International Journal of Software Engineering
and Knowledge Engineering
Vol. 34, No. 4 (2024) 675–703
© World Scientific Publishing Company
DOI: 10.1142/S0218194023500675

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

SCR-LIBM: A Correctly Rounded Elementary Function Library in Double-Precision

Yang Qu[*], Jinchen Xu[†], Bei Zhou[‡], Jiangwei Hao[§], Fei Li[¶]
and Zuoyan Zhang[||]

School of Cyber Science and Engineering
Information Engineering University, Zhengzhou 450001, P. R. China
[*]*qy19971130@163.com*
[†]*atao728208@126.com*
[‡]*13653970052@163.com*
[§]*haojiangweitimo@foxmail.com*
[¶]*feili2022@hotmail.com*
[||]*zhangzuoyan523@163.com*

Received 6 August 2023
Revised 8 October 2023
Accepted 4 November 2023
Published 7 December 2023

The MPFR and CR-LIBM math libraries are frequently utilized due to their ability to generate correctly rounded results for all double-precision inputs. However, it is worth noting that MPFR has a slower average performance, while CR-LIBM achieves correct rounding over two iterations, rendering it less stable. In addition, CR-LIBM has a poor performance in handling the worst-case of correct rounding. This paper implements a correctly rounded elementary function library called SCR-LIBM in double-precision, which is stable and efficient. Our key idea is to divide subdomains and use the low-degree Taylor polynomial to approximate the elementary function in each subdomain. We simulate the high-precision representation based on the double–double data format, and use the error-free transformation and Double-double algorithm to control the error in the process of polynomial approximation and output compensation. Our approach ensures that the elementary function is correctly rounded, without the need for redundant iterations. The experimental evaluation shows that the average performance of elementary functions implemented in SCR-LIBM is 8.534 times faster than that of MPFR, and 2.492 times faster than that of CR-LIBM when dealing with the worst-case of correct rounding. What's more, our SCR-LIBM is more stable than CR-LIBM.

Keywords: Elementary function; correct rounding; Taylor polynomial; Double-double algorithm.

1. Introduction

High performance computing (HPC) plays a crucial role in contemporary scientific and technological advancements. It is widely used in various domains such as

[†]Corresponding author.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

high-energy physics research, aerospace aircraft design, national economic forecasting and large-scale precision computing. The elementary function library [1] is an essential component of HPC software, providing implementations of elementary functions (e.g. $\exp(x)$, $\sin(x)$, $\log(x)$). According to the most recent IEEE-754 standard [2], elementary function libraries are advised to produce the correctly rounded value because the floating-point format is frequently used to approximate real numbers. Common used libraries are Intel's MKL [3], AMD's libm [4], Gnu's Glibc [5] and so on. These mainstream math libraries are mainly used for single and double precision, but most of them do not produce correctly rounded results for all inputs.

The correct rounding [6, 7] of an elementary function is defined as computing the value of the elementary function with the input and then rounding to the only result in the target representation according to the rounding rule. Developing a correctly rounded library is a challenging task because even a minor error in the computation of an elementary function can lead to an inaccurate result. According to [8], correct rounding has many benefits. First, it can preserve some mathematical properties of the function by minimizing the error, such as symmetry and monotonicity; second, correctly rounded elementary functions with the same floating-point format will always produce the same result, which is useful for software porting; third, it allows the accuracy of floating-point algorithms to be verified in standardized systems.

Certain critical industries, such as aerospace and automatic control, require the elementary function to provide high accuracy and fast execution performance. While the current double-precision elementary function libraries are capable of achieving correct rounding, their performance is deemed unsatisfactory. For instance, MPFR is poor in terms of average performance. Hence, Lefévre proposed algorithms to optimize the MPFR library [9]. In addition, IBM's MathLib [10] and CR-LIBM [11] are slow at handling the worst-case [12] of correct rounding. What's more, the performance of CR-LIBM is highly variable in some cases.

MPFR [13] is currently the dominant multi-precision package, using arbitrary multi-precision to achieve correct rounding. Experimental tests show that some elementary functions of MPFR cost 10 of 1000 of cycles. The average performance of MPFR is poor because it uses arbitrary precision, so it cannot take advantage of the floating-point performance offered by processors. MathLib provides correctly rounded elementary functions for double precision. However, Daramy [11] denotes that worst-case execution time of MathLib can increase to 20,000 times the normal execution time. This significant increase poses challenges in utilizing MathLib for real-time applications, CR-LIBM [11], a correctly rounded library in double-precision, developed by Daramy *et al.* CR-LIBM provides correct rounding by two iterations. The first iteration, which is called as Quick phase, provides 60–80 bits of accuracy (i.e. depending on the function) for processing most of the inputs. The second iteration, which is called as accurate phase, is dedicated to the worst-case of correct rounding. Although the first iteration of CR-LIBM is fast, it has poor

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

performance against the worst-case of correct rounding. The average worst-case execution performance of CR-LIBM is as high as 3200 cycles. In scenarios where CR-LIBM handles many inputs in close proximity, a notable circumstance arises: Certain inputs require only the first iteration for correct rounding, while others necessitate a second iteration for correct rounding. The situation described above leads to large fluctuations in the performance, with the $\sin(x)$ function of the CR-LIBM fluctuating up to 10 times in two iterations.

Therefore, it is difficult for the above elementary function libraries to meet the needs of the high real-time, fast interaction field. For example, real-time aerospace system [14] has strict limits on the average response time and worst-case execution time of each component. As an elementary function library providing computational support, its performance [15, 16] and stability directly affect the safety of the actual flight.

The main contributions of this paper are the following. First, we implement an elementary function library called SCR-LIBM in double-precision, which is efficient and stable. Second, we present an efficient algorithm to approximate elementary function. Our algorithm solves the problem that traditional elementary function libraries need to use iterations to achieve correct rounding in the worst-case, and reduces the upper limit of execution time in the worst-case. Without entering redundant iterations, our algorithm effectively avoids large fluctuations in the performance. Third, we use the multi-part mode for error control, which can achieve high accuracy and relatively good performance, to solve the problem that the average performance of elementary function library is poor.

The outline of this paper is as follows. In Sec. 2, we introduce the basics of floating-point representation, the implementation method of elementary functions, and error control methods for floating-point computations. In Sec. 3, we introduce the design theory of correctly rounded elementary functions. Two general phases are taken in order to correctly round the elementary function. In Sec. 4, we explain how to implement correctly rounded elementary functions in details. Specially, we propose a new polynomial approximation algorithm that divides subdomains and uses low-degree Taylor polynomial to approximate the elementary function in each subdomain. What's more, we present how to combine error-free transformation and Double-double algorithm to get the accurate result during polynomial approximation and output compensation. In Sec. 5, numerical experiments showing the accuracy and the performance of our SCR-LIBM; finally, this paper makes the conclusion and expectation in Sec. 6.

2. Background

We provide background on the floating-point representation, implementation methods of elementary functions, and error control methods for floating-point computations.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

2.1. *The floating-point representation*

The IEEE-754 (2008) standard [2] defines the formats of binary and decimal floating-point numbers, denormal values, some special values and the behavior of the four basic operators. Floating-point numbers are usually encoded in a scientific notation format in computers. The IEEE-754 (2008) standard specifies the encoding format of floating-point as a normalized binary scientific notation. The binary floating-point number is stored in the format of a symbolic numerical representation, which is divided into three parts from high to low in the computer: Sign bit, exponent bits and fraction bits. Overall, the IEEE-754 (2008) standard specifies several formats for binary floating-point numbers, as shown in Fig. 1, with single and double precision being the most commonly used data types.

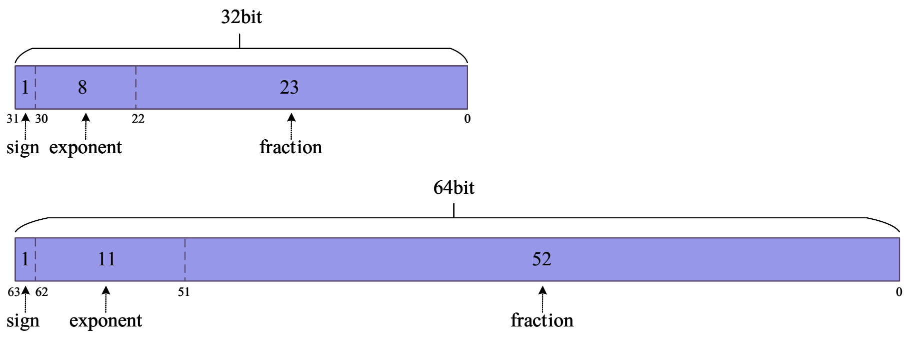

Fig. 1. Float and double floating-point format.

Real values must be rounded to the proper floating-point number in accordance with the rounding mode when they cannot be accurately represented. In total, the IEEE-754 (2008) standard specifies four rounding modes: Round toward positive infinity, round toward negative infinity, round toward zero and round to the nearest. Commonly, results are rounded to the nearest floating-point value using the "round to the nearest" mode. If two values are equally close, the even value is taken. In this paper, we assume that all the floating-point computations are performed in double-precision, with the "round to the nearest" mode.

2.2. *Methods for the implementation of elementary functions*

Currently, the table-lookup algorithm [17], the polynomial approximation method [18, 19], and the "reduction–approximation–reconstruction" algorithm are the methods used to implement elementary functions. Although the table-lookup algorithm is quite straightforward, it cannot be extended to increase precision. While the polynomial approximation method offers great accuracy, it is not appropriate for calculations involving huge approximation intervals.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

As Lim shows, using the "reduction–approximation–reconstruction" algorithm to implement functions has good effects [20]. Hence, our approach is based on the "reduction–approximation–reconstruction" algorithm. Typically, the "reduction–approximation–reconstruction" algorithm is composed of three parts: Range reduction, polynomial approximation and output compensation. According to the properties of the function, the mathematical rule, and the character of the floating point, range reduction is the process of converting inputs in the defined interval into a domain that is suitable for approximation. For example, the periodicity and symmetry of $\sin(x)$ can be used to transform the defined interval to $[0, \frac{\pi}{2}]$. After range reduction, one can approximate the value of the elementary function by choosing suitable approximation polynomials, such as the Taylor polynomial. The higher the degree of the polynomial, the more accurate the approximation will be, but it will also consume more computing resources. Finally, in order to obtain the final function value, output compensation incorporates previously computed intermediate values of range reduction and polynomial approximation. In addition to the above three steps, it is generally necessary to deal with the special cases of the function in advance. The special cases refer to the inputs that may cause errors and exceptions in the computation of the function.

2.3. *Error control method for floating-point computations*

Floating-point calculation errors can be divided into various types (e.g. representation error, rounding error). The most direct and reliable method of eliminating floating-point computation errors is to increase working precision. Increasing working precision can primarily be accomplished by high precision simulation of compiler, hardware and software algorithms. This paper describes how to simulate high precision using the software algorithm.

Software algorithms for high precision simulation can be divided into two categories: Multi-digit mode and multiple-component mode. The principle behind multi-digit mode is to use arrays of integers or floating-point numbers to represent mantissa, and use separate integer or floating-point number to represent exponent [21]. A more representative tool in multi-digital mode is MPFR [13]. The performance of MPFR is poor because the multi-digit mode specifies arbitrary precision bits. This prevents the arithmetic operation from utilizing the performance offered by the current processor. As for the multipart mode, it uses several standard floating-point numbers to represent the floating-point number with higher precision. Each mantissa bit and exponent bit in a floating-point number is unique. Several such floating-point numbers are added together to produce a floating-point number with higher precision. The expressible precision can only be an integer multiple of the precision of the component floating-point number, such as double–double [22] and Triple-double [23]. In [24], Graillat *et al.* have introduced the error-free transformation and compensated algorithms, which are the basis of the multiple-component mode. In addition, Graillat *et al.* have developed accurate compensated algorithms [25] by making use of

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

the error-free transformation. We recall the error-free transformation and the Double-double algorithm.

2.3.1. *Error-free transformation*

The error-free transformation is defined as in Definition 1.

Definition 1. If $\circ \in \{+, -, \times\}$, let a and b be two floating-point numbers such that $a, b \in F$, and $x = fl(a \circ b) \in F$, when there is no overflow or underflow and the rounding mode is round-to-even. Then

$$a \circ b = x + y.$$

The x represents the floating-point number closest to the result of the computation. The y represents an exact rounding error. The process of converting the floating-point pair (a, b) into the floating-point pair (x, y) is the error-free transformation.

FastTwoSum [26] and *TwoSum* are the two primary categories of error-free transformation addition. *TwoSum* does not restrict the size of the two added floating-point numbers, whereas *FastTwoSum* does. Hence, *TwoSum* is the optimal error-free transformation addition algorithm for summing two floating-point numbers. The most widely used algorithm for the product of two floating-point numbers is called *TwoProd*. *TwoProd* needs to use the *Split* algorithm to split the floating-point number into two floating-point numbers [26].

2.3.2. *Double-double algorithm*

The double–double [22] data format is often used for floating-point computation that requires twice the working precision. The following is the specific definition of the double–double data format.

Definition 2. Let x_h and x_l be two double-precision floating-point numbers, and x is the exact sum of x_h and x_l without rounding. The relationship between x, x_h and x_l can be expressed as

$$x = x_h + x_l.$$

When adding two double–double numbers, multiplying two double–double numbers, or dividing two double–double numbers, Double-double algorithms are used to get results that are double–double. Compared to the error-free transformation, Double-double algorithm is more accurate but has a higher performance overhead.

Summary. Under the guidelines of the IEEE-754 standard, implementing an elementary function for double precision with the "reduction–approximation–reconstruction" algorithm may suffer from floating-point calculation errors. To address the aforementioned issue, a dependable approach is to utilize software algorithms for high precision simulation. This method generally employs

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

error-free transformation and the Double-double algorithm to achieve a high level of accuracy.

3. The Framework of Correctly Rounded Elementary Functions in Double-Precision

Our SCR-LIBM uses two phases, based on and improved upon the traditional "reduction–approximation–reconstruction" algorithm, to implement correctly rounded elementary functions in double-precision. The first phase is the "Design of elementary function algorithm based on high precision". The second phase is the "Error control for floating-point computations of double precision". Figure 2 provides a pictorial overview of our method. Take the $\exp(x)$ function as an example. First, we filter special cases of the $\exp(x)$ function. Subsequently, range reduction is conducted in order to reduce the range of function arguments, hence facilitating the process of approximating the function. Next, we can obtain the approximation polynomial through the phase "Design of elementary function algorithm based on high precision". Once the approximation polynomial has been determined, correctly rounded elementary functions are implemented in the "Error control for floating-point computations of double precision" phase. The specific functions of these two phases will be introduced individually as follows.

- Design of elementary function algorithm based on high precision: First, we analyze the benefits and drawbacks of the Taylor polynomial, and then we propose a new approximation method that divides subdomains and uses the low-degree Taylor polynomial to approximate the function in the subdomains. Once the method of polynomial approximation has been established, it becomes imperative to ascertain the size of the subdomain and the degree of the Taylor polynomial. Polynomial calculation is performed using the high-precision library MPFR to prevent errors caused by floating-point computation. The determination of the size of a subdomain and the degree of the Taylor polynomial is governed by a circular condition, which ensures that the elementary functions achieve correct rounding. The final result of the first step is an approximation polynomial.
- Error control for floating-point computations of double precision: The approximation polynomial derived from phase 1 is utilized to approximate the function, ensuring correct rounding for elementary functions. However, the process of polynomial computations is done based on MPFR, resulting in poor performance of elementary functions. To solve this problem, we simulate the high-precision representation based on the double–double data format, and combine the error-free transformation and Double-double algorithm to control the error in the process of polynomial approximation and output compensation. The above method guarantees correct rounding of the elementary function while simultaneously enhancing function performance significantly.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

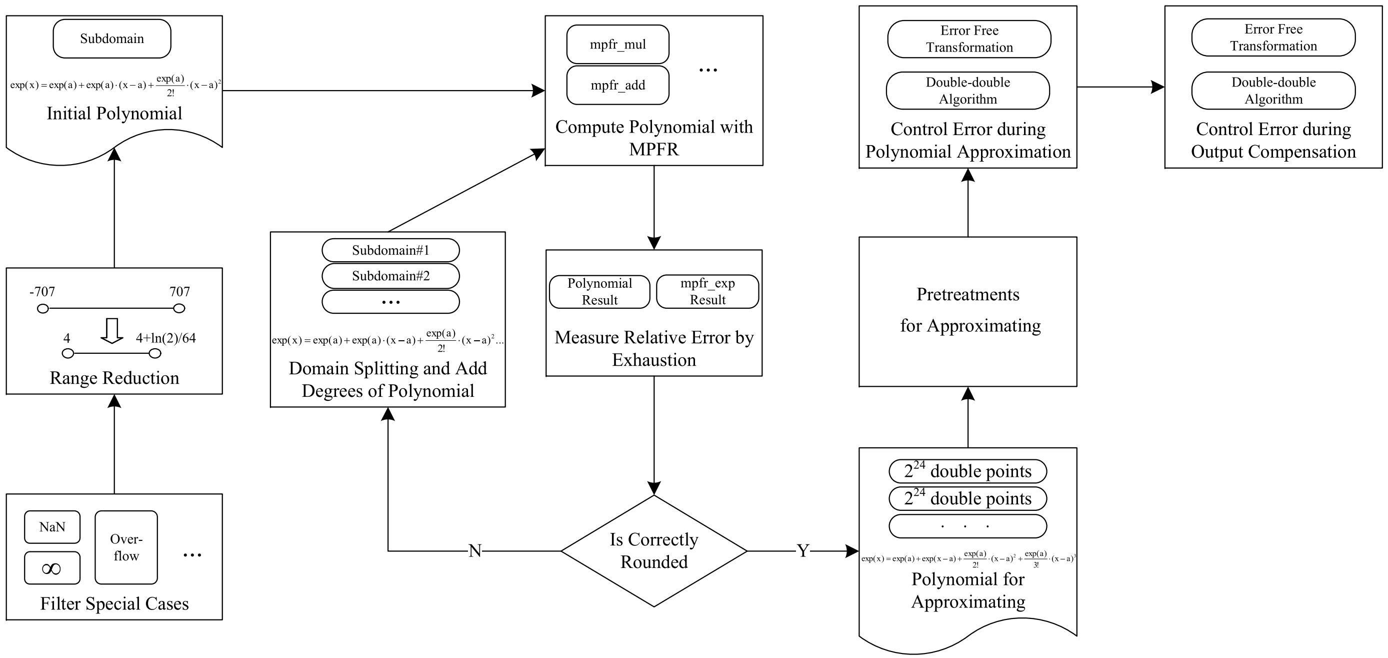
Subdomain
$\exp(x)=\exp(a)+\exp(a)\cdot(x-a)+\frac{\exp(a)}{2!}\cdot(x-a)^2$
Initial Polynomial
-707
707
4
4+ln(2)/64
Range Reduction
NaN
∞
Over-flow
...
Filter Special Cases
Subdomain#1
Subdomain#2
...
$\exp(x)=\exp(a)+\exp(a)\cdot(x-a)+\frac{\exp(a)}{2!}\cdot(x-a)^2\ldots$
Domain Splitting and Add Degrees of Polynomial
mpfr_mul
mpfr_add
...
Compute Polynomial with MPFR
Polynomial Result
mpfr_exp Result
Measure Relative Error by Exhaustion
Is Correctly Rounded
N
Y
2^{24} double points
2^{24} double points
. . .
$\exp(x)=\exp(a)+\exp(x-a)+\frac{\exp(a)}{2!}\cdot(x-a)^2+\frac{\exp(a)}{3!}\cdot(x-a)^3$
Polynomial for Approximating
Pretreatments for Approximating
Error Free Transformation
Double-double Algorithm
Control Error during Polynomial Approximation
Error Free Transformation
Double-double Algorithm
Control Error during Output Compensation

Fig. 2. The framework of correctly rounded elementary functions in double-precision.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

4. The Implementation of Correctly Rounded Elementary Functions in Double-Precision

In this section, the process of correctly rounding elementary functions of SCR-LIBM is presented in detail. Based on the "reduction–approximation–reconstruction" algorithm, we first filter special cases. Then, to reduce the input domain of the elementary function, we perform range reduction. After the above two steps, we propose a new polynomial approximation algorithm that dividing subdomains and using low-degree Taylor polynomial to approximate the elementary function in each subdomain. Finally, a combination of error-free transformation and Double-double algorithm is used for polynomial approximation and output compensation to get the correct rounding result.

4.1. *Filter special cases*

We first filter special cases of elementary functions in our SCR-LIBM, which can help avoid exceptions or errors and improve the accuracy and reliability of the calculated results. These special cases are specific to each elementary function and target representation. However, we can divide special cases into two main types [20]. The first type consists of inputs that produce undefined results like NaN or NaR. In addition, inputs that produce $\pm\infty$ can also be classified into the first type. For instance, $f(x) = \infty$ if $x = \infty$ for the $\exp(x)$ function. The second type is composed of inputs that produce some special results when approximating $f(x)$. These results are underflow, overflow, saturation error, or the same rounded result produced by ranges of inputs. For instance, all double inputs belong to $[707, \infty)$ produce results that round to ∞ (overflow) when approximating the $\exp(x)$ function. Therefore, it is necessary to filter out the special numbers in the input field according to the mathematical properties of the function.

4.2. *Range reduction*

Range reduction is the process of transforming inputs in the defined interval into a domain that is suitable for approximation according to the properties of the function, the mathematical law, and the character of the floating point. For instance, the $\exp(x)$ function in this paper is reduced three times, so that fewer subdomains can be divided in the process of polynomial approximation. For other functions, the times of range reduction are related to the mathematical properties of the function itself. So far, the 10 functions we have implemented only require range reduction three times at most. Taking the $\exp(x)$ function in our SCR-LIBM as an example, we show how to perform range reduction.

As shown in Fig. 3, this paper reduces the input domain of $\exp(x)$ function from $[-707, 707]$ into $[4, 4 + \frac{\ln(2)}{64}]$ by cubic reduction. The specific steps are as follows.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

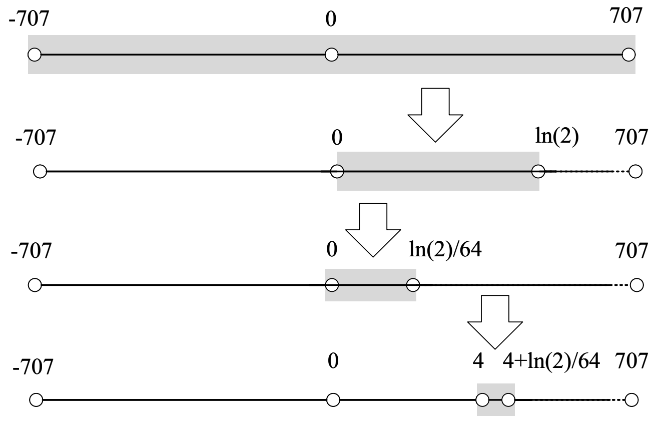

Fig. 3. The process of range reduction.

According to the properties of exp(x) function, Eq. (1) is used to rewrite exp(x) function equitably.

$$\exp(x) = \exp(x_0 + n * \ln(2)). \tag{1}$$

Proof. We have introduced two unknowns, x_0 and n. We let $x = x_0 + n * \ln(2)$, then we can get $\exp(x) = \exp(x_0 + n * \ln(2))$. □

In the above, x_0 and n can be expressed by Eq. (2). n is the amount of ln(2) in x, rounded down.

$$n = \left\lfloor \frac{x}{\ln(2)} \right\rfloor, \quad x_0 = x - n * \ln(2). \tag{2}$$

According to the property of the exponential function, Eq. (1) is further rewritten in the following equation:

$$\exp(x) = \exp(n * \ln(2)) * \exp(x_0) = 2^n * \exp(x_0). \tag{3}$$

Proof. According to the equation $\exp(a + b) = \exp(a) * \exp(b)$, we can get $\exp(x) = \exp(x_0 + n * \ln(2)) = \exp(n * \ln(2)) * \exp(x_0)$. Further, according to the equation $\exp(x * \ln(k)) = k^x$, we can get $\exp(n * \ln(2)) = 2^n$. Finally, we can get $\exp(x) = \exp(n * \ln(2)) * \exp(x_0) = 2^n * \exp(x_0)$. □

By the first range reduction, we get the reduced argument x_0 from x, and $x_0 \in [0, \ln(2))$. To further reduce the range of function parameters, the second range reduction is performed as follows: By inserting several nodes into $[0, \ln(2))$, we divide the interval into several smaller approximation intervals. For example, we insert 63 nodes into $[0, \ln(2))$ and we can get 64 smaller approximation intervals. By doing these, we rewrite x_0 as follows:

$$x_0 = \frac{j}{64} * \ln(2) + x'. \tag{4}$$

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

j is the value of the set $\{0, 1, 2, 3, \ldots, 63\}$ and the reduced argument $x' \in [0, \frac{\ln(2)}{64}]$. According to the above transformation, $\exp(x)$ function is rewritten as follows:

$$\exp(x) = 2^n * 2^{\frac{j}{64}} * \exp(x'). \tag{5}$$

Proof. Because $\exp(x) = 2^n * \exp(x_0)$, we can get $\exp(x) = 2^n * \exp(\frac{j}{64} * \ln(2) + x')$. According to $\exp(a + b) = \exp(a) * \exp(b)$, we can get $\exp\big(\frac{j}{64} * \ln(2) + x'\big) = \exp\big(\frac{j}{64} * \ln(2)\big) * \exp(x')$. Further, according to the equation $\exp(x * \ln(k)) = k^x$, we can get $\exp(\frac{j}{64} * \ln(2)) = 2^{\frac{j}{64}}$. Finally, we can get $\exp(x) = 2^n * 2^{\frac{j}{64}} * \exp(x')$. □

By the second range reduction, we get the reduced argument x' from x_0, and $x' \in [0, \frac{\ln(2)}{64}]$. After quadratic range reduction, there are still a large number of floating-point numbers in the reduced interval $[0, \frac{\ln(2)}{64}]$. In order to split fewer subdomains in the smaller reduced interval, the third range reduction is performed as follows: Using the algorithm of the exponential function, $\exp(x)$ function is rewritten according to the following equation:

$$\exp(x) = 2^n * 2^{\frac{j}{64}} * \exp(-4.0) * \exp(4.0 + x'). \tag{6}$$

Proof. According to the equation $\exp(a + b) = \exp(a) * \exp(b)$, we can get $\exp(x') = \exp(-4.0 + 4.0 + x') = \exp(-4.0) * \exp(4.0 + x')$. Finally, we can get $\exp(x) = 2^n * 2^{\frac{j}{64}} * \exp(-4.0) * \exp(4.0 + x')$. □

Now let $t = 4.0 + x'$, and $t \in [4.0, 4.0 + \frac{\ln(2)}{64}]$. Through the transformation of Eq. (6), the final $\exp(x)$ function is shown in the following equation:

$$\exp(x) = \exp(-4.0) * 2^n * 2^{\frac{j}{64}} * \exp(t). \tag{7}$$

By the third range reduction, we get the reduced argument t from x', and $t \in [4.0, 4.0 + \frac{\ln(2)}{64}]$. In addition to $\exp(t)$ in Eq. (7), the other parameters such as $\exp(-4.0), 2^n, 2^{\frac{j}{64}}$ can be stored in the table beforehand and read from the table when entering the calculation.

4.3. *The elementary function algorithm based on high precision*

After range reduction, we need to use appropriate methods to approximate the function. The common method is to use approximation polynomials. Unlike other approximation methods, the approximation of a function by a polynomial can be done efficiently using only addition, subtraction and multiplication operations. In this section, we propose a new polynomial approximation algorithm that dividing subdomains and using the low-degree Taylor polynomial to approximate function in the subdomain.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

4.3.1. *Taylor expansion in divided subdomains*

When the Taylor polynomial is utilized to approximate the function, the generation of the polynomial is simple. Furthermore, the error of the polynomial approximation has an upper bound. Hence, Taylor polynomial is often used for the approximation of functions. Currently, the mainstream math library typically reduces the parameters of the elementary function to near 0 and takes 0 as the pivot point, approximating the function with Maclaurin's series. However, as the input gets further away from 0, the error of the Taylor polynomial increases significantly. The only way to control the error is by raising the degree of the Taylor polynomial, though this action adversely impacts the efficiency of the elementary function.

To solve the aforementioned issues, we propose the following approach to approximate the elementary function. First, Taylor polynomial is used for the approximation of the elementary function. Second, a new expansion method is used to avoid the disadvantage of using 0 as the pivot point. Further, a number of small subdomains are constructed with different locations. In each subdomain, the starting-point of the subdomain is used as the Taylor expansion point. We use low-degree Taylor polynomial to approximate the elementary function in each subdomain. If the subdomain is sufficiently small, the elementary function can be approximated with less error and using only a few degrees.

As shown in Fig. 4, it is the image of using low-degree Taylor polynomial to approximate the $\exp(x)$ function in each subdomain. To be specific, each point of the same color in the image forms a subdomain and the length of each subdomain is 0.5 cm. From the figure we can see that there is a good approximation effect when the Taylor polynomial of degree 2 is used for the approximation of the elementary function.

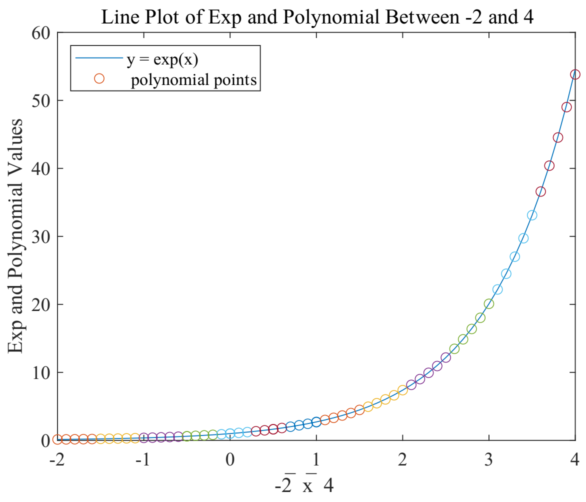

Fig. 4. The effect of polynomial approximation in each splitted subdomain.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

4.3.2. *Determining the approximation polynomial by MPFR*

When actually approximating the elementary function in double-precision, it is imperative to determine the size of each subdomain and the degree of the Taylor polynomial. Using such an approximation polynomial, the elementary function can be correctly rounded. However, floating-point computations such as addition, subtraction, multiplication and division will introduce errors during the process of verifying whether the chosen approximation polynomial can achieve correct rounding. This will ultimately impact the verification results. Therefore, as shown in Fig. 5, we set 128 precision bits of the high-precision library MPFR as the verification environment. In addition, we replaces floating-point computations with the corresponding algorithms of MPFR to prevent errors during polynomial computation, enabling us to focus solely on the errors of the approximation polynomial.

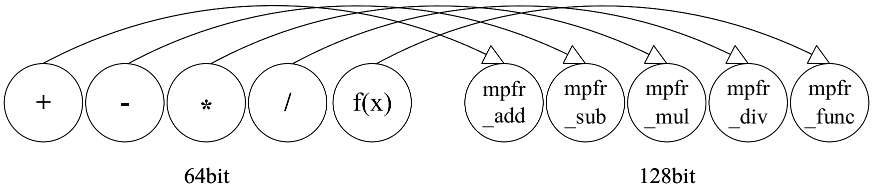

Fig. 5. MPFR replaces floating-point computations.

Equation (8) used by the Taylor polynomial to approximate the elementary function in each subdomain can be expressed as follows:

$$f(x) = f(a) + \frac{f'(a)}{1!}(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \frac{f'''(a)}{3!}(x-a)^3 + \ldots, \tag{8}$$

The more finely the subdomain is divided, the more precise the approximation result yielded by the Taylor polynomial, and the fewer degrees to which the Taylor polynomial needs to be expanded. But when the subdomain is too small, it will bring a lot of redundant calculations. As shown in Eq. (8), we must recalculate the function value, first derivative, second derivative and other values of the pivot point a when a subdomain is divided. The degree of derivation depends on the degree of Taylor polynomial. Therefore, it is necessary to choose a reasonable subdomain size and to minimize the degree of Taylor polynomial to reduce the algorithm overhead. The process of determining degrees of Taylor polynomial and the size of each subdomain is shown in Fig. 6.

Taking the log2(x) function in our SCR-LIBM as an example, we show how to determine the specific approximation polynomial. In this paper, the initial size of each subdomain is taken to be the arrangement distance of 2^{29} double-precision floating-point numbers. In each subdomain, we preliminary use Taylor polynomial of degree 2 to approximate the log2(x) function. Because of the large number of

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

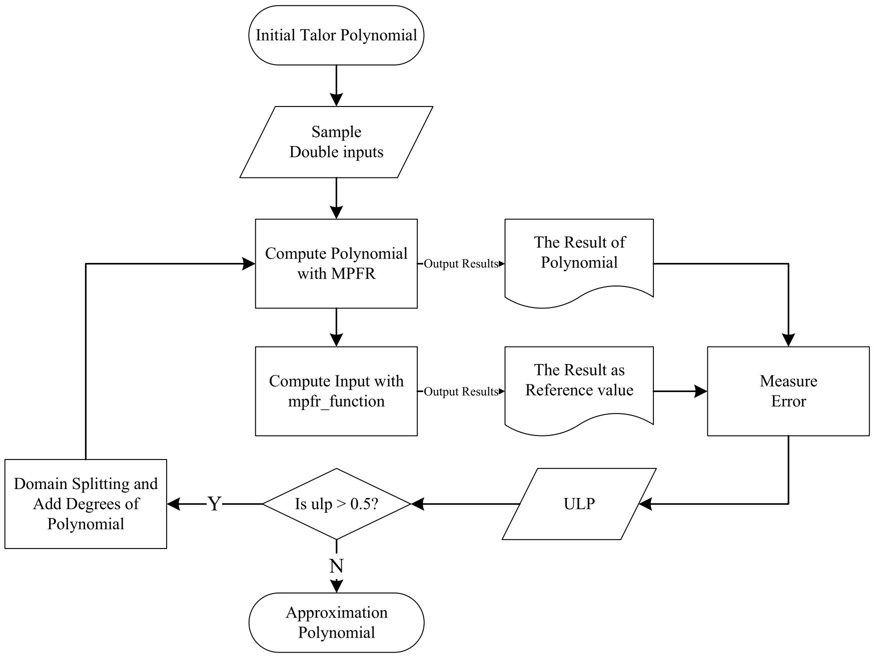

Fig. 6. The process of determining the approximation polynomial.

double-precision floating-point numbers, it is not practical to check whether the elementary function can be correctly rounded under each input. Hence, we need to sample a portion of inputs. Our practice is to select representative intervals of the function. Further, we check whether the approximation polynomial can realize the correct rounding of the elementary function in the selected intervals. According to the nature of the $\log 2(x)$ function, we select four representative intervals in the commonly used interval $[0.1, 10000.0]$ that provided by Intel. The specific intervals are:

(1) With 0.1 as the starting-point, the interval where floating-point numbers become larger.
(2) With 10000.0 as the starting-point, the interval where floating-point numbers become smaller.
(3) With 1 as the starting-point, the interval where floating-point numbers become larger.
(4) With 1 as the starting-point, the interval where floating-point numbers become smaller.

The first interval represents the smaller part of the double-precision inputs; The second interval represents the larger part of the double-precision inputs; In the third and fourth intervals, the value of the $\log 2(x)$ function is close to 0. The double-precision floating-point numbers used to represent the result are dense.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

When the polynomial is calculated according to Eq. (8), the double-precision input is shifted to determine the pivot point a of the corresponding subdomain. Then we calculate some arguments of the polynomial by MPFR. These arguments are function value, first derivative, second derivative of the pivot point a, the distance and distance square between the input and the pivot point a. Once we have obtained these arguments, we add them together using the MPFR to obtain the calculation result of the polynomial. For this double precision input, we use the mpfr_log2 function of MPFR to get the result as a reference value. This paper uses unit in the last place (ULP) as a measure of error. There are various definitions of ULP [27, 28], but we use Goldberg's definition here.

The calculation result of the polynomial above is compared with the reference value, and the relative error is calculated using the following equation:

$$\mathrm{Err}_{\mathrm{ulp}} = \frac{|\mathrm{Expression}(x)_{\mathrm{mpfr\_log2}} - \mathrm{Expression}(x)_{\mathrm{polynomial}}|}{\mathrm{ULP}_{\mathrm{mpfr\_log2}}}. \tag{9}$$

$\mathrm{Expression}(x)_{\mathrm{polynomial}}$ represents the calculation result of the polynomial. $\mathrm{Expression}(x)_{\mathrm{mpfr\_log2}}$ represents the reference value. Finally, $\mathrm{ULP}_{\mathrm{mpfr\_log2}}$ represents the error of the result calculated by MPFR. If $\mathrm{Err}_{\mathrm{ulp}}$ is small than 0.5, we can deem that the elementary function can achieve correct rounding under this input. Following the aforementioned process, double-precision inputs are verified in parallel in four representative intervals. If there is the input whose $\mathrm{Err}_{\mathrm{ulp}}$ is greater than 0.5, it shows that the elementary function cannot achieve correct rounding. Therefore, in order to reduce the error in approximating the function, it is necessary to either decrease the subdomain size or increase the degree of the Taylor polynomial. It is preferable to reduce the subdomain size, as increasing the degree of the Taylor polynomial augments the algorithm overhead. As shown in Fig. 7, we first divide an

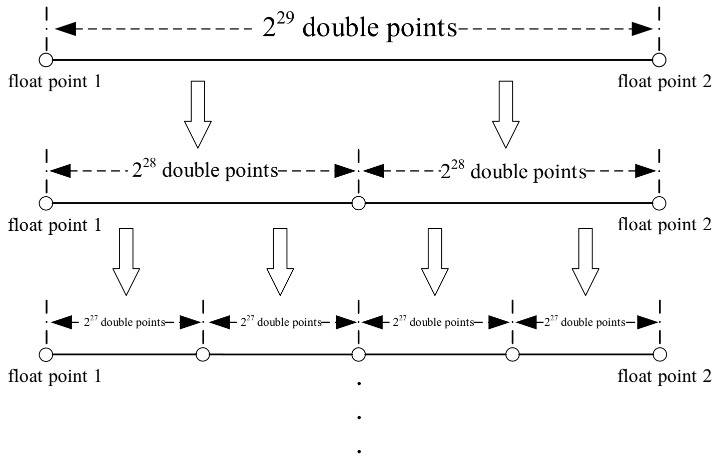

Fig. 7. Splitting subdomains.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

initial subdomain when we meet two adjacent single-precision floating-point every time. The size of each initial subdomain is the arrangement distance of 2^{29} double-precision floating-point numbers. The method of reducing the subdomain size is to split an initial subdomain into two to the power parts. For instance, we split an initial subdomain into two subdomains, four subdomains, eight subdomains and so on. Reducing the size of the subdomain has no effect on the double-precision input that is closer to the pivot point of the subdomain. However, it can reduce the error of the double-precision input that is farther away from the pivot point of the subdomain.

As shown in Fig. 7, we preferentially reduce the subdomain size. If there are too many subdomains, then we increase degrees of Taylor polynomial. In the case of high-precision calculation based on MPFR, we determined the approximation polynomial which can be used to achieve the correct rounding of the elementary function by the above process. The size of each subdomain is specifically determined: The arrangement distance of 2^{24} double-precision floating-point numbers. The degree of Taylor polynomial is determined: Three.

4.4. *Error control for floating-point computations of double precision*

In the case of high-precision calculation based on MPFR, Sec. 4.3.2 determines the approximation polynomial that can be used to achieve the correct rounding of the elementary function. However, when actually approximating the function, the algorithm's performance should be improved as much as possible. Frequent calls to MPFR can result in poor performance. Therefore, it is necessary to adopt certain error control methods to effectively reduce floating-point errors in polynomial calculation. The aim is to achieve the same computational effect as MPFR and to improve performance. The prerequisite is that the correct rounding of elementary functions is guaranteed.

Our method was inspired by the error control methods of floating-point computation in Sec. 2.3. To be specific, we simulate the high-precision representation based on the double–double data format, and combine the error-free transformation and Double-double algorithm to control the error in the process of polynomial approximation and output compensation. The following describes the specific flow of applying the error control method to achieve the correct rounding of the double-precision elementary function.

4.4.1. *Pretreatments for approximating*

Taking the $\exp(x)$ function in our SCR-LIBM as an example, we can calculate it using Eq. (7) from Sec. 4.2. According to Eq. (7), the reduced argument $t \in [4.0, 4.0 + \frac{\ln(2)}{64}]$. Therefore, it is only necessary to use Taylor polynomial to approximate $\exp(t)$ within this interval. We then obtain the calculation result of the polynomial and reconstruct the result to get the value of $\exp(x)$. We use the Taylor

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

polynomial of degree 3 to approximate $\exp(t)$ in each subdomain. Each subdomain size is the arrangement distance of 2^{24} double-precision floating-point numbers. Equation (10) which is used to approximate $\exp(t)$ is shown as follows:

$$\exp(t) = \exp(t_0) + \frac{\exp(t_0)}{1}(t - t_0) + \frac{\exp(t_0)}{2}(t - t_0)^2 + \frac{\exp(t_0)}{6}(t - t_0)^3. \qquad (10)$$

Proof. It is a Taylor polynomial of degree 3. By the equation $\exp(t) = \exp(t_0) + \frac{\exp'(t_0)}{1!}(t - t_0) + \frac{\exp''(t_0)}{2!}(t - t_0)^2 + \frac{\exp'''(t_0)}{3!}(t - t_0)^3$, we can get $\exp(t) = \exp(t_0) + \frac{\exp(t_0)}{1}(t - t_0) + \frac{\exp(t_0)}{2}(t - t_0)^2 + \frac{\exp(t_0)}{6}(t - t_0)^3$. □

When the polynomial is to be calculated by Eq. (10), some parameters of the polynomial are still unknown. Hence, it is necessary to perform pre-processing operations for approximating. We divide the pretreatments into three steps to determine the unknown parameters of the polynomial.

For each double-precision input x, there will be a corresponding t after range reduction. According to several formulas of range reduction steps, n and j are determined, respectively. In the process of calculating t, error-free transformation technology is used to ensure the accuracy of t. Getting the exact values of n, j and t is the first step of pretreatments. Having obtained the exact value of t, we determine the subdomain in which t falls at this time (i.e. the exact value of the pivot point t_0). In this paper, t is converted to an integer by the union. In addition, the single-precision floating-point numbers on either side of t are determined by the way the integer is moved first to the left and then to the right. There are 2^{29} double-precision floating-point numbers between each adjacent single-precision floating-point number. These double-precision floating-point numbers are divided into 32 subdomains (i.e. 2^{24} double-precision floating-point numbers in each subdomain). After obtaining the single-precision floating-point numbers on either side of t, we should figure out which subdomain t falls in. Equation (11) which is used to figure out which subdomain t falls in is shown as follows:

$$\left\lfloor \frac{\text{Integer}_t - \text{Integer}_{t\_\text{leftsingle}}}{2^{24}} \right\rfloor. \qquad (11)$$

Integer_t represents the hexadecimal integer of t. $\text{Integer}_{t\_\text{leftsingle}}$ represents the hexadecimal integer of single-precision floating-point number which is on the left side of t. Once we know which subdomain t falls in, we can find the pivot point t_0. Getting the exact values of t_0 is the second step of pretreatments. We should split 726,848 subdomains in the reduced interval $[4.0, 4.0 + \frac{\ln(2)}{64}]$. For the reason that the size of each subdomain is the arrangement distance of 2^{24} double-precision floating-point numbers. That is, there are a total of 726,848 pivot points. According to Eq. (10), an important parameter which is used to calculate $\exp(t)$ is the value of $\exp(t_0)$. It is beneficial for the correct rounding of the polynomial calculation if an accurate value of $\exp(t_0)$ can be obtained. For the $\exp(x)$ function, there are 726,848 values of

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

$\exp(t_0)$. In this paper, we use Sollya [29], a secure floating-point code development tool, to generate 726,848 exact values of $\exp(t_0)$. These results are stored into stack in the double–double data format. The above process is the third step of the pretreatments before the polynomial calculation.

4.4.2. *Polynomial calculation based on error control method*

By pretreatments for approximating, we get exact parameters of the polynomial. Further, we need to get the value of $\exp(t)$ by polynomial approximation. Since the value of $\exp(t_0)$ is known from Eq. (10), we extract the common factor $\exp(t_0)$ from Eq. (10) to reduce calculation times. Equation (10) is rewritten as the following equation:

$$\exp(t) = \exp(t_0) \cdot [C_0 + C_1 \cdot C_D + C_2 \cdot C_D^2 + C_3 \cdot C_D^3], \tag{12}$$

where C_0 is 1.0, C_1 is 1.0, C_2 is 0.5, C_3 is $\frac{1}{6}$ and C_D is the value of $(t - t_0)$. To further reduce calculation times, Eq. (12) was calculated using the Horner algorithm [30]. The polynomial was deformed as follows:

$$\exp(t) = \exp(t_0) \cdot (C_0 + C_D \cdot (C_1 + C_D \cdot (C_2 + C_3 \cdot C_D))). \tag{13}$$

Based on the analysis in Sec. 4.3, $\exp(t)$ can achieve correct rounding based on the condition that using MPFR to calculate Eq. (13). To achieve the same calculation effect as MPFR, this paper uses the error control method to calculate the polynomial. In terms of data types, to improve accuracy of representation, double–double data format is used for key parameters. We utilize double data format for parameters that are able to tolerate errors. Similarly, key steps are calculated using the Double-double algorithm. Error-tolerant steps are calculated using error-free transformation or even direct floating-point computations. The flow of polynomial approximation based on error control method is shown as Fig. 8.

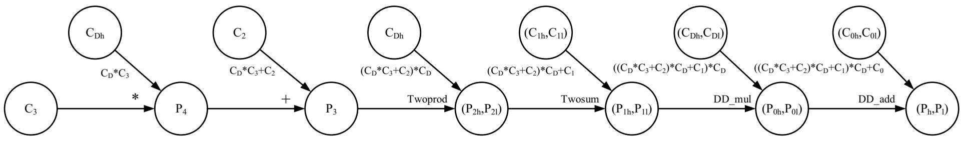

Fig. 8. The flow of polynomial approximation based on error control method.

The first is to determine the data type used to represent different polynomial parameters. C_D is $(t - t_0)$, which represents the distance between the reduced argument t and the pivot point of the subdomain. Since C_D participates in the computation several times, we determine the data type of C_D depending on the order of computation. We calculate $(t - t_0)$ using error-free transform addition. The result is represented as two double numbers, denoted as C_{Dh} and C_{Dl}. The C_D multiplied by C_3 is represented by C_{Dh}. Similarly, the C_D multiplied by C_2 is represented by C_{Dh}. The C_D multiplied by C_1 is represented by the double–double data format.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

(i.e. (C_{Dh}, C_{Dl})). C_2 and C_3 are represented by double, while C_0 and C_1 are represented by double–double data format. Once we have determined the data type used to represent different polynomial parameters, we then determine whether the steps are calculated using the Double-double algorithm or error-free transformation, or even direct floating-point computations. The first step calculated by the Horner [30] algorithm is the last expansion term of the Taylor polynomial. The higher the order of the expansion term of the polynomial, the smaller the value and the less the influence on the overall result of the polynomial calculation. Hence, we determine the degree of error control by the order of the polynomial calculation. The anterior steps use floating-point computations or error-free transformation. The later steps use the Double-double algorithm to improve the accuracy of the calculation. Therefore, for the calculation step $C_2 + C_3 \cdot C_D$, the data type used to represent the parameter is double. Because this calculation step is error-tolerant. It only needs to use the multiplication and addition of floating-point operations. For this multiplication step $(C_2 + C_3 \cdot C_D) \cdot C_D$, the result is to be represented as double–double data format. Therefore, the error-free transformation product algorithm is used for the calculation. Similarly, for the addition step $(C_2 + C_3 \cdot C_D) \cdot C_D + C_1$, we use the error-free transform addition algorithm for calculation. For the last two steps of the polynomial calculation, Double-double algorithm is used to improve the accuracy of the calculation. Calculated through the above steps, we can get the exact value of $(C_0 + C_D \cdot (C_1 + C_D \cdot (C_2 + C_3 \cdot C_D)))$, which we denote as (P_h, P_l). To get the result of $\exp(t)$, we need to multiply $\exp(t_0)$ by the value of (P_h, P_l). We will calculate this step in the output compensation part.

The polynomial approximation is completed by the above steps. Further, we carry out the reconstruction of the result. We rewrite Eq. (7) to form Eq. (14).

$$\exp(x) = \exp(-4.0) * 2^n * 2^{\frac{j}{64}} * \exp(t_0) * P. \tag{14}$$

Proof. We use the P to represent the result of $(C_0 + C_D \cdot (C_1 + C_D \cdot (C_2 + C_3 \cdot C_D)))$. Because $\exp(t) = \exp(t_0) \cdot (C_0 + C_D \cdot (C_1 + C_D \cdot (C_2 + C_3 \cdot C_D)))$, we can get $\exp(t) = \exp(t_0) \cdot P$. By Eq. (7) $\exp(x) = \exp(-4.0) * 2^n * 2^{\frac{j}{64}} * \exp(t)$, we can get $\exp(x) = \exp(-4.0) * 2^n * 2^{\frac{j}{64}} * \exp(t_0) * P$. □

The flow of output compensation based on error control method is shown as Fig. 9.

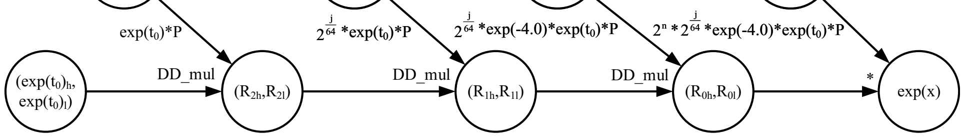

Fig. 9. The flow of output compensation based on error control method.

Here, P is the calculation result of polynomial approximation and is represented as (P_h, P_l). According to pretreatments for approximating in Sec. 4.4.1, values of all $\exp(t_0)$ were calculated in advance using Sollya and stored into stack in the double–double data format. Here, j is the value of the set $\{0, 1, 2, 3, \ldots, 63\}$, so there are only 64 possibilities in total. Hence, the value of $2^{\frac{j}{64}}$ can be accurately calculated in advance through the high-precision library. Further, these values are stored in the double–double data format. We can refer to the table for these values. Similarly, the value of $\exp(-4.0)$ can be accurately calculated in advance through the high-precision library and stored in the double–double data format. In the output compensation part, except for multiplying by 2^n, we use Double-double multiplication algorithm to calculate every step to ensure the accuracy of the calculation.

5. Experimental Evaluation

We use the Taylor polynomial of degree 3 to approximate the elementary function. Each subdomain size is the arrangement distance of 2^{24} double-precision floating-point numbers. We use error control method to control the error in the process of polynomial approximation and output compensation. These ensure that the elementary function is correctly rounded. In our SCR-LIBM, 10 elementary functions have been correctly rounded. To verify the effect of our SCR-LIBM, we will select rigorous and authoritative test methods to test these 10 elementary functions.

5.1. *Functions for test*

Four classes of elementary functions have been realized in SCR-LIBM, including three logarithmic functions, four trigonometric functions, two hyperbolic functions and one exponential function. Through the connection and similarity between the functions, SCR-LIBM can be rapidly expanded according to the 10 elementary functions that have been realized so far. For example, the inverse hyperbolic function is the composite function of the simple algebraic expression and the logarithmic function. It can be realized on the basis of the existing logarithmic function in this paper. Currently, mainstream libraries that can achieve correct rounding, such as RLIBM [31] for single-precision floating-point numbers and CR-LIBM for double-precision floating-point numbers, mostly implement elementary functions that are more commonly used. For the above reasons, SCR-LIBM implements elementary functions as shown in Table 1.

5.2. *Experimental setup*

The experimental environment used in this paper is listed in Table 2.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

Table 1. List of elementary functions.

No.	Function name	Function type
1	$\log(x)$	Logarithmic function
2	$\log2(x)$	Logarithmic function
3	$\log10(x)$	Logarithmic function
4	$\sin(x)$	Trigonometric function
5	$\cos(x)$	Trigonometric function
6	$\text{sinpi}(x)$	Trigonometric function
7	$\text{cospi}(x)$	Trigonometric function
8	$\sinh(x)$	Hyperbolic function
9	$\cosh(x)$	Hyperbolic function
10	$\exp(x)$	Exponential function

Table 2. Test environment of software and hardware.

Hardware environment	
CPU	Intel Xeon E5-2630 v4
Clock	2.10 GHz
Memory	64 GB
Cache sizes	64 KB L1, 1024 KB L2, 2816 KB L3
Software environment	
OS	Ubuntu 20.04.5 LTS
Compiler	GCC v9.4.0
GCC Optimize-options	"-O3"

5.3. *The accuracy test of double-precision elementary functions*

The number of double-precision floating-point numbers is too large. For the reason, we cannot verify that the elementary functions are rounded correctly under each input. In order to effectively verify the implemented elementary function, this paper performs a accuracy test as shown in Fig. 10.

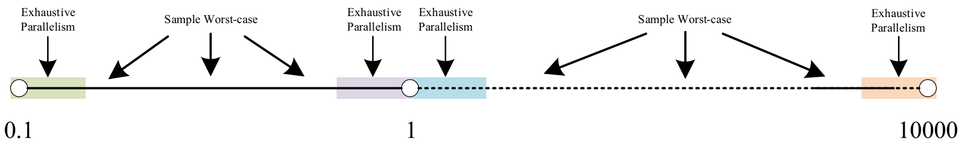

Fig. 10. The method of accuracy test.

Taking the $\log2(x)$ function as an example, we select representative intervals of the function. Further, we check whether the approximation polynomial can realize the correct rounding of the elementary function in the selected intervals. According to the nature of the $\log2(x)$ function, we select four representative intervals in the commonly used interval $[0.1, 10000.0]$ that were provided by Intel. As shown in

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

Fig. 10, four representative intervals are marked with different colors. The four representative intervals are as follows:

(1) With 0.1 as the starting-point, the interval where floating-point numbers become larger.
(2) With 10000.0 as the starting-point, the interval where floating-point numbers become smaller.
(3) With 1 as the starting-point, the interval where floating-point numbers become larger.
(4) With 1 as the starting-point, the interval where floating-point numbers become smaller.

The first interval represents the smaller part of the double-precision inputs; The second interval represents the larger part of the double-precision inputs; In the third and fourth intervals, the value of the $\log2(x)$ function is close to 0. The double-precision floating-point numbers used to represent the result are dense. If the result of the calculation deviates slightly from the reference value obtained by MPFR, the measured ULP is greater than 0.5. Hence, it is necessary to focus on verification at third and fourth intervals.

In four representative intervals, we detect the elementary function and whether it can achieve correct rounding under all inputs of selected intervals by parallel operation. By comparing the result of the realized elementary function with the reference value obtained by MPFR, the relative error is calculated. If the ULP measured is greater than 0.5, there is a double-precision input that cannot be rounded correctly.

Verification is carried out at four representative intervals. If there is no case that the double-precision input cannot be rounded correctly, the next verification step is taken.

The following accuracy test is an important part of our verification process. In addition to the four representative intervals for checking correct rounding, the remaining intervals of the $\log2(x)$ function are dedicated to testing the accuracy of the worst-case of correct rounding. In the remaining intervals of the $\log2(x)$ function, we specifically collected a subset of the worst-case inputs. As CR-LIBM provides correct rounding by two iterations and the second iteration is dedicated to the worst-case of correct rounding. Hence, we use CR-LIBM to get some of the worst-case inputs. The specific accuracy test procedure for worst-case inputs is as follows. In the interval without exhaustive parallel accuracy testing, a part of worst-case inputs are evenly sampled as a test set. The size of the test set is 10,000 worst-case inputs. Further, for these 10,000 worst-case inputs, the relative error is calculated by comparing the calculation results of the elementary function with the reference value obtained by MPFR. If the measured ULP is greater than 0.5, this worst-case input cannot be rounded correctly. These 10,000 worst-case inputs are evenly distributed over the interval that is not exhaustively parallel accuracy tested. And worst-case is recognized as the more difficult error case [32] to deal with incorrect rounding.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

For these two reasons, if ULP is less than 0.5 for all 10,000 inputs, it can be further explained that the elementary function implemented in this paper can achieve correct rounding.

According to the above overall testing process, accuracy test is performed on the functions implemented in SCR-LIBM, CR-LIBM library [11] and Glibc library [5]. The results of the tests are shown in Table 3.

Table 3. The result of accuracy test.

	SCR-LIBM		CR-LIBM		Glibc	
Func	Max ulp	Correctly rounded	Max ulp	Correctly rounded	Max ulp	Correctly rounded
log(x)	0.49999	Y	0.49999	Y	0.50027	N
log2(x)	0.49999	Y	0.49999	Y	1.59318	N
log10(x)	0.49999	Y	0.49999	Y	1.56431	N
sin(x)	0.49999	Y	0.49999	Y	0.50388	N
cos(x)	0.49999	Y	0.49999	Y	0.50735	N
sinpi(x)	0.49999	Y	0.49999	Y	N/A	N/A
cospi(x)	0.49999	Y	0.49999	Y	N/A	N/A
sinh(x)	0.49999	Y	0.49999	Y	1.66053	N
cosh(x)	0.49999	Y	0.49999	Y	1.49974	N
exp(x)	0.49999	Y	0.49999	Y	0.99911	N

According to the test results, all 10 functions implemented in SCR-LIBM and in CR-LIBM can be rounded correctly (i.e. the maximum ULP is less than 0.5). The test results show that Glibc cannot achieve correct rounding (i.e. sinpi(x) and cospi(x) do not exist in Glibc). It can be seen from the above results that the elementary function implemented in SCR-LIBM meets the requirement of correct rounding.

5.4. *Performance evaluation of double-precision elementary function*

5.4.1. *Average performance evaluation*

In this paper, the high-precision library MPFR and the double-precision correctly rounded library CR-LIBM are selected as the performance comparison objects. MPFR sets the precision bit to 128 bits to ensure that all double-precision inputs are rounded correctly. In order to maintain the consistency of the test, the same test interval is selected for each function. We sample 10,000 double-precision inputs and compared performance by testing the average performance cycles. In order to minimize the influence of the external environment, a data warm-up is performed before the function is tested (i.e. the function to be tested is called several times). The specific average performance evaluation formula is shown as follows:

$$\text{avgPERF} = \frac{1}{n}\sum_{i=1}^{n}\text{cycle}_i. \qquad (15)$$

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

The equation represents the sum of the performance cycles of 10,000 inputs and then averaged, where n represents the specific number of test set (i.e. in this paper is 10,000). cycle$_i$ represents the performance cycle of per double-precision input. Using Eq. (15), we evaluate the average performance and the test results are shown in Fig. 11.

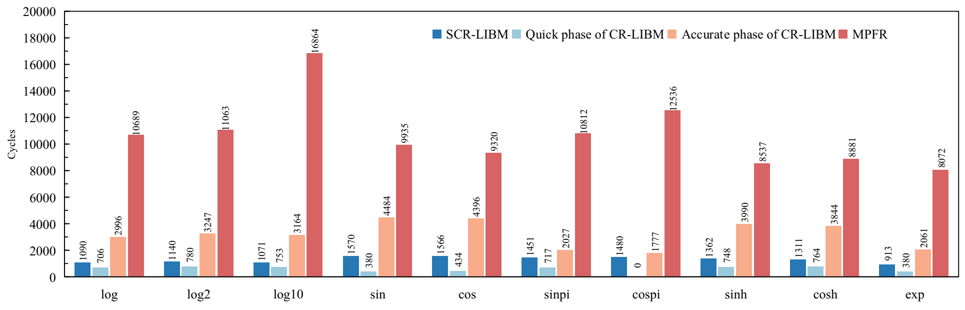

Fig. 11. The test result of average performance.

It can be found from the test results that the performance cost of MPFR is very large. Among these functions, the performance we measured for the log10(x) function was up to 17,000 cycles. Mainly because of the characteristics of multi-digit mode, MPFR can't effectively use the floating-point power of the current processor. Hence, this is an important reason why this paper uses the error control method instead of MPFR for floating-point calculation.

CR-LIBM provides correct rounding through two phases. We found that about 0.35% of double-precision inputs need to go to the second phase. What's more, it is worth noting that cospi(x) function in CR-LIBM only has the second phase, so we set its first-phase performance to 0. In conclusion, although the first phase of CR-LIBM is fast, it has poor performance in the second phase. In addition, the performance of the two phases is quite different. The performance of CR-LIBM's sin(x) and cos(x) functions is even more than 10 times different.

The 10 elementary functions implemented in SCR-LIBM have a huge advantage over MPFR in average performance, but are worse than CR-LIBM in average performance. This is because we have fully considered various input cases, including the worst case, when designing the correct rounding method for elementary functions. Hence, the correct rounding of elementary function is completed with higher precision, eliminating the problem of unstable performance and large fluctuation caused by dividing phases.

5.4.2. *Performance evaluation of worst-case*

Worst-case execution performance is also very important for the elementary function. It is directly related to whether the elementary function can be used in some

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

critical real-time applications, such as real-time avionics and embedded devices. Therefore, we perform the performance test for worst-case of correct rounding. The test set consists of 10,000 inputs that have been selected for the worst-case accuracy test in Sec. 5.3. Specifically, we select MPFR and CR-LIBM as the object of comparison. We calculate the average worst-case execution performance speedup of the function implemented in SCR-LIBM relative to MPFR and CR-LIBM. The speedup of each function of SCR-LIBM over MPFR and CR-LIBM is calculated by Eqs. (16) and (17). For each function, MPFR cycle, CR-LIBM cycle and SCR-LIBM cycle represent the average performance cycle when dealing with these worst-case inputs.

$$\text{Speedup} = \frac{\text{MPFRcycle}}{\text{SCR} - \text{LIBMcycle}}, \tag{16}$$

$$\text{Speedup} = \frac{\text{CR} - \text{LIBMcycle}}{\text{SCR} - \text{LIBMcycle}}. \tag{17}$$

The test results are shown in Fig. 12. Compared to CR-LIBM, the function implemented in SCR-LIBM has large performance advantages in worst-case execution performance. Specifically, compared to CR-LIBM, among the 10 functions implemented in SCR-LIBM, the minimum performance speedup is 1.201, the maximum is 2.954, and the average speedup is 2.492. It's worth noting that the sinpi(x) and cospi(x) functions in CR-LIBM have the better performance compared to other functions in CR-LIBM. This is despite the speedup of sinpi(x) and cospi(x) functions being smaller than that of other functions, but still greater than 1. Compared to CR-LIBM, SCR-LIBM is superior in dealing with the worst-case of correct rounding. In comparison with MPFR, the minimum performance speedup is 5.951, the maximum is 15.746, and the average speedup is 8.534. The maximum speedup is up to 15.746

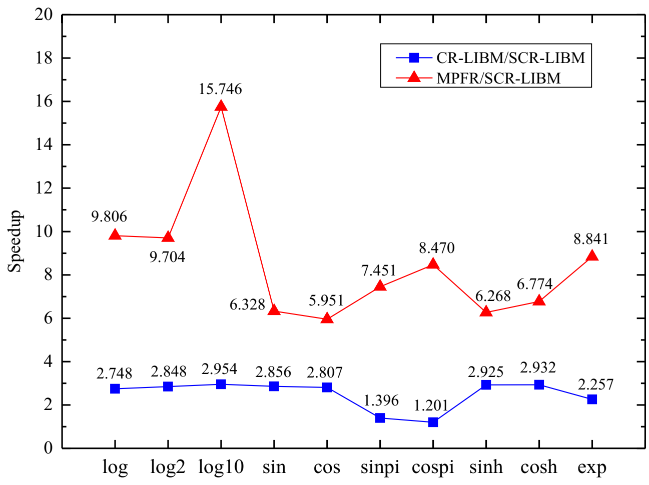

Fig. 12. The test result of worst-case execution performance.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

because the log10(x) in MPFR has a poor performance compared to other functions. In conclusion, the elementary function implemented in SCR-LIBM can be better applied to real-time applications with strict upper execution time requirements.

5.4.3. *Performance stability evaluation*

Overall performance stability of elementary functions is critical for some uses. Hence, we have compared the performance stability of SCR-LIBM and CR-LIBM. The exp(x) function is chosen as an example to show the performance stability comparison. To make a fair comparison, we use the same test set. The test set consists of 10,000 inputs ranging from -707.0 to 707.0, according to the commonly used interval provided by Intel. In addition, the test set consists of two types of inputs: A part of worst-case inputs and a part of common inputs.

We have tested SCR-LIBM and CR-LIBM for these 10,000 inputs and obtained the exp function performance cycles. Figure 13 presents the performance fluctuation of SCR-LIBM and CR-LIBM. According to the test result, it turned out that our SCR-LIBM is more stable than CR-LIBM.

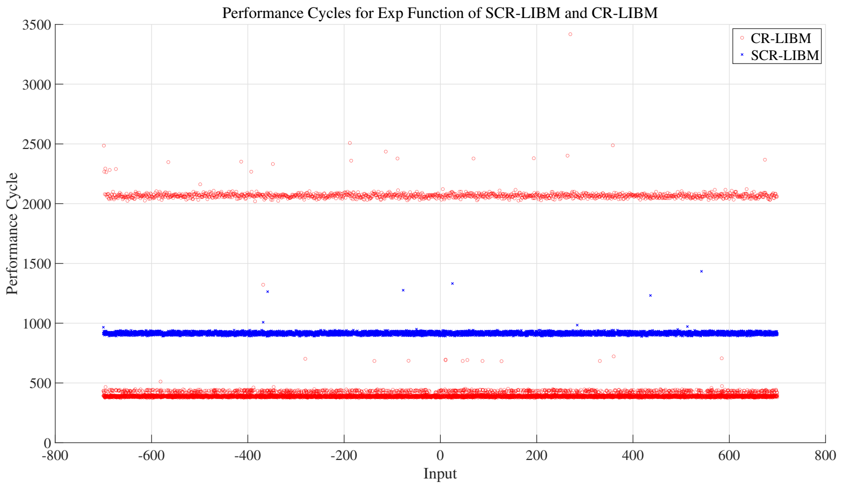

Fig. 13. Comparison of the performance stability of SCR-LIBM and CR-LIBM.

First, we observe that the CR-LIBM's performance cycle trend forms two lines. The upper line of the CR-LIBM is the performance cycle trend of dealing with the worst-case of correct rounding. Correspondingly, the lower line shows the performance cycle trend for common inputs. When CR-LIBM deals with continuous inputs, its performance can fluctuate up to almost 1800 cycles as shown in Fig. 13. Under the same input set for CR-LIBM, the performance cycle trend of our SCR-LIBM forms only one line. This means that performance is stable whether

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

SCR-LIBM is dealing with worst-case or common inputs. Second, there are some discrete points beside lines in the figure. These points are the result of fluctuations in the test under the influence of the external environment. In the experiments reported on Fig. 13, the number of discrete points belonging to SCR-LIBM is less than that of CR-LIBM. This phenomenon can indirectly reflect the stability of the function.

6. Conclusion and Future Work

We designed an efficient elementary function algorithm that divides subdomains and uses the low-degree Taylor polynomial to approximate function in the subdomain in this paper. Our algorithm solves the problem that traditional elementary function libraries need to use iterations to achieve correct rounding in the worst-case. In the process of realizing elementary functions under double precision, we simulate the high-precision representation based on the double–double data format, and combine the error-free transformation and Double-double algorithm to control the error in the process of polynomial approximation and output compensation. The test results show that the elementary functions implemented in SCR-LIBM can be rounded correctly. The average performance of SCR-LIBM is 8.534 times faster than MPFR. However, our average performance is worse than CR-LIBM. This is because we have fully considered various input cases, including the worst-case, when designing the correct rounding method for elementary functions. Aiming at the demand scenario of real-time application, we perform the performance test for worst-case of correct rounding. The test results show that SCR-LIBM has a huge advantage over MPFR and CR-LIBM when dealing with the worst-case of correct rounding. In addition, we carry out the experiment for the performance stability of SCR-LIBM and CR-LIBM. The experiment result shows that SCR-LIBM is more stable than CR-LIBM. Therefore, SCR-LIBM can be better applied to real-time applications.

In the future, we plan to optimize the performance of the implemented functions to improve the average performance. For instance, we will be looking at our hardware strategy in the future to increase efficiency, which is an excellent concept. In addition, other elementary functions will be rapidly extended according to the elementary functions that have been realized so far. Besides, it is difficult to prove that a math library can provide correctly rounded results for all double-precision inputs. Because the amount of double-precision floating-point number is too large, it is impossible to verify all inputs. Hence, we will pay more attention to the verification of math libraries. We plan to extend our accuracy testing method in order to cover more double-precision inputs.

Acknowledgment

This project did not receive any funding support.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.

ORCID

Yang Qu https://orcid.org/0009-0000-0575-896X
Jinchen Xu https://orcid.org/0000-0002-6275-2617
Bei Zhou https://orcid.org/0000-0003-1515-0602
Jiangwei Hao https://orcid.org/0000-0001-6012-0870
Fei Li https://orcid.org/0009-0002-1706-3625
Zuoyan Zhang https://orcid.org/0009-0005-2615-2541

References

1. J. Muller, Elementary functions and approximate computing, *Proc. IEEE* **108**(12) (2020) 2136–2149.
2. D. Zuras *et al.*, IEEE standard for floating-point arithmetic, in IEEE Std 754-2008, pp. 1–70 (2008), doi: 10.1109/IEEESTD.2008.4610935.
3. E. Wang *et al.*, Intel math kernel library, in *High-Performance Computing on the Intel Xeon PhiTM* (Springer, Cham, 2014), pp. 167–188.
4. AMD, AMD math library, https://developer.amd.com/amd-aocl/amd-math-library-libm/.
5. S. Loosemore *et al.*, *The GNU C Library Reference Manual* (version 2.23 edn.) (2016).
6. A. Godunov, Algorithms for calculating correctly rounded exponential function in double-precision arithmetic, *IEEE Trans. Comput.* **69**(9) (2020) 1388–1400.
7. J. P. Lim and S. Nagarakatte, One polynomial approximation to produce correctly rounded results of an elementary function for multiple representations and rounding modes, in *Proc. ACM Programming Languages*, 2022, pp. 1–28.
8. M. Schulte and E. Swartzlander, Exact rounding of certain elementary functions, in *Proc. 11th IEEE Symp. Computer Arithmetic*, 1993, pp. 138–145.
9. V. Lefèvre and P. Zimmermann, Optimized binary64 and binary128 arithmetic with GNU MPFR, in *Proc. 24th IEEE Symp. Computer Arithmetic*, 2017, pp. 18–26.
10. IBM, Accurate Portable MathLib, http://oss.software.ibm.com/mathlib/ (2008).
11. C. Daramy *et al.*, CR-LIBM: A correctly rounded elementary function library, in *Proc. SPIE Advanced Signal Processing Algorithms Architectures Implementations XIII*, Vol. 5205, 2003, pp. 458–464.
12. V. Lefèvre and J. Muller, Worst cases for correct rounding of the elementary functions in double precision, in *Proc.15th IEEE Symp. Computer Arithmetic*, 2001, pp. 111–118.
13. L. Fousse *et al.*, MPFR: A multiple-precision binary floating-point library with correct rounding, *ACM Trans. Math. Softw.* **33**(2) (2007) 13-es.
14. F. de Dinechin, A. Ershov and N. Gast, Towards the post-ultimate libm, in *Proc. 17th IEEE Symp. Computer Arithmetic*, 2005, pp. 288–295.
15. D. A. Celarier and D. W. Sando, An Ada math library for real-time avionics, *Ada Lett.* **XI** (1991) 274–284.
16. M. Schoeberl *et al.*, Patmos: A time-predictable microprocessor, *Real-Time Syst.* **54** (2018) 389–423.
17. H. de Lassus Saint-Geniès, D. Defour and G. Revy, Exact lookup tables for the evaluation of trigonometric and hyperbolic functions, *IEEE Trans. Comput.* **66**(12) (2017) 2058–2071.
18. L. Trefethen, *Approximation Theory and Approximation Practice*, Extended edn. (Society for Industrial and Applied Mathematics (SIAM), 2019).

19. N. Brisebarre and S. Chevillard, Efficient polynomial L-approximations, in *Proc. 18th IEEE Symp. Computer Arithmetic*, 2007, pp. 169–176.
20. J. P. Lim *et al.*, An approach to generate correctly rounded math libraries for new floating point variants, in *Proc. ACM Programming Languages*, 2021, pp. 1–30.
21. V. Lefèvre, Correctly rounded arbitrary-precision floating-point summation, in *Proc. 23th IEEE Symp. Computer Arithmetic*, 2016, pp. 71–78.
22. J. Muller and L. Rideau, 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.* **48** (2022) 1–24.
23. N. Fabiano, J. Muller and J. Picot, Algorithms for triple-word arithmetic, *IEEE Trans. Comput.* **68**(11) (2019) 1573–1583.
24. S. Graillat, Accurate floating-point product and exponentiation, *IEEE Trans. Comput.* **58**(7) (2009) 994–1000.
25. S. Graillat and F. Jézéquel, Tight interval inclusions with compensated algorithms, *IEEE Trans. Comput.* **69**(12) (2018) 1774–1783.
26. T. Dekker, A floating-point technique for extending the available precision, *Numer. Math.* **18** (1971) 224–242.
27. J.-M. Muller, On the definition of ulp(x), Technical Report 5504, INRIA (2005).
28. M. Cornea, ULPs and relative error, in *Proc. 24th IEEE Symp. Computer Arithmetic*, 2017, pp. 90–97.
29. S. Chevillard, M. Joldeş and C. Lauter, Sollya: An environment for the development of numerical codes, in *Proc. 3rd Int. Congress on Mathematical Software*, 2010, pp. 28–31.
30. J. Czekansky and T. Sauer, The multivariate Horner scheme revisited, *BIT Numer. Math.* **55** (2015) 1043–1056.
31. J. P. Lim and S. Nagarakatte, RLIBM-32: High performance correctly rounded math libraries for 32-bit floating point representations, in *PLDI '21: 42nd ACM SIGPLAN Int. Conf. Programming Language Design and Implementation, Virtual Event*, pp. 359–374 (2021).
32. D. Stehlé, V. Lefèvre, P. Zimmermann, Worst cases and lattice reduction, in *Proc. 16th IEEE Symp. Computer Arithmetic*, 2003, pp. 142–147.

Int. J. Soft. Eng. Knowl. Eng. 2024.34:675-703. Downloaded from www.worldscientific.com
by ZHEJIANG UNIVERSITY on 05/31/24. Re-use and distribution is strictly not permitted, except for Open Access articles.