A Compensated Estrin Scheme for Accurate Polynomial Evaluation

Guangping Yu[1], Stef Graillat[2(✉)], Hao Jiang[1], Chun Huang[1], and Tao Tang[1]

[1] College of Computer Science and Technology, National University of Defense Technology, Changsha 410073, China
{yuguangping,haojiang,chunhuang,taotang84}@nudt.edu.cn
[2] Sorbonne Université, CNRS, LIP6, 75005 Paris, France
stef.graillat@lip6.fr

Abstract. Polynomial evaluation is ubiquitous in scientific computing, yet simultaneously achieving high performance and numerical accuracy remains challenging. Traditional methods such as Horner's scheme are efficient but inherently sequential, while the Estrin scheme offers a parallel-friendly evaluation tree structure but does not inherently improve accuracy. This paper introduces a compensated Estrin scheme that combines the low-depth evaluation tree of the Estrin method with error compensation techniques based on error-free transformations. The proposed algorithm dynamically tracks and compensates for rounding errors at each stage of the evaluation, delivering accuracy close to double-double precision arithmetic while using only binary64 error-free transformations. A rigorous rounding error analysis proves that the relative forward error is bounded by $u + \mathrm{cond}(p, x) \cdot O(u^2)$, where u is the unit roundoff and $\mathrm{cond}(p, x)$ is the condition number of the polynomial evaluation. Numerical experiments on ill-conditioned polynomials demonstrate that the compensated Estrin scheme maintains high accuracy over a much larger range than the standard Estrin scheme, up to the regime where the condition number is too large for any binary64-output twice-working-precision-like method to remain accurate, providing a practical and efficient solution for accurate polynomial evaluation on modern architectures.

Keywords: Polynomial evaluation · Estrin scheme · Error compensation · Numerical accuracy

1 Introduction

Polynomial evaluation arises in interpolation, root-finding, finite element methods, elementary-function implementation, polynomial composition, and certified floating-point code generation [21–23,28,30]. Its accuracy is usually analyzed with condition numbers and rounding-error models [15,25]. Horner's scheme [13] minimizes arithmetic operations but is sequential. Estrin's scheme [4,6,29] reduces the dependency depth to $O(\log n)$, making it attractive on vector,

F. Boulier et al. (Eds.): CASC 2026, LNCS 16844, pp. 416–435, 2027.
https://doi.org/10.1007/978-3-032-34586-8_21

SIMD, superscalar, and parallel architectures [7], but it does not by itself improve numerical accuracy. Accurate and validated polynomial and rational-function evaluation has therefore been studied through compensated and related approaches [9,12]. Compensated algorithms, which use error-free transformations to track rounding errors and apply compensation, have been shown to deliver high accuracy without resorting to higher precision arithmetic. Examples include compensated Horner schemes [10,13], accurate summation and dot-product algorithms [27,32], accurate products and exponentiation [8], and faithfully rounded computations [20].

The compensated Estrin case is less direct than the compensated Horner case. Horner's scheme has a single sequential recurrence, so the local residual at one step is propagated through one following multiply-add chain. In contrast, Estrin's scheme has a tree structure: residuals produced at different nodes must be merged across levels, and the powers used by the tree are obtained by successive squarings whose own residuals also affect later levels. The algorithm below is therefore not just a reparenthesized compensated Horner scheme; it gives an explicit recurrence for the local products, sums, and squaring residuals in the Estrin tree.

The intended interface remains binary64 input and binary64 output; only the internal residual tracking is changed. This distinguishes the goal from general faithful-rounding frameworks such as the expression-tree approach of Lange and Rump [20], whose pair-arithmetic results apply under the no-inaccurate-cancellation framework. Related compensated summation, dot product, and MPI reduction implementations also show that error-free transformations can be effective on modern hardware [14,16].

We introduce a compensated Estrin scheme that combines the evaluation tree structure of the Estrin method with error compensation techniques. The main contributions are an Estrin-specific compensated recurrence, an operation count and condition-number forward-error bound of the form $u + \mathrm{cond}(p, x)O(u^2)$, a matching error analysis for the double-double Estrin baseline, and experiments comparing standard Estrin, Horner, compensated Horner, compensated Estrin, double-double Estrin, and binary128 Estrin evaluations.

2 Preliminaries

This section fixes the notation used in the error analyses.

2.1 Floating-Point Arithmetic

We assume floating-point arithmetic that conforms to the IEEE 754 standard [17], with round-to-nearest ties-to-even. The rounding-error analysis is carried out in the standard normal-range model: equivalently, the exponent range is treated as unbounded, or all exact intermediate results and rounded results considered in the analysis remain normal, with no overflow, underflow, or subnormal result. The set of normal floating-point numbers in the working precision is

Algorithm 1. TwoSum

```
function TwoSum(a, b)
    x ← a ⊕ b
    z ← x ⊖ a
    y ← (a ⊖ (x ⊖ z)) ⊕ (b ⊖ z)
    return (x, y)
end function
```

denoted by $\mathbb{F}$ and the unit roundoff by u. For $a, b \in \mathbb{F}$ and any basic operation $\circ \in \{+, -, \times\}$ whose exact nonzero result remains in this range, the floating-point result satisfies

$$\mathrm{fl}(a \circ b) = (a \circ b)(1 + \delta), \quad |\delta| \leq u. \tag{1}$$

This implies

$$|a \circ b - \mathrm{fl}(a \circ b)| \leq u\,|a \circ b|. \tag{2}$$

For bounding the accumulation of rounding errors, we use the standard quantity

$$\gamma_n = \frac{nu}{1 - nu}, \tag{3}$$

for $nu < 1$, see [15].

2.2 Error-Free Transformations and γ_n-Notation

Error-free transformations (EFTs) decompose a floating-point operation into a rounded result plus an exact compensating term. We use standard EFTs such as `TwoSum` and `TwoProduct` [5,19,25,26].

Lemma 1 (`TwoSum` and `FastTwoSum` [19]). *Given $a, b \in \mathbb{F}$, the function* `TwoSum` *returns $x, y \in \mathbb{F}$ such that*

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

`TwoSum` *uses 6 flops and has $x = \mathrm{fl}(a+b)$;* `FastTwoSum` *uses 3 flops under the assumption $|a| \geq |b|$.*

Lemma 2 (TwoProduct [5]). *Given $a, b \in \mathbb{F}$, the function* `TwoProduct` *returns $x, y \in \mathbb{F}$ such that*

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

The equality is exact, with $x = \mathrm{fl}(a \cdot b)$. The standard Dekker implementation uses the splitting technique of [5]; it costs 17 FLOPs in the operation-count model used below.

Algorithm 2. TwoProduct (Dekker Version)

function TwoProduct(a, b)
    $x \leftarrow a \otimes b$
    split a into $a_h + a_l$ and b into $b_h + b_l$
    $y \leftarrow ((a_h \otimes b_h \ominus x) \oplus a_h \otimes b_l \oplus a_l \otimes b_h) \oplus a_l \otimes b_l$
    return (x, y)
end function

Total: 17 flops.

Algorithm 3. TwoProduct (FMA-Based Version)

function TwoProdFMA(a, b)
    $x \leftarrow a \otimes b$
    $y \leftarrow \mathrm{FMA}(a, b, -x)$
    return (x, y)
end function

Total: 2 flops.

Lemma 3 (FMA Properties [2,3,26]). *The Fused-Multiply-and-Add (FMA) operator computes* $FMA(a, b, c) = \mathrm{fl}(a \cdot b + c)$. *Assuming no underflow/overflow occurs, it satisfies:*

$$\begin{aligned} FMA(a, b, c) &= (a \cdot b + c)(1 + \delta_1) \\ &= \frac{a \cdot b + c}{1 + \delta_2}, \quad |\delta_\nu| \le u. \end{aligned} \tag{4}$$

This operator is specified in IEEE floating-point arithmetic [17] and enables more efficient error-free transformations.

The following lemma fixes the standard γ_n notation for products of rounding factors, following Stewart [31].

Lemma 4. *Let* $|\delta_i| \le u$ *for* $i = 1, \ldots, n$ *and* $nu < 1$. *Then*

$$\prod_{i=1}^{n}(1 + \delta_i) = 1 + \theta_n, \quad |\theta_n| \le \gamma_n, \tag{5}$$

where γ_n *is defined in* (3). *Moreover,*

$$(1 + \gamma_i)(1 + \gamma_j) \le 1 + \gamma_{i+j}, \quad (1 + u)\,\gamma_i \le \gamma_{i+1}. \tag{6}$$

Throughout the paper, each occurrence of the symbol $\langle k \rangle$ denotes some product of k factors of the form $(1 + \delta_i)$ with possibly different local rounding errors satisfying $|\delta_i| \le u$. Thus each occurrence satisfies $\langle k \rangle = 1 + \theta_k$ with $|\theta_k|$ bounded by γ_k. When two such placeholders are multiplied, we write the result as a new occurrence of $\langle i + j \rangle$, meaning only that the product contains $i + j$ rounding factors and therefore has the corresponding γ_{i+j} bound; this is a bounding convention, not an identity between fixed products of preselected rounding errors.

These tools will be used repeatedly to bound the accumulation and propagation of rounding errors in the Estrin and compensated Estrin schemes.

Algorithm 4. Estrin Algorithm

```
function ESTRIN(C, x)
    n ← length(C) − 1
    L ← ⌊log2(n)⌋ + 1
    y ← x
    for j = 1 to L do
        m ← ⌊n/2⌋
        for i = 0 to m do
            if 2i + 1 ≤ n then
                D_i ← C_{2i} + C_{2i+1} · y
            else
                D_i ← C_{2i}
            end if
        end for
        C ← D,    n ← length(D) − 1
        if n = 0 then
            break
        end if
        y ← y · y
    end for
    return C_0
end function
```

3 Estrin Scheme

The Estrin evaluation scheme restructures a polynomial to reduce the dependency depth and obtain a low-depth evaluation tree.

3.1 Estrin Algorithm

Consider a degree-n polynomial

$$p(x) = \sum_{i=0}^{n} C_i \, x^i. \tag{7}$$

Estrin scheme groups terms at powers of two as follows. The corresponding floating-point evaluation algorithm is summarized in Algorithm 4 [6].

For example, for $p(x) = C_0 + C_1x + C_2x^2 + C_3x^3$, Horner's scheme evaluates

$$(((C_3x + C_2)x + C_1)x + C_0),$$

which is a single dependency chain. Estrin's scheme instead evaluates

$$(C_0 + C_1x) + (C_2 + C_3x)x^2,$$

so that the two linear factors can be evaluated independently before the final merge. This illustrates the lower-depth tree structure used below.

Estrin scheme reduces the evaluation depth from $O(n)$ to $O(\log n)$ at the cost of a modest increase in arithmetic operations.

3.2 Error Analysis of Estrin Scheme

To derive a rounding error bound for the Estrin scheme, we use the "error counter" notation introduced in Sect. 2.2. Namely, $\langle k \rangle$ denotes a product of k rounding factors and satisfies $\langle k \rangle = 1+\theta_k$ with $|\theta_k| \leq \gamma_k$. The following theorem records the forward-error bound used as the baseline for the new compensated analysis.

We also recall the usual condition number for polynomial evaluation, see [15]:

$$\mathrm{cond}(p, x) = \frac{\sum_{i=0}^{n} |C_i||x|^i}{|p(x)|}. \tag{8}$$

Theorem 1. *Let $p(x) = \sum_{i=0}^{n} C_i x^i$ be a polynomial of degree n with coefficients $C_i \in \mathbb{F}$. For any $x \in \mathbb{F}$, let $\widehat{p}(x)$ denote the result obtained by applying the Estrin scheme to p at x in floating-point arithmetic. The relative forward error bound satisfies*

$$\frac{|p(x) - \widehat{p}(x)|}{|p(x)|} \leq \gamma_{\lfloor \log_2(n) \rfloor + n + 1}\ \mathrm{cond}(p, x), \tag{9}$$

where $\mathrm{cond}(p, x)$ *is given in Eq.* (8).

Proof. The Estrin scheme evaluates a polynomial $p(x) = C_0 + C_1 x + C_2 x^2 + \cdots + C_n x^n$ by decomposing it into subexpressions. For a degree-three polynomial, using relative error counters, the computation can be written as

$$\begin{aligned} \widehat{p}_3(x) &= [(C_0 + C_1 x\langle 1 \rangle)\langle 1 \rangle + (C_2 + C_3 x\langle 1 \rangle)\langle 1 \rangle x^2 \langle 2 \rangle]\langle 1 \rangle \\ &= C_0\langle 2 \rangle + C_1 x\langle 3 \rangle + C_2 x^2\langle 4 \rangle + C_3 x^3\langle 5 \rangle. \end{aligned} \tag{10}$$

Similarly, we derive

$$\begin{cases} \widehat{p}_4(x) = C_0\langle 3 \rangle + C_1 x\langle 4 \rangle + C_2 x^2\langle 5 \rangle + C_3 x^3\langle 6 \rangle \\ \qquad\qquad + C_4 x^4\langle 5 \rangle, \\ \widehat{p}_5(x) = C_0\langle 3 \rangle + C_1 x\langle 4 \rangle + C_2 x^2\langle 5 \rangle + C_3 x^3\langle 6 \rangle \\ \qquad\qquad + C_4 x^4\langle 6 \rangle + C_5 x^5\langle 7 \rangle, \\ \widehat{p}_6(x) = C_0\langle 3 \rangle + C_1 x\langle 4 \rangle + C_2 x^2\langle 5 \rangle + C_3 x^3\langle 6 \rangle \\ \qquad\qquad + C_4 x^4\langle 7 \rangle + C_5 x^5\langle 8 \rangle + C_6 x^6\langle 9 \rangle. \end{cases} \tag{11}$$

By induction on the Estrin tree depth, one can show that for a degree-n polynomial, the term x^i is multiplied by an error counter $\langle k_i \rangle$ where $k_i \leq \lfloor \log_2(n) \rfloor + i + 1$. Therefore, the largest error counter occurs for x^n and is bounded by $\lfloor \log_2(n) \rfloor + n + 1$.

Thus we obtain:

$$\begin{aligned} \widehat{p}_n(x) = C_0\langle \lfloor \log_2(n) \rfloor + 1 \rangle + C_1 x\langle \lfloor \log_2(n) \rfloor + 2 \rangle \\ + \cdots + C_n x^n \langle \lfloor \log_2(n) \rfloor + n + 1 \rangle. \end{aligned} \tag{12}$$

By Lemma 4, the error terms satisfy $|\theta_k| \leq \gamma_k$. Hence the forward error bound for the Estrin scheme is

$$|p(x) - \widehat{p}(x)| \leq \gamma_{\lfloor \log_2(n) \rfloor + n + 1} \sum_{i=0}^{n} |C_i||x|^i. \tag{13}$$

Define the auxiliary polynomial $\tilde{p}(x) = \sum_{i=0}^{n} |C_i| x^i$. Then the right-hand side equals $\gamma_{\lfloor \log_2(n) \rfloor + n + 1} \tilde{p}(|x|)$. The relative error bound becomes

$$\frac{|p(x) - \widehat{p}(x)|}{|p(x)|} \leq \gamma_{\lfloor \log_2(n) \rfloor + n + 1} \cdot \frac{\tilde{p}(|x|)}{|p(x)|}. \tag{14}$$

The condition number $\mathrm{cond}(p, x)$ is defined as

$$\mathrm{cond}(p, x) = \frac{\tilde{p}(|x|)}{|p(x)|} = \frac{\sum_{i=0}^{n} |C_i||x|^i}{\left|\sum_{i=0}^{n} C_i x^i\right|}. \tag{15}$$

Substituting this definition yields the inequality stated in the theorem.

4 Compensated Estrin Scheme

In finite-precision arithmetic, the accuracy of numerical algorithms is fundamentally constrained by rounding errors in floating-point operations. This section derives Algorithm 5 and gives the new compensation-error and forward-error analysis for it.

4.1 Error Decomposition for Compensation

At the first level we set $\widehat{y}_1 = x$ and $\rho_1 = 0$. For $j \geq 1$, $\widehat{y}_j$ denotes the floating-point approximation to $x^{2^{j-1}}$, and ρ_j denotes the accumulated first-order residual of this power. Thus, up to terms of second order in u,

$$x^{2^{j-1}} = \widehat{y}_j + \rho_j + O(u^2 |x|^{2^{j-1}}). \tag{16}$$

The residual ρ_j is not only the last local `TwoProduct` residual. It also contains the first-order effect of previous squarings. Specifically, if

$$[\widehat{y}_{j+1}, \pi_{j+1}] = \texttt{TwoProduct}(\widehat{y}_j, \widehat{y}_j),$$

then

$$\rho_{j+1} = \pi_{j+1} + 2\widehat{y}_j \rho_j + O(u^2 |x|^{2^j}). \tag{17}$$

The algorithm evaluates the right-hand side of (17) in floating-point arithmetic; since π_{j+1} and ρ_j are already first-order quantities, the rounding made in this update contributes only to the second-order compensation error.

Let $C_i^{(j),*}$ be the exact value of the i-th Estrin subtree after j levels, with $C_i^{(0),*} = C_i$. At level j the exact merge is

$$C_i^{(j),*} = C_{2i}^{(j-1),*} + C_{2i+1}^{(j-1),*} x^{2^{j-1}}. \tag{18}$$

In the algorithm we have the floating-point approximations $\widehat{C}_{2i}^{(j-1)}, \widehat{C}_{2i+1}^{(j-1)}$, and $\widehat{y}_j$. Applying the two error-free transformations yields

$$[s_i, \alpha_i^{(j)}] = \texttt{TwoProduct}(\widehat{C}_{2i+1}^{(j-1)}, \widehat{y}_j), \tag{19}$$

$$[b_i, \beta_i^{(j)}] = \texttt{TwoSum}(\widehat{C}_{2i}^{(j-1)}, s_i). \tag{20}$$

The computed floating-point result is $\widehat{C}_i^{(j)} = b_i$. From the defining properties of EFT we obtain the *exact* representation

$$\widehat{C}_i^{(j)} = \widehat{C}_{2i}^{(j-1)} + \widehat{C}_{2i+1}^{(j-1)} \widehat{y}_j - \left(\alpha_i^{(j)} + \beta_i^{(j)}\right). \tag{21}$$

4.2 Compensated Estrin Algorithm

Define the first-order subtree error by

$$C_i^{(j),*} = \widehat{C}_i^{(j)} + \epsilon C_i^{(j)} + O(u^2 W_i^{(j)}), \tag{22}$$

where $W_i^{(j)}$ is the local absolute weight of the subtree. Using (16), the first-order exact merge at level j is

$$C_i^{(j),*} = \widehat{C}_{2i}^{(j-1)} + \widehat{C}_{2i+1}^{(j-1)} \widehat{y}_j + \epsilon C_{2i}^{(j-1)} + \epsilon C_{2i+1}^{(j-1)} \widehat{y}_j + \widehat{C}_{2i+1}^{(j-1)} \rho_j + O(u^2 W_i^{(j)}). \tag{23}$$

The omitted product $\epsilon C_{2i+1}^{(j-1)} \rho_j$ is second order. Combining (23) with (21) shows that the first-order local residual is

$$\epsilon D_i^{(j)} = \alpha_i^{(j)} + \beta_i^{(j)} + \widehat{C}_{2i+1}^{(j-1)} \rho_j. \tag{24}$$

This separates the local product and sum residuals $\alpha_i^{(j)} + \beta_i^{(j)}$ from the accumulated power residual ρ_j.

Define the first-order cumulative error at level j as $\epsilon C_i^{(j)}$. At the final level,

$$p(x) - \widehat{p}(x) = \epsilon C_0^{(L)} + O(u^2 \sum_i |C_i||x|^i).$$

Therefore evaluating and adding the compensation term removes the first-order part of the Estrin rounding error. The error satisfies the recurrence:

$$\epsilon C_i^{(j)} = \epsilon D_i^{(j)} + \epsilon C_{2i}^{(j-1)} + \epsilon C_{2i+1}^{(j-1)} \cdot \widehat{y}_j, \tag{25}$$

with local error $\epsilon D_i^{(j)} = \beta_i^{(j)} + \alpha_i^{(j)} + \widehat{C}_{2i+1}^{(j-1)} \cdot \rho_j$ and initial condition $\epsilon C_i^{(0)} = 0$. This recurrence enables iterative computation of the total error $\epsilon C_0^{(L)}$.

The compensated algorithm tracks both the approximate coefficients $\widehat{C}_i^{(j)}$ and the accumulated errors $\widehat{\epsilon C}_i^{(j)}$:

Algorithm 5. Compensated Estrin Algorithm

function CompEstrin(C, x)
  $d \leftarrow \text{length}(C) - 1$
  $L \leftarrow \lfloor \log_2(d) \rfloor + 1$
  $\widehat{C}^{(0)} \leftarrow C$
  $\widehat{y}_1 \leftarrow x$
  $\rho_1 \leftarrow 0$
  $\widehat{\epsilon C}^{(0)} \leftarrow 0$
  for $j = 1$ **to** L **do**
    $m \leftarrow \lfloor d/2 \rfloor$
    for $i = 0$ **to** m **do**
      if $2i + 1 \leq d$ **then**
        $[s_i, \alpha_i^{(j)}] \leftarrow \texttt{TwoProduct}(\widehat{C}_{2i+1}^{(j-1)}, \widehat{y}_j)$
        $[b_i, \beta_i^{(j)}] \leftarrow \texttt{TwoSum}(\widehat{C}_{2i}^{(j-1)}, s_i)$
        $\widehat{C}_i^{(j)} \leftarrow b_i$
        $\widehat{\epsilon D}_i^{(j)} \leftarrow (\beta_i^{(j)} \oplus \alpha_i^{(j)}) \oplus (\widehat{C}_{2i+1}^{(j-1)} \otimes \rho_j)$
      else
        $\widehat{C}_i^{(j)} \leftarrow \widehat{C}_{2i}^{(j-1)}$
        $\widehat{\epsilon D}_i^{(j)} \leftarrow 0$
        $\widehat{\epsilon C}_i^{(j)} \leftarrow \widehat{\epsilon C}_{2i}^{(j-1)}$
      end if
      if $2i + 1 \leq d$ **then**
        $\widehat{\epsilon C}_i^{(j)} \leftarrow \widehat{\epsilon D}_i^{(j)} \oplus \widehat{\epsilon C}_{2i}^{(j-1)} \oplus (\widehat{\epsilon C}_{2i+1}^{(j-1)} \otimes \widehat{y}_j)$
      end if
    end for
    $d \leftarrow m$
    if $j < L$ **then**
      $[\widehat{y}_{j+1}, \pi_{j+1}] \leftarrow \texttt{TwoProduct}(\widehat{y}_j, \widehat{y}_j)$
      $\rho_{j+1} \leftarrow \pi_{j+1} \oplus ((2 \otimes \widehat{y}_j) \otimes \rho_j)$
    end if
  end for
  return $\widehat{C}_0^{(L)} \oplus \widehat{\epsilon C}_0^{(L)}$
end function

With the standard 17-FLOP `TwoProduct`, the algorithm requires $19\lfloor \log_2(n) \rfloor + 29n + 1$ floating-point operations: 29 FLOPs for each of the n paired Estrin merges, 19 FLOPs for each compensated power update, and one final addition. The ρ_j update is essential: it carries the first-order residuals from the repeated squaring of the powers used by the Estrin tree. Without this term, the algorithm would be only a local compensation of the merge operations and would miss one source of first-order error.

4.3 Rigorous Error Analysis

The analysis separates the first-order residual generated by the Estrin tree from the rounding error incurred when that residual is evaluated. The recurrence above represents the exact first-order term $\epsilon C_0^{(L)}$, including the residuals from paired Estrin merges and from repeated squaring of the powers, and the algorithm computes its floating-point approximation $\widehat{\epsilon C}_0^{(L)}$. Theorem 2 bounds this latter evaluation error, and Theorem 3 gives the resulting forward-error bound for `CompEstrin`.

Theorem 2 (Compensation Error Bound). *Let $\widehat{\epsilon C}_0^{(L)}$ denote the computed compensation term and $\epsilon\, C_0^{(L)}$ the exact first-order error term, where L is equal to $\lfloor \log_2(n) \rfloor + 1$. Put*

$$K_n = \lfloor \log_2(n) \rfloor + n + 1, \quad M_n = 2\lfloor \log_2 \lfloor n/2 \rfloor \rfloor + 6.$$

Assuming $n \geq 2$, $K_n u < 1$, $M_n u < 1$, and no underflow or overflow occurs, we have:

$$|\epsilon C_0^{(L)} - \widehat{\epsilon C}_0^{(L)}| \leq 2\gamma_{M_n}\gamma_{K_n} \sum_{i=0}^{n} |C_i||x|^i, \tag{26}$$

where $\gamma_k = ku/(1-ku)$.

Proof. We write the exact compensation recurrence and the computed recurrence with consistent indices. Let $d_0 = n$ and $d_j = \lfloor d_{j-1}/2 \rfloor$, matching the scalar degree counter d in Algorithm 5. At level j, for each paired merge satisfying $2i + 1 \leq d_{j-1}$, define the local exact residual

$$e_i^{(j)} = \alpha_i^{(j)} + \beta_i^{(j)} + \widehat{C}_{2i+1}^{(j-1)}\rho_j. \tag{27}$$

For an unpaired term we set $e_i^{(j)} = 0$. Then the exact compensation satisfies

$$\epsilon C_i^{(j)} = e_i^{(j)} + \epsilon C_{2i}^{(j-1)} + \epsilon C_{2i+1}^{(j-1)}\widehat{y}_j, \tag{28}$$

with the last term omitted for an unpaired merge. The algorithm evaluates the same recurrence in floating-point arithmetic. Applying the γ-notation to this depth-L evaluation tree gives

$$|\epsilon C_0^{(L)} - \widehat{\epsilon C}_0^{(L)}| \leq \gamma_{2\lfloor \log_2 \lfloor n/2 \rfloor \rfloor + 6} \sum_{(j,i)\in\mathcal{I}} |e_i^{(j)}|\, w_i^{(j)}, \tag{29}$$

where $\mathcal{I}$ is the set of paired merges and $w_i^{(j)}$ is the absolute weight accumulated from that node to the root of the Estrin tree. These weights are products of powers of $|x|$ and sum against the corresponding coefficients in the same way as in the proof of Theorem 1.

It remains to bound the size of the exact first-order compensation. The residuals $\alpha_i^{(j)}$ and $\beta_i^{(j)}$ are the first-order product and addition residuals of the standard Estrin merge. The power residuals satisfy, by (17) and Lemma 4,

$$|\rho_j| \le \gamma_{2^{j-1}-1}|x|^{2^{j-1}} + O(u^2|x|^{2^{j-1}}).$$

Therefore the terms $e_i^{(j)}$, propagated with the weights $w_i^{(j)}$, are precisely the first-order residuals of the standard Estrin evaluation, including the residuals from the repeated squaring of the powers. Repeating the absolute-value propagation used in Theorem 1, and using a conservative factor 2 to absorb the separation between local merge and power residuals, gives

$$\sum_{(j,i)\in\mathcal{I}} |e_i^{(j)}|\, w_i^{(j)} \le 2\gamma_{K_n} \sum_{i=0}^{n} |C_i||x|^i. \tag{30}$$

Combining (29) and (30) gives

$$|\epsilon C_0^{(L)} - \widehat{\epsilon C}_0^{(L)}| \le 2\gamma_{M_n}\gamma_{K_n} \sum_{i=0}^{n} |C_i||x|^i, \tag{31}$$

which is the desired bound.

Theorem 3 (Forward Error Bound). *Let $p(x) = \sum_{i=0}^{n} C_i x^i$ be a polynomial and $\widehat{p}(x)$ its evaluation via the compensated Estrin algorithm. For $n \ge 2$, the relative forward error satisfies:*

$$\frac{|p(x) - \widehat{p}(x)|}{|p(x)|} \le u + 2\Gamma \cdot \mathrm{cond}(p, x), \tag{32}$$

where K_n and M_n are defined as in Theorem 2, $\Gamma = \gamma_{M_n}\gamma_{K_n+1}$, and $\mathrm{cond}(p,x) = \left(\sum_{i=0}^{n} |C_i||x|^i\right)/|p(x)|$ is the condition number.

Proof. From Theorem 2 and the definition of the compensated result, we have:

$$|C_{\mathrm{comp}} - p(x)| \le u|p(x)| + 2(1+u)\gamma_{M_n}\gamma_{K_n} \sum_{i=0}^{n} |C_i||x|^i. \tag{33}$$

Since $C_{\mathrm{comp}} = \widehat{p}(x)$ and, for the relevant range where the γ-notation is defined, $(1+u)\gamma_k \le \gamma_{k+1}$ with $k = K_n$, it follows that:

$$|\widehat{p}(x) - p(x)| \le u|p(x)| + 2\Gamma \sum_{i=0}^{n} |C_i||x|^i. \tag{34}$$

Dividing both sides by $|p(x)|$ and applying the definition of the condition number yields the stated bound.

5 Double-Double Estrin Scheme

5.1 Double-Double Arithmetic

A double-double number is a normalized double-word number represented as the unevaluated sum of two floating-point numbers:

$$a = a_h + a_l, \tag{35}$$

where $a_h, a_l \in \mathbb{F}$ and, for nonzero a, the leading component satisfies

$$a_h = \mathrm{RN}(a_h + a_l). \tag{36}$$

RN denotes rounding to nearest in the working binary64 format. The exact zero is represented by $(0, 0)$. Such double-word and multiple-precision representations are standard in software extended-precision systems [1]. Here we use the normalized double-word definition from the error analysis of Joldes, Muller, and Popescu [18]. It implies the usual nonoverlap relation $|a_l| \leq u\,|a_h|$ under the normal-range assumptions, but the nonoverlap inequality alone is not used here as the definition.

Basic operations on double-double numbers are built from standard error-free transformations such as `TwoSum` and `TwoProduct` [18]. We use the published double-word addition and multiplication kernels of Joldes, Muller, and Popescu [18], together with the later formalization and refinements of Muller and Rideau [24], as baseline building blocks rather than restating their full algorithms. For normalized inputs, the accurate double-word addition kernel returns r_h, r_l such that

$$r_h + r_l = (a + b)(1 + \varepsilon_{\mathrm{add}}), \quad |\varepsilon_{\mathrm{add}}| \leq \eta_{\mathrm{add}}, \quad \eta_{\mathrm{add}} = \frac{3u^2}{1 - 4u}. \tag{37}$$

The standard double-word multiplication kernel satisfies, for a nonzero exact product,

$$r_h + r_l = (a \cdot b)(1 + \varepsilon_{\mathrm{mul}}), \quad |\varepsilon_{\mathrm{mul}}| \leq \eta_{\mathrm{mul}}, \quad \eta_{\mathrm{mul}} = 7u^2 + O(u^3). \tag{38}$$

Exact-zero cases are interpreted through the corresponding exact-zero statements of the cited kernels. Thus the internal arithmetic has no first-order relative error while the result is kept as a double-word pair; only conversion back to one binary64 number introduces the usual order-u rounding term.

5.2 Double-Double Estrin Algorithm

Using double-double arithmetic, we can implement an accurate Estrin scheme by replacing each floating-point addition and multiplication in the standard Estrin algorithm with their double-double counterparts. Coefficients are embedded as exact normalized double-word pairs at the beginning, and all intermediate quantities are kept in double-double format until the final pair is formed. To match

the binary64 input/output interface used in the reported experiments, the algorithm returns the leading component of this final pair. Thus the double-double baseline is deliberately the same Estrin tree evaluated with the cited double-word kernels. Its final internal double-word pair is (p_h, p_l). Because this pair is normalized, the returned component p_h is the round-to-nearest binary64 value of $p_h + p_l$. The analysis below separates this final binary64 output from the internal double-word result.

For the Estrin-level analysis, we collect the published kernel bounds into the following model. Let

$$\eta_{\mathrm{DD}} \geq \max(\eta_{\mathrm{add}}, \eta_{\mathrm{mul}}). \tag{39}$$

Then each nonzero double-word addition or multiplication returns the represented double-word value

$$z = (a \circ b)(1 + \delta), \quad |\delta| \leq \eta_{\mathrm{DD}}, \quad \circ \in \{+, \times\}, \tag{40}$$

where z denotes the unevaluated sum of the returned pair. Exact zero cases are interpreted through the exact-zero statements of the corresponding kernels. For the kernels above, $\eta_{\mathrm{DD}} = O(u^2)$. We use the same product-counting argument as in Lemma 4: a product of k such factors is $1 + \theta_k^{\mathrm{DD}}$, with

$$|\theta_k^{\mathrm{DD}}| \leq \frac{k\eta_{\mathrm{DD}}}{1 - k\eta_{\mathrm{DD}}} \quad (k\eta_{\mathrm{DD}} < 1). \tag{41}$$

Theorem 4 (Forward Error Bound for Double-Double Estrin) *Let $p(x) = \sum_{i=0}^{n} C_i x^i$ with $n \geq 1$, $C_i, x \in \mathbb{F}$, and $p(x) \neq 0$. Suppose that the normal-range assumptions hold, the inputs are embedded exactly as normalized double-word pairs, and the double-double kernels satisfy the model above. Let (p_h, p_l) be the final internal double-double pair formed by the double-double Estrin baseline, and let $\widehat{p}_{dd}(x) = p_h + p_l$ be its represented real value before the algorithm returns the binary64 component p_h. Put*

$$K_n = \lfloor \log_2 n \rfloor + n + 1. \tag{42}$$

If $K_n \eta_{\mathrm{DD}} < 1$, then

$$\frac{|p(x) - \widehat{p}_{dd}(x)|}{|p(x)|} \leq \frac{K_n \eta_{\mathrm{DD}}}{1 - K_n \eta_{\mathrm{DD}}} \operatorname{cond}(p, x), \tag{43}$$

so the standard Estrin K_n-factor is multiplied by an $O(u^2)$ double-double kernel error. Since the baseline returns the high component p_h of the normalized final pair as the final binary64 value, we also have

$$\frac{|p(x) - p_h|}{|p(x)|} \leq u + (1 + u)\frac{K_n \eta_{\mathrm{DD}}}{1 - K_n \eta_{\mathrm{DD}}} \operatorname{cond}(p, x), \tag{44}$$

and hence the binary64-output error is $u + \operatorname{cond}(p, x) O(u^2)$.

Proof. Since $C_i, x \in \mathbb{F}$, representing them by the pairs $(C_i, 0)$ and $(x, 0)$ gives exact normalized double-word inputs. The standard Estrin proof in Theorem 1 counts, for each monomial contribution, the rounded multiplications used to form the relevant powers and the rounded additions used to merge the evaluation tree. The same count applies here, with the binary64 local rounding factor replaced by the double-double kernel factor bounded by η_{DD}.

Indeed, the powers used by the Estrin tree are obtained by repeated squaring. The power $x^{2^{j-1}}$ used at level j therefore contains at most $2^{j-1}-1$ double-double product factors: each squaring doubles the inherited factors and contributes one new kernel factor. Hence the products needed to form the monomial $C_i x^i$ contribute at most i double-double kernel factors. The partial value containing this monomial is then merged through at most $\lfloor \log_2 n \rfloor + 1$ double-double additions. Thus each monomial reaches the root with at most $K_n = \lfloor \log_2 n \rfloor + n + 1$ kernel factors, and the final internal pair can be written in the same perturbation form as

$$\widehat{p}_{dd}(x) = \sum_{i=0}^{n} C_i x^i (1 + \theta_i^{\mathrm{DD}}), \quad |\theta_i^{\mathrm{DD}}| \le \frac{K_n \eta_{\mathrm{DD}}}{1 - K_n \eta_{\mathrm{DD}}}. \tag{45}$$

Subtracting the exact value $p(x) = \sum_i C_i x^i$ yields

$$|p(x) - \widehat{p}_{dd}(x)| \le \sum_{i=0}^{n} |C_i||x|^i |\theta_i^{\mathrm{DD}}| \le \frac{K_n \eta_{\mathrm{DD}}}{1 - K_n \eta_{\mathrm{DD}}} \sum_{i=0}^{n} |C_i||x|^i, \tag{46}$$

and division by $|p(x)|$ gives the first stated bound through (8). Since the selected double-double kernels satisfy $\eta_{\mathrm{DD}} = O(u^2)$, the pair-output bound has the standard Estrin K_n-dependence with u replaced by an $O(u^2)$ kernel error.

For the final binary64 output, let $r = \widehat{p}_{dd}(x) = p_h + p_l$. The reported value is the high component p_h. Since the final pair is normalized, $p_h = \mathrm{RN}(r)$; equivalently, $|p_l| = |r - p_h| \le u|r|$ under the same normal-range assumption. Therefore

$$\begin{aligned} |p(x) - p_h| &\le |p(x) - r| + |r - p_h| \\ &= |p(x) - r| + |p_l| \\ &\le |p(x) - r| + u|r| \\ &\le u|p(x)| + (1+u)|p(x) - r|. \end{aligned} \tag{47}$$

After division by $|p(x)|$ and using the pair-output bound,

$$\frac{|p(x) - p_h|}{|p(x)|} \le u + (1+u)\frac{K_n \eta_{\mathrm{DD}}}{1 - K_n \eta_{\mathrm{DD}}} \mathrm{cond}(p, x). \tag{48}$$

This is $u + \mathrm{cond}(p, x) O(u^2)$, which proves the binary64-output statement.

A straightforward operation count shows that the Estrin algorithm in double-double precision is substantially heavier at the kernel level. With a standard 17-FLOP `TwoProduct` and 3-FLOP `FastTwoSum`, the cited multiplication and addition kernels cost 24 and 20 FLOPs, respectively. Thus the double-double

Estrin baseline uses about $44n + 24\lfloor \log_2(n) \rfloor$ such FLOPs, up to small loop and final-conversion terms. For timing, we report both a Dekker-kernel implementation and a hardware-FMA-kernel implementation. Their operation-count models differ only in `TwoProduct`: 17 FLOPs for Algorithm 2 and 2 FLOPs for Algorithm 3.

6 Experimental Results

This section compares standard Estrin, Horner, compensated Horner, compensated Estrin, double-double Estrin, and binary128 Estrin evaluations.

6.1 Experimental Setup

Unless explicitly stated otherwise, the polynomial-evaluation algorithms use IEEE binary64 input and output. Reference values, relative errors, and condition numbers are computed with MPFR at 256-bit precision; MPFR is used only as the reference backend and is not included as an evaluated or timed polynomial algorithm. The binary128 comparison uses a standard Estrin evaluation performed with GCC `__float128` arithmetic. On the tested x86_64 platform this binary128 baseline is implemented through the GCC quad-precision runtime and `libquadmath`, not MPFR arbitrary-precision arithmetic. To obtain a challenging accuracy test case with increasing ill-conditioning, we consider the polynomial $p(x) = (x-1)^n$ for degrees $n = 3, 4, \ldots, 42$, evaluated at the point $x = \mathrm{fl}(1.333)$.

The timing results were measured on the XG41 server with an Intel Xeon Gold 6430 processor and GCC 13.3.0. The Dekker timing was compiled with `-O3`, `-march=native`, `-fno-fast-math`, and `-ffp-contract=off`; the hardware-FMA timing additionally used `-mfma`. The generated assembly was checked to contain scalar FMA instructions for the hardware-FMA path. This prevents implicit FMA contraction in the Dekker benchmark while making the second benchmark a hardware-FMA benchmark. Each reported time is averaged over repeated evaluations. The pure timing benchmark for `CompEstrin` and double-double Estrin uses well-scaled deterministic random coefficients at $x = 0.875$, runs degrees $8 \le n \le 4096$, and excludes MPFR, binary128 arithmetic, and error computation from the timed loop. The implementation used in the experiments is available at https://github.com/yuguangping/CompEstrin.

This classical test has rapidly growing condition number when x is close to 1. Accuracy is measured by the relative forward error $|\widehat{p}(x) - p(x)|/|p(x)|$, and $\mathrm{cond}(p, x)$ is evaluated as in Sect. 3. Figure 1, Table 1, and Fig. 2 summarize the accuracy, kernel-count, and scalar timing results.

6.2 Results and Discussion

For small degrees, all schemes yield relative errors close to the unit roundoff u. As n grows, standard Estrin follows the behavior predicted by Theorem 1, with linear dependence on $\mathrm{cond}(p, x)$ up to a factor of order $\gamma_{\lfloor \log_2(n) \rfloor + n + 1}$. Compensated Estrin remains close to u over a much wider range and is consistent with Theorem 3,

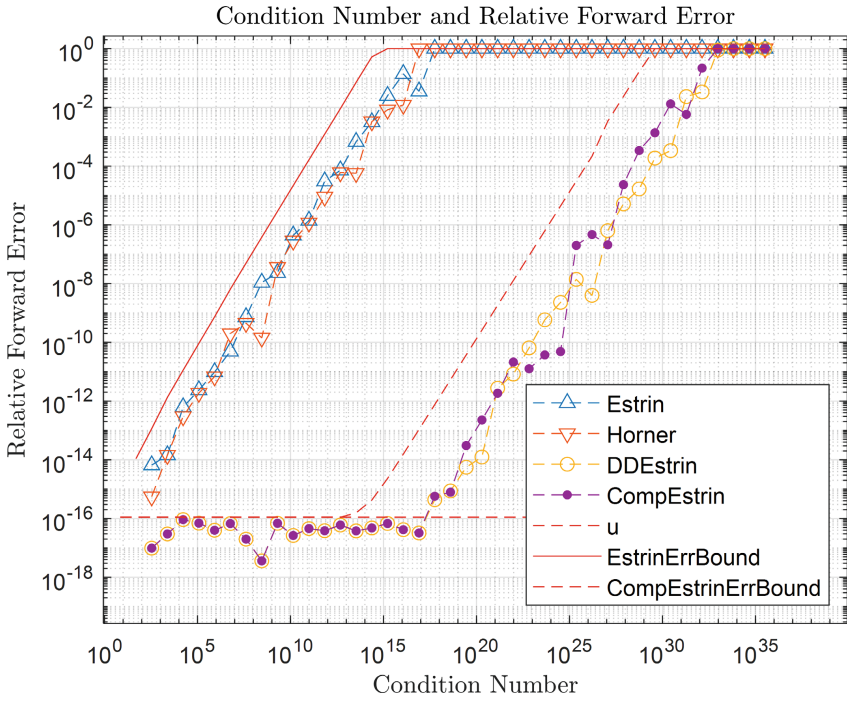

Fig. 1. Relative forward error versus the condition number for standard binary64 Estrin, Horner, the compensated Estrin scheme, and the double-double Estrin scheme, for $p(x) = (x-1)^n$ evaluated at $x = \mathrm{fl}(1.333)$.

Table 1. Representative accuracy and theoretical Dekker-kernel cost comparison. Relative errors are computed against 256-bit MPFR reference values for $p(x) = (x-1)^n$ at $x = \mathrm{fl}(1.333)$. The theoretical kernel-count ratios use `CompEstrin` as the baseline. A dash means that the binary64 Dekker-kernel count model does not apply.

n	$\mathrm{cond}(p,x)$	method	relative forward error	theoretical binary64 Dekker kernel count / `CompEstrin`
8	5.80×10^{6}	`CompEstrin`	6.73×10^{-17}	1.00
8	5.80×10^{6}	double-double Estrin	6.73×10^{-17}	1.46
8	5.80×10^{6}	GCC `__float128` Estrin	9.47×10^{-29}	–
16	3.37×10^{13}	`CompEstrin`	3.79×10^{-17}	1.00
16	3.37×10^{13}	double-double Estrin	3.79×10^{-17}	1.48
16	3.37×10^{13}	GCC `__float128` Estrin	1.12×10^{-22}	–
24	1.96×10^{20}	`CompEstrin`	4.06×10^{-13}	1.00
24	1.96×10^{20}	double-double Estrin	1.32×10^{-13}	1.49
24	1.96×10^{20}	GCC `__float128` Estrin	6.23×10^{-16}	–
32	1.14×10^{27}	`CompEstrin`	6.01×10^{-7}	1.00
32	1.14×10^{27}	double-double Estrin	1.45×10^{-6}	1.49
32	1.14×10^{27}	GCC `__float128` Estrin	1.10×10^{-8}	–
42	3.23×10^{35}	`CompEstrin`	3.96×10^{3}	1.00
42	3.23×10^{35}	double-double Estrin	1.74×10^{2}	1.50
42	3.23×10^{35}	GCC `__float128` Estrin	4.20×10^{-1}	–

whose leading terms are $u+\mathrm{cond}(p,x)O(u^2)$. The binomial stress test is stopped at $n = 42$, where $\mathrm{cond}(p,x) \approx 3.23 \times 10^{35}$; beyond this regime no binary64-output twice-working-precision-like method is expected to stay close to u.

Table 1 uses `CompEstrin` as the baseline for the theoretical binary64 Dekker-kernel ratios. The ratio is $N_{\mathrm{DD}}(n)/N_{\mathrm{CE}}(n)$, where $N_{\mathrm{CE}}(n) = 19\lfloor \log_2 n \rfloor + 29n + 1$ and $N_{\mathrm{DD}}(n) = 44n + 24\lfloor \log_2 n \rfloor$. With a hardware-FMA-kernel `TwoProduct`, the corresponding counts are $N_{\mathrm{CE}}^{\mathrm{FMA}}(n) = 4\lfloor \log_2 n \rfloor + 14n + 1$ and $N_{\mathrm{DD}}^{\mathrm{FMA}}(n) = 29n + 9\lfloor \log_2 n \rfloor$. The table tabulates the Dekker-kernel ratio only; Fig. 2 shows both kernel choices in the scalar timing benchmark (see Table 1).

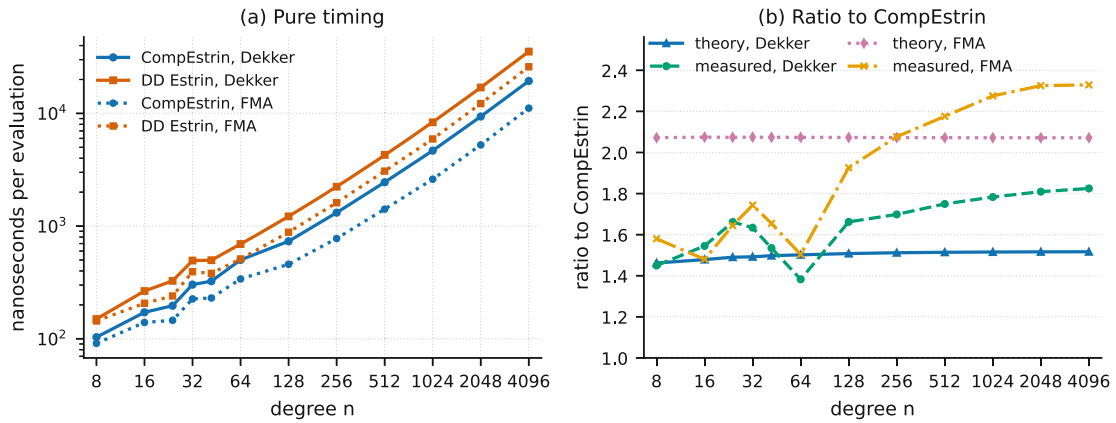

Fig. 2. Pure scalar timing benchmark at $x = 0.875$ with deterministic random coefficients. Panel (a) reports nanoseconds per evaluation for $8 \le n \le 4096$ using Dekker and hardware-FMA `TwoProduct` kernels. Panel (b) compares the measured double-double Estrin/`CompEstrin` time ratios with the corresponding theoretical binary64 FLOP-count ratios.

Table 2. Additional accuracy tests for alternating-sign, random, and Chebyshev polynomials at $x = \mathrm{fl}(1.333)$. The last column is the standard Estrin relative forward error divided by the `CompEstrin` relative forward error.

family	n	$\mathrm{cond}(p,x)$	Estrin error	Horner error	`CompEstrin` error	improvement factor
alternating	8	4.71	3.33×10^{-16}	1.81×10^{-16}	2.89×10^{-17}	11.5
alternating	16	4.00	5.02×10^{-16}	1.51×10^{-16}	2.38×10^{-17}	21.1
alternating	32	4.84	1.14×10^{-15}	4.84×10^{-16}	3.77×10^{-17}	30.1
alternating	42	6.60	1.00×10^{-15}	1.29×10^{-16}	4.65×10^{-17}	21.6
random	8	33.3	3.65×10^{-16}	1.44×10^{-15}	7.22×10^{-18}	50.5
random	16	22.5	5.96×10^{-16}	2.74×10^{-16}	4.80×10^{-17}	12.4
random	32	291	3.19×10^{-14}	9.90×10^{-16}	3.82×10^{-17}	834
random	42	9.99	3.94×10^{-16}	5.95×10^{-16}	6.09×10^{-18}	64.7
Chebyshev	8	11.3	4.00×10^{-16}	1.90×10^{-16}	6.44×10^{-18}	62.1
Chebyshev	16	128	2.16×10^{-15}	2.72×10^{-15}	7.08×10^{-17}	30.5
Chebyshev	32	1.65×10^{4}	3.02×10^{-14}	3.73×10^{-13}	1.47×10^{-17}	2.05×10^{3}
Chebyshev	42	3.42×10^{5}	4.42×10^{-13}	5.40×10^{-12}	3.06×10^{-17}	1.44×10^{4}

Figure 2 shows that, in this scalar C benchmark, double-double Estrin is slower than `CompEstrin` for both kernel choices. With the Dekker `TwoProduct`, the measured double-double Estrin/`CompEstrin` ratio ranges from 1.38 to 1.82; it is above the corresponding FLOP-count ratio ($\approx$ 1.46–1.52) at ten of the twelve tested degrees. With the hardware-FMA `TwoProduct`, the measured ratio ranges from 1.48 to 2.33. It is below the hardware-FMA FLOP-count ratio ($\approx$ 2.07) for the smaller degrees and above it for $n \geq 256$. Thus the operation counts should be read as kernel-level references rather than pointwise timing predictions; exposed independent subexpressions, latency, throughput, and instruction scheduling also affect the measured time [7]. This distinction is consistent with FMA-based compensated-Horner experiments, where the measured slowdowns of `CompHornerFMA` and `DDHornerFMA` over `HornerFMA` are reported separately from the theoretical FLOP-count ratios [11].

Table 2 reports additional accuracy tests for alternating-sign, random, and Chebyshev polynomials. These results indicate that the improvement is not specific to the binomial stress test. Across the tested alternating-sign, random, and Chebyshev families, `CompEstrin` consistently reduces the relative forward error compared with standard Estrin, while Horner is not uniformly more accurate than either Estrin variant.

7 Conclusion

This paper has presented a compensated Estrin scheme for polynomial evaluation in floating-point arithmetic. The method combines the low-depth evaluation tree of the Estrin scheme with error compensation based on error-free transformations. A detailed forward error analysis shows that the compensated scheme satisfies a bound of the form

$$\frac{|p(x) - \texttt{CompEstrin}(p, x)|}{|p(x)|} \leq u + \mathrm{cond}(p, x) \cdot O(u^2). \tag{49}$$

This indicates that the computed result is nearly as accurate as if $p(x)$ were evaluated in twice the working precision and then rounded to working precision.

An accurate Estrin scheme implemented in double-double arithmetic was also considered. We derived its forward error bound, distinguishing the $\mathrm{cond}(p, x) \cdot O(u^2)$ bound for the double-double pair result from the additional order-u term introduced when that pair is finally rounded back to binary64. Its behavior has been compared numerically with the standard and compensated Estrin schemes on a classical ill-conditioned test family. The numerical experiments confirm that the compensated Estrin scheme substantially improves the accuracy compared to the standard Estrin scheme, while producing errors close to those of the double-double Estrin implementation. In the scalar pure timing benchmark, it also has a lower measured cost than the double-double Estrin baseline, and it is faster than the software binary128 baseline on the tested platform.

Future work includes further optimization of the compensated Estrin scheme for specific hardware architectures, such as GPUs and SIMD units, and

extensions of the compensation strategy to other classes of functions beyond polynomials, for example rational functions or elementary functions implemented via polynomial or rational approximations.

Acknowledgments. The authors would like to thank the reviewers for their valuable feedback and suggestions. This work is supported by the National Key R&D Program of China (No. 2023YFB3001600) and partly by the FPT-4 project (ANR-24-CE46-7572) of the French National Agency for Research (ANR).

References

1. Bailey, D.H.: A Fortran 90-based multiprecision system. ACM Trans. Math. Softw. **21**(4), 379–387 (1995). https://doi.org/10.1145/212066.212075
2. Boldo, S., Muller, J.M.: Some functions computable with a fused-mac. In: Proceedings of the 17th Symposium on Computer Arithmetic, pp. 52–58. IEEE (2005)
3. Boldo, S., Muller, J.M.: Exact and approximated error of the FMA. IEEE Trans. Comput. **60**(2), 157–164 (2010)
4. De Lassus Saint-Genies, H., Revy, G.: Performances de schemas d'evaluation polynomiale sur architectures vectorielles. In: ComPAS: Conference en Parallelisme, Architecture et Systeme (2016)
5. Dekker, T.J.: A floating-point technique for extending the available precision. Numer. Math. **18**, 224–242 (1971). https://doi.org/10.1007/BF01397083
6. Estrin, G.: Organization of computer systems: the fixed plus variable structure computer. In: Papers Presented at the May 3–5, 1960, Western Joint IRE-AIEE-ACM Computer Conference, pp. 33–40. ACM (1960). https://doi.org/10.1145/1460361.1460365
7. Ewart, T., Cremonesi, F., Schürmann, F., Delalondre, F.: Polynomial evaluation on superscalar architecture, applied to the elementary function e^x. ACM Trans. Math. Softw. **46**(3), 1–22 (2020). https://doi.org/10.1145/3408893
8. Graillat, S.: Accurate floating-point product and exponentiation. IEEE Trans. Comput. **58**(7), 994–1000 (2009). https://doi.org/10.1109/TC.2008.215
9. Graillat, S.: An accurate algorithm for evaluating rational functions. Appl. Math. Comput. **337**, 494–503 (2018). https://doi.org/10.1016/j.amc.2018.05.039
10. Graillat, S., Ibrahimy, Y., Jeangoudoux, C.: A parallel compensated horner scheme for SIMD architecture. In: 2023 IEEE 30th Symposium on Computer Arithmetic (ARITH), pp. 131–138. IEEE (2023)
11. Graillat, S., Langlois, P., Louvet, N.: Improving the compensated Horner scheme with a fused multiply and add. In: Proceedings of the 2006 ACM Symposium on Applied Computing. pp. 1323–1327. SAC '06, ACM (2006). https://doi.org/10.1145/1141277.1141585
12. Graillat, S., Langlois, P., Louvet, N.: Algorithms for accurate, validated and fast polynomial evaluation. Jpn. J. Ind. Appl. Math. **26**(2), 191–214 (2009). https://doi.org/10.1007/BF03186531
13. Graillat, S., Louvet, N., Langlois, P.: Compensated Horner scheme. Tech. Rep. 04, Equipe de recherche DALI, Laboratoire LP2A, Universite de Perpignan Via Domitia, France (2005)
14. He, K., Huang, C., Jiang, H., Gu, T., Qi, J., Liu, J.: Design and implementation of a high-precision reduction function based on MPI. Comput. Eng. Sci. **43**(4), 594–602 (2021)

15. Higham, N.J.: Accuracy and Stability of Numerical Algorithms, 2nd edn. SIAM, Philadelphia (2002)
16. Huang, C., Jiang, H., Gu, T., Qi, J., Liu, W.: Implementation and optimization of high-precision summation and dot product algorithms on Phytium processor. Comput. Eng. Sci. **43**(1), 1–8 (2021)
17. IEEE: IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019. IEEE, New York (2019)
18. Joldes, M., Muller, J.M., Popescu, V.: Tight and rigorous error bounds for basic building blocks of double-word arithmetic. ACM Trans. Math. Softw. **44**(2) (2017). https://doi.org/10.1145/3121432
19. Knuth, D.E.: The Art of Computer Programming, Volume 2: Seminumerical Algorithms, 3 edn. Addison-Wesley (1998)
20. Lange, M., Rump, S.M.: Faithfully rounded floating-point computations. ACM Trans. Math. Softw. **46**(3), 1–20 (2020). https://doi.org/10.1145/3290955
21. Lauter, C., Mezzarobba, M.: Semi-automatic floating-point implementation of special functions. In: IEEE 22nd Symposium on Computer Arithmetic, pp. 58–65 (2015)
22. Moroz, G.: Fast polynomial evaluation and composition. Tech. Rep. RT-0453, INRIA (2013). arXiv:1307.5655
23. Muller, J.M.: Elementary Functions: Algorithms and Implementation. Birkhauser (1997)
24. Muller, J.M., Rideau, L.: Formalization of double-word arithmetic, and comments on "tight and rigorous error bounds for basic building blocks of double-word arithmetic". ACM Trans. Math. Softw. **48**(1), 1–24 (2022). https://doi.org/10.1145/3484514
25. Muller, J.M., et al.: Handbook of Floating-Point Arithmetic, 2 edn. Birkhauser (2018)
26. Nievergelt, Y.: Scalar fused multiply-add instructions produce floating-point matrix arithmetic provably accurate to the penultimate digit. ACM Trans. Math. Softw. **29**(1), 27–48 (2003)
27. Ogita, T., Rump, S.M., Oishi, S.: Accurate sum and dot product. SIAM J. Sci. Comput. **26**(6), 1955–1988 (2005)
28. Parhami, B.: Computer Arithmetic: Algorithms and Hardware Designs. Oxford University Press (2010)
29. Revy, G.: Analyse et implantation d'algorithmes rapides pour l'evaluation polynomiale sur les nombres flottants (2006)
30. Revy, G.: Implementation of binary floating-point arithmetic on embedded integer processors: polynomial evaluation-based algorithms and certified code generation. Ph.D. thesis, Universite de Lyon; Ecole Normale Superieure de Lyon (2009)
31. Stewart, G.W.: Introduction to Matrix Computations. Academic Press, New York (1973)
32. Yamanaka, N., Ogita, T., Rump, S.M., Oishi, S.: A parallel algorithm for accurate dot product. Parallel Comput. **34**(1–2), 392–410 (2008). https://doi.org/10.1016/j.parco.2008.02.002