HAL
archives-ouvertes.fr

Fast correct rounding of elementary functions in double precision using double-extended arithmetic

Dinechin, Florent De, David Defour, Christophe Lauter

▶ **To cite this version:**

Dinechin, Florent De, David Defour, Christophe Lauter. Fast correct rounding of elementary functions in double precision using double-extended arithmetic. [Research Report] Laboratoire de l'informatique du parallélisme. 2004, 2+12p. hal-02102114

HAL Id: hal-02102114
https://hal-lara.archives-ouvertes.fr/hal-02102114

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

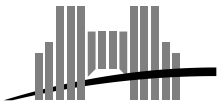
Fast correct rounding of elementary functions in double precision using double-extended arithmetic

Florent de Dinechin,
David Defour,
Christoph Lauter

March 2004

Research Report N° 2004-10

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr



Fast correct rounding of elementary functions in double precision using double-extended arithmetic

Florent de Dinechin,
David Defour,
Christoph Lauter

March 2004

Abstract

This article shows that IEEE-754 double-precision correct rounding of the most common elementary functions (exp/log, trigonometric and hyperbolic) is achievable on current processors using only double-double-extended arithmetic. This allows to improve by several orders of magnitude the worst case performance of a correctly-rounded mathematical library, compared to the current state of the art.
This article builds up on previous work by Lefèvre and Muller, who have shown that an intermediate accuracy of up to 158 bits is required for the evaluation of some functions. We show that the practical accuracy required can always be reduced to less than 119 bits, which is easy to obtain using well-known and well-proven techniques of double-double-extended arithmetic. As an example, a prototype implementation of the exponential function on the Itanium has a worst-case time about twice that of the standard, highly optimized `libm` by Intel, which doesn't offer correct rounding. Such a small performance penalty should allow correct rounding of elementary functions to become the standard.

Keywords: elementary functions, correct rounding, IEEE-754, double-extended precision

Résumé

Cet article montre que l'arrondi correct des fonctions élémentaires en double précision peut être obtenu rapidement en utilisant la précision double-étendue sur les processeurs qui la supportent. Ceci permet une accélération de plusieurs ordres de grandeur au pire cas par rapport aux bibliothèques avec arrondi correct existantes qui utilisent des techniques de multiprécision plus lourdes.
Cet article s'appuie sur les travaux de Lefèvre et Muller, qui ont montré que la précision intermédiaire nécessaire à l'implémentation d'une fonction élémentaire avec arrondi correct pouvait monter jusqu'à 158 bits. Cet article montre que la précision pratique peut toujours être ramenée à moins de 119 bits, une précision facile à atteindre en utilisant des techniques bien connues de double-double-étendue. Par exemple, une implémentation prototype de l'exponentielle avec arrondi correct sur Itanium a un temps d'exécution au pire cas dans un facteur 2 de l'implémentation standard, hautement optimisée, de la `libm` d'Intel sans arrondi correct. Une perte de performance si minime permet d'espérer une généralisation de l'arrondi correct des fonctions élémentaires en double précision.

Mots-clés: fonctions élémentaires, arrondi correct, IEEE-754, précision double étendue

Fast correct rounding of elementary functions in double precision using double-extended arithmetic

26th March 2004

1 Introduction

1.1 Correct rounding, and the benefits of floating-point standardization

The IEEE 754 standard, adopted in 1985 [1], enabled a certain level of *portability* of floating-point algorithms accross standard-compliant platforms. To achieve this, the standard defines two main floating point formats called single and double precision, and defines the exact semantics of a range of basic operations (addition, substraction, multiplication, division, remainder and square root). All these operations have to produce *correctly rounded* results, as if the operation were carried out in infinite precision and this intermediate result were then rounded. The standard defines four rounding modes: Round to nearest, round towards plus infinity, round towards minus infinity, and round towards zero. An algorithm using only these basic operations has the exact same numerical behaviour on any system which complies to the standard. Besides, this well-defined behaviour even in the presence of rounding errors also enables *proving* numerical properties in a portable way.

There are, however still many obstacles to portability and provability. The first one is that IEEE-754 compliance is a result of complex interactions between a language, a compiler, an operating system, and the hardware. A second obstacle is that standard compliance hinders performance: the compiler is not free to perform many arithmetic optimizations; the hardware is not free to use its more advanced features like extended-precision, or fused multiply-and-add (FMA) operations.

1.2 No standard behaviour for elementary functions

The last, and main obstacle to portability and provability is that only the basic arithmetic operators are specified in the standard. Most of real application code involves also elementary functions (exponential, logarithm, trigonometric functions, etc.) [15]. Current implementations of these functions provide a very good probability of correct rounding [14], but guaranteed correct rounding is very difficult to achieve because of the so-called *Table Maker's Dilemma*: the intermediate accuracy required to be sure that rounding the approximation is equivalent to rounding the mathematical result may be very high. This is the main reason why elementary functions were left out of the IEEE-754 standard. As an example, Lefèvre and Muller were able to establish that 158 bits of intermediate accuracy are required in the worst case for correctly rounding the exponential in double precision[12, 11]. As a double-precision number has a mantissa of 53 bits, this is three times the target precision.

1.3 State of the art : Multilevel strategies for correct rounding

Some libraries already provide correctly rounded elementary functions. They use a technique due to Ziv [17] which consists in evaluating the function in several steps. A first step evaluates

the function with a few bits more than the target precision. If this accuracy is not enough to decide rounding, a second, more accurate step is taken, and so on until the correct rounding can be returned. As the function almost always returns its result in the first step, which is similar to a standard library evaluation, the average overhead of this strategy is acceptable compared to a standard library. Examples of such library are Ziv's Ultimate Math Library at IBM[13], and the `crlibm` project of the Arénaire team[4].

The main drawback of this approach, however, is its worst case execution behaviour: until the work of Lefèvre the required accuracy could not even be usefully bounded. Lefèvre was able to compute the worst-case accuracy required for some functions, but even then the use of a multiple-precision algorithm in the second step means a slow and memory-hungry worst-case behaviour. For instance, the worst case performance of IBM's Ultimate Math Library can be several hundred times slower than the average time.

1.4 Contributions of the paper

The main result of this article is to show that the practical intermediate accuracy required for correctly rounding all the functions for which the worst cases are known is much less than the actual worst-case accuracy : in all the cases where the worst-case accuracy is greater than 119 bits, we are able to build an algorithm that only works with an intermediate accuracy of less than 119 bits. This result is detailed and proven in Section 2. We then need to recover the greater accuracy from the intermediate result: Section 3 shows how to do it in a few double-extended floating-point additions.

1.5 Significance : Towards mainstream correct rounding

Going down from 158 bits of intermediate accuracy to 119 may seem a small gain. The impact on worst-case performance, however, will be paramount on architectures with hardware double-extended support: In this case there is no need anymore for a generic multiple-precision package (such as `crlibm`'s SCS or the MPA library within IBM's Ultimate MathLib). Instead, it is possible to use the classical technique of representing a high-precision number as a sum of two floating-point numbers, which can hold 128 bits of precision, as represented on Figure 1.

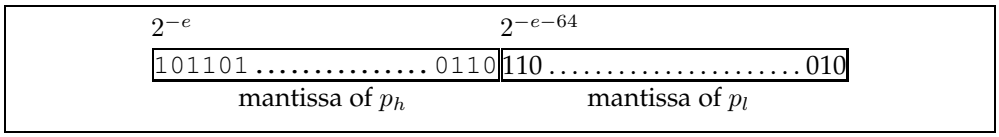


Figure 1: Representing a 128-bit number as the sum of two double-extended numbers $p_h + p_l$

Such architectures include the ubiquitous IA32 processor families by Intel and AMD, and the Intel Itanium processors. Some other instruction sets (Sparc, Power, PA-RISC) have provision for quad precision (with 112 bits of mantissa). Although it is currently mostly software-based, more and more hardware support will be available in the future.

The benefits of using double-double-extended (DDE) arithmetic for internal computations of elementary functions are the following:

- Algorithms for computing on numbers as the sum of two double-extended floating-point numbers are well known [7] and well proven [2].
- They are much simpler than more general multiple-precision arithmetic, and have a much smaller memory footprint.
- DDE is not required for the wole of the computation. Typically, for example, a function is approximated by a polynomial in Horner form, and only the last few iterations need to use double-double arithmetic. In contrast, using a multiple-precision library usually means that the whole computation is performed in high precision.

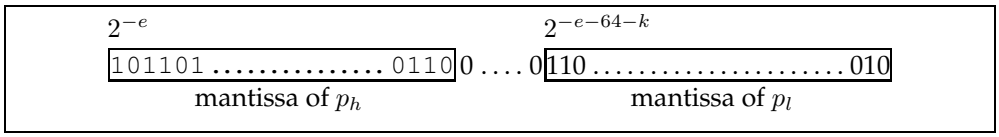


Figure 2: Representing more than 128 bits as the sum of two double-extended numbers $p_h + p_l$

- Many optimizations are possible when some constants (typically the small integers and the powers of two) are exactly representable as only one floating-point number.
- Another optimization technique, due to Gal [8] and depicted Figure 2, is to use use DDE constants which actually hold more than 128 bits of accuracy. This is not transposable to multiple-precision.
- As the target precision is always less than 119 bits and the DDE format offers 128 bits, there are at least 9 bits (usually more) to lose to approximation and rounding errors, which helps to design fast algorithms.
- The whole computation belongs to the floating-point pipeline. In particular, the conversion of the DDE result back to a correctly rounded double (detailed in Section 3) is more efficient than if an integer-based multiple-precision package is involved.
- In a Ziv-like two-step approach, the first step will be done in double-extended precision, and the second step will be computed in DDE. If both steps use the same arithmetic, they could also share some intermediate results. For instance, tabulated values may be stored as the sum of two double-extended, with the first step loading only the first term. This would reduce the memory footprint of the overall evaluation, with the additional benefit that the data for the second step would already be in the cache.

  This last point is the most speculative of all, as an algorithm exploiting this idea still has to be written.

Note that in a two-step Ziv approach, current libraries balance the very slow second step with the very low probability of calling it (typically in the order of 10^{-3} to 10^{-5}). By improving the performance of the second step, we also bring the average performance much closer to that of the first step, which is as fast as a standard `libm` evaluation.

This last point may be elaborated further: It is likely that with a good DDE-based implementation, the overhead of managing two Ziv levels will not be worth the benefit. In particular, the performance benefits cited above have to be put in the perspective of a deep memory architecture. As an illustration, we describe in Section 4 a prototype implementation of the correctly rounded exponential on Itanium which has a constant execution time of about twice that of the (heavily optimized, but not correctly rounded) standard implementation, and even less if the function call overhead is taken into account. This experiment is very dependent on the best-of-class floating-point features of the Itanium processors (two parallel double-extended fused multiply-and-add, multiple floating-point status registers, etc), but it may be the direction to follow.

The remainder of this article details and proves the previous claims.

2 A new accuracy bound for correct rounding

2.1 An example: the exponential

Let us consider the following synthetic presentation of the results by Muller and Lefèvre concerning the exponential [11].

Theorem 2.1 *Let x be a double precision floating point value and $y = e^x$ the exact value of its exponential. Let y^* be an approximation to y so that the distance between the mantissas of y and y^* can be majored by ϵ. Then:*

- *For $|x| < 2^{-54}$, correctly rounding x consists in returning 1, $1 + 2^{-53}$, or $1 - 2^{-53}$, depending on the rounding mode and the sign of x;*
- *For $2^{-54} \leq |x| < 2^{-44}$, if $\epsilon < 2^{-53-104-1} = 2^{-158}$, for all rounding modes, rounding y^* to double precision is equivalent to rounding y;*
- *For $2^{-44} \leq |x| < 2^{-30}$, if $\epsilon < 2^{-53-84-1} = 2^{-138}$, for all rounding modes, rounding y^* is equivalent to rounding y;*
- *For $|x| \geq 2^{-30}$, if $\epsilon < 2^{-53-59-1} = 2^{-113}$, for all rounding modes, rounding y^* to double precision is equivalent to rounding y.*

We remark that the closer to zero, the higher the intermediate accuracy required. This allows the following observation:

- For $|x| < 2^{-44}$, where the most difficult cases are, we can compute e^x as $e^x \approx 1 + p(x)$, where p is a polynomial approximating $e^x - 1$ with $|p(x)| < 2^{-43}$ on this interval. Thanks to these 43 known zeroes, to ensure the 158 bis of accuracy required to fulfill Theorem 2.1, we need $p(x)$ to be accurate to $158 - 43 = 115$ bits.
- The intermediate case is similar: when $2^{-44} \leq |x| < 2^{-30}$, we evaluate $e^x \approx 1 + p(x)$, with $|p(x)| < 2^{-29}$ on this interval. To ensure the critical accuracy of 2^{-138} we only need $p(x)$ to be accurate to $138 - 29 = 109$ bits.
- In the other cases, we need an approximation to the exponential which is exact to 113 bits.

We see that we have reduced the accuracy required in all the worst cases to less than 115 bits.

This result can be generalized to all the common elementary functions: There is a correlation between the presence of harder-than-119-bit cases, and the possibility of evaluating the function either as $1 + p(x)$ or as $x + p(x)$ on some intervals.

2.2 Correct rounding never requires more than 119 bits

The main result of this paper is the following theorem:

Theorem 2.2 *For all the elementary functions f studied by Lefèvre and Muller, the interval on which the worst cases have been computed can always be decomposed into two sets I_e and I_h such that*

- *on I_e, evaluating the function with 119 bits of accuracy is enough to ensure correct rounding,*
- *on I_h, the function requires a worst case accuracy larger than 119 bits, but then it can be written $f(x) = 1 + p(x)$ or $f(x) = x + p(x)$, and the evaluation of $p(x)$ with 119 bits of relative accuracy provides enough accuracy on f to ensure correct rounding of $f(x)$.*

Proof The proof consists simply in rewriting the results published by Lefèvre and Muller [11]. Tables 1 to 3 summarize the relevant results. All the special cases carefully studied in [11] are omitted here. As the worst cases for some functions have only been computed on a sub-interval of their definition domain, these tables also give the limits of Theorem 2.2. In these tables, the bold number is the accuracy required for the implementation. Of course, from an implementation point of view, this per-function information is more useful than the synthetic maximum of 119 stated in the above theorem. In these tables, $\mathbb{F}$ denotes the set of double-precision floating-point numbers.

Function	Interval	critical accuracy on f
$y = \log(x)$	$x \in \mathbb{F}$	**118** bits
$y = 2^x$	$x \in \mathbb{F}$	**113** bits
$y = \log_2(x)$	$x \in \mathbb{F}$	**109** bits
$y = \arccos(x)$	$\cos(\frac{12867}{8192}) \leq \lvert x\rvert \leq 1$	**116** bits
$y = \cosh(x)$	$2^{-6} \leq \lvert x\rvert \leq 2^5$	**111** bits
$y = \operatorname{arccosh}(x)$	$\cosh(2^{-6}) \leq \lvert x\rvert \leq \cosh(2^5)$	**115** bits

Table 1: Critical accuracy for functions not concerned with this article.

Function	Interval	critical accuracy on f	critical accuracy on p
$y = e^x$	$2^{-54} \leq \lvert x\rvert < 2^{-44}$	158 bits	**115** bits
	$2^{-44} \leq \lvert x\rvert < 2^{-30}$	138 bits	**109** bits
	$\lvert x\rvert \geq 2^{-30}$	**113** bits	
$y = \cos(x)$	$2^{-25} \leq \lvert x\rvert \leq 2^{-22}$	142 bits	**98** bits
	$2^{-22} \leq \lvert x\rvert \leq 2^{-18}$	136 bits	**100** bits
	$2^{-18} \leq \lvert x\rvert \leq 2^{-17}$	114 bits	**80** bits
	$2^{-17} \leq \lvert x\rvert \leq \frac{12867}{8192}$	**112** bits	

Table 2: Critical accuracy for functions of the type $f(x) = 1 + p(x)$.

Function	Interval	critical accuracy on f	critical accuracy on p
$y = \sin(x)$	$\lvert x\rvert < 2^{-17}$	126 bits	**110** bits
	$2^{-17} \leq \lvert x\rvert \leq 2 + \frac{4675}{8192}$	**119** bits	
$y = \arcsin(x)$	$\sin(2^{-24}) \leq \lvert x\rvert \leq \sin(2^{-19})$	126 bits	**107** bits
	$\sin(2^{-19}) \leq \lvert x\rvert \leq \sin(2^{-18})$	120 bits	**102** bits
	$\sin(2^{-18}) \leq \lvert x\rvert \leq 1$	**118** bits	
$y = \tan(x)$	$2^{-25} \leq \lvert x\rvert \leq 2^{-18}$	132 bits	**115** bits
	$2^{-18} \leq \lvert x\rvert \leq \arctan(2)$	**111** bits	
$y = \arctan(x)$	$\tan(2^{-25}) \leq \lvert x\rvert \leq \tan(2^{-18})$	126 bits	**109** bits
	$\tan(2^{-18}) \leq \lvert x\rvert \leq 2$	**113** bits	
$y = \sinh(x)$	$2^{-25} \leq \lvert x\rvert \leq 2^{-18}$	126 bits	**109** bits
	$2^{-18} \leq \lvert x\rvert \leq 2^{-12}$	114 bits	**103** bits
	$2^{-12} \leq \lvert x\rvert$	**109** bits	
$y = \operatorname{arcsinh}(x)$	$\sinh(2^{-25}) \leq \lvert x\rvert \leq \sinh(2^{-12})$	126 bits	**114** bits
	$\sinh(2^{-12}) \leq \lvert x\rvert$	**118** bits	

Table 3: Critical accuracy for functions of the type $f(x) = x + p(x)$.

There is a disclaimer in this theorem, as Muller and Lefèvre haven't yet completed the study of the worst cases for all the functions on the whole of their definition interval. The fundamental result, however, is the following: *there is no bad surprise in the critical accuracy required for correctly rounding a function.*

In other terms, where classical probabilistic arguments [8, 15] tell us that the expected worst cases should require around $53 + 64 = 117$ bits of accuracy, functions with actual worst cases which exceed these bounds (up to 158 bits so far) can be reduced easily to functions which don't. This result can be explained easily, although this explanation does not constitute a proof: The probabilistic argument assumes a uniform distribution of the distance of the exact value of $f(x)$ to the nearest floating-point number. This asumption, mathematically unfounded in all cases, becomes obviously wrong on I_h. To get something that resembles a uniform distribution, one has to consider $p(x)$ instead of $f(x)$.

Note that this theorem does not show how to recover the correctly rounded value of $f(x)$ from $p(x)$. Section 3 will deal with this question in the case when double-extended arithmetic is available.

2.3 Implementation considerations

We have already stated that the information relevant to the implementation of a function is the bold numbers corresponding to this function in Tables 1 to 3, not the worst case 119 used in the theorem. This remark can be refined.

Consider again the exponential. The same polynomial $p(x)$ (a variation on the Taylor approximation [15]) will usually be used for all the small arguments. This is the case in the exponential described in Section 4. In this case, the two first lines of Table 2 are only relevant to the proof of accuracy, not to an implementation. Conversely, if 115 bits is too difficult an accuracy target, the smaller interval can be broken down into more intervals for the purpose of the proof. The lower bound on the expected critical accuracy on $p(x)$ is $53 + 54 = 107$ bits on a binade, however. This remark also applies to other functions.

The last implementation concern is to be able to recover the value of $f(x)$ correctly rounded to double precision from the approximation of $p(x)$. This is the subject of the next section

3 Correct rounding using double-double-extended arithmetic

This section gives two theorems showing how to recover the correctly rounded value of $1 + p(x)$ and $x + p(x)$ as if the full worst-case accuracy was available, using only double-double precision arithmetic.

This section assumes that the system used provides double-extended arithmetic with IEEE-754-compliant rounding (i.e. rounding the infinitely precise result) for the addition. This is the case of the IA32 processor families by Intel and AMD, as well as the HP/Intel IA-64 (Itanium) family. On these systems, the mantissa consists of 64 bits.

In the following $\overset{64}{\underset{rn}{\oplus}}$ and $\overset{64}{\underset{rn}{\ominus}}$ denote the IEEE-compliant machine addition and subtraction in double-extended, round to nearest mode; $\overset{53}{\underset{rx}{\oplus}}$ denotes the machine operation that adds two double-extended numbers and rounds the exact result to an IEEE double using the user-selected rounding mode. These operations are machine operations on architectures which support double-extended precision. Similarly, $\overset{53}{\underset{rx}{\circ}}$ is the corresponding rounding function.

We also use the well-known Fast2Sum algorithm with double-extended numbers, with the condition improved by Boldo [2]:

Theorem 3.1 *Let a and b two double-extended floating-point numbers such that the exponent of b is smaller than or equal to the exponent of a.*

Then the following instruction sequence (called Fast2Sum algorithm):

$$s_h \leftarrow a \overset{64}{\underset{rn}{\oplus}} b$$
$$s_l \leftarrow b \overset{64}{\underset{rn}{\ominus}} (s_h \overset{64}{\underset{rn}{\ominus}} a)$$

computes two double-extended floating-point numbers s_h and s_l such that $s_h + s_l = a + b$ exactly, and $s_h = a \overset{64}{\underset{rn}{\oplus}} b$.

Note that all the Fast2Sum algorithm in this article works on double-extended numbers.

The cost of a Fast2Sum is 3 dependent floating-point additions. There also exists a Fast2Mult algorithm which computes the exact product of two floating-point values in 10 additions and 7 multiplications [7], or in only two fused multiply-and-add operations [3].

3.1 Final rounding in the case $1 + p(x)$

Theorem 3.2 *Let p_h and p_l be two double-extended numbers such that there exists an integer k and a real ϵ such that:*

- $p_h + p_l = p(x)(1 + \epsilon)$
- $p_h = p_h \overset{64}{\underset{rn}{\oplus}} p_l$
- $|p_h| < 2^{-k}$
- $2 < k < 52$
- *ϵ is smaller than 2^{-m}, where $m \leq 119$ is the worst-case accuracy given in bold in Table 2.*

Then for any rounding mode rx, the instruction sequence:

$$t_h + t_l \leftarrow Fast2Sum(1, p_h)$$
$$r \leftarrow t_l \overset{64}{\underset{rn}{\oplus}} p_l$$
$$z \leftarrow t_h \overset{53}{\underset{rx}{\oplus}} r$$

computes in z the value $1 + p(x)$ correctly rounded to double precision in the rounding mode rx.

This instruction sequence represents a total of 5 dependent floating-point additions.

The assumption $p_h = p_h \overset{64}{\underset{rn}{\oplus}} p_l$ is not restrictive at all, in particular it is satisfied if $p_h + p_l$ are the result of a Fast2Sum or Fast2Mult algorithm, as is the case in a polynomial evaluation in Horner form. It is also easy to satisfy if $p_h + p_l$ are tabulated values.

Proof We know from Table 2 that $\overset{53}{\underset{rx}{\circ}}(1 + p(x)) = \overset{53}{\underset{rx}{\circ}}(1 + p_h + p_l)$. We therefore need to prove that $z = \overset{53}{\underset{rx}{\circ}}(1 + p_h + p_l)$

Figures 3 and 4 give an overview of the mantissas involved in instruction sequence above.

On these figures, the meaning of the vertical lines is the following: A number strictly smaller in magnitude than 2^{-n} has all the bits of its mantissa to the right of the vertical line at 2^{-n}. In other words the bit of magnitude 2^{-n} is the bit immediately to the left of the line.

The Fast2Sum ensures that $t_h = 1 \overset{64}{\underset{rn}{\oplus}} p_h$ and $t_h + t_l = 1 + p_h$, therefore we have $1 + p_h + p_l = t_h + t_l + p_l$.

Let k be the largest integer such that $|p_h| < 2^{-k}$. This means that the mantissa of p_h and 1 are indeed aligned as on the figures.

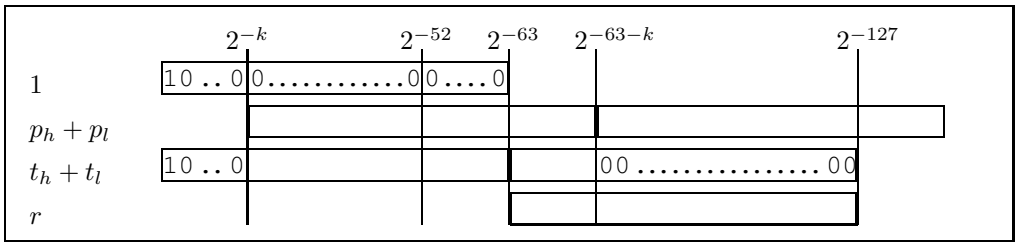


Figure 3: Case $p_h \geq 0$

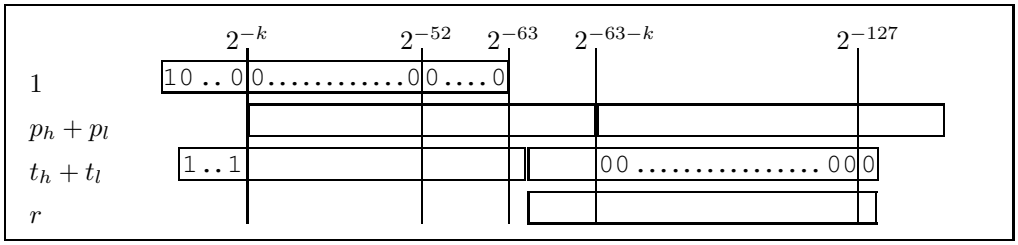


Figure 4: Case $p_h < 0$

Lemma 3.3 $1/2 < |t_h| < 2$ *(or, the mantissa of* t_h *and* 1 *are indeed aligned as on the figures).*

Proof From $|p_h| < 2^{-k} < 2^{-2}$.

Lemma 3.4 $|t_l| < 2^{-63}$.

Proof Using Lemma 3.3, this is ensured by the Fast2Sum producing $t_h + t_l$.

Lemma 3.5 *All the bits of* t_l *of magnitude strictly smaller than* 2^{-63-k} *are zeroes (this also appears on the figure).*

Proof The Fast2Sum producing $t_h + t_l$ ensures that $t_h + t_l = 1 + p_h$ exactly. Therefore all the bits of t_l of magnitude smaller than that of the last bit of p_h are zeroes. As k is the largest integer such that $p_h < 2^{-k}$ the last bit of the mantissa of p_h has magnitude 2^{-63-k}.

Lemma 3.6 $|p_l| < 2^{-63-k}$ *(on the figure,* p_l *is always on the right of the vertical line* 2^{-63-k}*).*

Proof From $p_h = p_h \overset{64}{\underset{rn}{\oplus}} p_l$ and $p_h < 2^{-k}$.

Lemma 3.7 *The sum* $r \leftarrow t_l \overset{64}{\underset{rn}{\oplus}} p_l$ *involves no carry propagation.*

Proof Lemmata 3.5 and 3.6 show that there is no overlap of non-zero bits from p_l and t_l.

Lemma 3.8 $|r| < 2^{-63}$.

Proof This is a direct consequence of lemmata 3.4 and 3.7.

Lemma 3.9 $t_h \overset{53}{\underset{rx}{\oplus}} r$ *is either* $\overset{53}{\underset{rx}{\circ}}(t_h)$*, or its successor, or its predecessor in the set of double-precision floating-point numbers.*

Proof This is a direct consequence of lemmata 3.3 and 3.8.

Lemma 3.9 tells us that the information carried by r is purely qualitative: The IEEE-compliant hardware needs only to know if r is strictly positive, null, or strictly negative, to decide which of the three possible results to return in z, depending also on the rounding mode rx. In the following we denote by *sign* this information.

Now the operation $r \leftarrow t_l \overset{64}{\underset{rn}{\oplus}} p_l$ may involve a rounding error, so some of the accuracy of $1 + p_h + p_l$ has been lost in the arguments t_h and r of the final rounding addition $t_h \overset{53}{\underset{rx}{\oplus}} r$. We now need to prove that this doesn't prevent correct rounding. What we will prove is that the sign of r (whether it is null, strictly positive, or strictly negative) is always equal to that of the exact sum $t_l + p_l$.

Let us define

$$\delta \;=\; (1 + p_h + p_l) - t_h \;=\; t_l + p_l$$

and

$$\epsilon_r = (t_l + p_l) - r \quad .$$

Lemma 3.10 $\overset{53}{\underset{rx}{\circ}}(1 + p_h + p_l) \;=\; \overset{53}{\underset{rx}{\circ}}(t_h + \delta)$.

Proof By definition of δ.

Now what we compute is not $\overset{53}{\underset{rx}{\circ}}(t_h + \delta)$ but $z \;=\; t_h \overset{53}{\underset{rx}{\oplus}} r \;=\; \overset{53}{\underset{rx}{\circ}}(t_h + r)$ in IEEE-compliant rounding. We therefore now prove that $\overset{53}{\underset{rx}{\circ}}(t_h + \delta) \;=\; \overset{53}{\underset{rx}{\circ}}(t_h + r)$. For this it is enough to prove that $|\delta| < 2^{-63}$ just as r, and that the sign of r is always the sign of δ

Lemma 3.11 $\delta = r + \epsilon_r$.

Proof $r = t_l \overset{64}{\underset{rn}{\oplus}} p_l \;=\; t_l + p_l - \epsilon_r = \;\delta - \epsilon_r$.

Lemma 3.12 $|\delta| < 2^{-63}$.

Proof This is again a consequence of Lemmata 3.5 and 3.6 which show that there is no overlap of non-zero bits from p_l and t_l, and from Lemma 3.4 ($|t_l| < 2^{-63}$).

Lemma 3.13 $|\epsilon_r| < 2^{-127}$.

Proof From $\epsilon_r = (t_l + p_l) - (t_l \overset{64}{\underset{rn}{\oplus}} p_l)$, and $|t_l| < 2^{-63}$ (Lemma 3.4) and $|p_l| < 2^{-63-k}$ (Lemma 3.6).

Lemma 3.14 *If* $|\delta| \geq 2^{-127}$ *then* $\overset{53}{\underset{rx}{\circ}}(t_h + \delta) \;=\; \overset{53}{\underset{rx}{\circ}}(t_h + r)$

Proof As $|\epsilon_r| < 2^{-127}$ (Lemma 3.13), if $|\delta| \geq 2^{-127}$ then the sign of r is the sign of δ. Lemmata 3.3, 3.8 and 3.12 ensure that only this information is significant in both rounding operations.

Lemma 3.15 *If* $|\delta| < 2^{-127}$ *then* $\overset{53}{\underset{rx}{\circ}}(t_h + \delta) \;=\; \overset{53}{\underset{rx}{\circ}}(t_h + r)$

Proof From $|\delta| < 2^{-127}$, $|\epsilon_r| < 2^{-127}$ (Lemma 3.13) and $r = \delta - \epsilon_r$ (Lemma 3.11) we deduce that $|r| < 2^{-126}$. It implies that $t_l = 0$, as all its non-zero bits are of magnitude greater than 2^{-63-k} with $k < 52$ (Lemma 3.5). This in turn implies that $r \leftarrow t_l \overset{64}{\underset{rn}{\oplus}} p_l$ was exact, or $\epsilon_r = 0$, therefore $\delta = r$.

Lemmata 3.15 and 3.14 show that in all cases, $t_h \overset{53}{\underset{rx}{\oplus}} r$ is the correct rounding of $1 + p_h + p_l$, and therefore of $1 + p(x)$. This concludes the proof of the theorem.

3.2 Final rounding in the case $x + p(x)$

Theorem 3.16 *Let x be a double-precision number cast into a double-extended, and let p_h and p_l be two double-extended numbers such that there exists an integer k and a real ϵ such that:*

- $p_h + p_l = p(x)(1 + \epsilon)$
- $p_h = p_h \overset{64}{\underset{rn}{\oplus}} p_l$
- $|p_h| < 2^{-k}x$
- $2 < k < 52$
- *ϵ is smaller than 2^{-m}, where $m \leq 119$ is the worst-case accuracy given in bold in Table 3.*

Then for any rounding mode rx the instruction sequence:

$$t_h + t_l \leftarrow Fast2Sum(x, p_h)$$
$$r \leftarrow t_l \overset{64}{\underset{rn}{\oplus}} p_l$$
$$z \leftarrow t_h \overset{53}{\underset{rx}{\oplus}} r$$

computes in z the value $x + p(x)$ correctly rounded to double precision in the rounding mode rx, provided no underflow occurs in the computation of t_h.

Note that the underflow disclaimer is not a problem in the context of implementing elementary functions : All the functions of the type $x + p(x)$ can be written with a Taylor series $x + cx^2 + o(x^2)$, and the cases when x is so small that underflow would occur are handled separately by returning x or its successor or predecessor, depending on the rounding mode. These cases are filtered out by an "if $x < m_x$" with m_x typically in the order of 2^{-53} (it depends on c). Therefore the theorem assumes $x \geq m_x$, which combined with the worst-case accuracy bound ensures that no underflow occurs for any of the floating-point involved (here an exact value of 0 is not considered an underflow).

Proof The proof is very similar to that of Theorem 3.2 with all absolute magnitudes replaced with relatives magnitude. The main difference is that if e_x is the exponent of x, then the exponent of $t_h = x \overset{64}{\underset{rn}{\oplus}} p(x)$ may be e_x or $e_x + 1$ or $e_x - 1$. This third case is depicted on Figure 5. However, the value of x has disappeared after the first Fast2Sum. If the relative magnitudes are relative to t_h and not to x, even the various constants in the proof are unchanged.

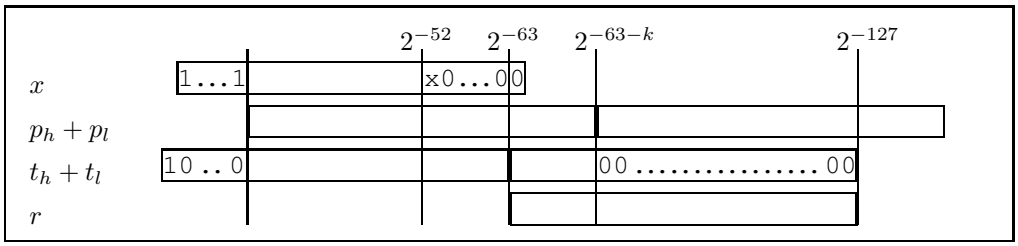


Figure 5: Case when the exponent of t_h is 1 plus that of x. Here the magnitudes are relative to the exponent of t_h.

4 Application: A correctly rounded exponential on the Itanium

In this section, we describe an implementation of a correctly rounded exponential on the Intel Itanium, without any claim that it is the best possible. We give an overview of the algorithm, which relies heavily on classical double-double arithmetic [7, 2]. The details of implementations and detailed proofs of accuracy are skipped: They have been published previously as a separate research report [10]. Note however that the proofs of the three theorems in the present article extend and correct the proof of the correct rounding property in this previous report.

4.1 Overview of the algorithm

The exponential is evaluated using the following equation (inspired by an algorithm in [16]).

$$e^x \approx 2^M \cdot 2^{\frac{i_1}{2^7}} \cdot 2^{\frac{i_2}{2^{14}}} \cdot (1 + p(x_h)) \cdot (1 + x_l) \tag{1}$$

The terms of this equation are detailed below.

4.2 Range reduction

The argument $x \in [-745.13; 709.78]$ (the range in which e^x is representable in double-precision and not rounded to zero) is decomposed as the sum of three terms:

$$x \approx k \cdot \frac{\ln 2}{2^{14}} + x_h + x_l$$

where x_h and x_l are two double-extended numbers such that $|x_h| < \frac{\log 2}{2^{15}}$, $|x_l| < 2^{-64}|x_h|$, and

$$k = \left\lfloor x \cdot \frac{2^{14}}{\ln 2} \right\rceil$$

The integer k is itself decomposed into three terms:

$$k = 2^{14} \cdot M + 2^7 \cdot i_1 + i_2 \tag{2}$$

where $i_1, i_2 \in \mathbb{N} \cap \left[0; 2^7 - 1\right]$ and $M \in \mathbb{N}$.

4.3 Tabulated values

The terms $2^{\frac{i_1}{2^7}}$ and $2^{\frac{i_2}{2^{14}}}$ are precomputed and stored in two 128-entry tables. Each entry consists of two double-extended numbers providing 128 bits of precision: $t_{h1} + t_{l1} \approx 2^{\frac{i_1}{2^7}}$ and $t_{h2} + t_{l2} \approx 2^{\frac{i_2}{2^{14}}}$.

We use here the lucky property that the scale constant $\frac{\ln 2}{2^{14}}$ can be stored on two double-extended numbers (128 bits) with an absolute error less than 2^{-150} instead of the expected $2^{-129-14} = 2^{143}$: This is depicted Figure 2.

The choice of a range reduction using a scale constant of $\frac{\ln 2}{2^{14}}$ is a compromise between several factors:

- The size of the two tables which can be optimized when an even scale constant exponent, 14, is used;
- The degree of the polynomial, which depends on the range of the final reduced argument x_h;
- The range of the lower part of the reduced argument x_l.

4.4 Polynomial evaluation

The polynomial $p(x_h)$ approximates the exponential of the reduced argument as a degree 6 modified Remez polynomial [15]. It is evaluated as

$$p(x_h) = x_h + \frac{1}{2}x_h^2 + (a_{h4} + a_{l4})\, x_h^3 + x_h^4 \cdot (a_5 + x_h \cdot (a_6 + a_7 x_h))$$

The values for a_{h4}, a_{l4}, a_5, a_6 and a_7 are near to respectively $\frac{1}{6}$, $\frac{1}{24}$, $\frac{1}{120}$ and $\frac{1}{720}$. The reason for not using Horner's scheme all along is its sequentiality: Splitting up the evaluation as shown allows better use of the pipelines on the Intel Itanium .

The approximation to $p(x_h)$ is computed as the sum of two double extended precision numbers: $p(x_h) \approx p_h + p_l$. This is the most time consuming phase of the algorithm. The final operation of this evaluation is a Fast2Sum, which ensures that p_h is $p_h + p_l$ correctly rounded to the nearest.

4.5 Reconstruction

The reconstruction consists in computing the product of all the terms of Eq. 1. It relies heavily on the Fast2Sum and Fast2Mult algorithms, the latter being very efficient thanks to the fused multiply-and-add operators of the Itanium architecture[3].

4.6 Correct rounding

The overall error of the whole algorithm is shown to be less than 2^{-116} in [10] (the proof is about 30 pages long). Looking back at Table 2, we conclude that correct rounding is ensured for any $|x| > 2^{-30}$.

For $|x| < 2^{-30}$, it is enough to remark that:

- In the range reduction, for $|x| < 2^{-30}$ we will have $k = 0$, $x_h = x$ and $x_l = 0$, and this step is exact.
- Concerning the tabulated values we have $i_1 = i_2 = 0$, therefore $2^{\frac{i_1}{2^7}} = 1$ and $2^{\frac{i_2}{2^7}} = 1$, which can be stored exactly as a 1 and a 0 (no approximation error here).
- $p_h + p_l$ is an approximation to $e^x - 1$ accurate to 2^{-121} in all cases (a 6-bit overkill according to Table 2, and probably more if we consider the remarks made in Section 2.3).
- The reconstruction is exact thanks to the tabulated 1: It reduces to multiplications by 1 and 0 and additions of 0; Besides it performs the sequence of operations given in Theorem 3.2 (well disguised as 6 fused multiply-and-add and 6 additions).

This is enough to prove that correct rounding is also achieved for $|x| < 2^{30}$.

Retrospectively, a more sensible approach would have been to have an "`if` $|x| < 2^{-30}$" in the code, which would accelerate the latter case by skipping the range reduction and reconstruction phase altogether (in our case it may actually be an "`if` $|x| < 2^{-15}$"). It would also shorten the proof since we wouldn't have to show that the reconstruction is exact in this case (another 4 pages in [10], including a few serious errors). However, it is precisely by studying why this implementation was always returning a correctly rounded result that we came to the more general results which are the subject of the present article.

Finally, note that in a Ziv-like two-level evaluation, it might make sense, for the average performance, if this "`if`" belongs to the first step and jumps directly to an accurate enough polynomial evaluation and reconstruction. However such considerations are very dependent on the cost of breaking the control-flow.

4.7 Performance

This implementation was tested against a battery of tools for accuracy and IEEE-compliance, including the exceptional cases.

Speedwise, the implementation performs in 92 cycles.

- This can be compared to the 42 cycles of the highly optimized Intel implementation, which is accurate to 0.502 ulps (*units in the last place*) only, or a relative accuracy of 2^{-61}. The reader should put these numbers in perspective with the cost of a function call on the Itanium, which adds 30 cycles to the above numbers, or with the handling of some floating-point exceptional cases through software exceptions, which can be several hundreds of cycles.

- It can also be compared to the best previous correctly rounded exponential by Defour [6], which has a 4721 cycles worst case time on the Itanium. The implementation of IBM's Ultimate MathLib has an even slower worst case, since it doesn't benefit from the work of Lefèvre and Muller and uses less efficient multiple-precision techniques. We have here an improvement on the worst case time by a factor 50 over the previous best.

Another factor of performance is memory consumption: The size of our table is $2 \cdot 128 \cdot 2 \cdot 16 = 8192$ bytes which is a relatively great value [16, 6] for an exponential. For example, the reference implementation by Intel requires only 264 bytes of table. On the other hand, the table could fit 12 times in the L2 cache of the Itanium processor which can be accessed in 6 cycles [9] (this processor cannot load floating point registers from L1 cache). Furthermore, this size is less than the sizes of the tables used by the other correctly rounding implementations.

As a final note concerning this implementation, consider that it has been written before the results presented here were known, and could probably be improved further with this knowledge. In particular the polynomial evaluation seems more accurate than strictly needed. The table size might also be reduced.

5 Conclusion

This article shows that, provided double-extended IEEE-compliant hardware is available, correct rounding of the elementary functions should no longer be prevented by performance considerations: it still has some overhead, but so has IEEE-754 compliance for the basic operators. The IEEE-754 standard is now universally accepted because its benefits far exceed this performance overhead. We believe that the result presented here bring elementary functions to this critical point, with an average-case overhead which becomes negligible, and a worst-case overhead in a factor smaller than 10. To support this quantified claims, of course, we need to implement more functions: the exponential is notably the elementary functions which is easiest to implement (although it is the function with the worst case of all, as far as correct rounding is concerned).

Note that, although the performance overhead of correct rounding becomes very low, the *implementation* overhead is huge: One needs to prove the accuracy of an implementation in the last details, which is very tedious and, of course, error prone. Besides, almost any improvement to the code will ruin most of the proof, making the maintenance of such code even more tedious. This may be the reason why, to our knowledge, no published proof exists for IBM's ultimate MathLib. One important contribution of this article is therefore the proof of the correctly rounding reconstruction, which can be exploited for many functions – and other precisions, if needed in the future. One of us is currently working on tools for the automation of proofs such as those written by hand for the two correctly-rounded exponentials mentionned in this article [4, 10].

Including the elementary functions in the revision of the IEEE standard has been proposed in [5]. Their study raises several points, one of which is relevant to the present paper, and more precisely to the "disclaimer" in our Theorem 2.2: The worst cases of required accuracy are not known for all the functions, on all their definition intervals. This is still work in progress, and for

some functions (most notably the periodic ones for the large arguments) an algorithmic breakthrough will be needed for this quest to be completed. Fortunately, these intervals are those on which evaluating the function makes little mathematical sense anyway: Few programs need the sine of a floating-point number so large that its floating-point successor is several times π away. The proposal suggests to standardise on several levels of accuracy, the lowest being the current situation, and the highest (guaranteed correct rounding everywhere) being currently out of reach for the trigonometric functions (but achieved by our exponential, for instance).

However, stated as *"There is no bad surprise in the required worst-case accuracy"*, our main result will hold: The results given in Tables 1 to 3 are here to stay, and even if Lefèvre and Muller do indeed find a bad surprise in the future, it can always be filtered out by an implementation.

Acknowledgements

The authors would like to Andrey Naraikin and his group at Intel Corporation for welcoming Christoph Lauter in Nizhny-Novgorod. Thanks also go to Sylvie Boldo, Peter Markstein, Guillaume Melquiond, and Jean-Michel Muller for many helpful discussions. The support of Hewlett-Packard through the donation of an Itanium-based system is also gratefully acknowledged.

References

[1] ANSI/IEEE. Standard 754-1985 for binary floating-point arithmetic, 1985.

[2] S. Boldo and M. Daumas. A mechanically validated technique for extending the available precision. In *35th Asilomar Conference on Signals, Systems, and Computers*, Pacific Grove, California, 2001. IEEE Computer Society Press.

[3] M. Cornea, J. Harrison, and P.T.P Tang. *Scientific Computing on Itanium-based Systems*. Intel Press, 2002.

[4] C. Daramy, D. Defour, F. de Dinechin, and J.M. Muller. CR-LIBM: The evaluation of the exponential. Technical Report RR2003-37, LIP, École Normale Supérieure de Lyon, July 2003. Available at ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR2003/RR2003-37.ps.gz.

[5] D. Defour, G. Hanrot, V. Lefèvre, J.-M. Muller, N. Revol, and P. Zimmermann. Proposal for a standardization of mathematical functions. *Numerical algorihms*, 2004.

[6] David Defour. *Fonctions élémentaires: algorithmes et implémentations efficaces pour l'arrondi correct en double précision*. PhD thesis, École Normale Supérieure de Lyon, Lyon, France, September 2003.

[7] Theodorus J. Dekker. A floating point technique for extending the available precision. *Numerische Mathematik*, 18(3):224–242, 1971.

[8] S. Gal. Computing elementary functions: A new approach for achieving high accuracy and good performance. In *Accurate Scientific Computations, LNCS 235*, pages 1–16. Springer Verlag, 1986.

[9] Intel Corporation. *Intel Itanium Architecture, Software Developer's Manual, Volume 3: Instruction Set Reference*, dec 2001. Intel Document Number: 245319-003.

[10] C. Lauter. A correctly rounded implementation of the exponential function on the Intel Itanium architecture. Technical Report 2003-54, LIP, École Normale Supérieure de Lyon, November 2003. Available at ftp://ftp.ens-lyon.fr/pub/LIP/Rapports/RR/RR2003/RR2003-54.ps.gz.

[11] V. Lefèvre and J.-M Muller. Worst cases for correct rounding of the elementary functions in double precision. http://perso.ens-lyon.fr/jean-michel.muller/Intro-to-TMD.htm, 2004.

[12] V. Lefèvre, J.-M. Muller, and A. Tisserand. Towards correctly rounded transcendentals. *IEEE Transactions on Computers*, 47(11):1235–1243, November 1998.

[13] IBM Accurate Portable Math. Library. `http://oss.software.ibm.com/mathlib/`.

[14] P. Markstein. *IA-64 and Elementary Functions : Speed and Precision*. Hewlett-Packard Professional Books. Prentice Hall, 2000. ISBN: 0130183482.

[15] J.-M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser, Boston, 1997.

[16] P. T. P. Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 15(2):144–157, June 1989.

[17] A. Ziv. Fast evaluation of elementary mathematical functions with correctly rounded last bit. *ACM Transactions on Mathematical Software*, 17(3):410–423, September 1991.