Improving the Compensated Horner Scheme with a Fused Multiply and Add

Stef Graillat Philippe Langlois Nicolas Louvet
Laboratoire LP2A, Université de Perpignan Via Domitia
52, avenue Paul Alduy, F-66860 Perpignan, France
[graillat, langlois, nlouvet]@univ-perp.fr

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 propose an accurate compensated Horner scheme specially designed to take advantage of the Fused Multiply and Add. We prove that the computed result is as accurate as if computed in twice the working precision. The algorithm we present is fast since it only requires well optimizable floating point operations, performed in the same working precision as the given data.

Categories and Subject Descriptors

G.4 [**Mathematics of Computation**]: Mathematical Software—*Reliability and robustness*; G.4 [**Mathematics of Computation**]: Mathematical Software—*Efficiency*

General Terms

Algorithms, Reliability, Performance

Keywords

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

1. INTRODUCTION

One of the three main processes associated with polynomials is evaluation, the two other ones being interpolation and root finding. Higham [7, chap. 5] devotes an entire chapter to polynomials and more especially to polynomial evaluation. The small backward error the Horner scheme introduce 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 at x is

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

When the computing precision is $\mathbf{u}$, evaluating $p(x)$ is ill-conditioned when $1 \ll \operatorname{cond}(p, x) \le \mathbf{u}^{-1}$. If the coefficients of p are exact numbers in precision $\mathbf{u}$, we can also consider extremely ill-conditioned evaluations, *i.e.*, such that $\operatorname{cond}(p, x) > \mathbf{u}^{-1}$.

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 [1, 10]. 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.

In [4] 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

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

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 rounding back to the working precision. The key tool to introduce more accuracy is what Ogita, Rump and Oishi call error-free transformations (EFT) in [12]: "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 number is computable using the well know algorithm 2Sum by Knuth [9]. 2Product by Veltkamp and Dekker [3] 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 $ab+c$ with only one final rounding error. 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 2ProductFMA presented hereafter computes this EFT in only two flops when a FMA is available [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, the authors have proven that the EFT for the FMA has to be expressed as 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

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

Both 2ProductFMA and 3FMA can be used to improve the compensated Horner scheme presented in [4].

In this paper, we focus on the use of 2ProductFMA, and we present the corresponding compensated Horner scheme denoted by CompHornerFMA. The proposed algorithm is as accurate as the classic Horner scheme performed in twice the working precision, *i.e.*, the accuracy of the computed result sastifies Relation (2). Experimental results show that time penalty due to this improvement of the precision is quite reasonable: our algorithm is not only fast in terms of floating point operations (flops) count, but also in terms of execution time.

By lack of the space most of our proofs are omitted. Nevertheless, the interested reader can refers to our research report [5] (available at http://webdali.univ-perp.fr), where our compensated Horner scheme based on 3FMA is also detailed.

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 EFT we use for the Horner scheme. In Section 4, we describe our compensated algorithm for the polynomial evaluation. The computed result is 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 implementation, 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. The symbols $\oplus$, $\ominus$, $\otimes$ and $\oslash$, representing respectively the floating point addition, subtraction, multiplication and division (*e.g.*, $a \oplus b = \mathrm{fl}\,(a+b)$). 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 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 occurs, 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

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

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

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

For any positive integer k, we denote $\gamma_k = \frac{k\,\mathbf{u}}{1-k\,\mathbf{u}}$. When using this notation, we always implicitly assume $k\,\mathbf{u} < 1$.

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

The accuracy of the computed evaluation is linked to the condition number of the polynomial evaluation,

$$\frac{|p(x) - \mathsf{Horner}\,(p,x)\,|}{|p(x)|} \le \gamma_{2n} \frac{\overline{p}(x)}{|p(x)|} = \gamma_{2n}\,\mathrm{cond}(p,x), \tag{5}$$

where $\overline{p}(x) = \sum_{i=0}^{n} |a_i||x^i|$ (see [7, p.95]). 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).

Table 1: Description of the experimented routines, and of the experimental environments
Experimental environments:
- (I): Intel Itanium I, 733MHz, GNU GCC 2.96
- (II): Intel Itanium II, 900MHz, GNU GCC 3.3.5
- (III): Intel Itanium II, 1.6GHz, GNU GCC 3.3.3

Experimented routines:
- HornerFMA (Algorithm 2)
- CompHornerFMA (Algorithm 6)
- DDHornerFMA: internal double-double computation

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
```

With this method, the flops count is divided by two, and the error bound is slightly improved, since now,

$$\frac{|p(x) - \mathsf{HornerFMA}\,(p, x)\,|}{|p(x)|} \le \gamma_n \operatorname{cond}(p, x). \qquad (6)$$

3. ERROR FREE TRANSFORMATIONS

3.1 EFT for the elementary operations

For $\circ$ in $\{+, -, \times\}$, it is well known that the elementary rounding error y occuring in the computation of $x = \mathrm{fl}\,(a \circ b)$ 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 (not on approximate floating point counterparts). Ogita *et al.* [12] call such a transformation an *error free transformation* (EFT).
The EFT for the addition is given by the well known 2Sum algorithm by Knuth [9]. 2Sum (Algorithm 3) requires 6 flops (floating point operations).

ALGORITHM 3. *EFT for the sum.*

```
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 algorithm 2Product by Dekker and Veltkamp [3]. 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 [11, 12]. 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 2Product can be replaced by Algorithm 4 which requires only 2 flops.

ALGORITHM 4. *EFT for the product.*

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

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

THEOREM 1 ([12]). *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 3). Then,*

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

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

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

3.2 An EFT for the Horner scheme

We now propose an EFT for the polynomial evaluation with the Horner scheme. This EFT is based on algorithms 2Sum and 2ProductFMA. The proof of Theorem 2 can be found in [5].

ALGORITHM 5. *EFT for the Horner scheme*

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

THEOREM 2. *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 Algorithm 5 computes both*

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

and we write

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

Then,

$$p(x) = \mathsf{Horner}\,(p, x) + (p_\pi + p_\sigma)(x), \qquad (7)$$

and we have

$$(\overline{p_\pi} + \overline{p_\sigma})(x) \le \gamma_{2n} \overline{p}(x). \qquad (8)$$

Relation (7) means that EFTHorner is an EFT for the polynomial evaluation with the Horner scheme. Algorithm 5 requires $8n$ flops.

4. COMPENSATED HORNER SCHEME

From Theorem 2 the global forward error affecting the floating point evaluation of p at x according to the Horner scheme is

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

Table 2: Measured time performances for CompHornerFMA and DDHornerFMA.

environment	CompHornerFMA/HornerFMA			
	min.	mean	max.	theo.
(I)	2.4	2.9	3.0	10
(II)	2.2	2.7	2.8	10
(III)	2.2	2.7	2.8	10
environment	**DDHornerFMA/HornerFMA**			
	min.	mean	max.	theo.
(I)	4.8	7.1	7.4	20
(II)	5.1	8.2	8.4	20
(III)	5.1	8.2	8.4	20

where the two polynomials p_π and p_σ are exactly computed by EFTHorner (Algorithm 5), together with the approximate 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

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

We say that $c = \text{fl}\,(e(x))$ is a corrective term for Horner (p, x). The corrected result res is expected to be more accurate than the first result Horner (p, x) as proved in the sequel of the section. We compute an approximate of the correcting term c by evaluating the polynomial whose coefficients are those of $p_\pi + p_\sigma$ rounded to the nearest floating point value.

4.1 The compensated algorithm

EFTHorner is used in the following algorithm to compute the corrected result. We have proven hereafter that the result of a polynomial evaluation computed with the compensated Horner scheme (Algorithm 6) 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.

ALGORITHM 6. *Compensated Horner scheme*

```
function [res] = CompHornerFMA (p, x)
[h, p_ε, p_φ] = EFTHorner (p, x)
c = HornerSumFMA (p_ε, p_φ, x)
res = h ⊕ c
```

THEOREM 3. *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* CompHornerFMA (p, x) *computed by Algorithm 6. Then,*

$$|\text{CompHornerFMA}\,(p, x) - p(x)| \le \mathbf{u}|p(x)| + (1 + \mathbf{u})\gamma_n^2 \overline{p}(x).$$

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 3 with the expression of the condition number (1) for the polynomial evaluation gives the following relation,

$$\frac{|\text{CompHornerFMA}\,(p, x) - p(x)|}{|p(x)|} \le \mathbf{u} + \gamma_{2n}^2 \,\text{cond}(p, x). \quad (9)$$

PROOF. From Theorem 2, $p(x) = h + (p_\pi + p_\sigma)(x)$, thus

$$|\text{CompHornerFMA}\,(-)\,p(x)| \le \mathbf{u}|p(x)| + (1+\mathbf{u})|(h+c) - p(x)|.$$

Since $c = \text{HornerFMA}\,(p_\pi \oplus p_\sigma, x)$, a standard error analysis yields

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

Next, we use Relation (8) to obtain

$$|(h + c) - p(x)| \le \gamma_n \gamma_{2n} \overline{p}(x) \le \gamma_{2n}^2 \overline{p}(x).$$

This proves the result. For a more detailed proof see [5]. □

In practical applications, we have $\gamma_{2n}^2 \approx \mathbf{u}^2$. 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 $\text{cond}(p, x) \lesssim \mathbf{u}^{-1}$, 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 (9) 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.

5. EXPERIMENTAL RESULTS

All our experiments are performed using IEEE-754 double precision. Since the double-doubles [6] 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 comparisions. 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 $|a_l| \le \mathbf{u}|a_h|$. 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, based on the implementation proposed by the authors of [6]. We notice that the double-double arithmetic naturally benefites from the availability of a FMA instruction: DDHornerFMA uses 2ProductFMA in the inner loop of the Horner scheme. DDHornerFMA requires $20n$ flops.

Accuracy of the compensated Horner scheme

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. Here we have

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

and $\text{cond}(p_n, x)$ grows exponentially with respect to n. In the experiments reported on Figure 1, we choose $x = \text{fl}\,(1.333)$; $\text{cond}(p_n, x)$ varies from 10^2 to 10^{40}, that corresponds to the degree range $n = 3, \ldots, 42$. These huge condition numbers have a sense since here the coefficients of p and the value x are floating point numbers.

We experiment both HornerFMA, CompHornerFMA and DDHornerFMA (see Table 1). Figure 1 presents the relative accuracy $|y - p_n(x)|/|p_n(x)|$ of the evaluation y computed by the three algorithms. The dotted lines represent the *a priori* error estimates (6) and (9). We observe that our compensated algorithm exhibits the expected behavior. The full precision solution is computed as long as the condition number is smaller that $\mathbf{u}^{-1} \approx 10^{16}$. 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 $\mathbf{u}$.

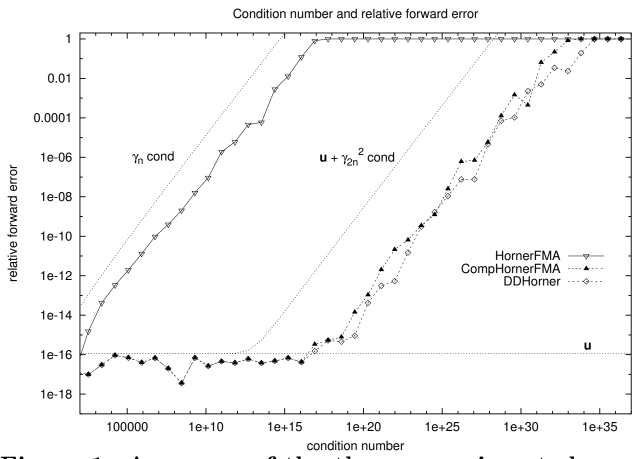

Figure 1: Accuracy of the three experimented routines.

Time performances

All the algorithms are implemented in C-code. 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. For each degree, the routines are tested on the same polynomial with the same argument. Table 2 displays the time ratios of CompHornerFMA and DDHornerFMA over HornerFMA. We have reported the minimum, the mean and the maximum these ratios, together with the theoretical ratios resulting from the number of flops involved by each algorithm. First, we have to notice that the measured slowdown factor introduced either by CompHornerFMA or DDHornerFMA is always significantly smaller than theoretically expected. This interesting property seems to be due to the fact that the classic algorithm performs only one operation with each coefficient of the polynomial, whereas CompHornerFMA and DDHornerFMA perform much more operations with each coefficient. The results reported in Table 2 show that our compensated algorithm CompHornerFMA is about 3 times slower than the classic Horner scheme. The same slowdown factor is about 7 for algorithm DDHornerFMA. Thus, from a practical point of view, we can state that the proposed algorithm is more than twice faster than the Horner scheme with double-doubles.

6. CONCLUDING REMARKS

We have presented an accurate evaluation of univariate polynomials in IEEE-754 floating point arithmetic when a FMA is available. We have proved that the accuracy is similar to the one given by the Horner scheme performed in doubled working precision. This CompHornerFMA algorithm uses only basic floating point operations, and only the same working precision as the data. It uses no branch nor access to the mantissa that can be time consuming on modern architectures. As a result, it is fast not only in terms of flops count but also in terms of measured computing time. In particular, CompHornerFMA runs only about three times slower than the classic Horner scheme, but faster than other existing alternatives that garantee the same output accuracy.

We have noticed that another improvement of the compensated Horner scheme exists with a FMA. Using 3FMA we also improve the compensated Horner scheme with the same output accuracy [5]. Nevertheless, experimental results reported in [5] show that such an implementation runs about twice slower than CompHornerFMA. Algorithm CompHornerFMA here presented seems thus to be the most efficient alternative to improve the accuracy of the Horner scheme.

7. REFERENCES

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

[2] S. Boldo and J.-M. 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] T. J. Dekker. A floating-point technique for extending the available precision. *Numer. Math.*, 18:224–242, 1971.

[4] S. Graillat, P. Langlois, and N. Louvet. Compensated Horner scheme. Research Report 4, Équipe de recherche DALI, Laboratoire LP2A, Université de Perpignan Via Domitia, France, 52 avenue Paul Alduy, 66860 Perpignan cedex, France, July 2005. Submitted to *SIAM J. Sci. Comput.*

[5] S. Graillat, P. Langlois, and N. Louvet. Improving the compensated Horner scheme with a fused multiply and add. Research Report 5, Équipe de recherche DALI, Laboratoire LP2A, Université de Perpignan Via Domitia, France, 52 avenue Paul Alduy, 66860 Perpignan cedex, France, Nov. 2005.

[6] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating point arithmetic. In N. Burgess and L. 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] N. 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] D. E. Knuth. *The Art of Computer Programming: Seminumerical Algorithms*, volume 2. Addison-Wesley, Reading, MA, USA, third edition, 1998.

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

[11] Y. 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.

[12] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM J. Sci. Comput.*, 26(6):1955–1988, 2005.