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 $\widehat{r}$ is a computed result, how to find a correcting term $\widehat{c}$ such that $\overline{r} = \widehat{r} \oplus \widehat{c}$ is more accurate than $\widehat{r}$?

- Aim of this presentation is:
  - to recall the principle of so-called compensated algorithms,
  - to present some details about the compensated Horner algorithm [1].

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

[1] S. Graillat, Ph. Langlois, N. Louvet. *Compensated Horner Scheme*, Research Report DALI-LP2A, Université de Perpignan, aug. 2005 (submitted)

Outline

Introduction

Why do we need compensated algorithms ?

Backward stable algorithms *vs.* condition number

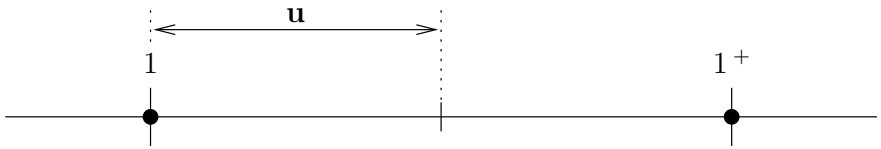
Backward stable algorithms:

- The accuracy of the computed solution satisfies

$$\text{accuracy} \lesssim \text{condition number} \times \mathbf{u},$$

  where

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



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

- 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 $\widehat{r}_0 = \text{Horner}\,(p, x)$

$\widehat{r}_n = a_n$

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

$\quad \widehat{r}_i = \widehat{r}_{i+1} \otimes x \oplus a_i$

end

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

$\widehat{r}_n = a_n$

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

  $\widehat{r}_i = \widehat{r}_{i+1} \otimes x \oplus a_i$

end

Relative accuracy of the evaluation with the Horner scheme:

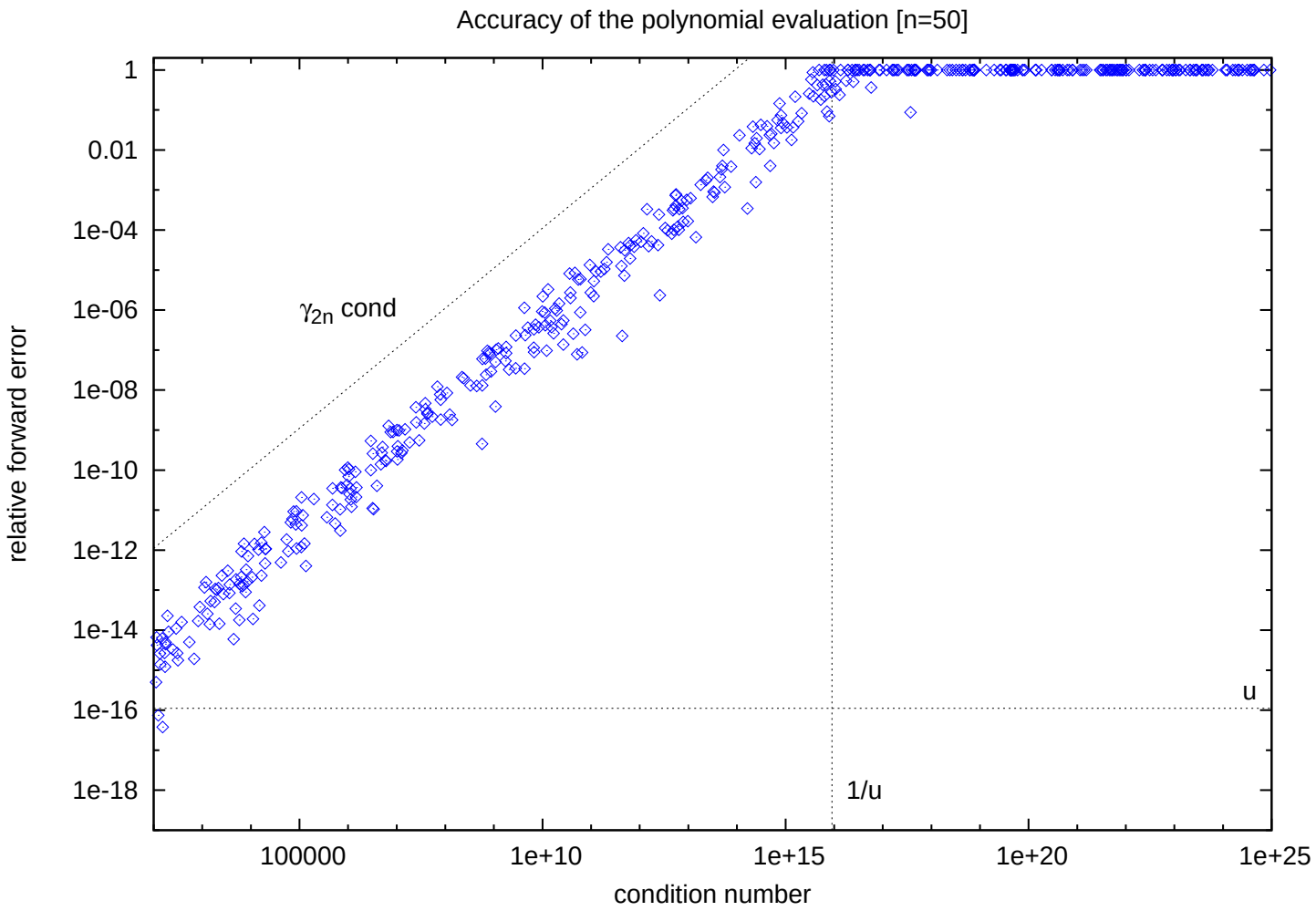
$$\frac{|\text{Horner}\,(p, x) - p(x)|}{|p(x)|} \leq \underbrace{\gamma_{2n}}_{\approx 2n\mathbf{u}} \text{cond}(p, x)$$

- $\mathbf{u}$ 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

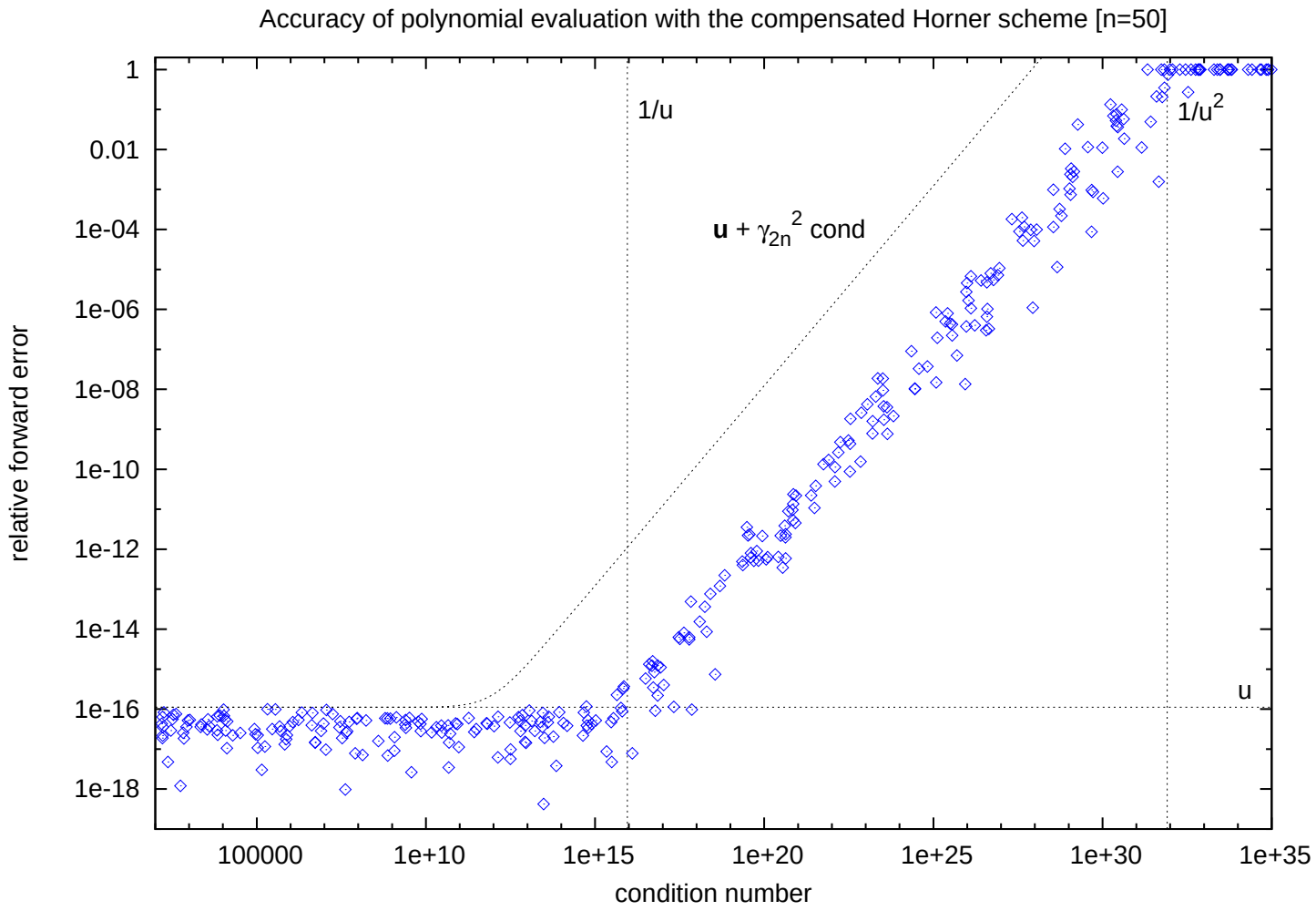
Algorithms that correct the generated rounding errors

- 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:

$$\text{accuracy} \lesssim \mathbf{u} + \text{condition number} \times \mathbf{u}^2$$

- 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 $\lesssim \mathbf{u} +$ condition number $\times\ \mathbf{u}^2$.



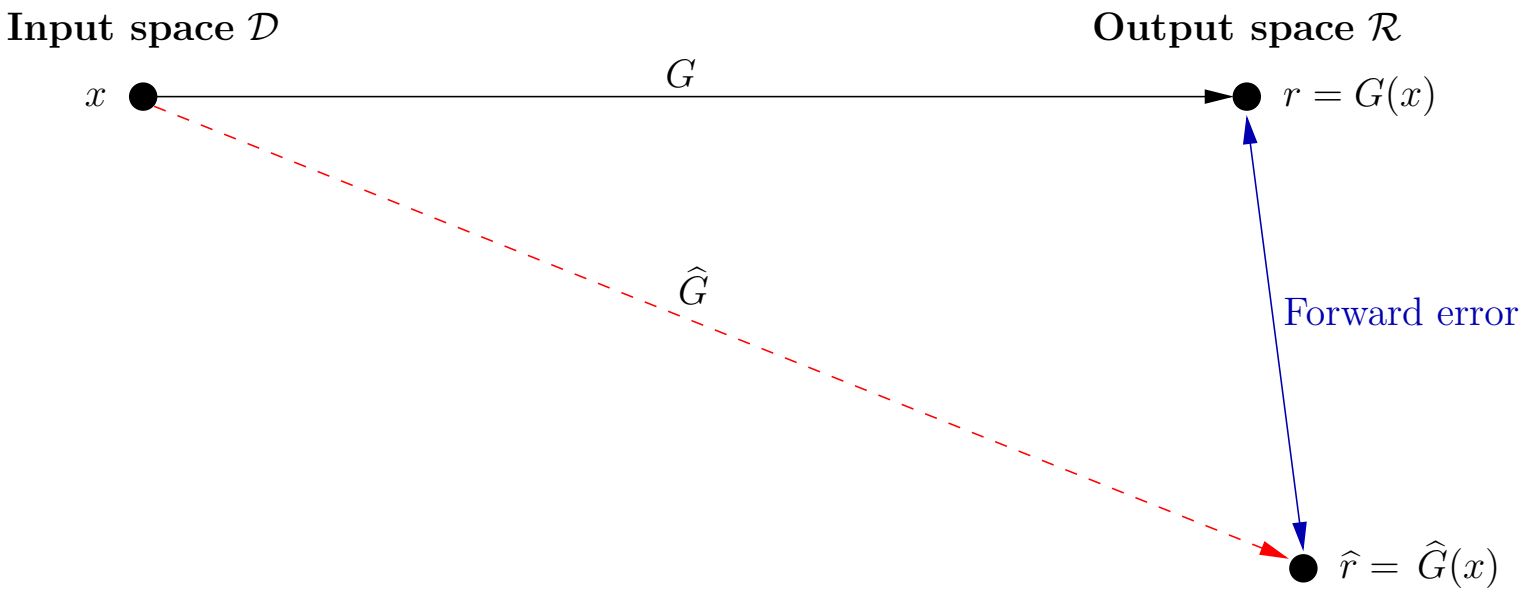
How is computed the compensated result?

Principle of compensated Algorithms

The compensated result is $\overline{r} = \widehat{r} \oplus \widehat{c}$. The correcting term $\widehat{c}$ is

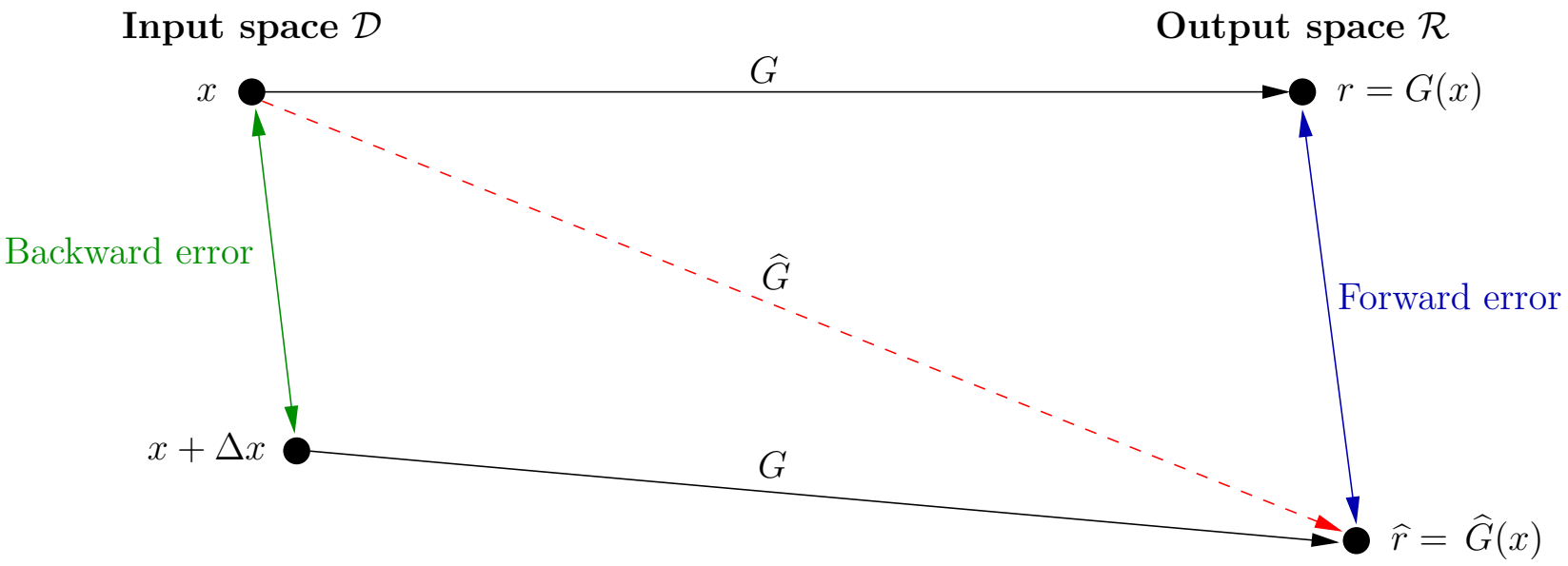
- an approximate of the forward error in $\widehat{r}$,
- computed thanks to "Error-Free Transformations".

Compensated result



- Forward error analysis

Compensated result

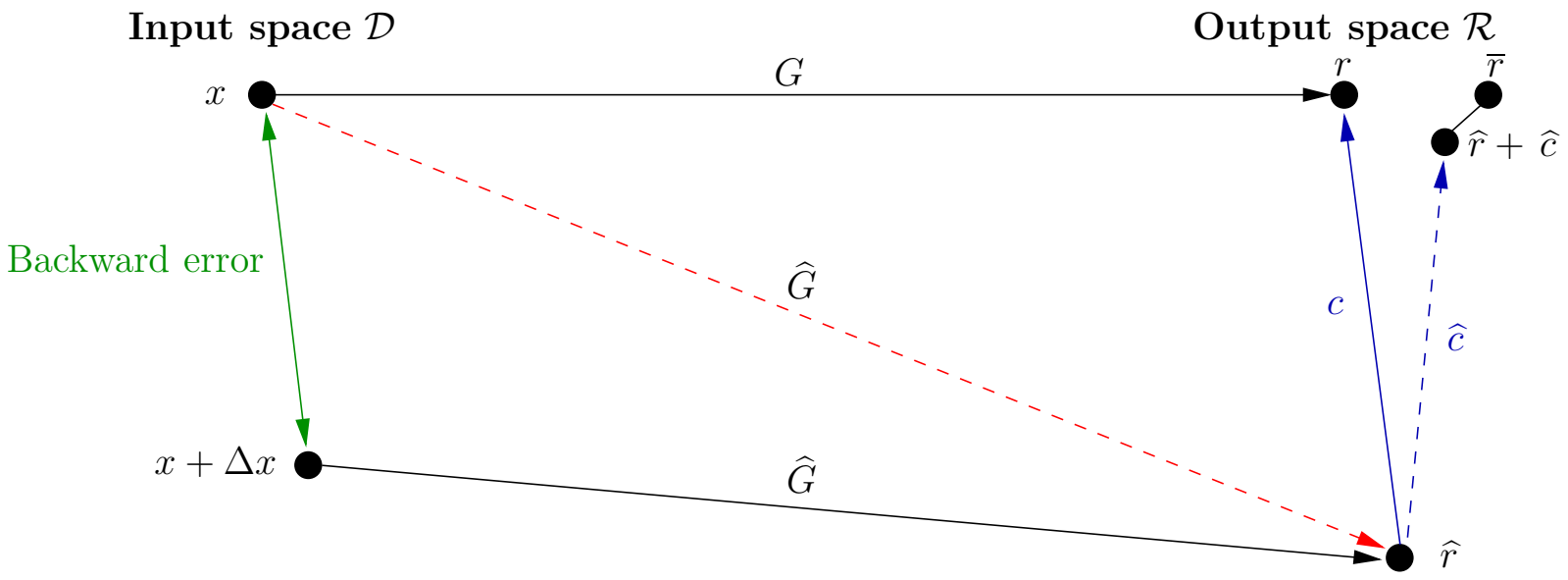


- Forward error analysis
- Backward error analysis

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

  $\widehat{r} = G(x + \Delta x)$

Compensated result



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

- First, compute an approximate $\widehat{c}$ of the forward error $c = r - \widehat{r}$.
  $\widehat{c}$ is a correcting term for $\widehat{r}$.
- Then, a compensated result $\overline{r} = \widehat{r} \oplus \widehat{c}$.

How to compute the correcting term $\widehat{c}$?

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

- 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

$$\text{Horner}\,(p, x) = \sum_{i=0}^{n} (1 + \delta_i) a_i x^i, \quad \text{with} \quad |\delta_i| \leq \gamma_{2n} \approx 2n\mathbf{u}$$

  - Forward error result:

$$c = p(x) - \text{Horner}\,(p, x) = -\sum_{i=0}^{n} \delta_i a_i x^i \quad \Rightarrow \quad |c| \leq \gamma_{2n} \sum_{i=0}^{n} |a_i||x|^i.$$

As the δ_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.

$+$	$(x, y) = 2\mathrm{Sum}\,(a, b)$ such that $a + b = x + y$ and $x = a \oplus b$	6 flops	Knuth (74)
$\times$	$(x, y) = 2\mathrm{Prod}\,(a, b)$ such that $a \times b = x + y$ and $x = a \otimes b$	17 flops	Veltkamp Dekker (71)

with $a, b, x, y \in \mathbb{F}$.

How to compute the correcting term $\widehat{c}$ thanks to EFT?

Algorithm (Horner scheme)

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

$\widehat{r}_n = a_n$

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

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

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

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

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

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

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

How to compute the correcting term $\widehat{c}$ thanks to EFT?

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

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

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

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

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

- 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 $\widehat{c}$.

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

$\widehat{r}_n = a_n$

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

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

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

$\quad p_\pi[i] = \pi_i;\ p_\sigma[i] = \sigma_i$

This algorithm is an EFT for the Horner algorithm since

$$p(x) = \text{Horner}\,(p, x) + \underbrace{(p_\pi + p_\sigma)(x)}_{=c}.$$

Compensated Horner Algorithm

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

Algorithm (Compensated Horner scheme)

function $\overline{r} = \text{CompHorner}\,(p, x)$

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

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

$\overline{r} = \widehat{r} \oplus \widehat{c}$

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

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

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,

$$p(x) = \text{Horner}\,(p, x) + \underbrace{(p_\pi + p_\sigma)(x)}_{=\mathsf{c}}.$$

Sketch of proof for the compensated Horner algorithm

We recall:

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

Since the compensated result is $\overline{r} = \widehat{r} \oplus \widehat{c}$,

$$|p(x) - \overline{r}| = |p(x) - (1+\varepsilon)(\widehat{r} + \widehat{c})|, \quad \text{with} \quad |\varepsilon| \leq \mathbf{u}.$$

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

$$\begin{aligned} |p(x) - \overline{r}| &= |p(x) - (1+\varepsilon)(p(x) + \widehat{c} - c)| \\ &\leq \mathbf{u}|p(x)| + (1+\mathbf{u})|\widehat{c} - c| \end{aligned}$$

How can we bound $|c - \widehat{c}|$?

Sketch of proof for the compensated Horner algorithm

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

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

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,

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

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

$$|c - \widehat{c}| \leq \gamma_{2n-1}\gamma_{2n}\,\widetilde{p}(x).$$

Nota:

$$\gamma_k = \frac{k\mathbf{u}}{1 - k\mathbf{u}} = k\mathbf{u} + \mathcal{O}\mathbf{u}^2.$$

Sketch of proof for the compensated Horner algorithm

Then,

$$\begin{aligned} |p(x) - \overline{r}| &\leq \mathbf{u}|p(x)| + (1+\mathbf{u})\gamma_{2n-1}\gamma_{2n}\,\widetilde{p}(x), \\ &\leq \mathbf{u}|p(x)| + \gamma_{2n}^2\,\widetilde{p}(x). \end{aligned}$$

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

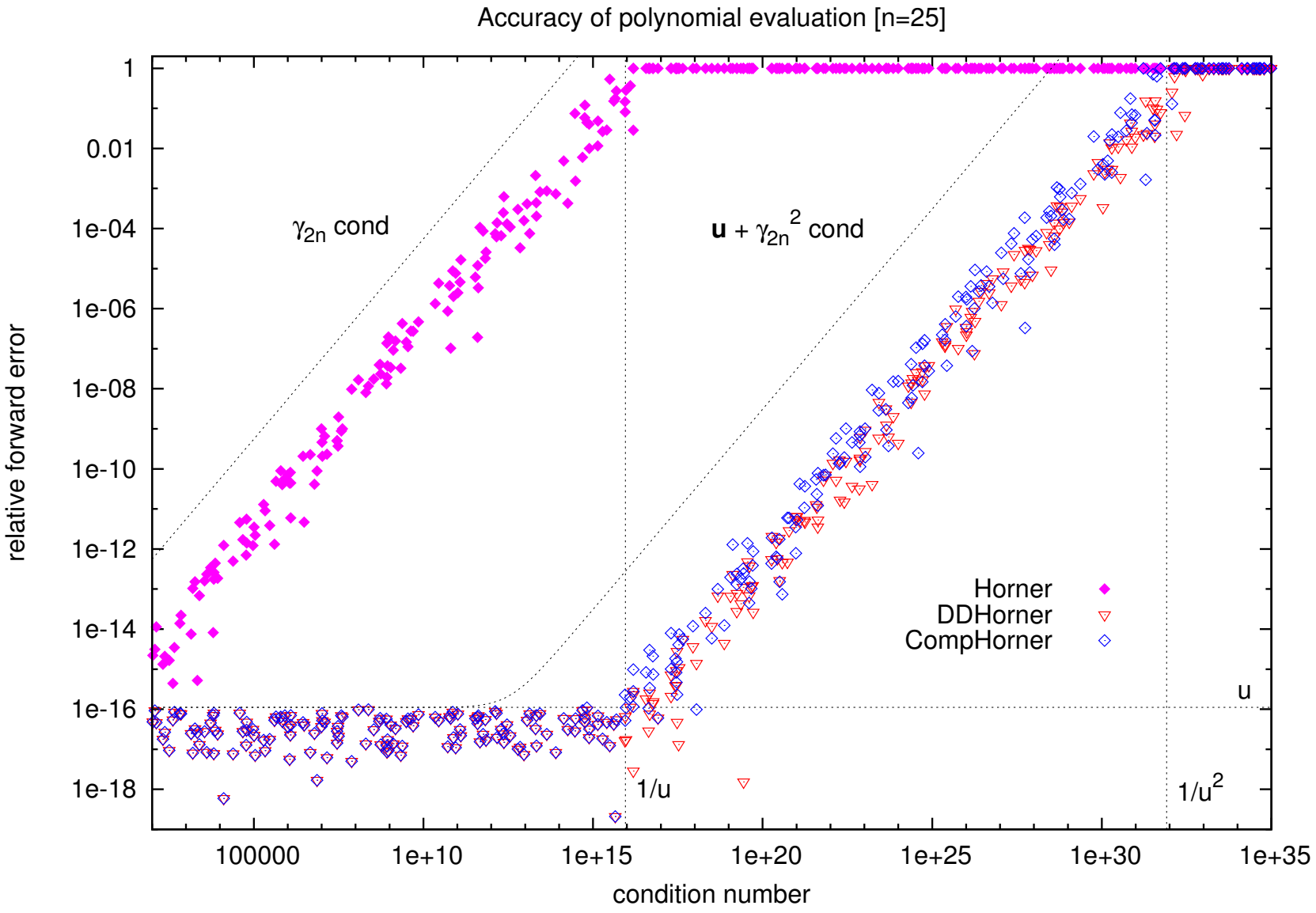
Theorem

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

$$\frac{|p(x) - \overline{r}|}{|p(x)|} \leq \mathbf{u} + \gamma_{2n}^2\,\mathrm{cond}(p,x).$$

Again, $\gamma_{2n}^2 \approx 4n^2\mathbf{u}^2$.

Accuracy of the result $\lesssim \mathbf{u} +$ condition number $\times\ \mathbf{u}^2$.



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.

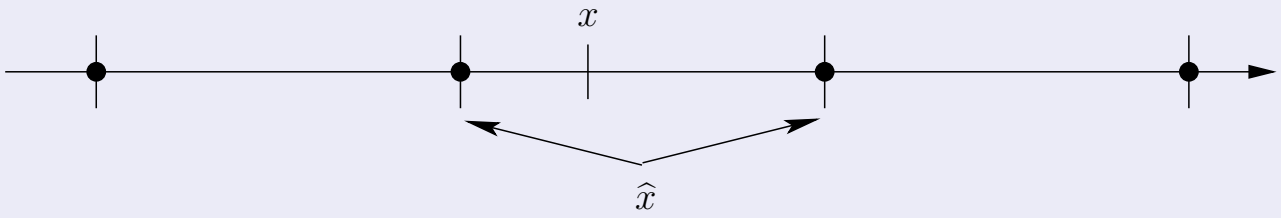
Faithful rounding with the CHS

Faithful rounding

Definition

A floating point number $\widehat{x}$ is said to be a faithful rounding of a real number x if

- either $\widehat{x} = x$,
- or $\widehat{x}$ is one of the two floating point neighbours of x.



The error bound

$$\frac{|\text{CompHorner}\,(p,x) - p(x)|}{|p(x)|} \leq \mathbf{u} + \underbrace{\gamma_{2n}^2}_{2n^2\mathbf{u}^2} \text{cond}(p,x)$$

is too large for reasoning about faithful rounding.

An *a posteriori* test

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

- The following error bound on the computed correcting term holds:

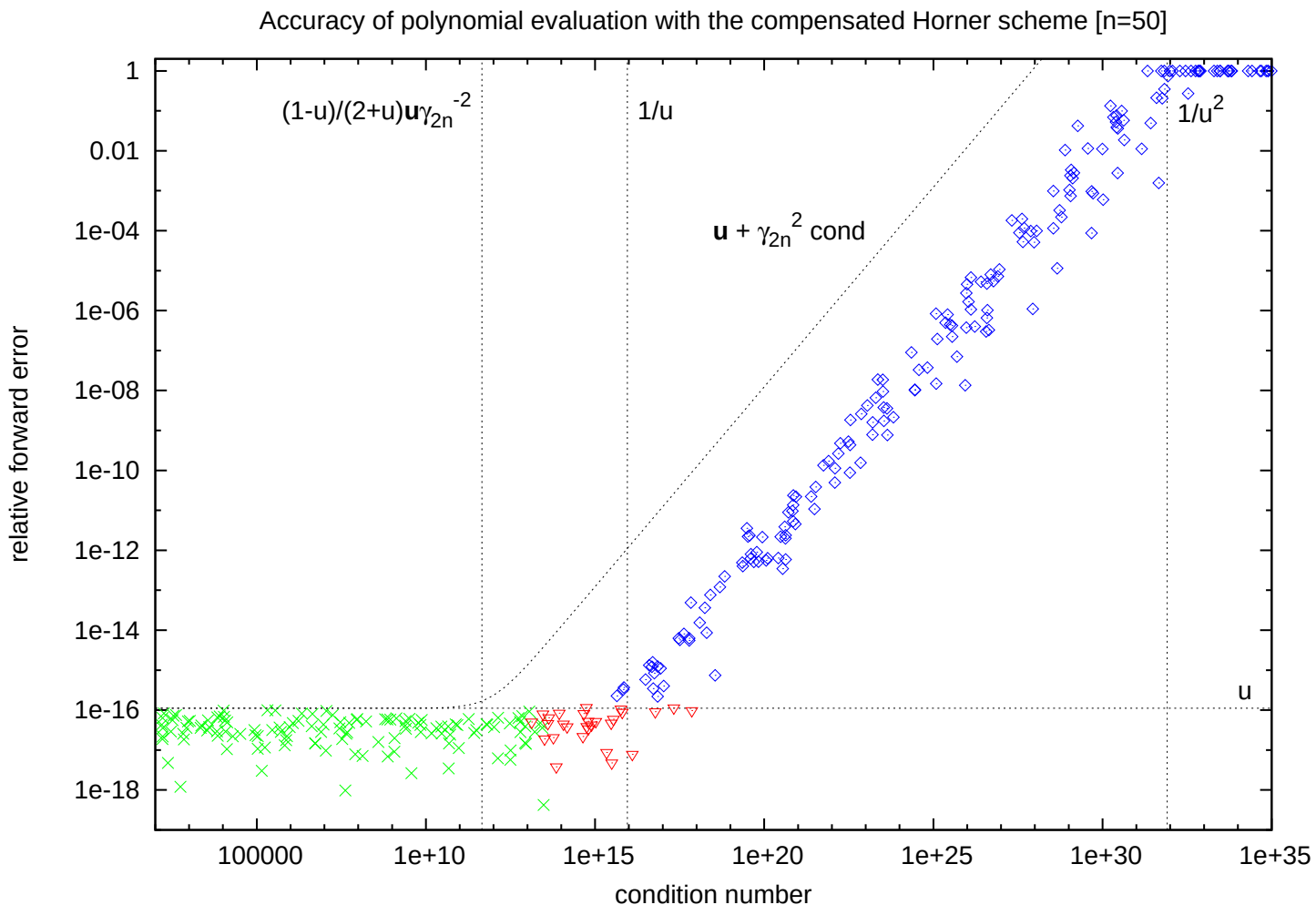
$$|c - \widehat{c}| \leq \text{fl}\left(\frac{\widehat{\gamma}_{2n-1}\text{Horner}\,(|p_\pi| \oplus |p_\sigma|, |x|)}{1 - 2(n+1)\mathbf{u}}\right) =: \widehat{\beta}.$$

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

Theorem

$$\widehat{\beta} < \frac{\mathbf{u}}{2}|\overline{r}| \Rightarrow |c - \widehat{c}| < \frac{\mathbf{u}}{2}|\overline{r}| \Rightarrow \overline{r} \textit{ is a faithful rounding of } p(x).$$

An *a posteriori* test



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]):

		$\frac{\text{CompHorner}}{\text{Horner}}$	$\frac{\text{CompHornerIsFaith}}{\text{Horner}}$	$\frac{\text{DDHorner}}{\text{Horner}}$
Pentium 4, 3.00 GHz	GCC 3.3.5	3.77	5.52	10.00
	ICC 9.1	3.06	5.31	8.88
Athlon 64, 2.00 GHz	GCC 4.0.1	3.89	4.43	10.48
Itanium 2, 1.4 GHz	GCC 3.4.6	3.64	4.59	5.50
	ICC 9.1	1.87	2.30	8.78
		$\sim \mathbf{2-4}$	$\sim \mathbf{4-6}$	$\sim \mathbf{5-10}$

[2] Average ratios for polynomials of degree 5 to 200.

Some comparisons

How does the more accurate algorithms compare to each other?

		$\frac{\text{CompHornerIsFaith}}{\text{CompHorner}}$	$\frac{\text{DDHorner}}{\text{CompHorner}}$	$\frac{\text{DDHorner}}{\text{CompHornerIsFaith}}$
Pentium 4, 3.00 GHz	GCC 3.3.5	1.46	2.66	1.89
	ICC 9.1	1.71	2.92	1.72
Athlon 64, 2.00 GHz	GCC 4.0.1	1.14	2.70	2.38
Itanium 2, 1.4 GHz	GCC 3.4.6	1.27	1.51	1.27
	ICC 9.1	1.24	4.67	3.84
		$\leq \mathbf{2}$	$\sim \mathbf{2-5}$	$\sim \mathbf{2-5}$

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 developped 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.

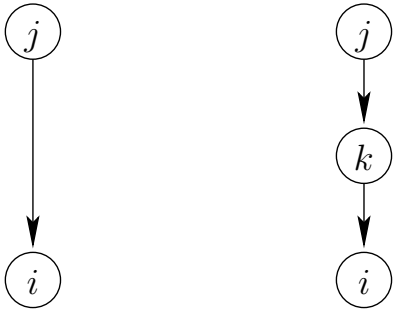
  How to determine wheter an instruction is dependent on another?

Dependences between instructions

- Three different types of dependences:
  - control dependences,
  - name dependences,
  - data dependences (or true dependences).
- A control dependence determine 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 produce a result that may be used by instruction i,
  - their exist 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 = CompHorner(P, x)
  s_n = a_i ;  c_n = 0
  for i = n-1 : -1 : 0

    [p_i , π_i] = 2Prod(s_{i+1}, x)

    [s_i , σ_i] = 2Sum(p_i , a_i)
    c_i = c_{i+1} ⊗ x ⊕ (π_i ⊕ σ_i)

  end
  r = s_0 ⊕ c_0
```

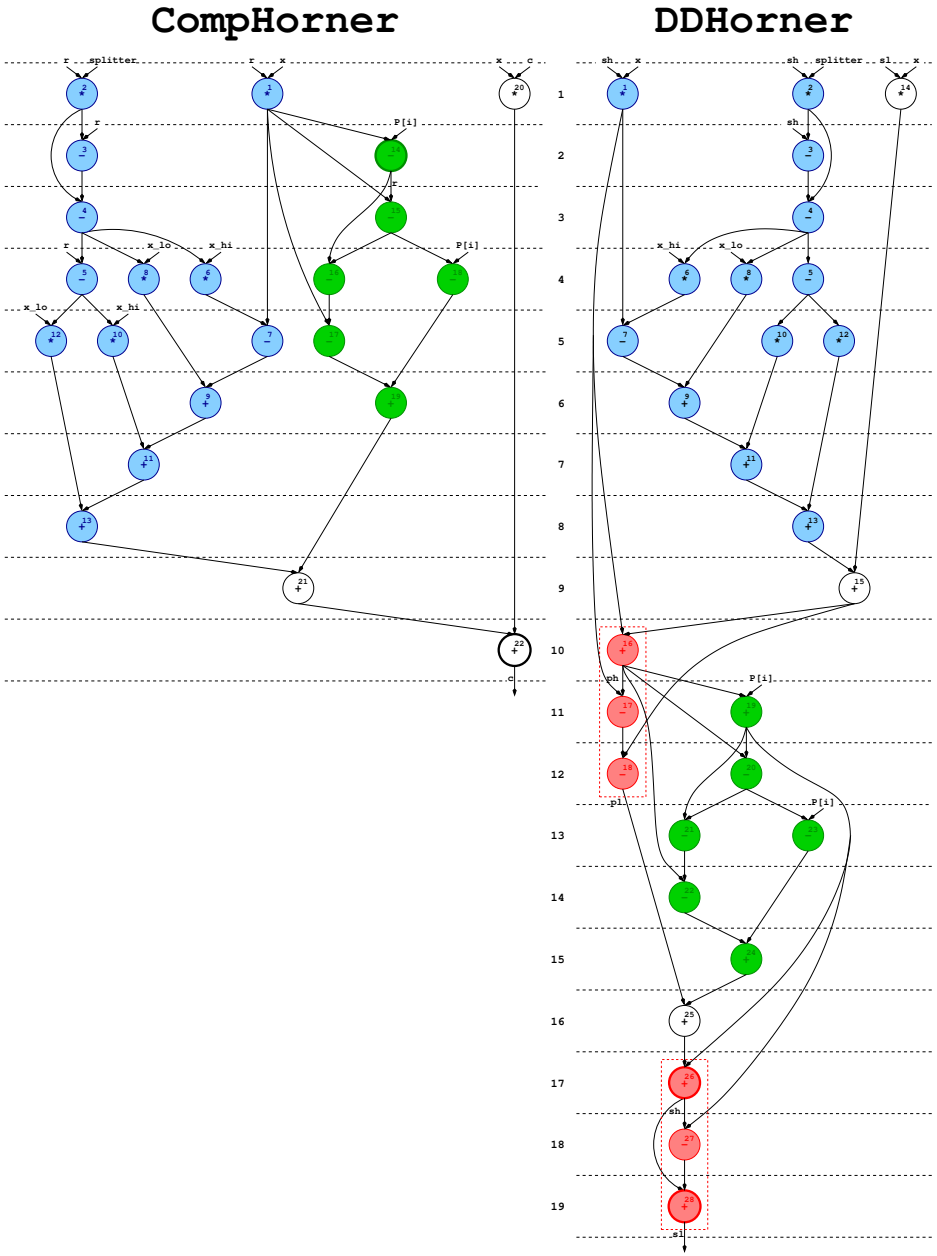
```
function r = DDHorner(P, x)
  sh_n = a_i ;  sl_n = 0
  for i = n-1 : -1 : 0
    %% [ph_i , pl_i]=[sh_{i+1}, sl_{i+1}] ⊗ x
    [th, tl] = 2Prod(sh_{i+1}, x)
    tl = sl_{i+1} ⊗ x ⊕ tl
    [ph_i , pl_i] = Fast2Sum(th, tl)

    %% [sh_i , sl_i]=[ph_i , pl_i] ⊕ a_i
    [th, tl] = 2Sum(ph_i , a_i)
    tl = tl ⊕ pl_i
    [sh_i , sl_i] = Fast2Sum(th, tl)
  end
  r = sh_0
```

Difference between DDHorner and CompHorner

```
function r = CompHorner'(P, x)
  s_n = a_i ;  c_n = 0
  for i = n-1 : -1 : 0

    [p_i , π_i] = 2Prod(s_{i+1}, x)
    t_i = c_{i+1} ⊗ x ⊕ π_i



    [s_i , σ_i] = 2Sum(p_i , a_i)
    c_i = t_i ⊕ σ_i

  end
  r = s_0 ⊕ c_0
```

```
function r = DDHorner(P, x)
  sh_n = a_i ;  sl_n = 0
  for i = n-1 : -1 : 0
    %% [ph_i , pl_i]=[sh_{i+1} , sl_{i+1}] ⊗ x
    [th, tl] = 2Prod(sh_{i+1}, x)
    tl = sl_{i+1} ⊗ x ⊕ tl
    [ph_i , pl_i] = Fast2Sum(th, tl)

    %% [sh_i , sl_i]=[ph_i , pl_i] ⊕ a_i
    [th, tl] = 2Sum(ph_i , a_i)
    tl = tl ⊕ pl_i
    [sh_i , sl_i] = Fast2Sum(th, tl)
  end
  r = sh_0
```

Difference in the data flow graph



- 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 pratical performance!