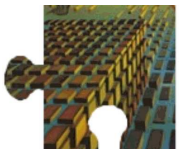

DALI
Digits, Architectures et Logiciels Informatiques

Équipe de Recherche DALI

Laboratoire LP2A, EA 3679
Université de Perpignan Via Domitia

Improving the Compensated Horner Scheme with a Fused Multiply and Add

S. Graillat, Ph. Langlois, N. Louvet
DALI-LP2A Laboratory.
Université de Perpignan Via Domitia.
stef.graillat@univ-perp.fr
philippe.langlois@univ-perp.fr
nicolas.louvet@univ-perp.fr

9 novembre 2005

Research Report N° RR2005-06

Université de Perpignan Via Domitia
52 avenue Paul Alduy, 66860 Perpignan cedex, France
Téléphone : +33(0)4.68.66.20.64
Télécopieur : +33(0)4.68.66.22.87
Adresse électronique : dali@univ-perp.fr

UPVD
Université de Perpignan Via Domitia

Improving the Compensated Horner Scheme with a Fused Multiply and Add

S. Graillat, Ph. Langlois, N. Louvet
DALI-LP2A Laboratory.
Université de Perpignan Via Domitia.
stef.graillat@univ-perp.fr
philippe.langlois@univ-perp.fr
nicolas.louvet@univ-perp.fr

9 novembre 2005

Abstract

Several different techniques and softwares intend to improve the accuracy of results computed in a fixed finite precision. Here we focus on a method to improve the accuracy of the polynomial evaluation. It is well known that the use of the Fused Multiply and Add operation available on some microprocessors like Intel Itanium improves slightly the accuracy of the Horner scheme. In this paper, we compare two accurate compensated Horner schemes specially designed to take advantage of the Fused Multiply and Add. These improvements are similar to the approach applied to the summation and the dot product by Ogita, Rump and Oishi. We also use a recent algorithm by Boldo and Muller that computes the exact result of a Fused Multiply and Add operation as the unevaluated sum of three floating point numbers. Such an Error-Free Transformation is an interesting tool to introduce more accuracy efficiently. We prove that the computed results are as accurate as if computed in twice the working precision. The algorithms we present are fast since they only require well optimizable floating point operations, performed in the same working precision as the given data.

Keywords: IEEE-754 floating point arithmetic, error-free transformations, extended precision, polynomial evaluation, Horner Scheme, Fused Multiply and Add

Résumé

Différentes techniques logicielles ont pour but d'améliorer la précision d'un résultat calculé en précision finie. Nous nous intéressons icià une méthode pour améliorer la précision de l'évaluation polynomiale par la méthode de Horner. L'utilisation de l'instruction "Fused Multiply and Add" (FMA), disponible sur certains microprocesseurs comme l'Itanium d'Intel, améliore légérement la précision du résultat calculé par le schéma de Horner. Dans ce papier, nous comparons deux schémas de Horner précis, spécialement étudiés pour tirer parti du FMA. Ces améliorations sont très similaires aux approches proposées par Ogita, Oishi et Rump pour la sommation et le produit scalaire. Nous utilisons égalament un algorithme récent de Boldo et Muller qui calcule le résultat exacte d'un opération FMA sous la forme d'une somme non-évaluée de trois flottants. Une telle "Error-Free Transformation" (EFT) est un outil de choix pour introduire plus de précision efficacement. Nous montrons que les résultats caclculés ont une précision égale à celle qu'ils auraient s'ils avaient été calculés en utilisant une précision double de la précision courante. Nos algorithmes sont simples et n'utilisent que les opérations classiques, avec la même précision que celle des données.

Mots-clés: arithmétique flottante IEEE-754, précision étendue, évaluation polynomiale, schéma de Horner

Contents

1 Introduction

One of the three main processes associated with polynomials is evaluation, the two other ones being interpolation and root finding. The classic Horner scheme is the optimal algorithm with respect to algebraic complexity for evaluating a polynomial with given coefficients in the monomial basis. Higham [7, chap. 5] devotes an entire chapter to polynomials and more especially to polynomial evaluation.
The small backward error of the Horner scheme when evaluated in finite precision justifies its practical interest in floating point arithmetic for instance. It is well known that the computed evaluation of $p(x)$ is the exact value at x of a polynomial obtained by making relative perturbations of at most size $2n\,\mathbf{u}$ to the coefficients of p, where n denotes the polynomial degree and $\mathbf{u}$ the finite precision of the computation [7]. Nevertheless, the computed result can be arbitrary less accurate than the working precision $\mathbf{u}$ when evaluating $p(x)$ is ill-conditioned. This is the case for example in the neighborhood of multiple roots where all the digits or even the order of the computed value of $p(x)$ could be false. The classic condition number that describes the evaluation of $p(x) = \sum_{i=0}^{n} a_i x^i$, with complex coefficients,

$$\mathrm{cond}(p,x) = \frac{\sum_{i=0}^{n} |a_i||x|^i}{|\sum_{i=0}^{n} a_i x^i|} = \frac{\overline{p}(x)}{|p(x)|}. \tag{1}$$

When the computing precision is $\mathbf{u}$, evaluating $p(x)$ is ill-conditioned when $1 \ll \mathrm{cond}(p,x) \leq \mathbf{u}^{-1}$. If the coefficients of p are exact numbers in precision $\mathbf{u}$, we can also consider extremely ill-conditioned evaluations, *i.e.*, such that $\mathrm{cond}(p,x) > \mathbf{u}^{-1}$.
A possible way to improve the accuracy of the computed evaluation is to increase the working precision. For this purpose, numerous multiprecision libraries are available when the computing precision $\mathbf{u}$ is not sufficient to guarantee a prescribed accuracy [3, 1, 12]. Fixed-length expansions such as "double-double" or "quad-double" libraries [6] are actual and effective solutions to simulate twice or four times the IEEE-754 double precision [8]. For example a double-double number is an unevaluated sum of two IEEE-754 double precision numbers and its associated arithmetic provides at least 106 bits of significand. These fixed-length expansions are currently embedded in major developments such as for example within the new extended and mixed precision BLAS [10].
In [5] we have presented a fast and accurate algorithm for the polynomial evaluation. This compensated Horner scheme only requires an IEEE-754 like floating point arithmetic, and uses a single working precision with rounding to the nearest. We have proven that the computed result r is of the same accuracy as if computed in doubled working precision. This means that the accuracy of the computed result r satisfies

$$\frac{|r - p(x)|}{|p(x)|} \leq \mathbf{u} + (\alpha\,\mathbf{u})^2\,\mathrm{cond}(p,x), \tag{2}$$

with α a moderate constant. The second term in the right hand side of Relation (2) reflects computations in doubled working precision, and the first one the final rounding back to the working precision. This improvement of the Horner scheme is similar to the approach applied to the summation and the dot product algorithms in [15, 14]. The key tool to introduce more accuracy is what Ogita, Rump and Oishi call error-free transformations (EFT) in [14]: "it is for long known that the approximation error of a floating point operation is itself a floating point number". It means that for two floating point numbers a and b, and $\circ$ an arithmetic operator in $\{+,-,\times\}$, it exists a floating point number e, computable with floating point operations, such that

$$a \circ b = \mathrm{fl}\,(a \circ b) + e,$$

where $\mathrm{fl}\,(\cdot)$ denotes floating point computation. The EFT of the sum of two floating point numbers is computable using the well know algorithm 2Sum by Knuth [9]. 2Prod by Veltkamp and Dekker [4] is also available for the EFT of the product.
The Fused Multiply and Add instruction (FMA) is available on some current processors, such as the IBM Power PC or the Intel Itanium. Given a, b and c three floating point point values, this instruction computes the expression $a \times b + c$ with only one final rounding error [11]. This is particulary interesting in the context of polynomial evaluation, since it allows us to perform the Horner scheme faster and more accurately.
The FMA can be used to improve algorithms based on error-free transformations in two ways. First, it allows us to compute the EFT for the product of two foating point values in a very efficient way: algorithm 2ProdFMA presented hereafter computes this EFT in only two flops when a FMA is available [13, 11].

On the other hand, an algorithm that computes an EFT for the FMA has been recently proposed by Boldo and Muller [2]. In particular, they have proven that the EFT for the FMA is the sum of three floating point numbers. Assuming an IEEE-754 like floating point arithmetic with round to the nearest rounding mode, algorithm 3FMA computes three floating point numbers x, y and z such that

$$a \times b + c = x + y + z \quad \text{with} \quad x = \mathsf{FMA}(a, b, c).$$

Both 2ProdFMA and 3FMA can be used to improve the compensated Horner scheme presented in [5]. Each of these alternatives have to be considered to find the most efficient algorithm.
In this paper, we describe the two corresponding alternatives: CompHornerFMA (based on 2ProdFMA) and CompHorner3FMA (based on 3FMA). We prove that the proposed algorithms are as accurate as the classic Horner scheme performed in twice the working precision, *i.e.*, the accuracy of the computed results satisfies Relation (2). Experimental results show that time penalty due to these improvements of the precision are quite reasonable: our algorithms are not only fast in terms of floating point operations (flops) count, but also in terms of execution time.
The sequel of the paper is organized as follows. We present the classic assumptions about floating point arithmetic and our notations for error analysis in Section 2. In Section 3, we briefly review the algorithms for the EFT of the summation and the product of two floating point numbers, and we present the algorithm for the EFT of the FMA. We also introduce the two EFT we use for the Horner scheme. In Section 4, we describe our compensated algorithms for the polynomial evaluation, and we prove that the computed results are of the same accuracy as if computed in doubled working precision. Numerical experiments for extremely ill-conditioned evaluations are presented in Section 5 to exhibit the practical efficiency of our implementations, with respect to both accuracy and computing time.

2 Floating point arithmetic and Horner scheme

2.1 Standard model

The notations used throughout the paper are presented hereafter. Most of them come from [7, chap. 2]. As in the previous section, $\mathrm{fl}(\cdot)$ denotes the result of a floating point computation, where all the operations inside the parenthesis are performed in the working precision. We also introduce the symbols $\oplus$, $\ominus$, $\otimes$ and $\oslash$, representing respectively the floating point addition, subtraction, multiplication and division (*e.g.*, $a \oplus b = \mathrm{fl}(a + b)$). We adopt MATLAB like notations for our algorithms.
Throughout the paper, we assume a floating point arithmetic adhering to the IEEE-754 floating point standard [8]. As already said, we also assume that a FMA is available. We constraint all the computations to be performed in one working precision, with the round to the nearest rounding mode. We assume that no overflow nor underflow occur during the computations. $\mathbf{u}$ denotes the relative error unit, that is half the spacing between 1 and the next representable floating point value. For IEEE-754 double precision with round to the nearest, we have $\mathbf{u} = 2^{-53} \approx 1.11 \cdot 10^{-16}$.
When no underflow nor overflow occur, the following standard model describes the accuracy of every considered floating point computation. For two floating point numbers a and b and for $\circ$ in $\{+, -, \times, /\}$, the floating point evaluation $\mathrm{fl}(a \circ b)$ of $a \circ b$ is such that

$$\mathrm{fl}(a \circ b) = (a \circ b)(1 + \varepsilon_1) = (a \circ b)/(1 + \varepsilon_2), \text{with} \quad |\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}. \tag{3}$$

Given a, b and c three floating point values, the result of $\mathsf{FMA}(a, b, c)$ is the exact result $a \times b + c$ rounded to the nearest floating point value. Therefore, we also have

$$\mathsf{FMA}(a, b, c) = (a \times b + c)(1 + \varepsilon_1) = (a \times b + c)/(1 + \varepsilon_2), \text{with} \quad |\varepsilon_1|, |\varepsilon_2| \leq \mathbf{u}. \tag{4}$$

To keep track of the $(1 + \varepsilon)$ factors in our error analysis, we use the $(1 + \theta_k)$ and γ_k notations presented in [7, chap. 3]. For any positive integer k, θ_k denotes a quantity bounded according to

$$|\theta_k| \leq \gamma_k = \frac{k\,\mathbf{u}}{1 - k\,\mathbf{u}}.$$

When using these notations, we always implicitly assume $k\,\mathbf{u} < 1$. In the sequel of the paper, we will essentially use the following relations for our error analysis,

$$(1 + \theta_k)(1 + \theta_j) \leq (1 + \theta_{k+j}), \quad k\,\mathbf{u} \leq \gamma_k, \quad \gamma_k \leq \gamma_{k+1}.$$

2.2 The Horner scheme

The Horner scheme is the classic method to evaluate a polynomial $p(x) = \sum_{i=0}^{n} a_i x^i$ (Algorithm 1). For any floating point value x we denote $\mathsf{Horner}\,(p, x)$ the result of the floating point evaluation of the polynomial p at x using the Horner scheme.

Algorithm 1. Horner scheme

function $[r_0] = \mathsf{Horner}\,(p, x)$
$r_n = a_n$
for $i = n-1 : -1 : 0$
  $r_i = r_{i+1} \otimes x \oplus a_i$
end

A forward error bound for the result of Algorithm 1 is (see [7, p.95])

$$|p(x) - \mathsf{Horner}\,(p, x)\,| \leq \gamma_{2n}\overline{p}(x), \tag{5}$$

where $\overline{p}(x) = \sum_{i=0}^{n} |a_i||x^i|$. So, the accuracy of the computed evaluation is linked to the condition number of the polynomial evaluation,

$$\frac{|p(x) - \mathsf{Horner}\,(p, x)\,|}{|p(x)|} \leq \gamma_{2n}\,\mathrm{cond}(p, x). \tag{6}$$

Clearly, the condition number $\mathrm{cond}(p, x)$ can be arbitrarily large. In particular, when $\mathrm{cond}(p, x) > \gamma_{2n}^{-1}$, we cannot guarantee that the computed result $\mathsf{Horner}\,(p, x)$ contains any correct digit.
If a FMA instruction is available on the considered architecture, then we can change the line $r_i = r_{i+1} \otimes x \oplus a_i$ in Algorithm 1 by $r_i = \mathsf{FMA}\,(r_{i+1}, x, a_i)$. This gives the following algorithm HornerFMA (Algorithm 2).

Algorithm 2. Horner scheme with a FMA

function $[r_0] = \mathsf{HornerFMA}\,(p, x)$
$r_n = a_n$
for $i = n-1 : -1 : 0$
  $r_i = \mathsf{FMA}\,(r_{i+1}, x, a_i)$
end

This enables to improve slightly the error bound, since now,

$$\frac{|p(x) - \mathsf{HornerFMA}\,(p, x)\,|}{|p(x)|} \leq \gamma_{n}\,\mathrm{cond}(p, x). \tag{7}$$

With this method, the flops count is also divided by two.

2.3 Sum of two polynomials

Here we state a preliminary lemma we use to compute corrective terms for the polynomial evaluation in Section 4. Let $p(x) = \sum_{i=0}^{n} a_i x^i$ and $q(x) = \sum_{i=0}^{n} b_i x^i$ be two polynomials of degree n with floating point coefficients, and let x be a floating point value. An approximate of $(p+q)(x)$ can be computed with Algorithm 3, which requires $2n+1$ flops.

Algorithm 3. Evaluation of the sum of two polynomials.

function $[v_0] = \mathsf{HornerSumFMA}\,(p, q, x)$
$v_n = a_n \oplus b_n$
for $i = n-1 : -1 : 0$
  $v_i = \mathsf{FMA}\,(v_{i+1}, x, (a_i \oplus b_i))$
end

Lemma 1. *Let us consider the floating point evaluation of* $(p+q)(x)$ *computed with* $\mathsf{HornerSumFMA}\,(p,q,x)$ *(Algorithm 3). The computed result satisfies the following forward error bound,*

$$|\mathsf{HornerSumFMA}\,(p,q,x) - (p+q)(x)| \leq \gamma_{n+1}(\overline{p+q})(x).$$

Proof. A straightforward error analysis shows that

$$v_0 = (1+\theta_{n+1})(a_n+b_n)x^n+(1+\theta_{n+1})(a_{n-1}+b_{n-1})x^{n-1}+(1+\theta_n)(a_{n-2}+b_{n-2})x^{n-2}+\cdots+(1+\theta_1)(a_0+b_0).$$

It follows that

$$|v_0 - (p+q)(x)| \leq \gamma_{n+1}\sum_{i=0}^{n}|a_i+b_i||x^i| = \gamma_{n+1}(\overline{p+q})(x),$$

since the quantities θ_i verify $|\theta_i| \leq \gamma_i$. □

3 Error free transformations

In this section, we review well known results concerning the EFT of the elementary floating point operations $+$, $-$ and $\times$. A recent algorithm by Boldo and Muller is described, for the EFT of the FMA operation. We also present two different EFT for the polynomial evaluation using the Horner scheme performed with the FMA.

3.1 EFT for the elementary operations

Let $\circ$ be in $\{+,-,\times\}$, a and b be two floating point numbers, and $x = \mathrm{fl}\,(a \circ b)$. The elementary rounding error in the computation of x is

$$y = (a \circ b) - \mathrm{fl}\,(a \circ b)\,, \tag{8}$$

that is the difference between the exact result and the computed result of the operation. For $\circ$ in $\{+,-,\times\}$, the elementary rounding error y is a floating point value, and is computable using only floating point operations. Thus, for $\circ$ in $\{+,-,\times\}$, any pair of floating point inputs (a,b) can be transformed into an output pair (x,y) of floating point numbers such that

$$a \circ b = x + y \quad \text{and} \quad x = \mathrm{fl}\,(a \circ b)\,.$$

Let us emphasize that this relation between these four floating point values relies on real operators and exact equality. Ogita *et al.* [14] call such a transformation an error-free transformation.
The EFT for the addition is given by the well known $\mathsf{2Sum}$ algorithm by Knuth [9]. $\mathsf{2Sum}$ (Algorithm 4) requires 6 flops (floating point operations).

Algorithm 4. EFT of the sum of two floating point numbers.

function $[x,y] = \mathsf{2Sum}\,(a,b)$
  $x = a \oplus b$
  $z = x \ominus a$
  $y = (a \ominus (x \ominus z)) \oplus (b \ominus z)$

For the EFT of the product, we could use the well know $\mathsf{2Prod}$ algorithm by Dekker and Veltkamp [4]. This algorithm requires 17 flops, with no branch nor access to the mantissa. But as the FMA instruction is available, we can use the following method instead [13, 14]. For a, b and c three floating point values, we recall that $\mathsf{FMA}\,(a,b,c)$ is the exact result $a \times b + c$ rounded to the nearest floating point value. Since $y = a \times b - a \otimes b$, then $y = \mathsf{FMA}\,(a,b,-(a \otimes b))$ and $\mathsf{2Prod}$ can be replaced by Algorithm 5 which requires only 2 flops.

Algorithm 5. EFT of the product of two floating point numbers with a FMA.

function $[x,y] = \mathsf{2ProdFMA}\,(a,b)$
  $x = a \otimes b$
  $y = \mathsf{FMA}\,(a,b,-x)$

We sum up the properties of these algorithms in the following theorem.

Theorem 2 ([14]). *Given two floating point numbers a and b, let x and y the two floating point values such that $[x, y] = \mathsf{2Sum}(a, b)$ (Algorithm 4). Then,*

$$a + b = x + y, \quad \textit{with} \quad x = a \oplus b, \quad |y| \leq \mathbf{u}|x| \textit{and} \quad |y| \leq \mathbf{u}|a + b|.$$

Given two floating point numbers a and b, let x and y the two floating point values such that $[x, y] = \mathsf{2ProdFMA}(a, b)$ (Algorithm 5). Then,

$$a \times b = x + y, \quad \textit{with} \quad x = a \otimes b, \quad |y| \leq \mathbf{u}|x| \textit{and} \quad |y| \leq \mathbf{u}|a \times b|.$$

An algorithm that computes an EFT for the FMA has been recently given by Boldo and Muller [2]. The EFT of a FMA operation cannot be represented as a sum of two floating point numbers, as it is the case for the addition and for the product. Therefore, the following algorithm 3FMA produces three floating point numbers. For efficiency reasons, 3FMA does not perform any renormalisation of the final result, as proposed in [2].

Algorithm 6. EFT for the FMA operation.

function $[x, y, z] = \mathsf{3FMA}\,(a, b, c)$
  $x = \mathsf{FMA}\,(a, b, c)$
  $(u_1, u_2) = \mathsf{2ProdFMA}\,(a, b)$
  $(\alpha_1, z) = \mathsf{2Sum}\,(b, u_2)$
  $(\beta_1, \beta_2) = \mathsf{2Sum}\,(u_1, \alpha_1)$
  $y = (\beta_1 \ominus x) \oplus \beta_2$

Algorithm 6 requires 17 flops. It satisfies the following properties.

Theorem 3 ([2]). *Given a, b, and c three floating point values, let x, y and z be the three floating point numbers such that $[x, y, z] = \mathsf{3FMA}\,(a, b, c)$. Then we have*

- $a \times b + c = x + y + z$ *exactly, with* $x = \mathsf{FMA}\,(a, b, c)$
- $|y + z| \leq \mathbf{u}|x|$ *and* $|y + z| \leq \mathbf{u}|ax + b|$,
- $y = 0$ *or* $|y| > |z|$.

We notice that the algorithms presented in this subsection require only well optimizable floating point operations. They do not use branches, nor access to the mantissa that can be time consuming.

3.2 Two EFT for the Horner scheme

We propose here two EFT for the polynomial evaluation with the Horner scheme. Both are used in the next section to develop two compensated algorithms.

3.2.1 One EFT with 2ProdFMA

We describe here an improvement of the EFT for the polynomial evaluation with the Horner scheme proposed in [5]. 2ProdFMA is simply used instead of 2Prod to compute the EFT more efficiently.

Theorem 4. *Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial of degree n with floating point coefficients, and let x be a floating point value. Then following Algorithm 7 computes both*

- *the floating point value* $\mathsf{Horner}\,(p, x)$ *(Algorithm 1), and*
- *two polynomials p_π and p_σ, of degree $n - 1$, with floating point coefficients,*

and we write

$$[\mathsf{Horner}\,(p, x)\,, p_\pi, p_\sigma] = \mathsf{EFTHornerFMA}\,(p, x)\,.$$

Then,

$$p(x) = \mathsf{Horner}\,(p, x) + (p_\pi + p_\sigma)(x). \tag{9}$$

This relation means that EFTHornerFMA *is an EFT for the polynomial evaluation with the Horner scheme. Algorithm 7 requires $8n$ flops.*

Algorithm 7. EFT for the Horner scheme using algorithm 2ProdFMA

function $[\mathsf{Horner}\,(p,x)\,, p_\pi, p_\sigma] = \mathsf{EFTHornerFMA}(p,x)$
$s_n = a_n$
for $i = n-1:-1:0$
  $[p_i, \pi_i] = \mathsf{2ProdFMA}\,(s_{i+1}, x)$
  $[s_i, \sigma_i] = \mathsf{2Sum}\,(p_i, a_i)$
  Let π_i be the coefficient of degree i in p_π
  Let σ_i be the coefficient of degree i in p_σ
end
$\mathsf{Horner}\,(p,x) = s_0$

of Theorem 4. We consider the for loop of Algorithm 7. For $i = 0, \ldots, n-1$, since 2Sum and 2ProdFMA are EFT, it follows from Theorem 2 that $s_{i+1}x = p_i + \pi_i$ and $p_i + a_i = s_i + \sigma_i$. Thus we have

$$s_i = s_{i+1}x + a_i - \pi_i - \sigma_i, \quad \text{for} \quad i = 0, \ldots, n-1.$$

Since $s_n = a_n$, the whole for loop yields

$$s_0 = \left[\sum_{i=0}^{n} a_i x^i\right] - \left[\sum_{i=0}^{n-1} \pi_i x^i\right] - \left[\sum_{i=0}^{n-1} \sigma_i x^i\right],$$

and $\mathsf{Horner}\,(p,x) = p(x) - (p_\pi + p_\sigma)(x)$. □

Proposition 5. *Given $p(x) = \sum_{i=0}^{n} a_i x^i$ a polynomial of degree n with floating point coefficients, and x a floating point value. Let y be the floating point value, p_π and p_σ be the two polynomials of degree $n-1$, with floating point coefficients, such that $[y, p_\pi, p_\sigma] = \mathsf{EFTHornerFMA}(p,x)$ (Algorithm 7). Then,*

$$(\overline{p_\pi} + \overline{p_\sigma})(x) \leq \gamma_{2n}\overline{p}(x).$$

Proof. Applying the standard model of floating point arithmetic (3), for $i = 1, \ldots, n$, the two computations in the loop of Algorithm 7 verify

$$\begin{aligned} |p_{n-i}| &= |s_{n-i+1} \otimes x| \leq (1+\mathbf{u})|s_{n-i+1}||x| \\ \text{and} \quad |s_{n-i}| &= |p_{n-i} \oplus a_{n-i}| \leq (1+\mathbf{u})(|p_{n-i}| + |a_{n-i}|). \end{aligned}$$

A standard error analysis of Algorithm 7 based on the two previous relations yields

$$p_{n-i} = (1+\theta_{2i-1})a_n x^i + (1+\theta_{2i-2})a_{n-1}x^{i-1} + (1+\theta_{2i-4})a_{n-2}x^{i-2} + \cdots + (1+\theta_2)a_{n-i+1},$$

and

$$s_{n-i} = (1+\theta_{2i})a_n x^i + (1+\theta_{2i-1})a_{n-1}x^{i-1} + (1+\theta_{2i-3})a_{n-2}x^{i-2} + \cdots + (1+\theta_1)a_{n-i},$$

for $i = 1, \ldots, n$. Each of the θ_j in the previous formulae is bounded according to $|\theta_j| \leq \gamma_j$. Thus, for $i = 1, \ldots, n$, we deduce

$$\begin{aligned} |p_{n-i}||x^{n-i}| &\leq (1+\gamma_{2i-1})\overline{p}(x) \\ \text{and} \quad |s_{n-i}||x^{n-i}| &\leq (1+\gamma_{2i})\overline{p}(x). \end{aligned}$$

From Theorem 2, since 2Sum and 2ProdFMA are EFT, for $i = 0, \cdots, n-1$, we have $|\pi_i| \leq \mathbf{u}|p_i|$ and $|\sigma_i| \leq \mathbf{u}|s_i|$. Therefore,

$$\begin{aligned} (\overline{p_\pi} + \overline{p_\sigma})(x) &= \sum_{i=0}^{n-1} (|\pi_i| + |\sigma_i|)|x^i| \\ &\leq \mathbf{u}\sum_{i=1}^{n} (|p_{n-i}| + |\sigma_{n-i}|)|x^{n-i}|, \end{aligned}$$

and we obtain

$$\begin{aligned} (\overline{p_\pi} + \overline{p_\sigma})(x) &\leq \mathbf{u}\sum_{i=1}^{n} (2 + \gamma_{2i-1} + \gamma_{2i})\,\overline{p}(x) \\ &\leq 2n\,\mathbf{u}\,(1+\gamma_{2n})\,\overline{p}(x). \end{aligned}$$

Since $2n\,\mathbf{u}(1+\gamma_{2n}) = \gamma_{2n}$, we finally prove the result. □

Table 1: Description of the experimented routines, and of the experimental environments

environment	description
(I)	Intel Itanium I, 733MHz, GNU Compiler Collection 2.96
(II)	Intel Itanium II, 900MHz, GNU Compiler Collection 3.3.5
(III)	Intel Itanium II, 1.6GHz, GNU Compiler Collection 3.3.3

routine	description
HornerFMA	IEEE-754 double precision with the FMA (Algorithm 2)
CompHornerFMA	Compensated algorithm (Algorithm 9 based on EFTHornerFMA)
CompHorner3FMA	Compensated algorithm (Algorithm 10 based on EFTHorner3FMA)
DDHornerFMA	Horner scheme performed with the double-double format + FMA

3.2.2 One EFT with 3FMA

We propose here a second EFT that uses 3FMA.

Theorem 6. *Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial of degree n with floating point coefficients, and let x be a floating point value. Then following Algorithm 8 computes both*

- *the floating point evaluation* $\mathsf{HornerFMA}\,(p, x)$ *(Algorithm 2), and*
- *two polynomials p_ε and p_φ, of degree $n-1$, with floating point coefficients,*

and we write

$$[\mathsf{HornerFMA}\,(p, x)\,, p_\varepsilon, p_\varphi] = \mathsf{EFTHorner3FMA}\,(p, x)\,.$$

Then,

$$p(x) = \mathsf{HornerFMA}\,(p, x) + (p_\varepsilon + p_\varphi)(x). \tag{10}$$

This relation means that EFTHorner3FMA *is an EFT for the polynomial evaluation with the Horner scheme. Algorithm 8 requires $17n$ flops.*

Algorithm 8. EFT for polynomial evaluation with the Horner scheme using 3FMA

function $[\mathsf{HornerFMA}\,(p, x)\,, p_\varepsilon, p_\varphi] = \mathsf{EFTHorner3FMA}(p, x)$
$u_n = a_n$
for $i = n-1 : -1 : 0$
  $[u_i, \varepsilon_i, \varphi_i] = \mathsf{3FMA}\,(u_{i+1}, x, a_i)$
  Let ε_i be the coefficient of degree i in p_ε
  Let φ_i be the coefficient of degree i in p_φ
end
$\mathsf{HornerFMA}\,(p, x) = u_0$

of Theorem 6. As 3FMA is an EFT, from Theorem 3 it follows that $u_{i+1}x + a_i = u_i + \varepsilon_i + \varphi_i$. Thus, for $i = 0, \ldots, n-1$, we have $u_i = u_{i+1}x + a_i - \varepsilon_i - \varphi_i$. Since $u_n = a_n$, the whole loop yields

$$u_0 = \left[\sum_{i=0}^{n} a_i x^i\right] - \left[\sum_{i=0}^{n-1} \varepsilon_i x^i\right] - \left[\sum_{i=0}^{n-1} \varphi_i x^i\right],$$

that is $\mathsf{HornerFMA}\,(p, x) = p(x) - (p_\varepsilon + p_\varphi)(x)$. □

Proposition 7. *Given $p(x) = \sum_{i=0}^{n} a_i x^i$ a polynomial of degree n with floating point coefficients, and x a floating point value. Let y be the floating point value, p_ε and p_φ be the two polynomials of degree $n-1$, with floating point coefficients, such that $[y, p_\varepsilon, p_\varphi] = \mathsf{EFTHorner3FMA}(p, x)$ (Algorithm 8). Then,*

$$(\overline{p_\varepsilon + p_\varphi})(x) \leq \gamma_n \overline{p}(x).$$

Proof. A standard error analysis of Algorithm 8 yields

$$u_{n-i} = (1+\theta_i)a_n x^i + (1+\theta_i)a_{n-1}x^{i-1} + (1+\theta_{i-1})a_{n-2}x^{i-2} + \cdots + (1+\theta_1)a_{n-i},$$

for $i = 1, \ldots, n$. Each of the θ_j in the previous formula is bounded according to $|\theta_j| \leq \gamma_j$, and $\gamma_i > \gamma_{i-1} > \cdots > \gamma_1$. Thus, for $i = 1, \ldots, n$, we write

$$\begin{aligned} |u_{n-i}||x^{n-i}| &\leq (1+\gamma_i)\left[|a_n||x^n| + \cdots + |a_{n-i}||x^{n-i}|\right] \\ &\leq (1+\gamma_i)\overline{p}(x). \end{aligned} \tag{11}$$

On the other hand, since $[u_i, \varepsilon_i, \varphi_i] = \mathsf{3FMA}\,(u_{i+1}, x, a_i)$, from Theorem 3 we have $|\varepsilon_{n-i} + \varphi_{n-i}| \leq \mathbf{u}|u_{n-i}|$, for $i = 1, \ldots, n$. Thus we write

$$\begin{aligned} (\overline{p_\varepsilon + p_\varphi})(x) &= \sum_{i=0}^{n-1} |\varepsilon_i + \varphi_i||x^i| \\ &= \sum_{i=1}^{n} |\varepsilon_{n-i} + \varphi_{n-i}||x^{n-i}| \\ &\leq \mathbf{u}\sum_{i=1}^{n} |u_{n-i}||x^{n-i}|. \end{aligned}$$

Relation (11) yields

$$\begin{aligned} (\overline{p_\varepsilon + p_\varphi})(x) &\leq \mathbf{u}\sum_{i=1}^{n}(1+\gamma_i)\overline{p}(x) \\ &\leq n\,\mathbf{u}(1+\gamma_n)\overline{p}(x). \end{aligned}$$

Since $n\,\mathbf{u}(1+\gamma_n) = \gamma_n$, this proves the proposition. □

4 Compensating the Horner scheme

We first present CompHornerFMA, the compensated algorithm with EFTHornerFMA. Next we describe more briefly algorithm CompHorner3FMA that uses EFTHorner3FMA.

4.1 Algorithm CompHornerFMA

From Theorem 4 the global forward error affecting the floating point evaluation of p at x according to the Horner scheme is

$$e(x) = p(x) - \mathsf{Horner}\,(p, x) = (p_\pi + p_\sigma)(x),$$

where the two polynomials p_π and p_σ are exactly computed by EFTHornerFMA (Algorithm 7), together with the approximate $\mathsf{HornerFMA}\,(p, x)$. Therefore, the key of the following algorithm is to compute an approximate of the global error $e(x)$ in working precision, and then to compute a corrected result

$$res = \mathsf{Horner}\,(p, x) \oplus \mathrm{fl}\,(e(x))\,.$$

We say that $c = \mathrm{fl}\,(e(x))$ is a corrective term for the first result $\mathsf{HornerFMA}\,(p, x)$. The corrected result res is expected to be more accurate than $\mathsf{HornerFMA}\,(p, x)$ as proved in the sequel of the section. We compute the corrective term c by evaluating the polynomial whose coefficients are those of $p_\pi + p_\sigma$ rounded to the nearest floating point value: for this we use HornerSumFMA (Algorithm 3).

Algorithm 9. Compensated Horner scheme with EFTHornerFMA

function $[res] = \mathsf{CompHornerFMA}\,(p, x)$
$[h, p_\pi, p_\sigma] = \mathsf{EFTHornerFMA}\,(p, x)$
$c = \mathsf{HornerSumFMA}\,(p_\pi, p_\sigma, x)$
$res = h \oplus c$

Table 2: Measured time performances for CompHorner3FMA, CompHornerFMA, DDHornerFMA.

environment	CompHorner3FMA/HornerFMA				CompHornerFMA/HornerFMA				DDHornerFMA/HornerFMA			
	min.	mean	max.	theo.	min.	mean	max.	theo.	min.	mean	max.	theo.
(I)	3.5	5.0	5.2	19	2.4	2.9	3.0	10	4.8	7.1	7.4	20
(II)	3.2	4.8	7.5	19	2.2	2.7	2.8	10	5.1	8.2	8.4	20
(III)	4.5	5.3	5.6	19	2.2	2.7	2.8	10	5.1	8.2	8.4	20

We prove hereafter that the result of a polynomial evaluation computed with Algorithm 9 is as accurate as if computed by the classic Horner scheme using twice the working precision and then rounded to the working precision. CompHornerFMA requires $10n - 1$ flops.

Theorem 8. *Given a polynomial $p = \sum_{i=0}^{n} a_i x^i$ of degree n with floating point coefficients, and x a floating point value. We consider the result* $\mathsf{CompHornerFMA}\,(p, x)$ *computed by Algorithm 9. Then,*

$$|\mathsf{CompHornerFMA}\,(p, x) - p(x)| \leq \mathbf{u}|p(x)| + (1 + \mathbf{u})\gamma_n\gamma_{2n}\overline{p}(x).$$

Proof. We denote $\delta = |\mathsf{CompHornerFMA}\,(p, x) - p(x)|$. From Algorithm 9, and applying the standard model of floating point arithmetic, we have

$$\begin{aligned} \delta &= |(h \oplus c) - p(x)| \\ &= |(1 + \varepsilon)(h + c) - p(x)|, \end{aligned}$$

with $|\varepsilon| \leq \mathbf{u}$. From Theorem 4, $p(x) = h + (p_\pi + p_\sigma)(x)$, thus

$$\delta \leq \mathbf{u}|p(x)| + (1 + \mathbf{u})|(h + c) - p(x)|.$$

Since $c = \mathsf{HornerSumFMA}\,(p_\pi, p_\sigma, x)$ with p_π and p_σ two polynomials of degree $n - 1$, Lemma 1 yields

$$|(h + c) - p(x)| \leq \gamma_n(\overline{p_\pi + p_\sigma})(x) \leq \gamma_n(\overline{p_\pi} + \overline{p_\sigma})(x).$$

Next, we apply Proposition 5 to obtain

$$|(h + c) - p(x)| \leq \gamma_n\gamma_{2n}\overline{p}(x),$$

and finally,

$$\delta \leq \mathbf{u}|p(x)| + (1 + \mathbf{u})\gamma_n\gamma_{2n}\overline{p}(x).$$

This proves the result. □

It is very interesting to interpret the previous theorem with respect to the condition number of the polynomial evaluation of p at x. Combining the error bound in Theorem 8 with the expression of the condition number (1) for the polynomial evaluation gives the following relation,

$$\frac{|\mathsf{CompHornerFMA}\,(p, x) - p(x)|}{|p(x)|} \leq \mathbf{u} + (1 + \mathbf{u})\gamma_n\gamma_{2n}\,\mathrm{cond}(p, x). \tag{12}$$

In practical applications, we have $(1 + \mathbf{u})\gamma_n\gamma_{2n} \approx \mathbf{u}^2$. In other words, the bound for the relative error of the computed result is essentially $\mathbf{u}^2$ times the condition number of the polynomial evaluation, plus the inevitable summand $\mathbf{u}$ for rounding the result to the working precision. In particular, if $\mathrm{cond}(p, x) \lesssim \mathbf{u}^{-1}$, then the relative accuracy of the result is bounded by a constant of the order $\mathbf{u}$. This means that the compensated Horner scheme computes an evaluation accurate to the last few bits as long as the condition number is smaller than $\mathbf{u}^{-1}$. Besides that, Relation (12) tells us that the computed result is as accurate as if computed by the classic Horner scheme with twice the working precision, and then rounded to the working precision.

4.2 Algorithm CompHorner3FMA

The principle of algorithm CompHorner3FMA is exactly the same as CompHornerFMA, but instead of EFTHornerFMA, we use EFTHorner3FMA to compute a corrected result. Following algorithm CompHorner3FMA requires $19n$ flops.

Algorithm 10. Compensated Horner scheme using EFTHorner3FMA

function $[res] = \mathsf{CompHorner3FMA}\,(p, x)$
$[h, p_\varepsilon, p_\varphi] = \mathsf{EFTHorner3FMA}\,(p, x)$
$c = \mathsf{HornerSumFMA}\,(p_\varepsilon, p_\varphi, x)$
$res = h \oplus c$

As for CompHornerFMA, we prove the following theorem.

Theorem 9. *Given a polynomial $p = \sum_{i=0}^{n} a_i x^i$ of degree n with floating point coefficients, and x a floating point value. We consider the result $\mathsf{CompHorner3FMA}\,(p, x)$ computed by Algorithm 10. Then,*

$$|\mathsf{CompHorner3FMA}\,(p, x) - p(x)| \leq \mathbf{u}|p(x)| + (1 + \mathbf{u})\gamma_n^2 \overline{p}(x).$$

Proof. Let δ denotes $|\mathsf{CompHorner3FMA}\,(p, x) - p(x)|$. From Algorithm 10 and Theorem 6, we have again

$$\delta \leq \mathbf{u}|p(x)| + (1 + \mathbf{u})|(h + c) - p(x)|.$$

Since $c = \mathsf{HornerSumFMA}\,(p_\varepsilon, p_\varphi, x)$ with p_ε and p_φ two polynomials of degree $n - 1$, Lemma 1 yields

$$|(h + c) - p(x)| \leq \gamma_n(\overline{p_\varepsilon + p_\varphi})(x).$$

Next, we apply Proposition 7 to obtain

$$|(h + c) - p(x)| \leq \gamma_n^2 \overline{p}(x),$$

and finally

$$\delta \leq \mathbf{u}|p(x)| + (1 + \mathbf{u})\gamma_n^2 \overline{p}(x).$$

This proves the result. □

Again, combining the error bound in Theorem 8 with the expression of the condition number (1) for the polynomial evaluation gives

$$\frac{|\mathsf{CompHorner3FMA}\,(p, x) - p(x)|}{|p(x)|} \leq \mathbf{u} + (1 + \mathbf{u})\gamma_n^2 \operatorname{cond}(p, x). \tag{13}$$

Since $(1 + \mathbf{u})\gamma_n^2 \approx \mathbf{u}^2$, the previous remarks about error bound (12) apply.While CompHorner3FMA needs almost two times more flops than CompHornerFMA, we notice that the error bounds (12) and (13) are similar. Actually, our experimental results confirm that CompHornerFMA is more efficient than CompHorner3FMA in terms of computing time while being similarly accurate.

5 Experimental results

All our experiments are performed using IEEE-754 double precision. Since the double-doubles [6, 10] are usually considered as the most efficient portable library to double the IEEE-754 double precision, we consider it as a reference in the following comparisons. For our purpose, it suffices to know that a double-double number a is the pair (a_h, a_l) of IEEE-754 floating point numbers with $a = a_h + a_l$ and $|a_l| \leq \mathbf{u}|a_h|$. This property implies a renormalisation step after each arithmetic operation. We denote by DDHornerFMA our implementation of the Horner scheme with the double-double format, derived from the implementation proposed by the authors of [10]. We notice that the double-double arithmetic naturally benefits from the availability of a FMA instruction: DDHornerFMA uses 2ProdFMA in the inner loop of the Horner scheme. DDHornerFMA requires $20n$ flops. Using the double-double library proposed in [6], we can slightly reduce this flops count, but it has almost no impact on the measured computing times.

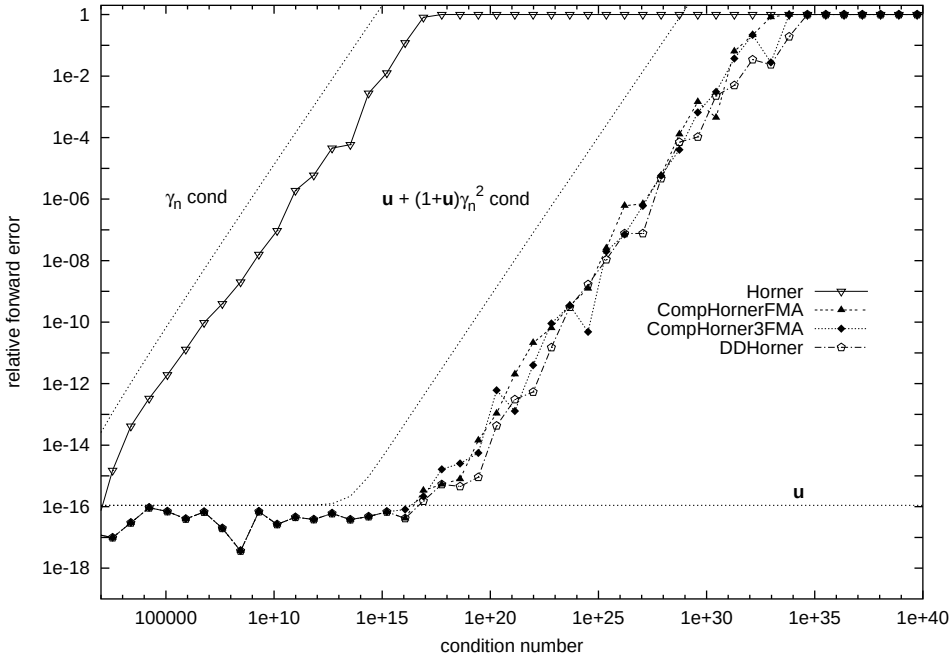

Figure 1: Accuracy of the three experimented routines.

5.1 Accuracy tests

We test the expanded form of the polynomial $p_n(x) = (x-1)^n$. The argument x is chosen near to the unique real root 1 of p_n. The condition number is

$$\mathrm{cond}(p_n, x) = \frac{\overline{p_n}(x)}{|p_n(x)|} = \left|\frac{1+|x|}{1-x}\right|^n,$$

and $\mathrm{cond}(p_n, x)$ grows exponentially with respect to n. In the experiments reported on Figure 1, we have chosen $x = \mathrm{fl}(1.333)$ to provide a floating point value with many non-zero bits in its mantissa. The value of $\mathrm{cond}(p_n, x)$ varies from 10^2 to 10^{40}, that corresponds to degrees range $n = 3, \ldots, 42$. These huge condition numbers have a sense since the coefficients of p and the value x are floating point numbers.
We experiment both HornerFMA, CompHorner3FMA, CompHornerFMA and DDHornerFMA (see Table 1). For each polynomial p_n, the exact value $p_n(x)$ is approximated with a high accuracy thanks to the MPFR library [12]. Figure 1 presents the relative accuracy $|y - p_n(x)|/|p_n(x)|$ of the evaluation y computed by the three algorithms. We set to the value one relative errors greater than one, which means that almost no useful information is available in the computed result. We also display the *a priori* error estimates (7), (13) and (12). We observe that our compensated algorithms exhibits the expected behavior. The full precision solution is computed as long as the condition number is smaller than $\mathbf{u}^{-1} \approx 10^{16}$. Then, for condition numbers between $\mathbf{u}^{-1}$ and $\mathbf{u}^{-2} \approx 10^{32}$, the relative error degrades to no accuracy at all, as the computing precision is $\mathbf{u}$.

5.2 Time performances

All the algorithms are implemented in C-code. We use the same programming techniques for the implementations of three routines CompHornerFMA, CompHorner3FMA and DDHornerFMA. The experimental environments are listed in Table 1. Our measures are performed with polynomials whose degrees vary from 5 to 450 by steps of 5. We choose the coefficients and the arguments at random. For each degree, the routines are tested on the same polynomial with the same argument. Table 2 displays the time ratios of the compared algorithms over HornerFMA. We have reported the minimum, the mean and the maximum of these ratios. The theoretical ratios are also reported, resulting from the number of flops involved by each algorithm.
We notice that the measured slowdown factors introduced are always significantly smaller than theoretically expected. Our compensated algorithms CompHorner3FMA and CompHornerFMA are both significantly faster than DDHorner. Algorithm CompHornerFMA seems to be the most efficient alternative to

improve the accuracy of the Horner scheme. It runs about 1.8 times faster than CompHorner3FMA and more than two times faster than DDHorner that uses the double-double library.

6 Concluding remarks

We have presented two accurate algorithms for the evaluation of univariate polynomials in floating point arithmetic. The only assumptions we made are that the floating point arithmetic is conformed to the IEEE-754 standard, and that a Fused Multiply-Add instruction is available.We have proven that the accuracy of the results computed by our compensated algorithms is similar to the one given by the Horner scheme performed in doubled working precision.
Our algorithms use only basic floating point operations, and only the same precision as the given data. They use no branch nor access to the mantissa that can be time consuming. As a result, they are fast not only in terms of flops count, but also in terms of computing time: the slowdown factors due to these improvements of the accuracy are much smaller than theoretically expected.
According to our experiments, algorithm CompHornerFMA seems to be the most efficient alternative to improve the accuracy of the Horner scheme: it runs only three times slower than the classic Horner scheme performed with a FMA, and more than two times faster than other existing alternatives that guarantee the same output accuracy.

References

[1] David H. Bailey. Algorithm 719, multiprecision translation and execution of Fortran programs. *ACM Trans. Math. Software*, 19(3):288–319, 1993.

[2] Sylvie Boldo and Jean-Michel Muller. Some functions computable with a fused-mac. In IEEE, editor, *Proceedings of the 17th IEEE Symposium on Computer Arithmetic, 2005, Cape Cod, Massachusetts, USA*. IEEE Computer Society Press, 2005.

[3] Richard P. Brent. A Fortran multiple-precision arithmetic package. *ACM Trans. Math. Software*, 4(1):57–70, 1978.

[4] Theodorus J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971.

[5] Stef Graillat, Philippe Langlois, and Nicolas Louvet. Compensated Horner scheme, 2005. Submitted to *SIAM J. Sci. Comput.*

[6] Yozo Hida, Xiaoye S. Li, and David H. Bailey. Algorithms for quad-double precision floating point arithmetic. In Neil Burgess and Luigi Ciminiera, editors, *Proceedings of the 15th Symposium on Computer Arithmetic, Vail, Colorado*, pages 155–162, Los Alamitos, CA, USA, 2001. Institute of Electrical and Electronics Engineers.

[7] Nicholas J. Higham. *Accuracy and Stability of Numerical Algorithms*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition, 2002.

[8] IEEE Standards Committee 754. *IEEE Standard for binary floating-point arithmetic, ANSI/IEEE Standard 754-1985*. Institute of Electrical and Electronics Engineers, Los Alamitos, CA, USA, 1985. Reprinted in SIGPLAN Notices, 22(2):9-25, 1987.

[9] Donald Ervin Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, MA, USA, third edition, 1998.

[10] Xiaoye S. Li, James W. Demmel, David H. Bailey, Greg Henry, Yozo Hida, Jimmy Iskandar, William Kahan, Suh Y. Kang, Anil Kapur, Michael C. Martin, Brandon J. Thompson, Teresa Tung, and Daniel J. Yoo. Design, implementation and testing of extended and mixed precision BLAS. *ACM Trans. Math. Software*, 28(2):152–205, 2002.

[11] Peter Markstein. *IA-64 and elementary functions: speed and precision*. Hewlett-Packard professional books. Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 2000.

[12] The MPFR library. Available at http://www.mpfr.org.

[13] Yves Nievergelt. Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. *ACM Trans. Math. Software*, 29(1):27–48, 2003.

[14] Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.

[15] Michèle Pichat. Correction d'une somme en arithmétique à virgule flottante. (French) [correction of a sum in floating-point arithmetic]. *Numer. Math.*, 19:400–406, 1972.