Compensated Algorithms

Philippe Langlois, Nicolas Louvet

Equipe DALI, Université de Perpignan
http://webdali.univ-perp.fr
Motivation

- Results computed in floating point arithmetic are possibly corrupted by rounding errors.

- Compensated algorithms = algorithms that correct the rounding errors generated during the computation:
  If $\hat{r}$ is a computed result, how to find a correcting term $\hat{c}$ such that $\bar{r} = \hat{r} \oplus \hat{c}$ is more accurate than $\hat{r}$?

- Aim of this presentation is:
  - to recall the principle of so-called compensated algorithms,
  - to present some details about the compensated Horner algorithm.

- Context: IEEE-754 fp arithmetique, rounding to the nearest, no underflow.

---

Outline

1 Introduction

2 Principle of Compensated Algorithms

3 Sketch of proof for the compensated Horner algorithm

4 Conclusion

5 More slides
   - Faithful rounding with the CHS
   - Practical efficiency
Introduction

Why do we need compensated algorithms?
Backward stable algorithms vs. condition number

Backward stable algorithms:

- The accuracy of the computed solution satisfies

\[ \text{accuracy} \lesssim \text{condition number} \times u, \]

where

- \( u \) is the computing precision:
  - IEEE-754 double, 53-bits mantissa, rounding to the nearest \( \Rightarrow u = 2^{-53} \).

- the condition number quantify the difficulty to solve the problem accurately.

Examples: summation, dot product, Horner algorithm, substitution for triangular system solving.
Accuracy of the Horner scheme

We consider the polynomial

\[ p(x) = \sum_{i=0}^{n} a_i x^i, \]

with \( a_i \in \mathbb{F}, x \in \mathbb{F} \)

Algorithm (Horner scheme)

function \( \hat{r}_0 = \text{Horner}(p, x) \)
\( \hat{r}_n = a_n \)
for \( i = n - 1 : -1 : 0 \)
\( \hat{r}_i = \hat{r}_{i+1} \otimes x \oplus a_i \)
end

Relative accuracy of the evaluation with the Horner scheme:

\[ |\text{Horner}(p, x) - p(x)| \leq \gamma 2^n \approx 2^n u \]

\( u \) is the computing precision
\( \text{cond}(p, x) \) denotes the condition number of the evaluation:

\[ \text{cond}(p, x) = \sum \left| a_i x^i \right| / \left| p(x) \right| \geq 1. \]

Compensated Algorithms – N. Louvet
April 18, 2007 6 / 33
Accuracy of the Horner scheme

We consider the polynomial

\[ p(x) = \sum_{i=0}^{n} a_i x^i, \]

with \( a_i \in \mathbb{F}, \, x \in \mathbb{F} \)

Relative accuracy of the evaluation with the Horner scheme:

\[
\frac{|\text{Horner} (p, x) - p(x)|}{|p(x)|} \leq \gamma_{2n} \text{cond}(p, x) \approx 2nu
\]

- \( u \) is the computing precision
- \( \text{cond}(p, x) \) denotes the condition number of the evaluation:

\[
\text{cond}(p, x) = \frac{\sum |a_i x^i|}{|p(x)|} \geq 1
\]
Accuracy $\lesssim$ condition number of the problem $\times u$

How to manage ill-conditioned cases?
Compensated algorithms

Algorithms that correct the generated rounding errors

- Many examples:
  - Compensated summation: Neumaier (74), Sum2 in Ogita-Rump-Oishi (05)
  - Compensated Horner algorithm
  - Compensated substitution for triangular system solving

- Accuracy as if computed in twice the working precision:

\[
\text{accuracy} \lesssim u + \text{condition number} \times u^2
\]

- More efficient than fixed length expansion libraries (double-double)
- Not considered here: Kahan's compensated summation (65), Priest’s doubly compensated summation (92)
Accuracy of the result $\lesssim u + \text{condition number} \times u^2$.

How is computed the compensated result?
Principle of compensated Algorithms

The compensated result is $\tilde{r} = \hat{r} \oplus \hat{c}$. The correcting term $\hat{c}$ is

- an approximate of the forward error in $\hat{r}$,
- computed thanks to “Error-Free Transformations”.
Compensated result

\[ \text{Input space } \mathcal{D} \quad x \rightarrow G \quad \text{Output space } \mathcal{R} \quad r = G(x) \]

\[ \hat{G} : \hat{r} = \hat{G}(x) \]

- Forward error analysis
Compensated result

Forward error analysis
Backward error analysis

Identify $\hat{r}$ as the exact solution of a perturbed problem:

$\hat{r} = \hat{G}(x + \Delta x)$
Compensated result

How to improve the quality of the computed result $\hat{r}$?

- First, compute an approximate $\hat{c}$ of the forward error $c = r - \hat{r}$. $\hat{c}$ is a correcting term for $\hat{r}$.
- Then, a compensated result $\bar{r} = \hat{r} \oplus \hat{c}$. 
How to compute the correcting term $\hat{c}$?

$\hat{c}$ is an approximate of the forward error $c = r - \hat{r}$

- **Classical error analysis:**
  
  Let $p(x) = \sum_{i=0}^{n} a_i x^i$ be a polynomial with floating point coefficient.
  
  - **Backward error result:** Horner algorithm computes
    
    $\text{Horner}(p, x) = \sum_{i=0}^{n} (1 + \delta_i) a_i x^i,$
    
    with $|\delta_i| \leq \gamma 2^n \approx 2n u$

  - **Forward error result:**
    
    $c = p(x) - \text{Horner}(p, x) = -\sum_{i=0}^{n} \delta_i a_i x^i \Rightarrow |c| \leq \gamma 2^n \sum_{i=0}^{n} |a_i||x|^i.$

  As the $\delta_i$ are unknown, classical error analysis does not solve our problem.

  But we can do better thanks to Error-Free Transformations!
Error-free transformations

**Error-Free Transformations (EFT)** are **properties** and **algorithms** to compute the rounding errors **at the current working precision**.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Formula</th>
<th>Flops</th>
<th>Author</th>
<th>Reference</th>
</tr>
</thead>
</table>
| Addition  | \((x, y) = 2\text{Sum}(a, b)\)  
\[a + b = x + y \text{ and } x = a \oplus b\] | 6 | Knuth | (74) |
| Multiplication | \((x, y) = 2\text{Prod}(a, b)\)  
\[a \times b = x + y \text{ and } x = a \otimes b\] | 17 | Veltkamp | |

with \(a, b, x, y \in \mathbb{F}\).
How to compute the correcting term $\hat{c}$ thanks to EFT?

**Algorithm (Horner scheme)**

function $\hat{r}_0 = \text{Horner}(p, x)$

$$\hat{r}_n = a_n$$

for $i = n - 1 : -1 : 0$

$$\hat{p}_i = \hat{r}_{i+1} \otimes x \ % \ \text{rounding error } \pi_i \in \mathbb{F} \Rightarrow \hat{p}_i = \hat{r}_{i+1}x - \pi_i$$

$$\hat{r}_i = \hat{p}_i \oplus a_i \ % \ \text{rounding error } \sigma_i \in \mathbb{F} \Rightarrow \hat{r}_i = \hat{p}_i + a_i - \sigma_i$$

For $i = n - 1 : -1 : 0$,

$$\hat{r}_i = \hat{r}_{i+1}x + a_i - \pi_i - \sigma_i.$$  

Since $\hat{r}_n = a_n$,

$$\text{Horner}(p, x) = \sum_{i=0}^{n} (a_i - \pi_i - \sigma_i)x^i, \quad \text{with} \quad \pi_n = \sigma_n = 0.$$
How to compute the correcting term $\hat{c}$ thanks to EFT?

$\hat{c}$ is an approximate of the forward error $c = r - \hat{r}$

- Express $\hat{r}$ w.r.t. the data and the elementary rounding errors:
  
  Horner algorithm computes

  $\text{Horner}(p, x) = \sum_{i=0}^{n} (a_i - \pi_i - \sigma_i)x^i$, \quad with \quad $\pi_n = \sigma_n = 0$.

- Deduce an expression for the forward error $c = r - \hat{r}$:

  $c = p(x) - \text{Horner}(p, x) = \sum_{i=0}^{n-1} (\pi_i + \sigma_i)x^i$.

- If we manage to find a closed form formula for $c$ w.r.t
  
  - the data,
  
  - the elementary rounding errors (exactly computable thanks to EFT),

  we can easily compute a correcting term $\hat{c}$. 

An EFT for the Horner Algorithm

We have

\[
c = \sum_{i=0}^{n-1} (\pi_i + \sigma_i)x^i = (p_\pi + p_\sigma)(x),
\]

with \( p_\pi(x) = \sum \pi_i x^i \) and \( p_\sigma(x) = \sum \sigma_i x^i \).

Algorithm (EFT for Horner)

function \([\hat{r}_0, p_\pi, p_\sigma] = \text{EFTHorner}(p, x)\)

\( \hat{r}_n = a_n \)

for \( i = n - 1 : -1 : 0 \)

\[
[\hat{p}_i, \pi_i] = 2\text{Prod}(\hat{r}_{i+1}, x)
\]

\[
[\hat{r}_i, \sigma_i] = 2\text{Sum}(\hat{p}_i, a_i)
\]

\( p_\pi[i] = \pi_i; \ p_\sigma[i] = \sigma_i \)

This algorithm is an EFT for the Horner algorithm since

\[
p(x) = \text{Horner}(p, x) + (p_\pi + p_\sigma)(x) = c.
\]
Compensated Horner Algorithm

Since $c = (p_\pi + p_\sigma)(x)$, we compute $\hat{c}$ as Horner $(p_\pi \oplus p_\sigma, x)$.

**Algorithm (Compensated Horner scheme)**

```plaintext
function $\bar{r} = \text{CompHorner} (p, x)$

[$\hat{r}, p_\pi, p_\sigma] = \text{EFTHorner} (p, x) \quad \% \quad \hat{r} = \text{Horner} (p, x)$

$\hat{c} = \text{Horner} (p_\pi \oplus p_\sigma, x)$

$\bar{r} = \hat{r} \oplus \hat{c}$
```

Next question: how to prove something about the accuracy of $\bar{r}$?

(Accuracy of the result $\lesssim u + \text{condition number} \times u^2$)

Difficult to answer in a general manner...
Sketch of proof for the compensated Horner algorithm

The key “ingredient” here is the EFT for the Horner algorithm,

\[ p(x) = \text{Horner}(p, x) + (p_\pi + p_\sigma)(x). \]

\[ = c \]
Sketch of proof for the compensated Horner algorithm

We recall:

- \( \hat{r} = \text{Horner}(p, x) \),
- \( c = (p_\pi + p_\sigma)(x) \) is the forward error in \( \hat{r} \),
- \( \hat{c} = \text{Horner}(p_\pi \oplus p_\sigma, x) \) is the computed correcting term,
- \( \bar{r} = \hat{r} \oplus \hat{c} \) is the compensated result.

Since the compensated result is \( \bar{r} = \hat{r} \oplus \hat{c} \),

\[
|p(x) - \bar{r}| = |p(x) - (1 + \varepsilon)(\hat{r} + \hat{c})|, \quad \text{with} \quad |\varepsilon| \leq u.
\]

Using the EFT for the Horner scheme, \( \hat{r} = p(x) - c \),

\[
|p(x) - \bar{r}| = |p(x) - (1 + \varepsilon)(p(x) + \hat{c} - c)| \\
\leq u|p(x)| + (1 + u)|\hat{c} - c|
\]

How can we bound \( |c - \hat{c}| \)?
Sketch of proof for the compensated Horner algorithm

$|c - \hat{c}|$ is the forward error in $\hat{c} = \text{Horner}(p_\pi \oplus p_\sigma, x)$. Then

$$|c - \hat{c}| \leq \gamma_{2n-1}(\widetilde{p_\pi + p_\sigma})(x),$$

with $(\widetilde{p_\pi + p_\sigma})(x) = \sum_{i=0}^{n-1} |p_\pi + p_\sigma||x^i|$. We bound “largely” this term as follows,

$$(\widetilde{p_\pi + p_\sigma})(x) \leq \gamma_{2n} \tilde{p}(x),$$

with $\tilde{p}(x) = \sum_{i=0}^{n} |a_i||x^i|$. Therefore,

$$|c - \hat{c}| \leq \gamma_{2n-1}\gamma_{2n} \tilde{p}(x).$$

Nota:

$$\gamma_k = \frac{ku}{1 - ku} = ku + \mathcal{O}u^2.$$
Sketch of proof for the compensated Horner algorithm

Then,

\[ |p(x) - \tilde{r}| \leq u |p(x)| + (1 + u) \gamma_{2n-1} \gamma_{2n} \tilde{p}(x), \]
\[ \leq u |p(x)| + \gamma_{2n}^2 \tilde{p}(x). \]

Now we turn to relative accuracy, and we obtain the following theorem.

**Theorem**

*Given* \( p \) *a polynomial with floating point coefficients, and* \( x \) *a floating point value, let* \( \tilde{r} \) *be the compensated evaluation of* \( p(x) \) *computed with CompHorner. Then,*

\[ \frac{|p(x) - \tilde{r}|}{|p(x)|} \leq u + \gamma_{2n}^2 \text{cond}(p, x). \]

*Again,* \( \gamma_{2n}^2 \approx 4n^2u^2. \)
Accuracy of the result $\lesssim u + \text{condition number} \times u^2$. 
Conclusion

- We have recall how to define a correcting term: correcting term = approximate of the forward error.
- Compensating the Horner algorithm improves the accuracy:
  - the accuracy of the compensated result is the same as if the result was computed in doubled working precision.
  - Remark: this is also true for compensated summation or compensated triangular system solving.
Faithful rounding with the CHS
Faithful rounding

Definition

A floating point number \( \hat{x} \) is said to be a faithful rounding of a real number \( x \) if

- either \( \hat{x} = x \),
- or \( \hat{x} \) is one of the two floating point neighbours of \( x \).

The error bound

\[
\frac{|\text{CompHorner}(p, x) - p(x)|}{|p(x)|} \leq u + \gamma_{2n}^2 \text{cond}(p, x)
\]

\[\leq 2n^2 u^2 \]

is too large for reasoning about faithful rounding.
An *a posteriori* test

We recall:
- \( \bar{r} = \text{CompHorner}(p, x) \) is the compensated result,
- \( c = (p_\pi + p_\sigma)(x) \) is the exact (real) correcting term for \( \bar{r} \),
- \( \hat{c} = \text{Horner}(p_\pi \oplus p_\sigma, x) \) is the computed (floating point) correcting term.

The following error bound on the computed correcting term holds:

\[
|c - \hat{c}| \leq \text{fl} \left( \frac{\hat{\gamma}_{2n-1} \text{Horner}(|p_\pi| \oplus |p_\sigma|, |x|)}{1 - 2(n + 1)u} \right) =: \hat{\beta}.
\]

Then, we can perform a dynamic test for faithful rounding:

**Theorem**

\[
\hat{\beta} < \frac{u}{2} |\bar{r}| \Rightarrow |c - \hat{c}| < \frac{u}{2} |\bar{r}| \Rightarrow \bar{r} \text{ is a faithful rounding of } p(x).
\]
An *a posteriori* test

Accuracy of polynomial evaluation with the compensated Horner scheme [n=50]

$$\frac{(1-u)/(2+u)u\gamma_{2n}^{-2}}{1/u}$$

$$u + \gamma_{2n}^{-2} \text{ cond}$$
Practical efficiency
Overhead to obtain more accuracy

- Theoretical ratios (flops):

\[
\frac{\text{CompHorner}}{\text{Horner}} \sim 10.5 \quad \frac{\text{CompHornerIsFaith}}{\text{Horner}} \sim 13 \quad \frac{\text{DDHorner}}{\text{Horner}} \sim 14
\]

- Some practical ratios (running times \(^2\)):

<table>
<thead>
<tr>
<th></th>
<th>CompHorner Horner</th>
<th>CompHornerIsFaith Horner</th>
<th>DDHorner Horner</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Pentium 4, 3.00 GHz</strong></td>
<td>3.77</td>
<td>5.52</td>
<td>10.00</td>
</tr>
<tr>
<td>GCC 3.3.5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICC 9.1</td>
<td>3.06</td>
<td>5.31</td>
<td>8.88</td>
</tr>
<tr>
<td><strong>Athlon 64, 2.00 GHz</strong></td>
<td>3.89</td>
<td>4.43</td>
<td>10.48</td>
</tr>
<tr>
<td>GCC 4.0.1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Itanium 2, 1.4 GHz</strong></td>
<td>3.64</td>
<td>4.59</td>
<td>5.50</td>
</tr>
<tr>
<td>GCC 3.4.6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ICC 9.1</td>
<td>1.87</td>
<td>2.30</td>
<td>8.78</td>
</tr>
</tbody>
</table>

\(~2 – 4\) \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim \quad \sim

\(^2\)Average ratios for polynomials of degree 5 to 200.
Some comparisons

How does the more accurate algorithms compare to each other?

<table>
<thead>
<tr>
<th></th>
<th>CompHornerIsFaith CompHorner</th>
<th>DDHorner CompHorner</th>
<th>DDHorner CompHornerIsFaith</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pentium 4, 3.00 GHz GCC 3.3.5</td>
<td>1.46</td>
<td>2.66</td>
<td>1.89</td>
</tr>
<tr>
<td></td>
<td>ICC 9.1</td>
<td>1.71</td>
<td>2.92</td>
</tr>
<tr>
<td>Athlon 64, 2.00 GHz GCC 4.0.1</td>
<td>1.14</td>
<td>2.70</td>
<td>2.38</td>
</tr>
<tr>
<td>Itanium 2, 1.4 GHz GCC 3.4.6</td>
<td>1.27</td>
<td>1.51</td>
<td>1.27</td>
</tr>
<tr>
<td></td>
<td>ICC 9.1</td>
<td>1.24</td>
<td>4.67</td>
</tr>
<tr>
<td></td>
<td>( \leq 2 )</td>
<td>( \sim 2 ) – 5</td>
<td>( \sim 2 ) – 5</td>
</tr>
</tbody>
</table>
What is Instruction-Level Parallelism?

- All processors since about 1985, including those in the embedded space, use pipelining to overlap the execution of instructions and improve performance. This potential overlap among instruction is called instruction-level parallelism (ILP) since the instruction can be evaluated in parallel. (Hennessy & Patterson)

- A wide range of techniques have been developed to exploit the parallelism available among instructions (pipelining, superscalar architectures...)

- Amount of ILP available in a code:
  - if two instructions are parallel they can execute simultaneously in a pipeline,
  - if two instructions are dependent they must be executed in order.

How to determine whether an instruction is dependent on another?
Dependences between instructions

- Three different types of dependences:
  - control dependences,
  - name dependences,
  - data dependences (or true dependences).

- A control dependence determines the ordering of an instruction with respect to a branch instruction.

- A name dependence occurs when two instructions use the same register or memory location (name), but there is in fact no flow of data between instructions associated with that name.

- But here we are mainly interested by data dependences.
Data dependences

- An instruction $i$ is data dependent on an instruction $j$ if either
  - instruction $j$ produces a result that may be used by instruction $i$,
  - there exists a chain of dependences of the first type between $i$ and $j$.

- If two instructions are data dependent, they cannot execute simultaneously.
- Dependences are properties of programs: the presence of a data dependence in an instruction sequence reflects a data dependence in the source code.
- What about the dependences in CompHorner and DDHorner?
Difference between DDHorner and CompHorner

function $r = \text{CompHorner}(P, x)$
\[
\begin{align*}
  s_n &= a_i; \quad c_n = 0 \\
  \text{for} \ i = n - 1 : -1 : 0 \\
  [p_i, \pi_i] &= 2\text{Prod}(s_{i+1}, x) \\
  [s_i, \sigma_i] &= 2\text{Sum}(p_i, a_i) \\
  c_i &= c_{i+1} \otimes x \oplus (\pi_i \oplus \sigma_i)
\end{align*}
\]
end
\[
r = s_0 \oplus c_0
\]

function $r = \text{DDHorner}(P, x)$
\[
\begin{align*}
  sh_n &= a_i; \quad sl_n = 0 \\
  \text{for} \ i = n - 1 : -1 : 0 \\
  [ph_i, pl_i] &= [sh_{i+1}, sl_{i+1}] \otimes x \\
  [th, tl] &= 2\text{Prod}(sh_{i+1}, x) \\
  tl &= sl_{i+1} \otimes x \oplus tl \\
  [ph_i, pl_i] &= \text{Fast2Sum}(th, tl) \\
  [sh_i, sl_i] &= [ph_i, pl_i] \oplus a_i \\
  [th, tl] &= 2\text{Sum}(ph_i, a_i) \\
  tl &= tl \oplus pl_i \\
  [sh_i, sl_i] &= \text{Fast2Sum}(th, tl)
\end{align*}
\]
end
\[
r = sh_0
\]
Difference between DDHorner and CompHorner

function $r = \text{CompHorner}'(P, x)$
\begin{align*}
    s_n &= a_i; \quad c_n = 0 \\
    \text{for } i &= n - 1 : -1 : 0 \\
    [p_i, \pi_i] &= 2\text{Prod}(s_{i+1}, x) \\
    t_i &= c_{i+1} \otimes x \oplus \pi_i \\
    [s_i, \sigma_i] &= 2\text{Sum}(p_i, a_i) \\
    c_i &= t_i \oplus \sigma_i \\
\end{align*}
end

$r = s_0 \oplus c_0$

function $r = \text{DDHorner}(P, x)$
\begin{align*}
    s_h &= a_i; \quad s_l = 0 \\
    \text{for } i &= n - 1 : -1 : 0 \\
    &\% [p_h, p_l] = [s_{h+1}, s_{l+1}] \otimes x \\
    [t_h, t_l] &= 2\text{Prod}(s_{h+1}, x) \\
    t_l &= s_{l+1} \otimes x \oplus tl \\
    [p_h, p_l] &= \text{Fast2Sum}(t_h, t_l) \\
    \% [s_h, s_l] = [p_h, p_l] \oplus a_i \\
    [t_h, t_l] &= 2\text{Sum}(p_h, a_i) \\
    t_l &= t_l \oplus pl_i \\
    [s_h, s_l] &= \text{Fast2Sum}(t_h, t_l) \\
\end{align*}
end

$r = s_h_0$
We represent all data dependences in the inner loop of each algorithm.

More parallelism among floating point operations in CompHorner than in DDHorner.

Thus more potential ILP, and greater practical performance!