $\mathrm{cond}(\sum_{i=1}^{n} a_i) = S_n/|s_n|$.

2. Compensated Sum and Dot Product Algorithms Based on High-Precision Instructions

This section will first introduce the functions and technical details of the faddr and faddn instructions and describe accurate summation and dot with these two instructions. Then, the error bounds of the accurate summation and dot are presented.

2.1. The Faddr and Faddn Instructions

The faddr and faddn instructions appear in pairs and aim to achieve the rounding error of $\mathrm{fl}(a+b)$ with a few floating-point operations. Assuming that no carry occurs in fl(a + b), the functions of faddn and faddr are as shown in Figure 1. The descriptions of the instructions' functionalities are as follows.

FADD R Instruction

FADDR <Dn>, <Dm>, <Dd>

Assembly Symbols

<Dd> is the destination double-precision register.

<Dn> is the first source double-precision register.

<Dm> is the second source double-precision register.

Operation

The faddr instruction can extract the significant bits that are rounded away during floating-point addition fl(a + b), while a in register <Dn>, b in register <Dm>. This instruction adds the corresponding elements from the two registers, extracts the significant bits that are rounded away from the floating-point operation, and stores them in the destination register <Dd>, while taking the sign bit and exponent bits. This instruction is for double-precision operations. In the original addition result, let p denote the result in the destination register <Dd>,

$$p \approx 2^{\lfloor \log_2 |\mathrm{fl}(a+b)| \rfloor - \lfloor \log_2 |a+b-\mathrm{fl}(a+b)| \rfloor} (a + b - \mathrm{fl}(a+b)).$$

FADDN Instruction

FADDN <Dd>

Assembly Symbols

<Dd> is the source double-precision register and the destination double-precision register.

Operation

The faddn instruction will apply the appropriate bias to the exponent bits and normalize the significand of the element in register <Dd>. This instruction is for double-precision operations. Let p denote the result in the destination register <Dd>,

$$p \approx a + b - \mathrm{fl}(a + b).$$

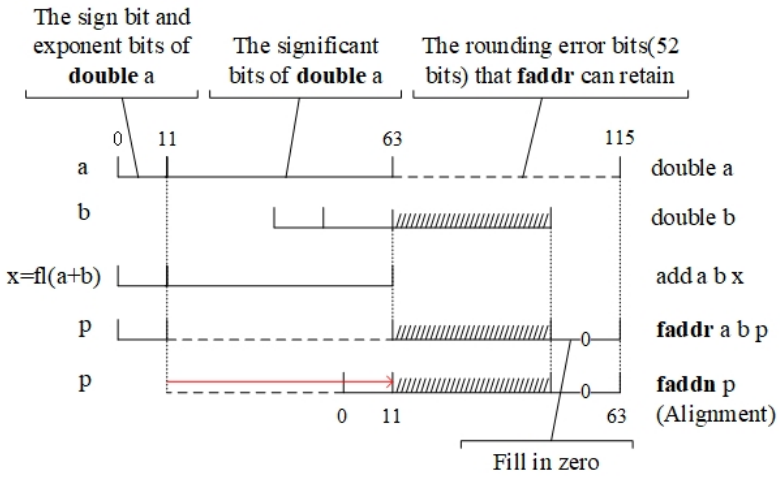

Figure 1. The operation of floating-point addition, addr, and faddn instructions.

Thus, with these two instructions, we can obtain round errors using fewer FLOPs. An example is the following Algorithm 5.

Algorithm 5 Err

function $p = \mathrm{Err}(a, b)$
faddr a b p
faddn p

Here, $x = \mathrm{fl}(a + b)$, and we hope that the result p from the faddr and faddn instructions satisfies Equation (7). However, there is a limitation on the hardware. As shown in Figures 2–5, when the working precision is double precision, $\mathrm{fl}(a + b)$ has 64 bits, with the last 52 bits as the signifcand. When using the faddr instruction to achieve the rounding error of $\mathrm{fl}(a + b)$ and directly round off the error, only the first 52 bits of the rounded significant bits will be preserved. Thus, p is an approximation of the rounding error, which satisfies

$$x + p \approx a + b \tag{11}$$

Under extreme conditions, some or even all of the rounding error will not be saved in p. Therefore, special processing is required when calculating the error bound. Several cases may arise.

Lemma 2. *Without loss of generality, assume $a > b$. Then, the relative size of b compared to $\mathrm{fl}(a + b)$ will affect the outcome of Algorithm 5. In the following four different cases, p has different properties.*

(a) When $|b| \geq 2^{\lfloor \log_2 |x| \rfloor}$, there is no rounding error. p, the rounding error, and zero are equal, as shown in Figure 2.

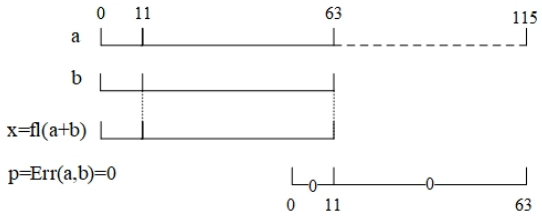

Figure 2. In case of $|b| \geq 2^{\lfloor \log_2 |x| \rfloor}$, p equals zero.

(b) When $|b| < 2^{\lfloor \log_2 |x| \rfloor}$ and $|b| \geq u \cdot 2^{\lfloor \log_2 |x| \rfloor}$, p will retain all rounding errors. Therefore, p equals the rounding error and satisfies Equation (7), as shown in Figure 3.

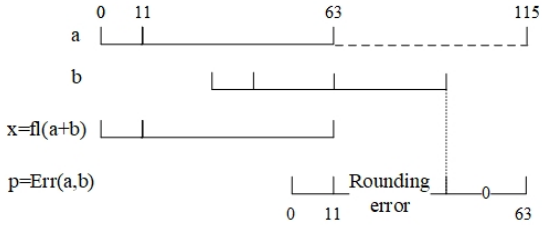

Figure 3. In case of $u \cdot 2^{\lfloor \log_2 |x| \rfloor} \leq |b| < 2^{\lfloor \log_2 |x| \rfloor}$, p equals rounding error.

(c) When $|b| < u \cdot 2^{\lfloor \log_2 |x| \rfloor}$ and $|b| \geq u^2 \cdot 2^{\lfloor \log_2 |x| \rfloor}$, due to the limitation of the instructions, part of the rounding error will be truncated. Therefore, p is the approximation of the rounding error, as shown in Figure 4.

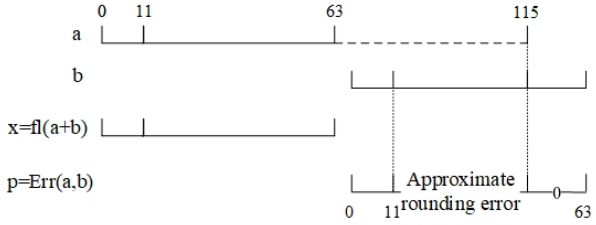

Figure 4. In case of $u^2 \cdot 2^{\lfloor \log_2 |x| \rfloor} \leq |b| < u \cdot 2^{\lfloor \log_2 |x| \rfloor}$, p is approximation of rounding error.

(d) When $|b| < u^2 \cdot 2^{\lfloor \log_2 |x| \rfloor}$, the rounding error is too small to be retained. Therefore, p equals zero, as shown in Figure 5.

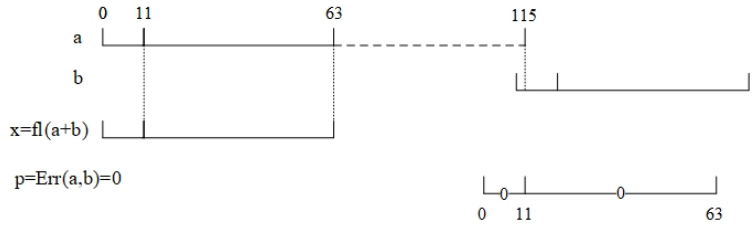

Figure 5. In case of $|b| < u^2 \cdot 2^{\lfloor \log_2 |x| \rfloor}$, p equals zero.

Let $q = a + b - x - p$. In all situations, $|p| \leq u|x|, |q| \leq u^2|x|$.

Proof. Because the rounding mode is rounding-towards-zero, $|p + q| = |p| + |q|$. By Theorem 1,

$$u|x| \geq |p + q| = |p| + |q| \geq |p|$$

In cases (a) and (b), x and p satisfy Equation (7), which means that $q = 0$. In case (c), $|q| \leq u|p| \leq u^2|x|$. In case (d), $|q| = |b| \leq u^2|x|$.

In summary, the lemma is established. □

2.2. Compensated Summation and Dot Product Based on High-Precision Instructions

Algorithm 5 uses these two instructions to obtain the approximations of the rounding error. We take the result of Algorithm 5 as the rounding error. The improved algorithm is presented as Algorithm 6.

Algorithm 6 rnTwoSum

function $[x, p] = \text{rnTwoSum}(a, b)$
$x = \text{fl}(a + b)$
$p = \text{Err}(a, b)$

For floating-point numbers $a_i, i = 1, 2, \ldots n$, $s_n = \sum_{i=1}^{n} a_i$ represents the exact result of the summation. Let $S_n = \sum_{i=1}^{n} |a_i|$, and $\pi_n = \text{fl}(\sum_{i=1}^{n} a_i)$ represents the floating-point sum result. Applying Algorithm 6 at each step of $\sum_{i=1}^{n} a_i$, and then accumulating all the error terms for compensation in the final summation result, we obtain Algorithm 7.

Algorithm 7 rnCompensatedSum

function Res = rnCompensatedSum($\mathbf{a}$)
$\pi_1 = a_1; r_1 = 0$
for $i = 2$ to n **do**
  $[\pi_i, p_i] = \text{rnTwoSum}(\pi_{i-1}, a_i)$
  $r_i = \text{fl}(r_{i-1} + p_i)$
end for
$\text{Res} = \pi_n + r_n$

Combining Algorithms 2 and 7, we achieve the compensated precision dot product based on the high-precision instructions (Algorithm 8).

Algorithm 8 rnCompensatedDot

function Res = CompensatedDot($\mathbf{a}, \mathbf{b}$)
$[\pi_1, r_1] = \text{TwoProductFMA}(a_1, b_1)$
for $i = 2$ to n **do**
  $[h_i, l_i] = \text{TwoProductFMA}(a_i, b_i)$
  $[\pi_i, p_i] = \text{rnTwoSum}(\pi_{i-1}, h_i)$
  $r_i = \text{fl}(r_{i-1} + (p_i + l_i))$
end for
$\text{Res} = \pi_n + r_n$

2.3. Error Analysis

Algorithms 7 and 8 reduce the number of floating-point operations for compensated summation and dot product, thus improving the computational efficiency. However, due to hardware limitations on the floating-point precision, there may be some loss of accuracy. Therefore, the applicability of Algorithms 7 and 8, i.e., whether these two algorithms can be applied in specific cases, depends on the analysis of the rounding error bounds for these algorithms. When the error meets the conditions, due to the efficiency of computation, the priority of these two algorithms will be higher than that of the traditional compensated summation and dot product.

In Algorithm 7, p_i represents the part of the rounding error directly in the signifcand from the 64th to the 115th bit, and let q_i represent the part beyond the 115th bit. Then, $(p_i + q_i)$ can be considered as the rounding error in the addition process. Consequently, we have $s_n = \sum_{i=2}^{n} p_i + \sum_{i=2}^{n} q_i + \pi_n$. The result obtained by accumulating a_i in Algorithm 7 is $res = \text{fl}(\pi_n + p_n) = \text{fl}(\pi_n + (\sum_{i=2}^{n} p_i))$.

To prove the conclusion about the relative error bound of Algorithm 7, we need to introduce a lemma.

Lemma 3. *Given floating-point numbers* $\pi_0,\ a_i \in \mathbb{F}, 1 \le i \le n.$ $\pi_i, p_i \in \mathbb{F}, 1 \le i \le n$, *are the results of the following loop calculation:*

$$[\pi_i, p_i] = rnTwoSum(\pi_{i-1}, a_i), i = 1, ..., n \tag{12}$$

Let $q_i = \pi_{i-1} + a_i - \pi_i - p_i$; *then, the following inequalities are confirmed:*

(a) $|q_i| \le u^2|\pi_i|$, $|p_i| \le u|\pi_i|$;

(b) *When* $\pi_0 = 0$, $\sum_{i=2}^{n}|p_i| \le \gamma_{n-1}\sum_{i=1}^{n}|a_i| = \gamma_{n-1}S_n$; *when* $\pi_0 \ne 0$, $\sum_{i=2}^{n}|p_i| \le \gamma_n\sum_{i=1}^{n}|a_i| = \gamma_n S_n$;

(c) *When* $\pi_0 = 0$, $\sum_{i=2}^{n}|q_i| \le u\gamma_{n-1}\sum_{i=1}^{n}|a_i| = u\gamma_{n-1}S_n$; *when* $\pi_0 \ne 0$, $\sum_{i=2}^{n}|q_i| \le u\gamma_n\sum_{i=1}^{n}|a_i| = u\gamma_n S_n$.

Proof. Due to Lemma 2, (a) is confirmed. To prove (b), we require the following.

First, when $n = 2$, both sides of the inequality are 0. Assume that (b) is true for some $n - 1 \ge 2$; then,

$$\begin{aligned}\sum_{i=2}^{n}|p_i| &= \sum_{i=2}^{n-1}|p_i| + |p_n| \\ &\le \gamma_{n-2}\sum_{i=1}^{n-1}|a_i| + |p_n| = \gamma_{n-2}S_{n-1} + |p_n|.\end{aligned} \tag{13}$$

By Lemmas 1 and 2,

$$|p_n| \le u|\pi_n| \le u(1+\gamma_{n-1})\sum_{i=1}^{n}|a_i| = u(1+\gamma_{n-1})S_n. \tag{14}$$

Combining Equations (13) and (14), we have

$$\begin{aligned}\sum_{i=2}^{n}|p_i| &\le \gamma_{n-2}S_{n-1} + u(1+\gamma_{n-1})S_n \\ &\le S_n(\gamma_{n-2} + u(1+\gamma_{n-1})) \\ &\le \gamma_{n-1}S_n.\end{aligned} \tag{15}$$

Hence, (b) is true. Similarly, (c) is true as well. □

Theorem 4. *All variables in Algorithm 7 are double-precision floating-point numbers, and the rounding mode is round-towards-zero. The absolute error bound of the sum obtained by this algorithm and* s_n *is denoted as* $|E_n| = |\mathrm{Res} - s_n|$. *The following inequality holds:*

$$|E_n|/|s_n| \le \gamma_{n-1}\gamma_n \mathrm{cond}(\sum_{i=1}^{n}a_i) + u. \tag{16}$$

where $\mathrm{cond}(\sum_{i=1}^{n}a_i) = \frac{\sum_{i=1}^{n}|a_i|}{|\sum_{i=1}^{n}a_i|}$.

Proof. In Algorithm 7, define $q_i = \pi_{i-1} + a_i - \pi_i - p_i$. Then, by Equation (1),

$$\begin{aligned} |E_n| &= |\mathrm{fl}(\pi_n + r_n) - s_n| \\ &= |(1+\epsilon)(\pi_n + r_n - s_n) + \epsilon s_n| \\ &= |(1+\epsilon)(\pi_n + \sum_{i=2}^{n} p_i - s_n) + (1+\epsilon)(p_n - \sum_{i=2}^{n} p_i) + \epsilon s_n| \\ &= |(1+\epsilon)\sum_{i=2}^{n} q_i + (1+\epsilon)(p_n - \sum_{i=2}^{n} p_i) + \epsilon s_n|. \end{aligned} \tag{17}$$

In Equation (1), $|\epsilon| \leq u$

$$|E_n| \leq (1+u)(|\sum_{i=2}^{n} q_i| + |p_n - \sum_{i=2}^{n} p_i|) + u|s_n|. \tag{18}$$

By Lemmas 1 and 3 we have

$$|p_n - \sum_{i=2}^{n} p_i| \leq \gamma_{n-2} \sum_{i=2}^{n} |p_i| \leq \gamma_{n-2} \cdot \gamma_{n-1} \sum_{i=2}^{n} |a_i|.$$

Therefore, combining inequality (18), we obtain

$$\begin{aligned} |E_n| &\leq (1+u)(|\sum_{i=2}^{n} q_i| + \gamma_{n-2} \sum_{i=2}^{n} |p_i|) + u|s_n| \\ &\leq (1+u)|\sum_{i=2}^{n} q_i| + (1+u)\gamma_{n-2}\gamma_{n-1}S_n + u|s_n|. \end{aligned} \tag{19}$$

It is not difficult to see that $(1+u)\gamma_{n-2} \leq \gamma_{n-1}$, and, by Lemma 3, $|\sum_{i=2}^{n} q_i| \leq \sum_{i=2}^{n} |q_i| \leq u\gamma_{n-1}S_n$. Therefore,

$$|E_n| \leq (1+u)u\gamma_{n-1}S_n + \gamma_{n-1}^2 S_n + u|s_n| = (\gamma_{n-1} + u + u^2)\gamma_{n-1}S_n + u|s_n|. \tag{20}$$

In Equation (20),

$$\gamma_{n-1} + u + u^2 < \gamma_{n-1} + u + u^2 + u\gamma_{n-1} \leq \gamma_{n-1} + u + u\gamma_n \leq \gamma_n, \tag{21}$$

thus,

$$|E_n|/|s_n| \leq \gamma_{n-1}\gamma_n S_n/|s_n| + u = \gamma_{n-1}\gamma_n \mathrm{cond}(\sum_{i=1}^{n} a_i) + u. \tag{22}$$

□

Theorem 4 shows that applying the error storage instruction in Algorithm 7 will affect the error bound of the final summation result. Similar conclusions hold for Algorithm 8.

Theorem 5. *All variables in Algorithm 8 are double-precision floating-point numbers, and the rounding mode is round-towards-zero. The absolute error bound of the algorithm's result with respect to s_n is denoted as $|E_n| = |\mathrm{Res} - \mathbf{a}^T\mathbf{b}|$; then, the following inequality holds:*

$$|E_n|/|\mathbf{a}^T\mathbf{b}| \leq \gamma_n\gamma_{n+1}\mathrm{cond}(\mathbf{a}^T\mathbf{b}) + u. \tag{23}$$

where $\mathrm{cond}(\mathbf{a}^T\mathbf{b}) = \frac{|\mathbf{a}^T||\mathbf{b}|}{|\mathbf{a}^T\mathbf{b}|}$.

Proof. In Algorithm 8, define $q_i = \pi_{i-1} + h_i - \pi_i - p_i$. From Equations (1), (2), and (4), we have

$$\begin{aligned} |E_n| &= |\mathrm{fl}(\pi_n + r_n) - \mathbf{a}^T\mathbf{b}| \\ &= |(1+\epsilon)(\pi_n + r_n) - \mathbf{a}^T\mathbf{b}| \\ &= |\epsilon\mathbf{a}^T\mathbf{b} + (1+\epsilon)(\pi_n + r_n - \mathbf{a}^T\mathbf{b})| \\ &\leq (1+u)|\pi_n + r_n - \mathbf{a}^T\mathbf{b}| + u|\mathbf{a}^T\mathbf{b}|. \end{aligned} \tag{24}$$

Moreover, h_i, l_i, π_i, r_i for $2 \leq i \leq n$ are intermediate variables after executing Algorithm 8. Combining the definition of q_i, we have the following equation:

$$p_i + q_i + l_i = (\pi_{i-1} + h_i - \pi_i) + (a_i b_i - h_i) = a_i b_i + \pi_{i-1} - \pi_i. \tag{25}$$

Then,

$$r_1 + \sum_{i=2}^{n}(p_i + q_i + l_i) = (a_1 b_1 - \pi_1) + (\sum_{i=2}^{n} a_i b_i + \pi_1 - \pi_n) = \mathbf{a}^T\mathbf{b} - \pi_n. \tag{26}$$

Substituting Equation (26) into Equation (24), we obtain

$$\begin{aligned} |E_n| &\leq (1+u)|r_n - r_1 - \sum_{i=2}^{n}(p_i + q_i + l_i)| + u|\mathbf{a}^T\mathbf{b}| \\ &= (1+u)|\mathrm{fl}(r_1 + \sum_{i=2}^{n}(p_i + l_i)) - r_1 - \sum_{i=2}^{n}(p_i + q_i + l_i)| + u|\mathbf{a}^T\mathbf{b}| \\ &\leq (1+u)|\mathrm{fl}(r_1 + \sum_{i=2}^{n}(p_i + l_i)) - r_1 - \sum_{i=2}^{n}(p_i + l_i)| + (1+u)\sum_{i=2}^{n}|q_i| + u|\mathbf{a}^T\mathbf{b}|. \end{aligned} \tag{27}$$

where, from Lemma 1,

$$\begin{aligned} |r_1 + \sum_{i=2}^{n}(p_i + l_i) - \mathrm{fl}(r_n + \sum_{i=2}^{n}(p_i + l_i))| &\leq \gamma_{n-1}(|r_1| + \sum_{i=2}^{n}|\mathrm{fl}(p_i + l_i)|) \\ &\leq \gamma_n(|r_1| + \sum_{i=2}^{n}|p_i + l_i|). \end{aligned} \tag{28}$$

From Equation (2) and Lemma 3, the following inequalities are confirmed:

$$\begin{aligned} &|r_1| \leq u|a_1 b_1|, \\ &\sum_{i=2}^{n}|l_i| \leq u\sum_{i=2}^{n}|a_i b_i|, \\ &\sum_{i=2}^{n}(|p_i| + |q_i|) \leq \gamma_{n-1}(|\pi_1| + \sum_{i=2}^{n}|h_i|) = \gamma_{n-1}\sum_{i=1}^{n}|\mathrm{fl}(a_i \cdot b_i|) \leq (1+u)\gamma_{n-1}|\mathbf{a}^T||\mathbf{b}|. \end{aligned} \tag{29}$$

Substituting this into Equation (28), we obtain

$$|r_1 + \sum_{i=2}^{n}(r_i + l_i) - \mathrm{fl}(r_1 + \sum_{i=2}^{n}(r_i + l_i))| \leq \frac{\gamma_n n \cdot u}{(1-(n-1)u)}|\mathbf{a}^T||\mathbf{b}|. \tag{30}$$

Combining Equation (27), we obtain

$$\begin{aligned} |E_n| &\leq \frac{(1+u)n \cdot u\gamma_n}{(1-(n-1)u)}|\mathbf{a}^T||\mathbf{b}| + (1+u)\sum_{i=2}^{n}|q_i| + u|\mathbf{a}^T\mathbf{b}| \\ &\leq \gamma_n^2|\mathbf{a}^T||\mathbf{b}| + (1+u)\sum_{i=2}^{n}|q_i| + u|\mathbf{a}^T\mathbf{b}|. \end{aligned} \tag{31}$$

According to Lemma 3, $\sum_{i=2}^{n}|q_i| \leq u\gamma_n\sum_{i=1}^{n}|a_i, b_i| \leq u\gamma_n|\mathbf{a}^T\mathbf{b}|$. Combining Equation (31), we have

$$\begin{aligned} |E_n| &\leq \gamma_n^2 |\mathbf{a}^T||\mathbf{b}| + u(1+u)\gamma_n |\mathbf{a}^T||\mathbf{b}| + u|\mathbf{a}^T\mathbf{b}| \\ &= \gamma_n(\gamma_n + u + u^2)\mathbf{a}^T||\mathbf{b}| + u|\mathbf{a}^T\mathbf{b}| \end{aligned} \tag{32}$$

and by Equation (21), we obtain

$$|E_n| \leq \gamma_n \gamma_{n+1} \mathbf{a}^T||\mathbf{b}| + u|\mathbf{a}^T\mathbf{b}|. \tag{33}$$

Then,

$$|E_n|/|\mathbf{a}^T\mathbf{b}| \leq \gamma_n \gamma_{n+1} \mathrm{cond}(\mathbf{a}^T\mathbf{b}) + u. \tag{34}$$

□

3. Numerical Experiment

When conducting simulation experiments on the FPGA platform, it may not fully reflect the program's performance. For example, the HAPS platform, using real DDR, cannot accurately evaluate program performance when the program involves accessing DDR. The FPGA platform provides modeling capabilities for DDR controllers and memory. Users can use the tools and libraries provided by the FPGA platform to build DDR models, including configuring the memory parameters, setting the timing, delays, etc. After loading the design code onto the FPGA chip, the FPGA platform will simulate the behavior of the DDR controller and memory. During program execution, FPGA will simulate DDR read/write operations, timing delays, data transfers, etc., to accurately simulate the program's performance. Therefore, the FPGA platform can not only test the correctness but also accurately reflect the performance. The high-precision instructions become ineffective under the round-to-nearest mode. Hence, it is necessary to switch the system's rounding mode to round-towards-zero mode. The code to modify the rounding mode on the ARMv8 platform is as Listing 1.

Listing 1. The code to modify the rounding mode on the ARMv8 platform.

```
void change_round_mode(u_int64_t x){
        __asm__ __volatile__(
        "mrs x0, FPCR                   \n\t"
        "ldr x4, =0xff3fffff            \n\t"
        "ldr x1, %[input]               \n\t"
        "and x2, x0, x4                 \n\t"
        "orr x3, x2, x1                 \n\t"
        "msr FPCR, x3                   \n\t"
        :
        :[input]"m"(x)
        :"x0","x1","x2","x3","x4");
}
void setRN(){
        change_round_mode(0x0); //rounding to nearest.
}
void setRu(){
        change_round_mode(0x400000); //rounding to plus infinity.
}
void setRD(){
        change_round mode(ex800000); //rounding to minus infinity.
}
void setRZ(){
        change_round mode(0xc00000); //rounding towards Zero.
}
```

3.1. FPGA Hardware Simulation Platform

The ARMv8 processor for this experiment is built on the FPGA simulation platform. The testing programs are compiled with the -O0 and -fno-fast-math options to avoid compiler optimization, because this may invalidate the accuracy of the algorithms. When testing the summation algorithm, let a_0 be a constant α, with the rest of the numbers in the array generated by the sine function within the domain $(-\pi/2, \pi/2)$, ensuring that the exact sum of this portion of the numbers is 0, thus obtaining the exact solution of the array sum as α. Similarly, when testing the dot product algorithm, let $a_0 * b_0$ be a constant α, with the remaining products being split into two halves that are opposite and equal in magnitude, resulting in the exact solution of the array sum as α. This can help us to calculate the condition number and relative error. Since the current instructions only support round-towards-zero, the algorithm will fail under round-to-nearest mode. Therefore, all numerical experiments are conducted under the round-towards-zero mode.

3.2. Experiment for Accurate Summation

First, we observe the calculation results of Algorithms 3 and 7 and regular summation for multiple sets of different α and array sizes n. We calculate the relative errors, compare the precision of Algorithms 3 and 7 with regular summation, and verify the correctness of Theorem 4 based on the condition numbers. Second, we compare the time required for Algorithms 3 and 7 to compute problems of the same scale and verify the performance improvement of algorithm optimization.

3.2.1. Accuracy Comparison for Accurate Summation

The length of arrays $\{a_i\}_{i=0}^{N}$ is $N+1$, $a_i = \sin(\pi/2 * (2i-1-N)/2N), i = 1, 2, \ldots, N$. $a_0 = \alpha$, α takes the values $10^{-5}, 10^{-10}$, and 10^{-15}. We calculate the sum of array a_n using ordinary summation and Algorithms 3 and 7. The relative error of the algorithm is as follows:

$$\text{relative error of algorithm} = |\alpha - \text{result of algorithm}|/\alpha. \tag{35}$$

where α is known to be the analytical solution of $\sum_{i=0}^{N} a_i$. The relative errors of each algorithm are shown in Table 1. With the rise in the condition number, the relative error follows an increasing tendency.

Consistent with theoretical analysis and intuition, as the condition number increases, the relative error of the algorithm also increases. Comparing the relative error of our algorithm and the classic sum, we can conclude that the precision of the results obtained by Algorithm 7 is higher compared to classic summation. Comparing the relative error of our algorithm and Algorithm 3 [6], it can be concluded that the computed results of Algorithms 3 and 7 are quite similar. For a few rows of data, the relative error of Algorithm 7 is lower. This may because of the features of the data. It also indirectly reflects that, at least for the computation of certain types of data, the precision of Algorithm 7 is as high as that of Algorithm 3.

Table 1. The condition numbers and the relative errors of each algorithm.

Condition Number	Relative Error of Algorithm 7 (Ours)	Relative Error of Algorithm 3 [6]	Relative Error of Classic Sum
6.51898903×10^{7}	0.00000000×10^{0}	0.00000000×10^{0}	$2.21007546 \times 10^{-7}$
2.60759465×10^{8}	0.00000000×10^{0}	0.00000000×10^{0}	$3.63575381 \times 10^{-6}$
1.04303784×10^{9}	0.00000000×10^{0}	0.00000000×10^{0}	$5.83473446 \times 10^{-5}$
4.17215134×10^{9}	0.00000000×10^{0}	0.00000000×10^{0}	$9.34421480 \times 10^{-4}$

Table 1. *Cont.*

Condition Number	Relative Error of Algorithm 7 (Ours)	Relative Error of Algorithm 3 [6]	Relative Error of Classic Sum
$1.27323954 \times 10^{10}$	0.00000000×10^{0}	0.00000000×10^{0}	$7.02761044 \times 10^{-3}$
$3.18309886 \times 10^{10}$	0.00000000×10^{0}	0.00000000×10^{0}	$5.47924835 \times 10^{-2}$
$6.36619772 \times 10^{10}$	0.00000000×10^{0}	0.00000000×10^{0}	$2.19175222 \times 10^{-1}$
$6.51898903 \times 10^{12}$	0.00000000×10^{0}	0.00000000×10^{0}	$2.20868596 \times 10^{-2}$
$2.60750465 \times 10^{13}$	0.00000000×10^{0}	0.00000000×10^{0}	$3.63419377 \times 10^{-1}$
$1.04303784 \times 10^{14}$	$6.46234854 \times 10^{-16}$	$6.46234854 \times 10^{-16}$	5.83543333×10^{0}
$4.17215134 \times 10^{14}$	$8.91804098 \times 10^{-15}$	$8.91804098 \times 10^{-15}$	9.34405709×10^{1}
$1.27323954 \times 10^{15}$	$7.50024900 \times 10^{-14}$	$7.50924900 \times 10^{-14}$	7.02764015×10^{2}
$3.18309886 \times 10^{15}$	$7.50024900 \times 10^{-14}$	$7.50924900 \times 10^{-14}$	5.47923314×10^{3}
$6.36619772 \times 10^{15}$	$2.19267486 \times 10^{-12}$	$2.19267486 \times 10^{-12}$	2.19175070×10^{4}
$2.60759465 \times 10^{18}$	$2.11611938 \times 10^{-12}$	$9.57846791 \times 10^{-12}$	3.63533640×10^{4}
$1.04303784 \times 10^{19}$	$8.57846791 \times 10^{-12}$	$1.11976044 \times 10^{-10}$	5.83497916×10^{5}
$4.17215134 \times 10^{19}$	$3.18771198 \times 10^{-10}$	$3.18771198 \times 10^{-10}$	9.34401167×10^{6}
$1.27323054 \times 10^{20}$	$3.62749365 \times 10^{-9}$	$3.62749365 \times 10^{-9}$	7.02763561×10^{7}
$3.18309886 \times 10^{20}$	$3.62749365 \times 10^{-9}$	$1.09506612 \times 10^{-8}$	5.47925087×10^{8}
$6.36619772 \times 10^{20}$	$3.21264849 \times 10^{-7}$	$3.21264849 \times 10^{-7}$	2.19175611×10^{9}

3.2.2. Runtime Performance Comparison for Accurate Summation

The lengths of arrays n range from 1025 to 1,000,001. The left y-axis is the runtime. The right y-axis is the acceleration ratio. Due to the large range of values for n, the x-axis takes $\ln(n)$. Comparing the runtimes of Algorithms 3 and 7, Algorithm 7 achieved a 1.69 speedup on average, as shown in Figure 6. Thus, we conclude that using Algorithm 7 can significantly improve the efficiency of accurate summation.

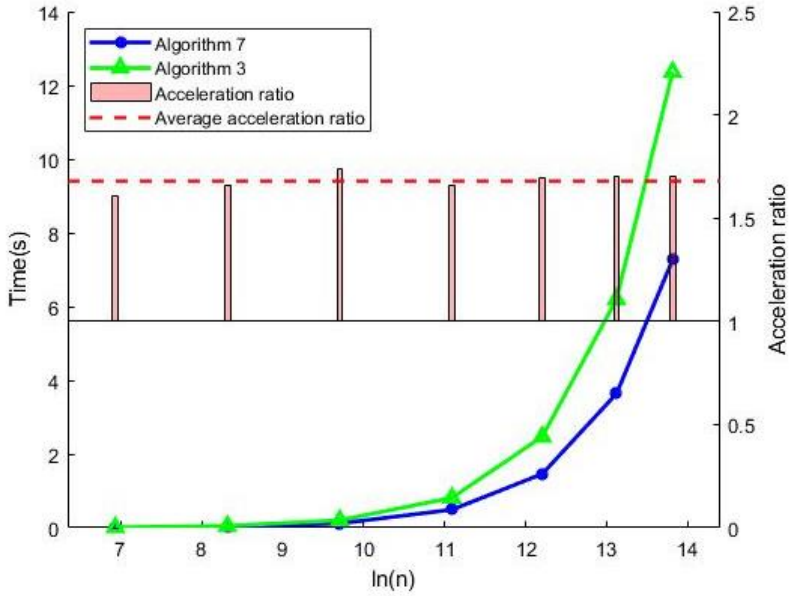

Figure 6. Runtimes of Algorithms 3 and 7.

3.3. Experiment for Accurate Dot Product

Similar to accurate summation, we observe the calculation results of Algorithms 4 and 8 and the regular dot product for multiple sets of different α and array sizes n. We compare the accuracy of Algorithms 4 and 8 with that of regular summation and verify the correctness of the error bounding Theorem 5 based on the condition numbers. We compare the time required for Algorithms 4 and 8 to compute problems of the same scale and verify the performance improvement of algorithm optimization.

3.3.1. Accuracy Comparison for Accurate Dot Product

The lengths of $\mathbf{a}$ and $\mathbf{b}$ are $2N+1$. Let $dr_i \in \mathbb{F}$ be a random number from 0.5 to 1.5, $i = 1, 2, \ldots, N$. $a_i = \sin(\pi/2 \times (2i-1-N)/2N)/dr_i = a_{2N+1-i}$, $b_i = dr_i = -b_{2N+1-i}$, for $i = 1, 2, \ldots, N$. $a_0 = \alpha$ and $b_0 = 1$ are constant, and α takes the values $10^{-3}, 10^{-6}$, and 10^{-9}. We calculate the sum of array a_n using ordinary summation and Algorithms 4 and 8. The analytical solution for $\mathbf{a}^T\mathbf{b}$ is known to be α. Then, the relative error of the algorithm is also computed by Equation (35). As with the results of the sum algorithms, with the rise in the condition number, the relative error follows an increasing tendency. In Table 2, we compare the relative errors of each algorithm under the same circumstances.

Comparing the relative errors of our algorithm and the classic dot, it can be recognized that the precision of Algorithm 8 is higher than that of the classic dot product. By observing the relative errors of our algorithm and Algorithm 4 [6], it can be seen that even the relative error of Algorithm 7 is lower for a few rows. We can see that Algorithm 8 is as accurate as Algorithm 4.

Table 2. The condition numbers and the relative errors of each algorithm.

Condition Number	Relative Error of Algorithm 8 (Ours)	Relative Error of Algorithm 4 [6]	Relative Error of Classic Dot
3.09609813×10^{9}	0.00000000×10^{0}	0.00000000×10^{0}	$3.64059902 \times 10^{-8}$
$4.93847415 \times 10^{10}$	0.00000000×10^{0}	0.00000000×10^{0}	$5.84336469 \times 10^{-7}$
$7.78222147 \times 10^{11}$	0.00000000×10^{0}	0.00000000×10^{0}	$9.34442157 \times 10^{-6}$
$3.09609813 \times 10^{12}$	0.00000000×10^{0}	0.00000000×10^{0}	$3.64271123 \times 10^{-5}$
$1.24410327 \times 10^{13}$	0.00000000×10^{0}	0.00000000×10^{0}	$1.49551930 \times 10^{-4}$
$4.93847415 \times 10^{13}$	0.00000000×10^{0}	0.00000000×10^{0}	$5.84243904 \times 10^{-4}$
$3.09609813 \times 10^{15}$	0.00000000×10^{0}	0.00000000×10^{0}	$3.64190617 \times 10^{-2}$
$1.24410327 \times 10^{16}$	$1.29172524 \times 10^{-14}$	$5.23042845 \times 10^{-14}$	$1.49552064 \times 10^{-1}$
$1.15696185 \times 10^{17}$	$2.97732081 \times 10^{-13}$	$1.19008129 \times 10^{-12}$	1.12452736×10^{0}
$7.22410536 \times 10^{17}$	$5.95210052 \times 10^{-12}$	$2.38306367 \times 10^{-11}$	8.76712204×10^{0}
$2.89045729 \times 10^{18}$	$4.76318389 \times 10^{-11}$	$1.00464464 \times 10^{-10}$	3.50684910×10^{1}
$1.24410327 \times 10^{19}$	$1.31147418 \times 10^{-11}$	$5.23462639 \times 10^{-11}$	1.49552170×10^{2}
$1.15696185 \times 10^{20}$	$2.97807148 \times 10^{-10}$	$1.19010342 \times 10^{-9}$	1.12453110×10^{3}
$7.22410535 \times 10^{20}$	$5.95217559 \times 10^{-9}$	$2.38305000 \times 10^{-8}$	8.76713306×10^{3}
$2.89045729 \times 10^{21}$	$4.76325403 \times 10^{-8}$	$1.90464327 \times 10^{-7}$	3.50685311×10^{4}

3.3.2. Runtime Performance Comparison for Accurate Dot Product

The lengths of arrays n range from 2049 to 2,000,001. The left y-axis is the runtime. The right y-axis is the acceleration ratio. Due to the large range of values for n, the x-axis takes $\ln(n)$. As shown in Figure 7, Algorithm 8 achieves a 1.14 speedup on average. Because

it requires more multiplications and FMA instructions, the proportion of Algorithm 6 is lower. Therefore, the acceleration ratio that Algorithm 8 achieves is not as high as that of Algorithm 7. Thus, we conclude that using Algorithm 8 can slightly improve the efficiency of the accurate dot operation.

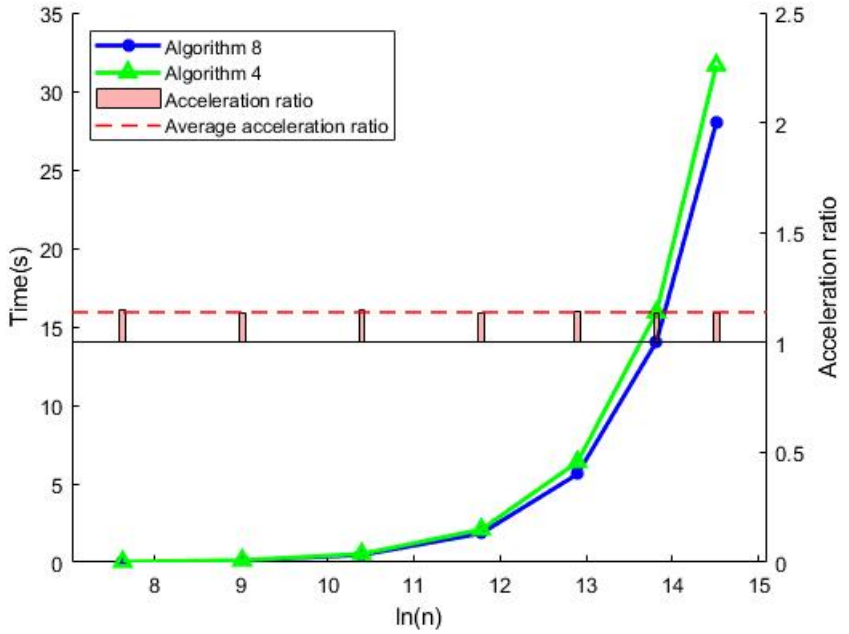

Figure 7. Runtimes of Algorithms 4 and 8.

4. Conclusions

Considering the error analysis along with the results of the numerical experiments, the following conclusions can be drawn (Table 3). When the rounding mode is round-towards-zero, the relative error bound of Algorithm 7 is $\gamma_{n-1}\gamma_n \mathrm{cond} + u$ and the relative error bound of Algorithm 8 (ours) is $\gamma_n\gamma_{n+1}\mathrm{cond} + u$ according to Theorems 4 and 5. The differences in the error bounds between our algorithms and the algorithms in [6] are not significant. The numerical experiments also indicate that the computed results of Algorithms 7 and 8 are very close in accuracy to those of Algorithms 3 and 4 [6]. Regarding the runtime performance, Algorithm 7 reduces the computation time by 40% compared to Algorithm 3 in [6]. Algorithm 8 reduces the computation time by 10% compared to Algorithm 4 in [6].

Table 3. Floating-point operations and error bounds for each algorithm.

Algorithm	FLOPs	Relative Error Bound
Algorithm 3	$7n-5$	$\gamma_{n-1}^2\mathrm{cond} + u$ (Theorem 2)
Algorithm 7	$4n-2$	$\gamma_{n-1}\gamma_n\mathrm{cond} + u$ (Theorem 4)
Algorithm 4	$10n-5$	$\gamma_n^2\mathrm{cond} + u$ (Theorem 3)
Algorithm 8	$7n-2$	$\gamma_n\gamma_{n+1}\mathrm{cond} + u$ (Theorem 5)

Author Contributions: Conceptualization, K.X. and H.J.; methodology, K.X. and H.W.; software, K.X.; validation, K.X., Q.L. and H.J.; formal analysis, K.X., H.J. and H.W.; investigation, K.X. and H.J.; resources, K.X.; data curation, K.X.; writing—original draft preparation, K.X.; writing—review and editing, K.X., H.J. and H.W.; visualization, K.X.; supervision, K.X.; project administration, K.X. and H.J.; funding acquisition, H.J. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the National Key R&D Program of China, grant number 2023YFB3001601.

Data Availability Statement: The data were generated with the submitted code.

Conflicts of Interest: The funders had no role in the design of the study; in the collection, analyses, or interpretation of the data; in the writing of the manuscript; or in the decision to publish the results.

References

1. Kahan, W. Pracniques: Further remarks on reducing truncation errors. *Commun. ACM* **1965**, *8*, 40. [CrossRef]
2. Knuth, D.E. *The Art of Computer Programming: Seminumerical Algorithms*, 3rd ed.; Addison-Wesley Longman Publishing Co., Inc.: Reading, MA, USA, 1997; Volume 2.
3. Boldo, S.; Graillat, S.; Muller, J.M. On the robustness of the 2sum and fast2sum algorithms. *ACM Trans. Math. Softw.* **2017**, *44*, 4. [CrossRef]
4. Dekker, T.J. A floating-point technique for extending the available precision. *Numer. Math.* **1971**, *18*, 224–242. [CrossRef]
5. Graillat, S.; Lefèvre, V.; Muller, J.M. Alternative split functions and dekker's product. In Proceedings of the 2020 IEEE 27th Symposium on Computer Arithmetic (ARITH), Portland, OR, USA, 7–10 June 2020; pp. 41–47. [CrossRef]
6. Ogita, T.; Rump, S.M.; Oishi, S. Accurate sum and dot product. *SIAM J. Sci. Comput.* **2005**, *26*, 1955–1988. [CrossRef]
7. Rump, S.M.; Ogita, T.; Oishi, S. Accurate floating-point summation part I: Faithful rounding. *SIAM J. Sci. Comput.* **2008**, *31*, 189–224. [CrossRef]
8. Rump, S.M.; Ogita, T.; Oishi, S. Accurate floating-point summation part II: Sign, k-fold faithful and rounding to nearest. *SIAM J. Sci. Comput.* **2009**, *31*, 1269–1302. [CrossRef]
9. Yamanaka, N.; Ogita, T.; Rump, S.M.; Oishi, S. A parallel algorithm for accurate dot product. *Parallel Comput.* **2008**, *34*, 392–410. [CrossRef]
10. Ozaki, K.; Ogita, T.; Oishi, S.; Rump, S.M. Error-free transformations of matrix multiplication by using fast routines of matrix multiplication and its applications. *Numer. Algorithms* **2012**, *59*, 95–118. [CrossRef]
11. Graillat, S.; Ménissier-Morain, V. Accurate summation, dot product and polynomial evaluation in complex floating point arithmetic. *Inf. Comput.* **2012**, *216*, 57–71. [CrossRef]
12. Kadric, E.; Gurniak, P.; Dehon, A. Accurate parallel floating-point accumulation. In Proceedings of the 2013 IEEE 21st Symposium on Computer Arithmetic, Austin, TX, USA, 7–10 April 2013; pp. 153–162. [CrossRef]
13. Demmel, J.; Nguyen, H.D. Fast reproducible floating-point summation. In Proceedings of the 2013 IEEE 21st Symposium on Computer Arithmetic, ARITH '13, Austin, TX, USA, 7–10 April 2013; pp. 163–172. [CrossRef]
14. Abdalla, D.M.; Zaki, A.M.; Bahaa-Eldin, A.M. Acceleration of accurate floating point operations using SIMD. In Proceedings of the 2014 9th international conference on computer engineering & systems (ICCES), Cairo, Egypt, 22–23 December 2014; pp. 225–230. [CrossRef]
15. Jankovic, J.; Subotic, M.; Marinkovic, V. One solution of the accurate summation using fixed-point accumulator. In Proceedings of the 2015 23rd Telecommunications Forum Telfor (TELFOR), Belgrade, Serbia, 24–26 November 2015; pp. 508–511. [CrossRef]
16. Goodrich, M.T.; Eldawy, A. Parallel algorithms for summing floating-point numbers. In Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures, SPAA '16, New York, NY, USA, 11–13 July 2016; pp. 13–22. [CrossRef]
17. Evstigneev, N.; Ryabkov, O.; Bocharov, A.; Petrovskiy, V.; Teplyakov, I. Compensated summation and dot product algorithms for floating-point vectors on parallel architectures: Error bounds, implementation and application in the Krylov subspace methods. *J. Comput. Appl. Math.* **2022**, *414*, 114434. [CrossRef]
18. Lange, M. Toward accurate and fast summation. *ACM Trans. Math. Softw.* **2022**, *48*, 28. [CrossRef]
19. Zhu, Y.K.; Hayes, W.B. Correct rounding and a hybrid approach to exact floating-point summation. *SIAM J. Sci. Comput.* **2009**, *31*, 2981–3001. [CrossRef]
20. Jiang, H.; Du, Q.; Guo, M.; Quan, Z.; Zuo, K.; Wang, F.; Yang, C.Q. Design and implementation of qgemm on armv8 64-bit multi-core processor. *Jisuanji Xuebao/Chinese J. Comput.* **2017**, *40*, 2018–2029. [CrossRef]
21. Lange, M.; Rump, S.M. Error estimates for the summation of real numbers with application to floating-point summation. *BIT Numer. Math.* **2017**, *57*, 927–941. [CrossRef]
22. Lange, M.; Rump, S. Sharp estimates for perturbation errors in summations. *Math. Comput.* **2018**, *88*, 349–368. [CrossRef]
23. Boldo, S.; Lauter, C.; Muller, J.M. Emulating round-to-nearest ties-to-zero "augmented" floating-point operations using round-to-nearest ties-to-even arithmetic. *IEEE T. Comput.* **2021**, *70*, 1046–1058. [CrossRef]
24. Muller, J.M.; Rideau, L. Formalization of double-word arithmetic, and comments on "tight and rigorous error bounds for basic building blocks of double-word arithmetic". *ACM Trans. Math. Softw.* **2022**, *48*, 15res. [CrossRef]

25. Brisebarre, N.; Muller, J.M.; Picot, J. Error in ulps of the multiplication or division by a correctly-rounded function or constant in binary floating-point arithmetic. In Proceedings of the 2023 IEEE 30th Symposium on Computer Arithmetic (ARITH), Portland, OR, USA, 4–6 September 2023; p. 88. [CrossRef]
26. Hubrecht, T.; Jeannerod, C.P.; Muller, J.M. Useful applications of correctly-rounded operators of the form ab + cd + e. In Proceedings of the 2024 IEEE 31st Symposium on Computer Arithmetic (ARITH), Malaga, Spain, 10–12 June 2024; pp. 32–39. [CrossRef]
27. *IEEE-Std-754-2019*; IEEE Standard for Floating-Point Arithmetic. Technical Report; IEEE Computer Society: New York, NY, USA, 2019. [CrossRef]
28. Higham, N.J. *Accuracy and Stability of Numerical Algorithms*, 2nd ed.; Society for Industrial and Applied Mathematics: Philadelphia, PA, USA, 2002.

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.