Compensated Horner algorithm in K times the working precision

Philippe Langlois, Nicolas Louvet

To cite this version:

HAL Id: inria-00267077

https://hal.inria.fr/inria-00267077

Submitted on 26 Mar 2008

Compensated Horner algorithm in K times the working precision

Philippe Langlois
Laboratoire ELIAUS, Université de Perpignan
philippe.langlois@univ-perp.fr

Nicolas Louvet
INRIA, LIP, ENS Lyon
nicolas.louvet@ens-lyon.fr

Abstract

We introduce an algorithm to evaluate a polynomial with floating point coefficients as accurately as the Horner scheme performed in K times the working precision, for K an arbitrary integer. The principle is to iterate the error-free transformation of the compensated Horner algorithm and to accurately sum the final decomposition. We prove this accuracy property with an *a priori* error analysis. We illustrate its practical efficiency with numerical experiments on significant environments and IEEE-754 arithmetic. Comparing to existing alternatives we conclude that this K-times compensated algorithm is competitive for K up to 4, *i.e.*, up to 212 mantissa bits.

Keywords: Accurate polynomial evaluation, compensated algorithms, error-free transformations, floating point arithmetic.

1 Motivations and synthesis

The Horner scheme is of an undeniable practical interest to evaluate a polynomial p at the point x with floating point arithmetic. Since it is backward stable, the Horner scheme returns results arbitrarily less accurate than the working precision $\mathbf{u}$ — when evaluating $p(x)$ is ill-conditioned. This is for example the case in the neighborhood of multiple roots where most of the digits, or even the order, of the computed value of $p(x)$ can be false.

When the IEEE-754 floating point arithmetic is available but not precise enough, "double-double" and "quad-double" libraries are classic software solutions to respectively simulate twice or four times the working precision [3]. The compensated Horner scheme is an efficient alternative introduced in [8]. This compensated evaluation yields the same accuracy as the Horner algorithm computed in doubled working precision.

We present another compensated algorithm that computes an approximate $\overline{r}$ of $p(x)$ that is as accurate as if computed in K-times the working precision ($K \geq 2$). The idea is to iterate the compensation of the Horner scheme proposed in [8], improving the accuracy of the computed result by a factor $\mathbf{u}$ at every iteration step. In the sequel we name CompHornerK this algorithm. Given $p(x) := \sum_{i=0}^{n} a_i x^i$, a polynomial with floating point coefficients, and x a floating point value, we prove the announced behavior, *i.e.*, the accuracy of the compensated result $\overline{r}$ computed with CompHornerK is bounded as follows,

$$\frac{|\overline{r} - p(x)|}{|p(x)|} \leq \mathbf{u} + \mathcal{O}(\mathbf{u}^K)\,\mathrm{cond}(p, x). \tag{1}$$

The classic condition number $\mathrm{cond}(p, x)$ describes the sensitivity of the polynomial evaluation,

$$\mathrm{cond}(p, x) := \frac{\widetilde{p}(x)}{|p(x)|} \geq 1, \tag{2}$$

with[1] $\widetilde{p}(x) := \sum_{i=0}^{n} |a_i x^i|$ [4]. CompHornerK only requires IEEE-754 floating point arithmetic with rounding to the nearest. Our main tool to improve the accuracy of the computed result is an "error-free transformation" (EFT) for the polynomial evaluation with the Horner algorithm.

The time penalty to improve the accuracy with CompHornerK is very reasonable for $K \leq 4$. Practical performances reported hereafter exhibit that CompHornerK is an efficient alternative to other software solutions, such as the quad-double library or MPFR. In particular, our optimized version of CompHornerK with $K = 4$ runs about 40% faster than the corresponding routine with quad-double arithmetic. This justifies the practical interest of the proposed algorithm when only a reasonable increase of the working precision is necessary.

Many problems in Computer Assisted Design (CAD) reduce to find the roots of a polynomial equation, which is subjected to accuracy problems when dealing with multiple roots. More accurate polynomial evaluation algorithms are used by some authors in this area [5]. Our CompHornerK algorithm may be used with success in such cases since no restriction applies to the magnitude of $|x|$, nor to the coefficients neither to the degree of the polynomial.

We start illustrating this motivation with a simple example. Let us consider the evaluation in the neighborhood of its multiple roots of $p(x) = (0.75 - x)^5(1 - x)^{11}$ —$p(x)$ is written in its expanded form. Double precision IEEE-754 arithmetic is used for these experiments and the coefficients of p in the monomial basis are exact IEEE-754 double precision numbers. We denote by Horner the classic Horner algorithm for polynomial evaluation. The top two drawings of Figure 1 illustrate the evaluation of $p(x)$ for 400 equally spaced points in the interval $[0.72, 1.09]$, using Horner and CompHornerK with $K = 2$. It is clear that the Horner evaluation is totally inaccurate and that twice the IEEE-754 double precision is sufficient here to provide the expected smooth drawing of the polynomial. On the bottom three drawings of Figure 1, we zoom the evaluation of $p(x)$ for 400 equally spaced points in the interval $[0.99975, 1.00025]$, using CompHornerK with $K = 2, 3$ and 4. We see that using twice or 3-times the IEEE double precision is not sufficient to obtain a smooth drawing at this scale. As it is well known these simple experiments clearly illustrate that it may be useful to increase the precision to provide an accurate polynomial evaluation in the neighborhood of its multiple roots.

In section 2 we recall some classic notations and well known results about the EFT of arithmetic operators. We also briefly recall the EFT for the Horner evaluation already described in [8]. In section 3 we use this result in as a basic block to design a new EFT for polynomial evaluation. In section 4 we describe the algorithm CompHornerK and perform its error analysis. Sections 5 and 6 are devoted to experimental results that respectively illustrate the actual accuracy and the practical time performances of the proposed CompHornerK algorithm.

2 Notations and previous results

Throughout this paper, we assume a floating point arithmetic adhering to the IEEE-754 floating point standard [6]. We constraint all the computations to be performed in one working precision, with the "round to the nearest" rounding mode. We also assume that no overflow nor underflow occurs during the computations. Next notations are standard (see [4, chap. 2] for example). $\mathbf{F}$ is the set of all normalized floating point numbers and $\mathbf{u}$ denotes the unit round-off, that is half

[1] We apply this tilde notation to other polynomials further in this paper.

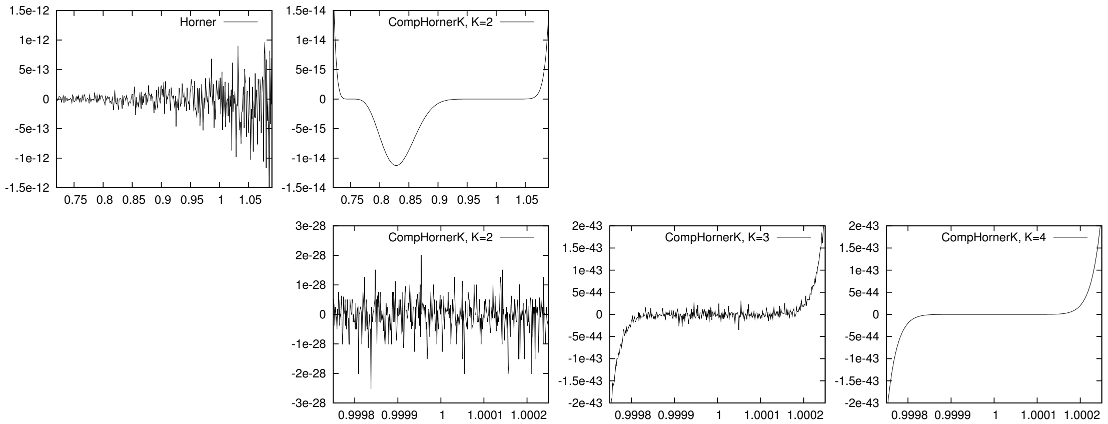


Figure 1: More iterations of CompHornerK are necessary to zoom the polynomial evaluation — here $p(x) = (0.75 - x)^5(1 - x)^{11}$ in its expanded form.

the spacing between 1 and the next representable floating point value. For IEEE-754 double precision with rounding to the nearest, we have $\mathbf{u} = 2^{-53} \approx 1.11 \cdot 10^{-16}$.

The notation $\mathrm{fl}(\cdot)$ denotes the result of a floating point computation where every operation inside the parenthesis is performed in the working precision. When no underflow nor overflow occurs, the following standard model describes the accuracy of every considered floating point computation: for $a, b \in \mathbf{F}$ and for $\circ \in \{+, -, \times, /\}$, with have

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

To keep track of the $(1 + \varepsilon)$ factors in the error analysis, we use the classic $(1 + \theta_k)$ and γ_k notations [4, chap. 3]. For any positive integer k, θ_k denotes a quantity bounded according to $|\theta_k| \leq \gamma_k := k\mathbf{u}/(1 - k\mathbf{u})$. When using these notations, we always implicitly assume $k\mathbf{u} < 1$. In further error analysis, we essentially use the following relations: $(1 + \theta_k)(1 + \theta_j) \leq (1 + \theta_{k+j})$, $k\mathbf{u} \leq \gamma_k$, and $\gamma_k \leq \gamma_{k+1}$.

Now we briefly review well known results about the error-free transformations (EFT) for elementary arithmetic operators. For the EFT of the addition we use the well known TwoSum algorithm by Knuth [7, p.236] that requires 6 flop (floating point operations). TwoProd by Veltkamp and Dekker [1] performs the EFT of the product and requires 17 flop. The next theorem summarizes the properties of TwoSum and TwoProd.

Theorem 1 ([9]). *Let a, b in $\mathbf{F}$ and $\circ \in \{+, \times\}$. Let $x, y \in \mathbf{F}$ such that $[x, y] = \mathsf{TwoSum}(a, b)$ if $\circ = +$, $[x, y] = \mathsf{TwoProd}(a, b)$ otherwise. Then,*

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

We also recall the EFT for the Horner algorithm presented by Langlois and Louvet [8]. This EFT of the polynomial evaluation with the Horner algorithm exhibits the exact rounding error generated by the Horner algorithm together with an algorithm to compute it. We use this EFT as a basic block in the next section to design a new EFT for polynomial evaluation.

Algorithm 2 ([8]). EFT for the Horner algorithm

```
function $[s_0, p_\pi, p_\sigma] = \mathsf{EFTHorner}(p, x)$
  $s_n = a_n$
  for $i = n - 1 : -1 : 0$
    $[p_i, \pi_i] = \mathsf{TwoProd}(s_{i+1}, x)$
    $[s_i, \sigma_i] = \mathsf{TwoSum}(p_i, a_i)$
    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
```

Theorem 3 ([8])**.** *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 2 computes both i) the floating point evaluation $\mathsf{Horner}(p, x)$ and ii) two polynomials p_π and p_σ, of degree $n-1$, with floating point coefficients, such that $[\mathsf{Horner}(p, x), p_\pi, p_\sigma] = \mathsf{EFTHorner}(p, x)$. If no underflow occurs,*

$$p(x) = \mathsf{Horner}(p, x) + (p_\pi + p_\sigma)(x). \tag{4}$$

Moreover we have, using previously defined tilde notations,

$$(\widetilde{p_\pi + p_\sigma})(x) \leq \gamma_{2n}\widetilde{p}(x). \tag{5}$$

Relation (4) means that algorithm $\mathsf{EFTHorner}$ is an EFT for polynomial evaluation with the Horner algorithm.

3 A new EFT for polynomial evaluation

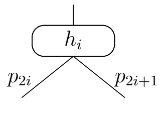
In the sequel of the paper, p_1 is a polynomial of degree n with floating point coefficients, and x is a floating point value. Given an integer $K \leq 2$, we now define a new EFT for polynomial evaluation. The principle of this EFT is to apply algorithm $\mathsf{EFTHorner}$ (Algorithm 2) recursively to $K-1$ levels.

3.1 Recursive application of $\mathsf{EFTHorner}$

Further developments will be easier to read introducing a graphical representation of one application of the $\mathsf{EFTHorner}$ transformation (Algorithm 2). Given p_i a polynomial of degree d with floating point coefficients and x a floating point number, we consider the floating point value h_i and the polynomials p_{2i} and p_{2i+1} of degree at most $d-1$ such that $[h_i, p_{2i}, p_{2i+1}] = \mathsf{EFTHorner}(p_i, x)$. From Theorem 3, we have $h_i = \mathsf{Horner}(p_i, x)$ and

$$p_i(x) = h_i + (p_{2i} + p_{2i+1})(x).$$

We represent this EFT of $p_i(x)$ with the following cell where edges are polynomials (one entry and two outputs) and the node is a floating point value.



Now we describe the principle of the $\mathsf{EFTHornerK}$ algorithm as the binary tree of depth K represented with Figure 2. For levels 1 to $K-1$, we recursively apply $\mathsf{EFTHorner}$. At the last level K this gives 2^{K-1} polynomials here represented as rectangles.

When $\mathsf{EFTHorner}$ is applied to a polynomial of degree d then the two generated polynomials are of degree $d-1$. Since p_1 is of degree n and $\mathsf{EFTHorner}$ is applied to $K-1$ levels, the polynomials computed on the level K are of degree at most $n-K+1$. In particular, if $n-K+1 = 0$ then the polynomials computed at the leaves of the binary tree are constants and so it is useless to apply again $\mathsf{EFTHorner}$. Therefore, to simplify the discussion we will always assume $2 \leq K \leq n+1$ in the sequel.

To easily identify the nodes in this binary tree, we define the following sets of indices.

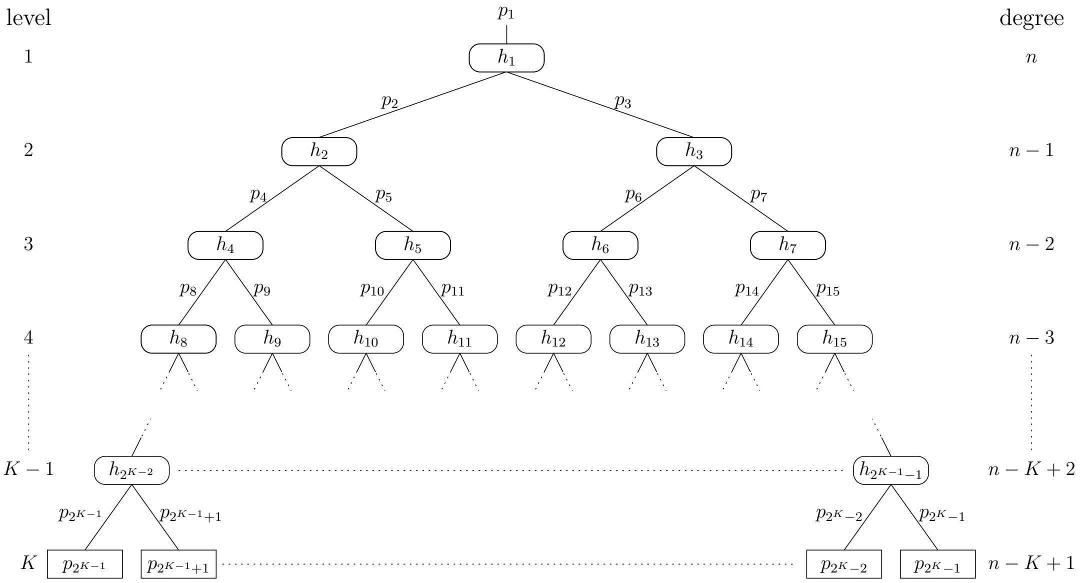


Figure 2: Representation of EFTHornerK as a binary tree.

- $\mathcal{N}_T^K = \{1, \ldots, 2^K - 1\}$ is the set of all the nodes in the tree, and $\mathrm{card}(\mathcal{N}_T^K) = 2^K - 1$;
- $\mathcal{N}_I^K = \{1, \ldots, 2^{K-1} - 1\}$ is the set of the internal nodes, and $\mathrm{card}(\mathcal{N}_I^K) = 2^{K-1} - 1$;
- $\mathcal{N}_L^K = \{2^{K-1}, \ldots, 2^K - 1\}$ is the set of the leaves, and $\mathrm{card}(\mathcal{N}_L^K) = 2^{K-1}$.

In particular we have $\mathcal{N}_T = \mathcal{N}_I \cup \mathcal{N}_L$ and $\mathcal{N}_I \cap \mathcal{N}_L = \emptyset$. We avoid exponent K in the set notations except when necessary. The recursive application of EFTHorner to $K-1$ levels is then defined by

$$[h_i, p_{2i}, p_{2i+1}] = \mathsf{EFTHorner}(p_i, x), \quad \text{for} \quad i \in \mathcal{N}_I, \tag{6}$$

with $h_i \in \mathbf{F}$ for $i \in \mathcal{N}_I$ and p_i being a polynomial with floating point coefficients for every $i \in \mathcal{N}_T$. According to Theorem 3, every h_i defined by the previous relation is the evaluation of the polynomial p_i at x by the Horner algorithm, *i.e.*,

$$h_i = \mathsf{Horner}(p_i, x), \quad \text{for} \quad i \in \mathcal{N}_I. \tag{7}$$

Since EFTHorner is an EFT for the Horner algorithm, Theorem 3 also yields

$$p_i(x) = h_i + (p_{2i} + p_{2i+1})(x), \quad \text{for} \quad i \in \mathcal{N}_I. \tag{8}$$

The floating point values $h_{i \in \mathcal{N}_I}$ and the polynomials $p_{i \in \mathcal{N}_L}$ are computed thanks to the next EFTHornerK algorithm.

Algorithm 4. Recursive application of EFTHorner to $K-1$ levels

```
function $[h_{i\in\mathcal{N}_I}, p_{i\in\mathcal{N}_L}] = \mathsf{EFTHornerK}(p_1, x)$
  for $i \in \mathcal{N}_I$,    $[h_i, p_{2i}, p_{2i+1}] = \mathsf{EFTHorner}(p_i, x)$
```

3.2 Numerical properties of EFTHornerK

First we prove that EFTHornerK (Algorithm 4) is actually an EFT for the evaluation of $p_1(x)$.

Theorem 5. *Given an integer K with $2 \leq K \leq n+1$, we consider the floating point numbers $h_{i\in\mathcal{N}_I}$ and the polynomials $p_{i\in\mathcal{N}_L}$, such that $[h_{i\in\mathcal{N}_I}, p_{i\in\mathcal{N}_L}] = \mathsf{EFTHornerK}(p_1, x)$ (Algorithm 4). The following relation holds,*

$$p_1(x) = \sum_{i\in\mathcal{N}_I} h_i + \sum_{i\in\mathcal{N}_L} p_i(x). \tag{9}$$

Algorithm $\mathsf{EFTHornerK}$ computes the evaluation $h_i = \mathsf{Horner}(p_i, x)$ of every polynomial p_i, for $i \in \mathcal{N}_I$. For the proof of Theorem 5, we also need to consider the evaluation of the polynomials $p_i(x)$, for $i \in \mathcal{N}_L$. So let us also denote $h_i = \mathsf{Horner}(p_i, x)$, for $i \in \mathcal{N}_L$.

Proof of Theorem 5. We proceed by induction on K. For $K = 2$, according to Theorem 3 we have $p_1(x) = \mathsf{Horner}(p_1, x) + (p_2 + p_3)(x) = h_1 + p_2(x) + p_3(x)$, and therefore $[h_1, p_2, p_3] = \mathsf{EFTHorner}(p_1, x)$. Now let us assume that relation (9) is satisfied for K such that $2 \leq K < n+1$. Then we consider the polynomials $p_{2^K}, \dots, p_{2^{K+1}-1}$ such that $[h_i, p_{2i}, p_{2i+1}] = \mathsf{EFTHorner}(p_i, x)$, for $i \in \mathcal{N}_L^K$. For $i \in \mathcal{N}_L^K$, Theorem 3 proves that $p_i(x) = h_i + (p_{2i} + p_{2i+1})(x)$. Thus,

$$\sum_{i\in\mathcal{N}_L^K} p_i(x) = \sum_{i\in\mathcal{N}_L^K} h_i + \sum_{i\in\mathcal{N}_L^K} (p_{2i} + p_{2i+1})(x) = \sum_{i\in\mathcal{N}_L^K} h_i + \sum_{i\in\mathcal{N}_L^{K+1}} p_i(x).$$

Reporting the last equality in Relation (9), we obtain

$$p_1(x) = \sum_{i\in\mathcal{N}_I^K} h_i + \sum_{i\in\mathcal{N}_L^K} h_i + \sum_{i\in\mathcal{N}_L^{K+1}} p_i(x) = \sum_{i\in\mathcal{N}_I^{K+1}} h_i + \sum_{i\in\mathcal{N}_L^{K+1}} p_i(x),$$

which concludes the proof of Theorem 5. ■

Proposition 6. *With the same hypothesis as in Theorem 5, the following relations hold,*

$$\left| p_1(x) - \sum_{i\in\mathcal{N}_T} h_i \right| = \left| \sum_{i\in\mathcal{N}_L} p_i(x) - h_i \right| \leq \gamma_{2(n-K+1)} \gamma_{4n}^{K-1} \widetilde{p_1}(x). \tag{10}$$

The error generated when approximating $p_1(x)$ by the exact sum $\sum_{i\in\mathcal{N}_T} h_i$ is therefore equal to the sum of the errors generated when approximating every $p_i(x)$ by $h_i = \mathsf{Horner}(p_i, x)$, for $i \in \mathcal{N}_L$. The previous proposition also provides an *a priori* bound on this error with respect to $\widetilde{p_1}(x) = \sum_{i=0} |a_i||x|^i$.

Proof. From Relation (9), since $\mathcal{N}_T = \mathcal{N}_I \cup \mathcal{N}_L$ and $\mathcal{N}_I \cap \mathcal{N}_L = \emptyset$ we have

$$p_1(x) - \sum_{i\in\mathcal{N}_T} h_i = \sum_{i\in\mathcal{N}_I} h_i + \sum_{i\in\mathcal{N}_L} p_i(x) - \sum_{i\in\mathcal{N}_I} h_i - \sum_{i\in\mathcal{N}_L} h_i = \sum_{i\in\mathcal{N}_L} p_i(x) - h_i.$$

Let us denote by e the left hand side in Inequality (10). Then we have $e \leq \sum_{i\in\mathcal{N}_L} |p_i(x) - h_i|$. The polynomials $p_{i\in\mathcal{N}_L}$ are of degree $n - K + 1$. Since $h_i = \mathsf{Horner}(p_i, x)$, we have

$$|p_i(x) - h_i| \leq \gamma_{2(n-K+1)} \widetilde{p_i}(x), \quad \text{for} \quad i \in \mathcal{N}_L.$$

Since $[h_i, p_{2i}, p_{2i+1}] = \mathsf{EFTHorner}(p_i, x)$, from Theorem 3 we have $\widetilde{p_{2i}}(x) + \widetilde{p_{2i+1}}(x) \leq \gamma_{2n} \widetilde{p_i}(x)$, thus $\widetilde{p_{2i}}(x) \leq \gamma_{2n} \widetilde{p_i}(x)$ and $\widetilde{p_{2i+1}}(x) \leq \gamma_{2n} \widetilde{p_i}(x)$. Then it can be proved by induction that $\widetilde{p_i}(x) \leq \gamma_{2n}^{K-1} \widetilde{p_1}(x)$, for $i \in \mathcal{N}_L$, and thus

$$|p_i(x) - h_i| \leq \gamma_{2(n-K+1)} \gamma_{2n}^{K-1} \widetilde{p_1}(x), \quad \text{for} \quad i \in \mathcal{N}_L.$$

Since $\mathrm{card}(\mathcal{N}_L) = 2^{K-1}$, we obtain $e \leq \gamma_{2(n-K+1)} \gamma_{2n}^{K-1} 2^{K-1} \widetilde{p_1}(x)$, and from $\gamma_{2n}^{K-1} 2^{K-1} \leq \gamma_{4n}^{K-1}$, Inequality (10) holds. ■

Our approach is motivated by Inequality (10). This inequality shows that when the parameter K is incremented by one, the distance between $p_1(x)$ and the exact sum $\sum_{i\in\mathcal{N}_T} h_i$ decreases by a factor γ_{4n}, that is roughly an accuracy improvement by a factor $4n\mathbf{u}$. In the next section we propose to compute this sum with the floating point summation algorithm SumK from [9]. We prove that the computed result is as accurate as the evaluation of $p_1(x)$ computed by the Horner algorithm in K times the working precision.

4 Algorithm CompHornerK

In this section, we formulate the algorithm CompHornerK (Algorithm 8), and then we provide an *a priori* error analysis for the compensated evaluation performed by this algorithm.

4.1 Principle of the algorithm

As previously, the floating point values $h_{i\in\mathcal{N}_T}$ are defined according to the relations $[h_{i\in\mathcal{N}_I}, p_{i\in\mathcal{N}_L}] = \mathsf{EFTHornerK}(p_1, x)$, and $h_i = \mathsf{Horner}(p_i, x)$ for $i \in \mathcal{N}_L$. Then Inequality (10) shows that

$$\left| p_1(x) - \sum_{i\in\mathcal{N}_T} h_i \right| \le (4n\mathbf{u})^K \widetilde{p_1}(x) + \mathcal{O}(\mathbf{u}^{K+1}). \tag{11}$$

The principle of the algorithm we propose here is to compute an approximate $\overline{r}$ of $\sum_{i\in\mathcal{N}_T} h_i$ in K times the working precision, so that

$$\frac{|\overline{r} - p_1(x)|}{|p_1(x)|} \le \big(\mathbf{u} + \mathcal{O}(\mathbf{u}^2)\big) + \big((4n\mathbf{u})^K + \mathcal{O}(\mathbf{u}^{K+1})\big)\,\mathrm{cond}(p_1, x). \tag{12}$$

In the previous inequality the factor $(4n\mathbf{u})^K + \mathcal{O}(\mathbf{u}^{K+1})$ reflects that the intermediate computation is as accurate as if performed in precision $\mathbf{u}^K$. The first term $\mathbf{u} + \mathcal{O}(\mathbf{u}^2)$ reflects the final rounding of the result to the working precision $\mathbf{u}$.

For the final summation, we use algorithm SumK proposed by Ogita, Rump and Oishi in [9]. This algorithm allows us to compute an approximate value of $\sum_{i\in\mathcal{N}_T} h_i$ with the same accuracy as if it was computed in K times the working precision. The following theorem summarizes the properties of algorithm SumK.

Theorem 7 (proposition 4.10 in [9])**.** *Given a vector $z = (z_1, \dots, z_n)$ of n floating point values, let us define $s = \sum_{i=1}^n z_i$ and $\widetilde{s} = \sum_{i=1}^n |z_i|$. We assume $4n\mathbf{u} < 1$ and we denote by $\overline{s}$ the floating point number such that $\overline{s} = \mathsf{SumK}(z, K)$. Then, even in presence of underflow,*

$$|\overline{s} - s| \le (\mathbf{u} + 3\gamma_{n-1}^2)|s| + \gamma_{2n-2}^K \widetilde{s}. \tag{13}$$

Now we formulate our compensated algorithm CompHornerK. We prove in the next subsection that it is as accurate as the Horner algorithm performed in K times the working precision.

Algorithm 8. Compensated Horner algorithm providing K times the working precision.

```
function $\overline{r} = \mathsf{CompHornerK}(p_1, x, K)$
  $[h_{i\in\mathcal{N}_I}, p_{i\in\mathcal{N}_L}] = \mathsf{EFTHornerK}(p_1, x)$
  for $i \in \mathcal{N}_L$,    $h_i = \mathsf{Horner}(p_i, x)$
  $\overline{r} = \mathsf{SumK}(h_{i\in\mathcal{N}_T}, K)$
```

In algorithm EFTHornerK, polynomials $p_{i\in\mathcal{N}_I}$ are of degree at most n. Applying EFTHorner to each of these polynomials requires $\mathcal{O}(n)$ floating point operations (flop). Since $\mathrm{card}(\mathcal{N}_I) = 2^{K-1} - 1$, the cost of EFTHornerK is therefore $\mathcal{O}\big(n2^K\big)$ flop. In CompHornerK, the evaluation of the 2^{K-1} polynomials $p_{i\in\mathcal{N}_L}$ also requires $\mathcal{O}(n2^K)$ flop. Finally, the computation of $\mathsf{SumK}\,(h_{i\in\mathcal{N}_T}, K)$ involves $(6K-5)(2^{K-1}-1) = \mathcal{O}(n2^K)$ flop. Overall, the cost of algorithm CompHornerK is therefore $\mathcal{O}\big(n2^K\big)$ flop. We will further see that this exponential complexity does not reduce the practical efficiency of the proposed algorithm for reasonable values of K, *e.g.*, while $K \leq 4$.

4.2 *A priori* error bound

In the following theorem we prove an *a priori* bound for the forward error in the compensated result computed by CompHornerK.

Theorem 9. *Let K be an integer such that $2 \leq K \leq n+1$. We assume $(2^K - 2)\gamma_{2n+1} \leq 1$. Then the forward error in the compensated evaluation of $p_1(x)$ with $\overline{r} = \mathsf{CompHornerK}(p_1, x)$ (Algorithm 8) is bounded as follows,*

$$|\overline{r} - p_1(x)| \leq \big(\mathbf{u} + 3\gamma_{2^K-2}^2 + \gamma_{2^{K+1}-4}^K\big)\,|p_1(x)| + \Big(\gamma_{4n}^K + \gamma_{2n+1}\gamma_{2^{K+1}-4}^K + \gamma_{4n}^{K+1}\Big)\,\widetilde{p_1}(x). \tag{14}$$

For the proof of Theorem 9, we use the following lemma to bound the absolute condition number for the final summation of the floating point numbers $h_{i\in\mathcal{N}_T}$.

Lemma 10. *With the notations of Algorithm 8, assuming $(2^K-2)\gamma_{2n+1} \leq 1$, we have $\sum_{i\in\mathcal{N}_T} |h_i| \leq |p_1(x)| + \gamma_{4n}\widetilde{p_1}(x)$.*

Proof. We decompose the sum as follows, $\sum_{i\in\mathcal{N}_T} |h_i| = |h_1| + \sum_{i\in\mathcal{N}_T-\{1\}} |h_i|$. Since $h_1 = \mathsf{Horner}(p_1, x)$, we have $|h_1| \leq |p_1(x)| + \gamma_{2n}\widetilde{p_1}(x)$. Moreover, for $i \in \mathcal{N}_T - \{1\}$ we also have $h_i = \mathsf{Horner}(p_i, x)$ with p_i of degree at most $n-1$ and $\widetilde{p_i}(x) \leq \gamma_{2n}\widetilde{p_1}(x)$, thus

$$|h_i| \leq |p_i(x)| + \gamma_{2(n-1)}\widetilde{p_i}(x) \leq (1 + \gamma_{2(n-1)})\widetilde{p_i}(x) \leq (1+\gamma_{2(n-1)})\gamma_{2n}\widetilde{p_1}(x) \leq \gamma_{2n+1}\widetilde{p_1}(x).$$

Therefore $\sum_{i\in\mathcal{N}_T} |h_i| \leq |p_1(x)| + \gamma_{2n}(1+(2^K-2)\gamma_{2n+1})\widetilde{p_1}(x)$. By assumption $(2^K-2)\gamma_{2n+1} \leq 1$, so that $\gamma_{2n}(1+(2^K-2)\gamma_{2n+1}) \leq 2\gamma_{2n} \leq \gamma_{4n}$, which proves the lemma. ■

Proof of Theorem 9. Defining the terms $e_1 := \Big|p_1(x) - \sum_{i\in\mathcal{N}_T} h_i\Big|$ and $e_2 := \Big|\sum_{i\in\mathcal{N}_T} h_i - \overline{r}\Big|$, we have $|\overline{r} - p_1(x)| \leq e_1 + e_2$. According to Proposition 6 it follows that $e_1 \leq \gamma_{2n}\gamma_{4n}^{K-1}\widetilde{p_1}(x)$. The second term e_2 denotes the error occurring in the final summation with algorithm SumK. Using the error bound (13), we deduce $e_2 \leq (\mathbf{u} + 3\gamma_{2^K-2}^2)|s| + \gamma_{2^{K+1}-4}^K\widetilde{s}$, with $s = \sum_{i\in\mathcal{N}_T} h_i$ and $\widetilde{s} = \sum_{i\in\mathcal{N}_T} |h_i|$. Using Theorem 5 and Proposition 6, we have $|s| \leq |p_1(x)| + \gamma_{2n}\gamma_{4n}^{K-1}\widetilde{p_1}(x)$. On the other hand $\widetilde{s}$ is bounded according to Lemma 10. Thus we write

$$\begin{aligned} e_2 &\leq (\mathbf{u} + 3\gamma_{2^K-2}^2)\Big(|p_1(x)| + \gamma_{2n}\gamma_{4n}^{K-1}\widetilde{p_1}(x)\Big) + \gamma_{2^{K+1}-4}^K\left(|p_1(x)| + \gamma_{4n}\widetilde{p_1}(x)\right) \\ &\leq \big(\mathbf{u} + 3\gamma_{2^K-2}^2 + \gamma_{2^{K+1}-4}^K\big)\,|p_1(x)| + \Big(\gamma_{4n}\gamma_{2^{K+1}-4}^K + \gamma_{4n}^{K+1}\Big)\,\widetilde{p_1}(x). \end{aligned}$$

Therefore we have the inequality

$$|\overline{r} - p_1(x)| \leq \big(\mathbf{u} + 3\gamma_{2^K-2}^2 + \gamma_{2^{K+1}-4}^K\big)\,|p_1(x)| + \Big(\gamma_{2n}\gamma_{4n}^{K-1} + \gamma_{4n}\gamma_{2^{K+1}-4}^K + \gamma_{4n}^{K+1}\Big)\,\widetilde{p_1}(x),$$

which proves Theorem 9. ■

5 Numerical behavior of CompHornerK

To exhibit the numerical behavior of CompHornerK (Algorithm 8) with respect to the condition number we have generated a set of 700 polynomials with condition number ranging from 10^2 to 10^{100}. The coefficients of these polynomials are exact in IEEE-754 double precision we used as the working precision in these experiments. The method for generating these extremely ill-conditioned polynomials is the same as in [8]. We report with Figure 3 the relative accuracy of every polynomial evaluation performed with CompHornerK, with respect to the condition number $\text{cond}(p, x)$ and for successive iterates K in $2, \dots, 6$. We also represent on this figure the *a priori* relative error bound (12).

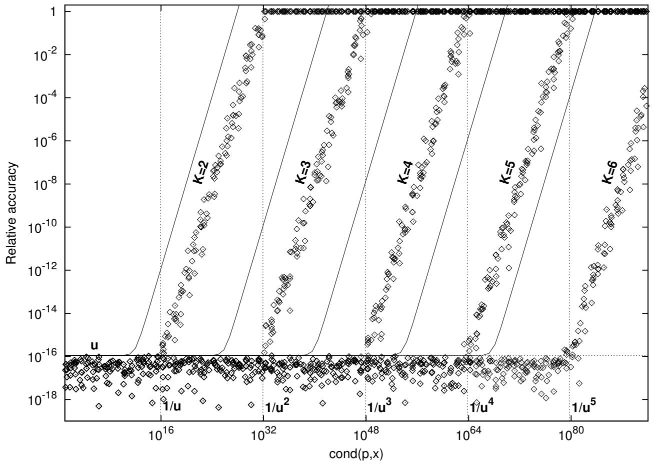


Figure 3: Relative accuracy of the compensated evaluation performed with CompHornerK (Algorithm 8) with respect to $\text{cond}(p, x)$, for $K = 2, \dots, 6$.

As expected from the results presented in Section 4, algorithm CompHornerK is in practice as accurate as the Horner algorithm performed in K times the working precision with a final rounding to the working precision. For every considered value of K the relative accuracy of the compensated evaluation is of the order of the working precision $\mathbf{u}$ as long as $\text{cond}(p, x)$ is smaller than $\mathbf{u}^{-K}$. When $K = 2$ we also notice that CompHornerK exhibits the same numerical behavior as the compensated algorithm presented in [8].

We also observe that the *a priori* bound (12) of the relative error in the computed evaluation is always pessimistic compared to the actual (measured) error by many orders of magnitude. Moreover this error bound is more and more pessimistic when the parameter K increases — this phenomenon is also observed in [9] for the compensated dot product algorithm DotK.

6 Time Performances

Let us first emphasize that the running time of the considered algorithms do not depend on the coefficients of the polynomial, nor on the argument x, but only on the degree n. We use the following experimental environments.

(I) Intel Pentium 4, 3.0GHz, GNU Compiler Collection 4.1.2, fpu x87;
(II) AMD Athlon 64, 2 GHz, GNU Compiler Collection 4.1.2, fpu sse;
(III) Itanium 2, 1.5 GHz, GNU Compiler Collection 4.1.1;
(IV) Itanium 2, 1.5 GHz, Intel C Compiler 9.1.

In the first part of our experiments, we study the performances of algorithm CompHornerK (Algorithm 8) assuming that K is an argument of the implemented routine. Since we use IEEE-754 double precision as working precision, CompHornerK simulates a precision of the order of $K \times 53$ bits. We compare CompHornerK to the Horner algorithm implemented with the MPFR library[2] [2] using a precision of $K \times 53$ bits; we denote by MPFRHornerK this implementation. For our measures we use a set of 39 random polynomials of degree varying from 10 to 200, by step of 5. For every considered degree n we measure the overhead introduced by the algorithms CompHornerK and MPFRHornerK compared to the classic Horner algorithm (we measure the ratio of the running time of CompHornerK over the running time of Horner, and we perform the same measurement for MPFRHornerK). We report the average overheads for both algorithms with respect to K on the left side of Figure 4.

CompHornerK is clearly not competitive compared to MPFRHornerK for large values of K. Nevertheless, in our experiments CompHornerK runs always faster than MPFRHornerK while K is smaller than 4. This illustrates the practical interest of CompHorner for simulating a small improvement of the working precision.

Next we study an optimized version of CompHornerK *a priori* setting a value for K. We name CompHorner4 the corresponding implementation for $K = 4$. Setting the parameter K to a particular value does not change the principle of the algorithm but allows the compiler to perform more optimizations and to provide better practical performances. In these experiments, we compare CompHorner4 to the Horner algorithm implemented with the quad-double library[3] that also simulates 4 times the IEEE-754 precision [3]. We denote by QDHorner the Horner algorithm implemented with quad-double arithmetic. For a fair comparison, our implementation of QDHorner inlines the quad-double arithmetic described in [3] and is also compiled with the same optimizing option as CompHorner4. We also compare CompHorner4 to MPFRHorner4 using the MPFR library with a working precision of 212 bits.

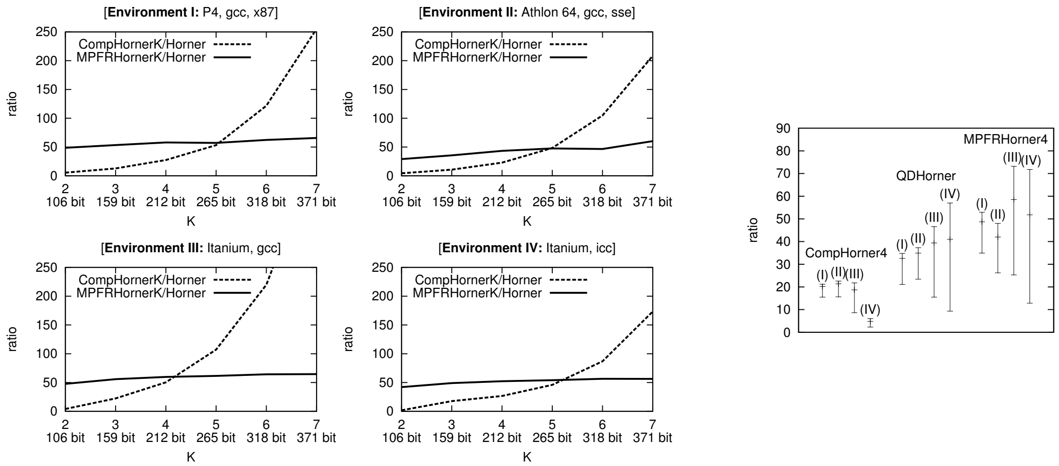


Figure 4: Average measured overheads for CompHornerK and MPFRHornerK (left) and measured overheads for CompHorner4, QDHorner and MPFRHorner4 (right).

As in the previous experiments, we use a set of 39 random polynomials of degree varying from 10 to 200 by step of 5. For every polynomial, we measure the overhead of CompHorner4 compared to the classic Horner algorithm. For every environment listed above we report the minimum, the average and the maximum values of this overhead on the right side of Figure 4. We also

[2] The MPFR library is available at `http://www.mpfr.org/`. We use version 2.2.1 in our experiments.

[3] The quad-double library is available at `http://crd.lbl.gov/~dhbailey/mpdist/`.

report the same average overheads for QDHorner and MPFRHorner4.

Our CompHorner4 is always significantly faster than both QDHorner and MPFRHorner. In particular CompHorner4 runs about 8 times faster than QDHorner in the environment (IV) which is the Itanium architecture with the Intel compiler.

References

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

[2] G. Hanrot, V. Lefèvre, P. Pélissier, and P. Zimmermann. The MPFR library. Available at `http://www.mpfr.org/`.

[3] Y. Hida, X. S. Li, and D. H. Bailey. Algorithms for quad-double precision floating point arithmetic. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 155–162, 2001.

[4] N. J. Higham. *Accuracy and stability of numerical algorithms.* Society for Industrial and Applied Mathematics (SIAM), Philadelphia, PA, second edition, 2002.

[5] C. M. Hoffmann, G. Park, J.-R. Simard, and N. F. Stewart. Residual iteration and accurate polynomial evaluation for shape-interrogation applications. In *Proceedings of the 9th ACM Symposium on Solid Modeling and Applications*, pages 9–14, 2004.

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

[7] D. E. Knuth. *The Art of Computer Programming, Volume 2, Seminumerical Algorithms.* Addison-Wesley, Reading, MA, USA, third edition, 1998.

[8] P. Langlois and N. Louvet. How to ensure a faithful polynomial evaluation with the compensated Horner algorithm. In *Proceedings of the 18th IEEE Symposium on Computer Arithmetic*, pages 141–149, 2007.

[9] T. Ogita, S. M. Rump, and S. Oishi. Accurate sum and dot product. *SIAM Journal on Scientific Computing*, 26(6):1955–1988, 2005.