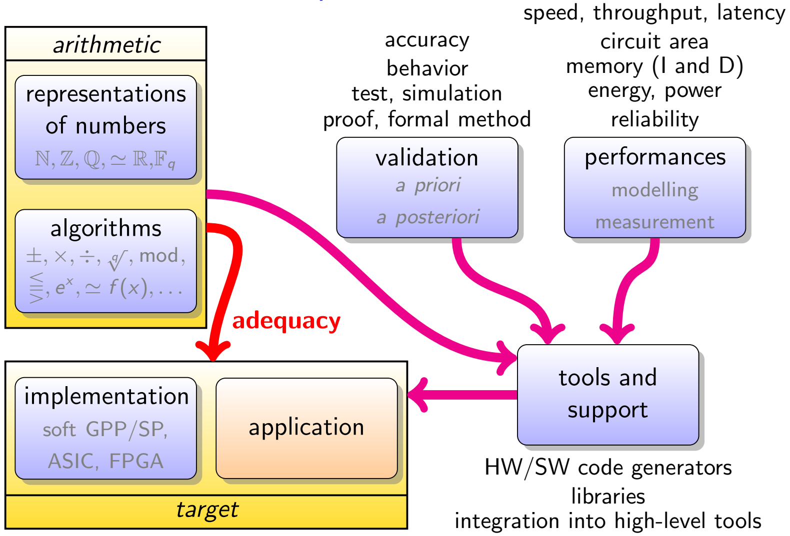
Optimizing Polynomial Approximations for Function Evaluation

Arnaud Tisserand

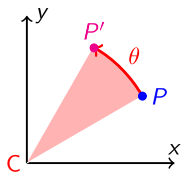
CNRS

ARCAD Meeting, January 2023

Computer Arithmetic



Motivations

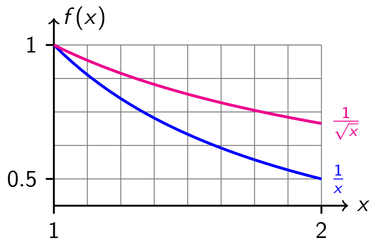


$$P' = \mathcal{R}(P, C, \theta)$$

$$\begin{pmatrix} x' \\ y' \end{pmatrix} = \begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix}$$

Initial approximations for floating-point units:

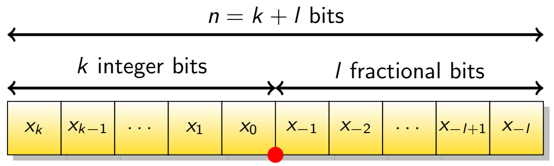
- reciprocal $\frac{1}{x}$ for division
- inverse square root $\frac{1}{\sqrt{x}}$ for square root



and many other functions: $\exp(x), \log(x), \arctan(x), \cosh(x), \ldots$

Radix-2 Representations of Values

- Fixed-point format (kQl):

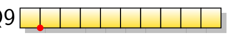
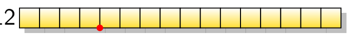


$$= \sum_{i=-l}^{k} x_i 2^i = X$$

- Representation R:

$$X = (x_{k-1}x_{k-2}\ldots x_1 x_0 . x_{-1}x_{-2}\ldots x_{l-1}x_l)_{\mathrm{R}}$$

Examples:

- $()_2$ binary representation, $x_i \in \{0, 1\}$
  e.g. $3.125 = (11.001)_2$
- $(\ )_{\mathrm{bs}}$ *borrow-save* redundant representation $x_i \in \{-1, 0, 1\}$, $-1 = \bar{1}$
  e.g. $31 = (11111.0)_2 = (10000\bar{1}.0)_{\mathrm{bs}}$
- 1Q9  4Q12 

Error and Accuracy

Question: how many bits are correct ?

$$\begin{cases} x_t & = (1.000\,000\,00)_2 & \textit{theoretical value} \\ x_c & = (0.111\,111\,11)_2 & \textit{value in the circuit} \\ |x_t - x_c| & = (0.000\,000\,01)_2 = 2^{-8} & \end{cases}$$

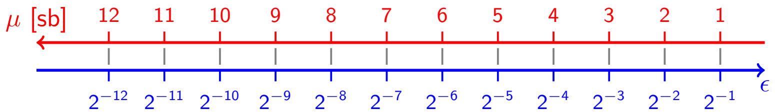
Error, ϵ: distance between 2 objects (e.g. $\epsilon = ||f(x) - p(x)||$)

Accuracy, μ: (fractional) number of bits required to represent values with an error $\leq \epsilon$

$$\mu = -\log_2 |\epsilon|$$

Notation: μ expressed in terms of correct or significant bits ([cb], [sb])

Example: error $\epsilon = 0.0000107$ is equivalent to accuracy $\mu = 16.5$ sb

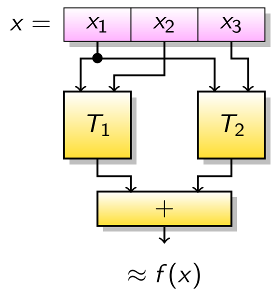


Function Evaluation Methods

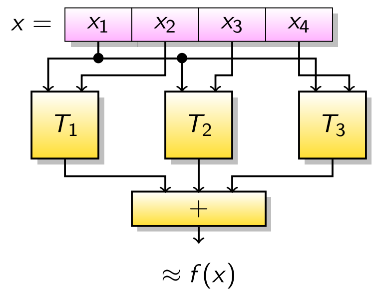
- Table based approximations
  - HW: require tables, $\pm$ (and possibly very small $\times_{\text{cst}}$)
  - ☺ very high throughput
  - ☹ large silicon area (limited to small accuracy)
- Shift and add algorithms (e.g. CORDIC)
  - HW: require $\pm$ and very small tables
  - ☺ small silicon area
  - ☺ scalable and flexible for multiple functions evaluation
  - ☹ long latency ($T(n) = O(n)$)
- Polynomial or rational approximations
  - HW: require $\pm$, $\times$ (possibly small tables for coefficients storage)
  - ☺ simple architecture
  - ☺ resource sharing for multiple functions evaluation
  - ☹ large silicon area due to multipliers

Table Based Approximations

Bipartite method:



Multipartite method:



Reference:
F. de Dinechin and A. Tisserand, *Multipartite Table Methods*, IEEE Transactions on Computers, March 2005, vol. 53, n. 3, pp. 319–330, DOI: 10.1109/TC.2005.54

Shift and Add Algorithms

CORDIC: COordinate Rotation DIgital Computer (H. Briggs 1624, J. Volder 1959 and S. Walther 1971), used for function approximation, DFT, filters, linear algebra (syst. solving, SVD), DDFS...

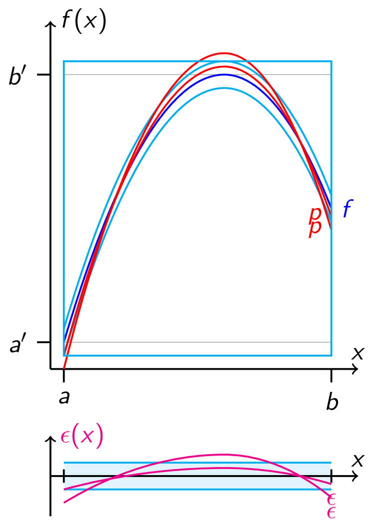
$$\begin{cases} x_{n+1} & = & x_n - m d_n y_n 2^{-\sigma(n)} \\ y_{n+1} & = & y_n + d_n x_n 2^{-\sigma(n)} \\ z_{n+1} & = & z_n - w_{\sigma(n)} \end{cases}$$

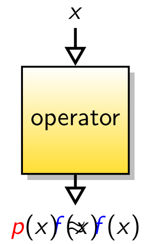
Some possible evaluation modes (depends on the configuration):

$$\begin{cases} x_n & \rightarrow & K(x_0 \cos z_0 - y_0 \sin z_0) \\ x_n & \rightarrow & K'(x_1 \cosh z_1 + y_1 \sinh z_0) \\ x_n & \rightarrow & K\sqrt{x_0^2 + y_0^2} \end{cases} \qquad \begin{cases} y_n & \rightarrow & y_0 + x_0 z_0 \\ z_n & \rightarrow & z_0 - \arctan \frac{y_0}{x_0} \\ z_n & \rightarrow & z_0 - \frac{y_0}{x_0} \\ z_n & \rightarrow & z_1 - \tanh^{-1} \frac{y_1}{x_1} \end{cases}$$

where $m \in \{0, 1\}$, $d_n \in \{\text{sign}(z_n), -\text{sign}(y_n)\}$,
$w_k \in \{\arctan(2^{-k}), 2^{-k}, \tanh^{-1}(w^{-k})\}$ are tabulated values and $\sigma(n) \in \{n, n-k\}$
where k is the largest integer s.t. $3^{k+1} + 2k - 1 \leq 2n$

Polynomial Approximations





x argument

[a, b] domain

f function

p polynomial

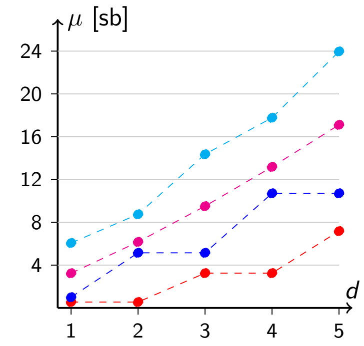
Question: what is the best p?

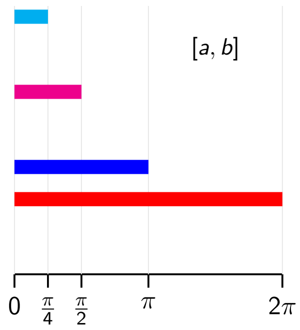
$\epsilon(x) = f(x) - p(x)$ — *ϵ approx. error*

$\epsilon(x) \leq \epsilon_{\text{target}}$ — ϵ_{target} *maximum allowed error*

Accuracy, Degree and Evaluation Cost

Degree-d minimax approximation polynomials to $\sin(x)$ with $x \in [a, b]$:





- higher accuracy ⟹ higher degree
- higher degree ⟹ more costly evaluation

Polynomial Evaluation Schemes

scheme	computations	# ±	# ×
direct	$p_0 + p_1x + p_2x^2 + p_3x^3$	3	5
Horner	$p_0 + (p_1 + (p_2 + p_3x)x)x$	3	3
Estrin	$p_0 + p_1x + (p_2 + p_3x)x^2$	3	4

Trade-off:

- direct scheme ⟶ high operation cost and smaller accuracy
- Horner scheme ⟶ smallest cost but sequential
- Estrin scheme ⟶ some internal parallelism

Question: what is the best evaluation scheme?

Round-off Errors

Round-off errors occur during most of computations:

- due to the finite accuracy during the computations
- small for a single operation (fraction of the LSB)
- accumulation of such errors may be a problem in long computation sequences
- need for a sufficient datapath width in order to limit round-off errors

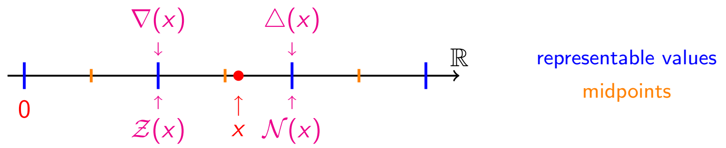
Examples: $1/3 = 0.33333333\ldots \to 0.3333$ or 0.3334 in $1\mathrm{Q}_{10}4$ format

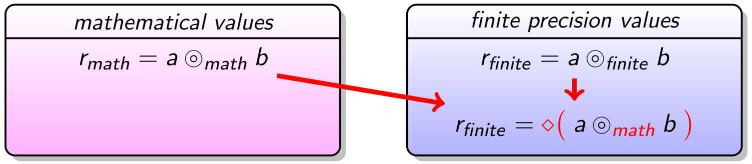
Question: what is the best datapath width?

Rounding Modes and Correct Rounding

Notations:

- $\circledcirc$ is an operation $\pm, \times, \div \ldots$
- $\diamond$ is the active rounding mode (or quantization mode)
  IEEE-754: $\triangle(x)$ towards $+\infty$ (up), $\nabla(x)$ towards $-\infty$ (down), $\mathcal{Z}(x)$ towards 0, $\mathcal{N}(x)$ towards the nearest





Bounding Round-off Errors

Problem: it is very difficult to get tight bounds

Solutions:

- worst case: assume 1/2 LSB error for each operation
  $\rightsquigarrow$ simple but very pessimistic
- qualification: exhaustive or selected simulations
  $\rightsquigarrow$ simple but only validated bounds for small systems
- specific tools: formal accurate analysis (and proof)
  $\rightsquigarrow$ we use gappa developed by Guillaume Melquiond

Gappa Overview

- developed by Guillaume Melquiond
- goal: formal verification of the correctness of numerical programs:
  - software and hardware
  - integer, floating-point and fixed-point arithmetic ($\pm$, $\times$, $\div$, $\sqrt{}$)
- uses multiple-precision interval arithmetic, forward error analysis and expression rewriting to bound mathematical expressions (rounded and exact operators)
- generates a theorem and its proof which can be automatically checked using a proof assistant (e.g. Coq or HOL Light)
- reports tight error bounds for given expressions in a given domain
- C++ code and free software licence (CeCILL $\simeq$ GPL)
- publication: ACM Transactions on Mathematical Software, n. 1, vol. 37, 2010, pp: 2:1–20, doi: 10.1145/1644001.1644003
- source code and doc: `http://gappa.gforge.inria.fr/`

Gappa Example

Degree-2 polynomial approximation to e^x over $[1/2, 1]$ and format 1Q9:

```
p0 = 571/512;    p1 = 275/512;    p2 = 545/512;

x = fixed<-9,dn>(Mx);

y1 fixed<-9,dn>= p2 * x + p1;
p  fixed<-9,dn>= y1 * x + p0;

Mp = (p2 * Mx + p1) * Mx + p0;

{
    Mx in [0.5,1]   /\   |Mp-Mf| in [0,0.001385]
->
    |p-Mf| in ?
}
```

Gappa-0.14.0 result ($[a, b]$, $x\{(\approx x)_{10}, \log_2 x\}$, $x\text{b}y = x2^y$):

```
Results for Mx in [0.5, 1] and |Mp - Mf| in [0, 0.001385]:
|p - Mf| in [0, 193518932894171697b-64 {0.0104907, 2^(-6.57475)}]
```

Still Pending Questions

Question: what is the best (or a good) p?

- mathematical p: *minimax approximations*
- implemented p: simple selection of representable coefficients
- links to other methods and tools

Question: what is the best (or a good) datapath width?

- basic optimization method
- better heuristics under development...

Question: what is the best (or a good) evaluation scheme?

- Horner or specific scheme examples...
- work still in progress...

Minimax Polynomial Approximations

- approximation error $\epsilon_{\text{app}} = ||f - p||_\infty = \max_{a \le x \le b} |f(x) - p(x)|$
- minimax polynomial approximation to f over $[a, b]$ is p^* such that:

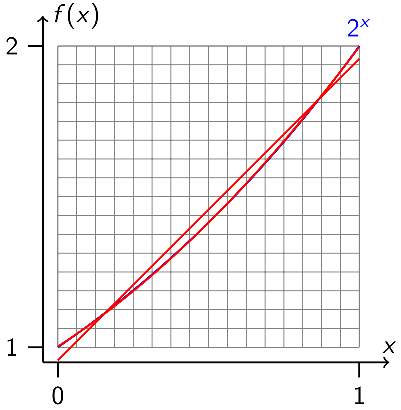
$$||f - p^*||_\infty = \min_{p \in \mathcal{P}_d} ||f - p||_\infty$$

- $\mathcal{P}_d$ set of polynomials with real coefficients and degree $\le d$
- p^* computed using an algorithm from Remez (numerically implemented in Maple, Matlab, sollya...)

Problems:

- p^* coefficients in $\mathbb{R} \Longrightarrow$ conversion to finite precision
- during p^* evaluation, some round-off errors add up to ϵ_{app}

Example $f(x) = 2^x$ and $x \in [0, 1]$



d	μ [sb]	ϵ_{app}
1	4.53	4.31×10^{-2}
2	8.65	2.48×10^{-3}
3	13.18	1.08×10^{-4}
4	18.04	3.71×10^{-6}
5	23.15	1.07×10^{-7}

$p^* = 0.999892965 + x \times (0.696457394 + x \times (0.224338364 +$
$p^* = 0.956964656 + 1.000065104780 + x \times 0.344001106)$
$x \times (0.079204240) + x \times 0.013697664)))$

Finite Precision Coefficients Selection Problem

Example: $f(x) = e^x$ over $[1/2, 1]$ with $d = 2$, the `remez` function from `sollya` gives:

$$p^* = 1.116019297\ldots + 0.535470348\ldots \times x + 1.065407185\ldots \times x^2$$

Question: what are "good" representable values for p_0, p_1 and p_2?

Problem: p^* is the best theoretical approximation to f (i.e. $p_i \in \mathbb{R}$)

Need: find good approximations with "machine-representable" coefficients

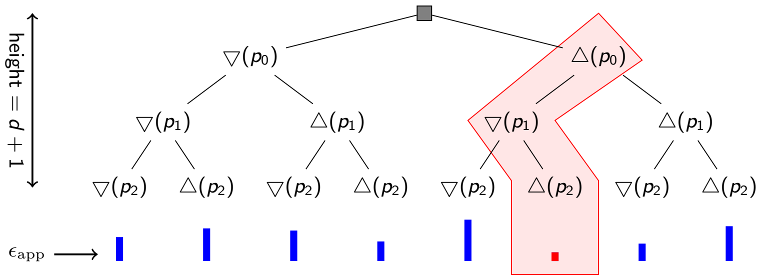
Above example with 1Q9 format (all values for domain $[1/2, 1]$):

- $\epsilon_{\text{app}} = ||f - p^*||_\infty \simeq 1.385 \times 10^{-3} \rightsquigarrow \simeq 9.4$ sb
- $\frac{571}{512} + \frac{137}{256}x + \frac{545}{512}x^2 \rightsquigarrow 8.1$ sb $\quad (\forall i$ use $\mathcal{N}(p_i))$
- $\frac{571}{512} + \frac{275}{512}x + \frac{545}{512}x^2 \rightsquigarrow 9.3$ sb $\quad$ (best selection)

Basic Coefficient Selection Method

Idea: search among all the rounding modes for all the p_i^*

- round up $p_i = \triangle(p_i^*)$, round down $p_i = \triangledown(p_i^*)$
- 2 values per coeff. $\Longrightarrow$ total of 2^{d+1} values (but d is small)
- for each polynomial p evaluate $\epsilon_{\text{app}} = ||f - p||_\infty$, then select polynomial(s) with the smallest ϵ_{app}



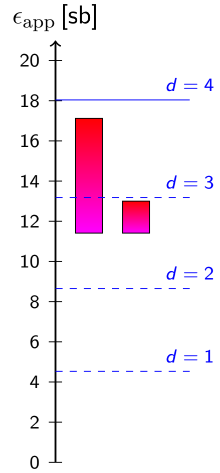
Result: $p(x) = \sum_{i=0}^{d} p_i x^i$ where all p_i are representable in target format

Example for $f(x) = 2^x$, $x \in [0, 1]$ and $d = 4$

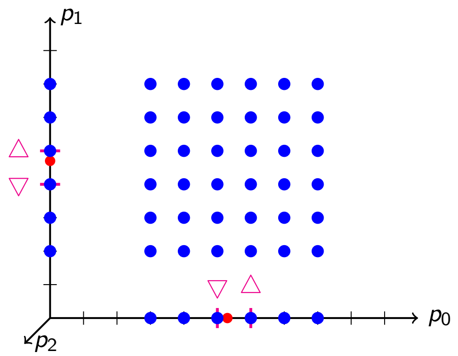
$\epsilon_{\text{app}}(p^*) \rightsquigarrow$ 18.04 sb

p	$\epsilon_{\text{app}}(p)$	p	$\epsilon_{\text{app}}(p)$
(▽, ▽, ▽, ▽, ▽)	12.00	(▽, ▽, ▽, ▽, △)	13.00
(▽, ▽, ▽, △, ▽)	13.00	(▽, ▽, ▽, △, △)	14.03
(▽, ▽, △, ▽, ▽)	13.00	(▽, ▽, △, ▽, △)	14.55
(▽, ▽, △, △, ▽)	14.99	(▽, ▽, △, △, △)	13.00
(▽, △, ▽, ▽, ▽)	13.00	(▽, △, ▽, ▽, △)	16.13
(▽, △, ▽, △, ▽)	17.12	(▽, △, ▽, △, △)	13.00
(▽, △, △, ▽, ▽)	15.71	(▽, △, △, ▽, △)	13.00
(▽, △, △, △, ▽)	13.00	(▽, △, △, △, △)	12.00
(△, ▽, ▽, ▽, ▽)	13.00	(△, ▽, ▽, ▽, △)	13.00
(△, ▽, ▽, △, ▽)	13.00	(△, ▽, ▽, △, △)	13.00
(△, ▽, △, ▽, ▽)	13.00	(△, ▽, △, ▽, △)	13.00
(△, ▽, △, △, ▽)	12.99	(△, ▽, △, △, △)	12.00
(△, △, ▽, ▽, ▽)	12.99	(△, △, ▽, ▽, △)	12.98
(△, △, ▽, △, ▽)	12.91	(△, △, ▽, △, △)	12.00
(△, △, △, ▽, ▽)	12.79	(△, △, △, ▽, △)	12.00
(△, △, △, △, ▽)	12.00	(△, △, △, △, △)	11.41

p represented by $(p_0, p_1, p_2, p_3, p_4)$



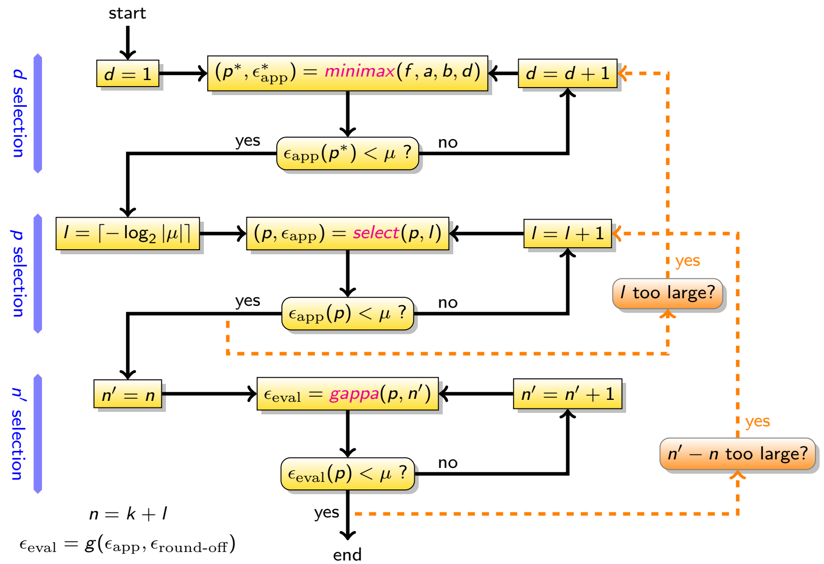
Improved Coefficient Selection Methods



Other selection methods:

- linear programming methods, e.g. `meplib` software `https://lipforge.ens-lyon.fr/projects/meplib/`
- euclidean lattices reduction (LLL), e.g. `sollya` software `http://sollya.gforge.inria.fr/`

Method Summary



Example: 2^x over $[0,1]$ and $\mu \leq 12$ sb (1/2)

Let us try with $d = 3$ (max. theoretical accuracy 13.18 sb):
$p^*(x) = 0.999892965 + 0.696457394x + 0.224338364x^2 + 0.079204240x^3$

Coefficients (fractional part) size selection:

l	12	13	14	15	16
ϵ_{app}	12.38	12.45	13.00	13.00	13.02
# polynomials	0	0	2	2	7

Coefficients selection: for $n = k + l = 1 + 14$ bits, we get:

(▽,▽,▽,▽)	11.41	(▽,▽,▽,△)	12.00
(▽,▽,△,▽)	12.00	(▽,▽,△,△)	12.84
(▽,△,▽,▽)	12.00	(▽,△,▽,△)	13.00
(▽,△,△,▽)	13.00	(▽,△,△,△)	12.36
(△,▽,▽,▽)	12.00	(△,▽,▽,△)	12.25
(△,▽,△,▽)	12.23	(△,▽,△,△)	12.23
(△,△,▽,▽)	12.13	(△,△,▽,△)	12.12
(△,△,△,▽)	12.05	(△,△,△,△)	11.64

Example: 2^x over $[0,1]$ and $\mu \leq 12$ sb (2/2)

Datapath size selection:

n'	14	15	16	17	18	19	20
ϵ_{eval} direct	11.24	11.86	12.32	12.62	12.79	12.89	12.94
ϵ_{eval} Horner	11.32	11.93	12.36	12.65	12.81	12.90	12.95

Solution: $d = 3$, $n = k + l = 1 + 14$ and $n' = 16$
Implementation results:

solution	area	period	#cycles	latency	power
wo. tools	1.00	1.00	4	1.00	1.00
w. tools	0.83	0.82	3	0.61	0.68

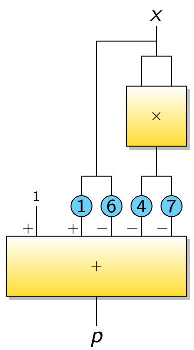
Example: $\sqrt{x}$ over $[1,2]$ and $\mu \leq 8$ sb

Selection of coefficients leading to sparse recodings

$p^* = 1.00076383 + 0.48388463x - 0.071198745x^2$

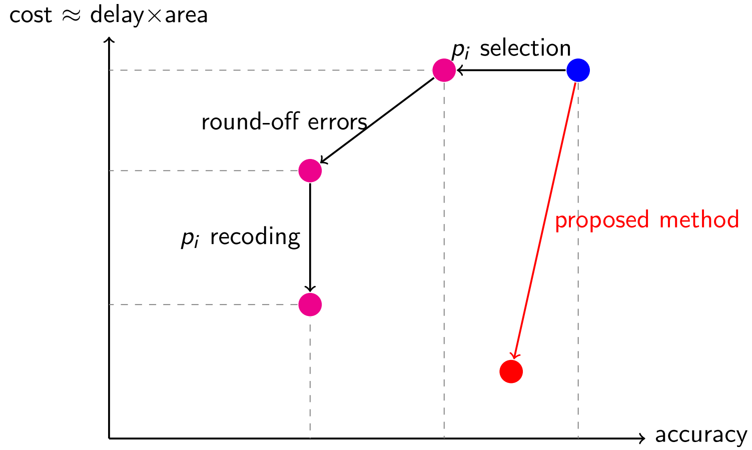
$p = 1 + (0.100001\bar{1})_2 x - (0.0001001)_2 x^2$

replace $\times$ by a small number of $\pm$



solution	area	period	#cycles	latency	power
wo. tools	1.00	1.00	2	1.00	1.00
w. tools	0.59	0.97	1	0.48	0.45

Summary



Important: non-optimal solutions BUT very good ones in practice