Automatic generation of polynomial-based hardware architectures for function evaluation

Mioara Joldeș

joint work with:

N. Brisebarre and **S. Chevillard** and **F. de Dinechin** and **B. Pașca**



Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.
- Use:
  - precomputed values stored in memory
  - additions
  - multiplications

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.
- Use:
  - precomputed values stored in memory
  - additions
  - multiplications

$\rightsquigarrow$ Approximation Polynomial

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.
- Use:
  - precomputed values stored in memory
  - additions
  - multiplications

$\rightsquigarrow$ Approximation Polynomial with machine representable coefficients

Reminder: Finite representation of numbers in a machine

"Look engineers. All we're asking for is an infinite number of transistors on a finite-sized chip..."

Reminder: Finite representation of numbers in a machine

"Look engineers. All we're asking for is an infinite number of transistors on a finite-sized chip..."

- Finite set of representable numbers M_p;
- $x \in M_p$ represents $I(x)$ partition of $\mathbb{R}$;

Reminder: Finite representation of numbers in a machine

"Look engineers. All we're asking for is an infinite number of transistors on a finite-sized chip..."

- Finite set of representable numbers M_p;
- $x \in M_p$ represents $I(x)$ partition of $\mathbb{R}$;

- Fixed Point Arithmetic
  - $M_p = \{x/2^k, x \in [-2^{p-1}, 2^{p-1} - 1)\}$
  - $I(x)$ constant size
  - Absolute error

Reminder: Finite representation of numbers in a machine

"Look engineers. All we're asking for is an infinite number of transistors on a finite-sized chip..."

- Finite set of representable numbers M_p;
- $x \in M_p$ represents $I(x)$ partition of $\mathbb{R}$;

- Fixed Point Arithmetic
  - $M_p = \{x/2^k, x \in [-2^{p-1}, 2^{p-1} - 1)\}$
  - $I(x)$ constant size
  - Absolute error

- Floating Point Arithmetic
  - $M_p = \{x/2^p \cdot 2^e, x \in \pm[-2^{p-1}, 2^p)\}$
  - $I(x)$ dynamic size corresponding to x
  - Relative error

Reminder: IEEE Precisions

See `http://en.wikipedia.org/wiki/IEEE_754-2008` or (older) `http://babbage.cs.qc.edu/courses/cs341/IEEE-754references.html`.

	precision	min. exponent	max. exponent
single (binary 32)	24	-126	127
double (binary 64)	53	-1022	1023
extended double	64	-16382	16383
quadruple (binary 128)	113	-16382	16383

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.

$\rightsquigarrow$ Computation of "machine-efficient" polynomial approximations

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.

$\rightsquigarrow$ Computation of "machine-efficient" polynomial approximations

$\rightsquigarrow$ Efficient evaluation scheme for polynomials

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.
  - $\rightsquigarrow$ Argument reduction (Payne & Hanek, Ng, Daumas *et al*): evaluation of a function φ over $\mathbb{R}$ or a subset of $\mathbb{R}$ is reduced to the evaluation of a function f over $[a, b]$.
  - $\rightsquigarrow$ Computation of "machine-efficient" polynomial approximations
  - $\rightsquigarrow$ Efficient evaluation scheme for polynomials

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.
  - $\rightsquigarrow$ Argument reduction (Payne & Hanek, Ng, Daumas *et al*): evaluation of a function φ over $\mathbb{R}$ or a subset of $\mathbb{R}$ is reduced to the evaluation of a function f over $[a, b]$.
  - $\rightsquigarrow$ Computation of "machine-efficient" polynomial approximations
  - $\rightsquigarrow$ Efficient evaluation scheme for polynomials
  - $\rightsquigarrow$ Manage approximation and evaluation errors

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.
  - $\rightsquigarrow$ Argument reduction (Payne & Hanek, Ng, Daumas *et al*): evaluation of a function φ over $\mathbb{R}$ or a subset of $\mathbb{R}$ is reduced to the evaluation of a function f over $[a, b]$.
  - $\rightsquigarrow$ Computation of "machine-efficient" polynomial approximations
  - $\rightsquigarrow$ Efficient evaluation scheme for polynomials
  - $\rightsquigarrow$ Manage approximation and evaluation errors
- General step chain, but customized for different target architectures. Eg. exploit FPGAs flexibility: custom precision, dedicated architecture for coarser operators, massive parallelism.

Context - Evaluation of a function φ on a machine

- Need to evaluate sin, exp, arctan, $\sqrt{}$ ... to an accuracy η.

$\rightsquigarrow$ Computation of "machine-efficient" polynomial approximations*

*N. Brisebarre and S. Chevillard, *Efficient polynomial L^∞- approximations, 18th IEEE SYMPOSIUM on Computer Arithmetic,169–176, Los Alamitos, USA, 2007*

Minimax Approximation

Reminder. Let $\varepsilon : [a, b] \to \mathbb{R}$, $||\varepsilon||_{[a,b]} = \sup_{a \leq x \leq b} |\varepsilon(x)|$.

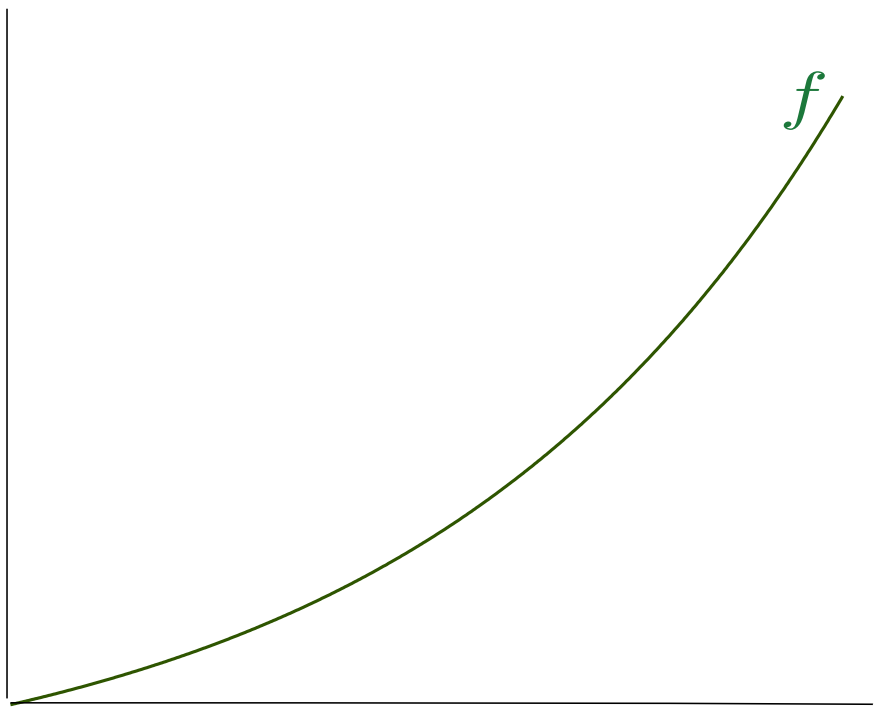
We denote $\mathbb{R}_n[X] = \{p \in \mathbb{R}[X]; \deg p \leq n\}$.

Minimax Approximation

Reminder. Let $\varepsilon : [a, b] \to \mathbb{R}$, $||\varepsilon||_{[a,b]} = \sup_{a \leq x \leq b} |\varepsilon(x)|$.

We denote $\mathbb{R}_n[X] = \{p \in \mathbb{R}[X]; \deg p \leq n\}$.

Minimax approximation: let $f : [a, b] \to \mathbb{R}$, $n \in \mathbb{N}$,

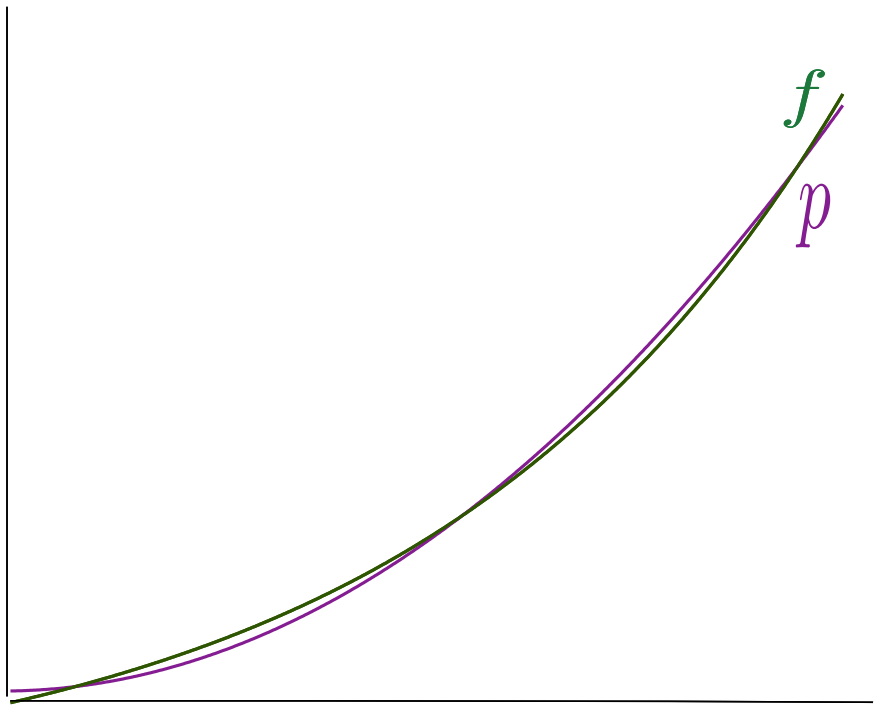


Minimax Approximation

Reminder. Let $\varepsilon : [a, b] \to \mathbb{R}$, $||\varepsilon||_{[a,b]} = \sup_{a \le x \le b} |\varepsilon(x)|$.

We denote $\mathbb{R}_n[X] = \{p \in \mathbb{R}[X]; \deg p \le n\}$.

Minimax approximation: let $f : [a, b] \to \mathbb{R}$, $n \in \mathbb{N}$, we search for $p \in \mathbb{R}_n[X]$



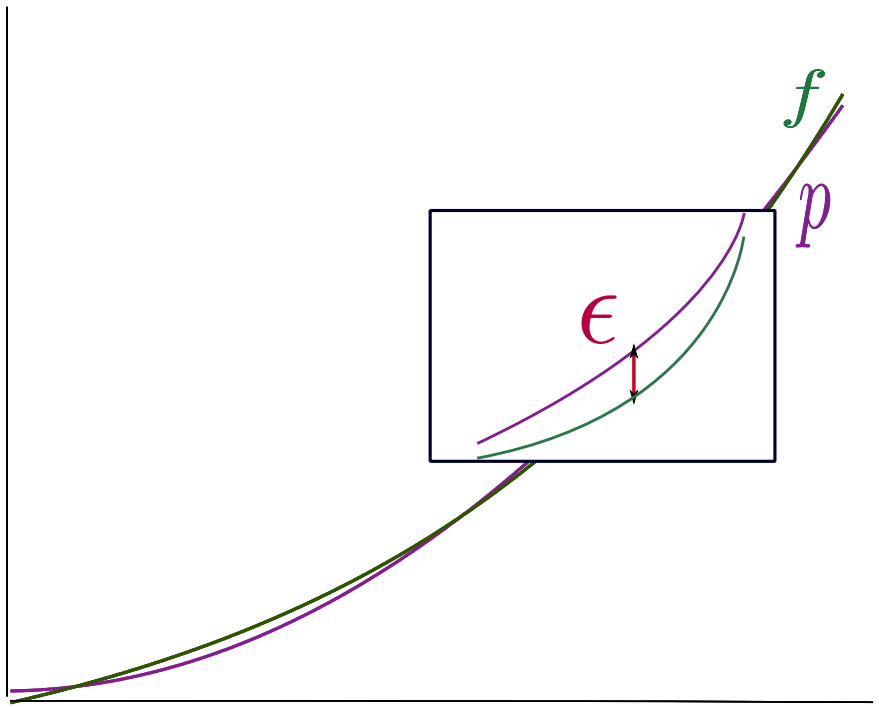
An algorithm due to Remez gives p (`minimax` function in Maple, also available in Sollya `http://sollya.gforge.inria.fr/`).

Minimax Approximation

Reminder. Let $\varepsilon : [a, b] \to \mathbb{R}$, $||\varepsilon||_{[a,b]} = \sup_{a \le x \le b} |\varepsilon(x)|$.

We denote $\mathbb{R}_n[X] = \{p \in \mathbb{R}[X]; \deg p \le n\}$.

Minimax approximation: let $f : [a, b] \to \mathbb{R}$, $n \in \mathbb{N}$, we search for $p \in \mathbb{R}_n[X]$ s.t. $||\varepsilon||_{[a,b]}$ is minimal, where $\varepsilon = p - f$.



An algorithm due to Remez gives p (`minimax` function in Maple, also available in Sollya `http://sollya.gforge.inria.fr/`).

Truncated Polynomials

Our context: the coefficients of the polynomials must be written on a finite (imposed) number of bits.

Truncated Polynomials

Our context: the coefficients of the polynomials must be written on a finite (imposed) number of bits.

Let $m = (m_i)_{0 \leq i \leq n}$ a finite sequence of integers. Let

$$\mathcal{P}_n^m = \{q = q_0 + q_1 X + \cdots + q_n X^n \in \mathbb{R}_n[X]; q_i \text{ integer multiple of } 2^{-m_i}, \forall i\}.$$

Truncated Polynomials

Our context: the coefficients of the polynomials must be written on a finite (imposed) number of bits.

Let $m = (m_i)_{0 \leq i \leq n}$ a finite sequence of integers. Let

$$\mathcal{P}_n^m = \{q = q_0 + q_1 x + \cdots + q_n x^n \in \mathbb{R}_n[X]; q_i \text{ integer multiple of } 2^{-m_i}, \forall i\}.$$

Question: find $p^\star \in \mathcal{P}_n^m$ which minimizes $||f - q||$, $q \in \mathcal{P}_n^m$.

Truncated Polynomials

Our context: the coefficients of the polynomials must be written on a finite (imposed) number of bits.

Let $m = (m_i)_{0 \leq i \leq n}$ a finite sequence of integers. Let

$$\mathcal{P}_n^m = \{q = q_0 + q_1 x + \cdots + q_n x^n \in \mathbb{R}_n[X]; q_i \text{ integer multiple of } 2^{-m_i}, \forall i\}.$$

Question: find $p^\star \in \mathcal{P}_n^m$ which minimizes $||f - q||$, $q \in \mathcal{P}_n^m$.

First idea. Remez $\rightarrow$ $p(x) = p_0 + p_1 x + \cdots + p_n x^n$. Every p_i rounded to $\hat{a}_i / 2^{m_i}$, the nearest integer multiple of 2^{-m_i} $\rightarrow$

$$\hat{p}(x) = \frac{\hat{a}_0}{2^{m_0}} + \frac{\hat{a}_1}{2^{m_1}} x + \cdots + \frac{\hat{a}_n}{2^{m_n}} x^n.$$

Truncated Polynomials

Our context: the coefficients of the polynomials must be written on a finite (imposed) number of bits.

Let $m = (m_i)_{0 \leq i \leq n}$ a finite sequence of integers. Let

$$\mathcal{P}_n^m = \{q = q_0 + q_1 x + \cdots + q_n x^n \in \mathbb{R}_n[X]; q_i \text{ integer multiple of } 2^{-m_i}, \forall i\}.$$

Question: find $p^\star \in \mathcal{P}_n^m$ which minimizes $||f - q||$, $q \in \mathcal{P}_n^m$.

First idea. Remez $\rightarrow$ $p(x) = p_0 + p_1 x + \cdots + p_n x^n$. Every p_i rounded to $\hat{a}_i / 2^{m_i}$, the nearest integer multiple of 2^{-m_i} $\rightarrow$

$$\hat{p}(x) = \frac{\hat{a}_0}{2^{m_0}} + \frac{\hat{a}_1}{2^{m_1}} x + \cdots + \frac{\hat{a}_n}{2^{m_n}} x^n.$$

Problem: $\hat{p}$ not necessarily a minimax approx. of f among the polynomials of $\mathcal{P}_n^m$.

Approximation of the Function cos over $[0, \pi/4]$ by a Degree-3 Polynomial

Maple or Sollya tell us that the polynomial

$$p = 0.9998864206 + 0.00469021603x - 0.5303088665x^2 + 0.06304636099x^3$$

is $\sim$ the best approximant to cos. We have
$\varepsilon = ||\cos - p||_{[0,\pi/4]} = 0.0001135879....$

We look for $a_0, a_1, a_2, a_3 \in \mathbb{Z}$ such that

$$\max_{0 \le x \le \pi/4} \left| \cos x - \left(\frac{a_0}{2^{12}} + \frac{a_1}{2^{10}} x + \frac{a_2}{2^6} x^2 + \frac{a_3}{2^4} x^3 \right) \right|$$

is minimal.

Approximation of the Function cos over $[0, \pi/4]$ by a Degree-3 Polynomial

Maple or Sollya tell us that the polynomial

$$p = 0.9998864206 + 0.00469021603x - 0.5303088665x^2 + 0.06304636099x^3$$

is $\sim$ the best approximant to cos. We have
$\varepsilon = ||\cos - p||_{[0,\pi/4]} = 0.0001135879....$

We look for $a_0, a_1, a_2, a_3 \in \mathbb{Z}$ such that

$$\max_{0 \le x \le \pi/4} \left| \cos x - \left(\frac{a_0}{2^{12}} + \frac{a_1}{2^{10}} x + \frac{a_2}{2^6} x^2 + \frac{a_3}{2^4} x^3 \right) \right|$$

is minimal.

The naive approach gives the polynomial

$$\hat{p} = \frac{2^{12}}{2^{12}} + \frac{5}{2^{10}} x - \frac{34}{2^6} x^2 + \frac{1}{2^4} x^3.$$

We have $\hat{\varepsilon} = ||\cos - \hat{p}||_{[0,\pi/4]} = 0.00069397....$

Approximation of the Function cos over $[0, \pi/4]$ by a Degree-3 Polynomial

Maple or Sollya computes a polynomial p which is $\sim$ the best approximant to cos. We have $\varepsilon = ||\cos - p||_{[0,\pi/4]} = 0.0001135879...$.
We look for $a_0, a_1, a_2, a_3 \in \mathbb{Z}$ such that

$$\max_{0 \le x \le \pi/4} \left| \cos x - \left(\frac{a_0}{2^{12}} + \frac{a_1}{2^{10}} x + \frac{a_2}{2^6} x^2 + \frac{a_3}{2^4} x^3 \right) \right|$$

is minimal.
The naive approach gives the polynomial $\hat{p}$ and
$\hat{\varepsilon} = ||\cos - \hat{p}||_{[0,\pi/4]} = 0.00069397...$

Approximation of the Function cos over $[0, \pi/4]$ by a Degree-3 Polynomial

Maple or Sollya computes a polynomial p which is $\sim$ the best approximant to cos. We have $\varepsilon = ||\cos - p||_{[0,\pi/4]} = 0.0001135879...$.
We look for $a_0, a_1, a_2, a_3 \in \mathbb{Z}$ such that

$$\max_{0 \le x \le \pi/4} \left| \cos x - \left(\frac{a_0}{2^{12}} + \frac{a_1}{2^{10}} x + \frac{a_2}{2^6} x^2 + \frac{a_3}{2^4} x^3 \right) \right|$$

is minimal.
The naive approach gives the polynomial $\hat{p}$ and $\hat{\varepsilon} = ||\cos - \hat{p}||_{[0,\pi/4]} = 0.00069397...$ But the best "truncated" approximant:

$$p^\star = \frac{4095}{2^{12}} + \frac{6}{2^{10}} x - \frac{34}{2^6} x^2 + \frac{1}{2^4} x^3$$

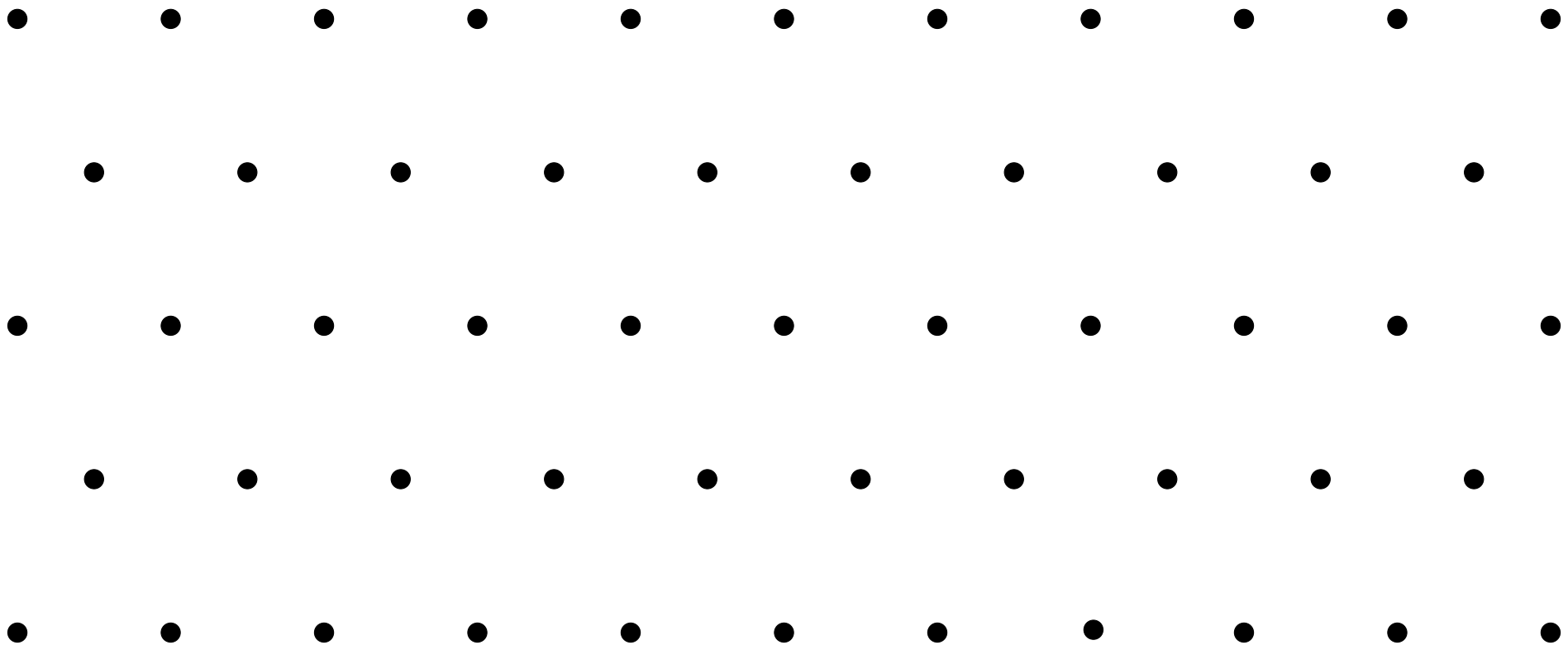
which gives $||\cos - p^\star||_{[0,\pi/4]} = 0.0002441406250$.

In this example, we gain $-\log_2(0.35) \approx 1.5$ bits of accuracy.

A First Approach through Linear Programming

It makes it possible to tackle with degree-8 or 10 polynomials: this is nice for hardware-oriented applications but not satisfying for all software-oriented applications.

A Second Approach through Lattice Basis Reduction



A Second Approach through Lattice Basis Reduction

Definition

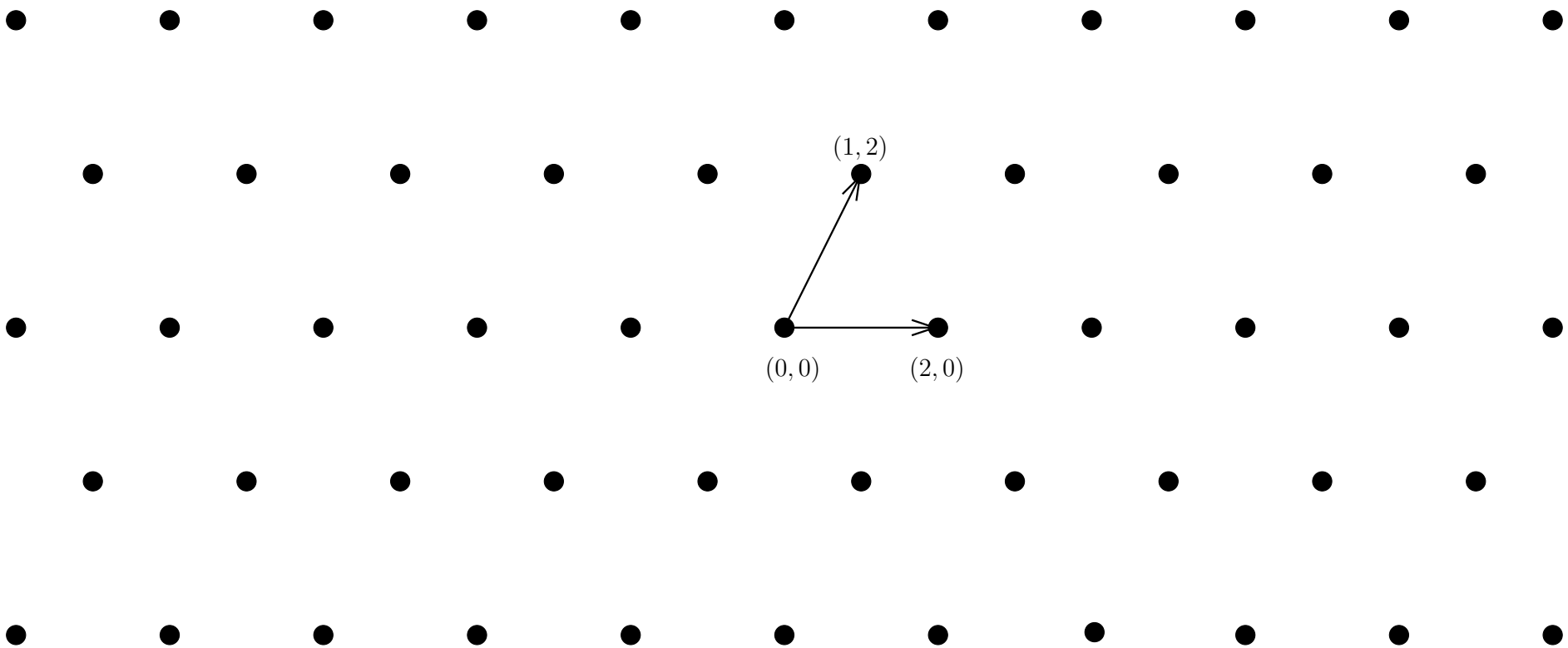
Let L be a nonempty subset of $\mathbb{R}^d$, L is a lattice iff there exists a set of vectors $b_1, \ldots, b_k$ $\mathbb{R}$-linearly independent such that

$$L = \mathbb{Z}.b_1 \oplus \cdots \oplus \mathbb{Z}.b_k.$$

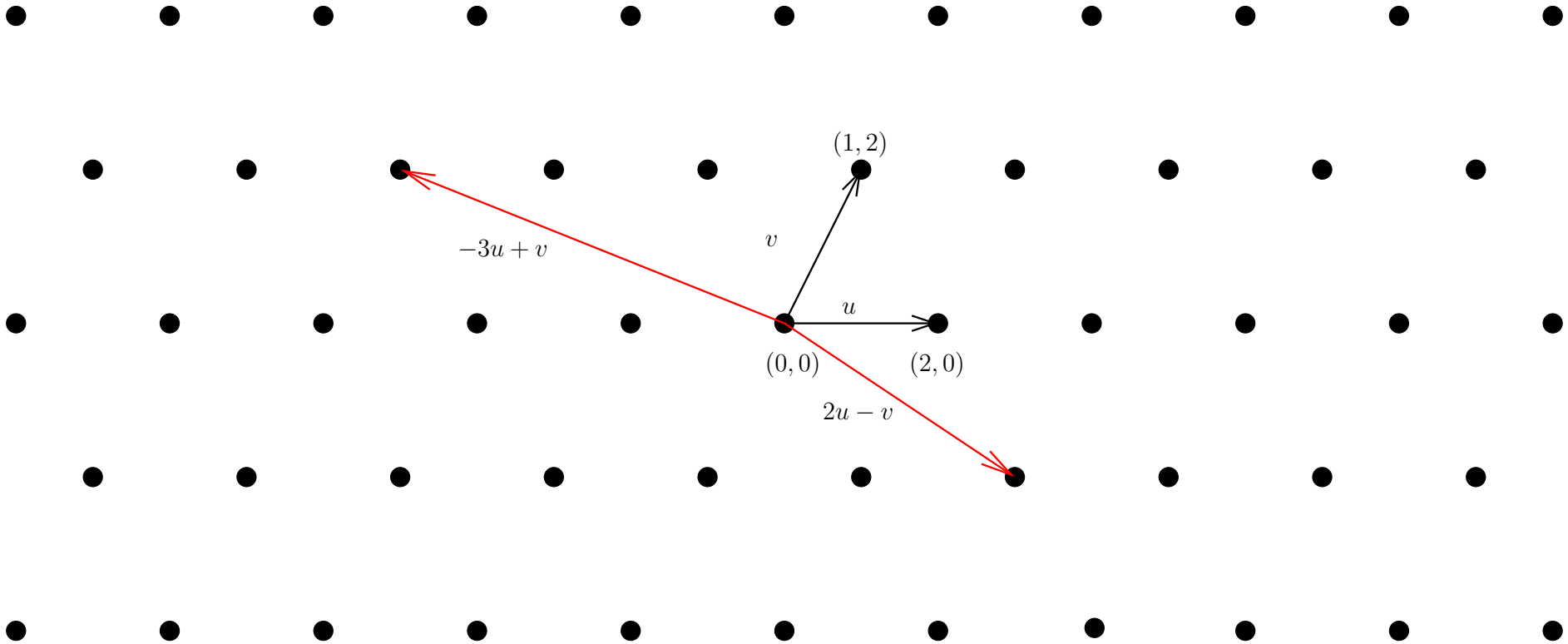
$(b_1, \ldots, b_k)$ is a basis of the lattice L.

Examples. $\mathbb{Z}^d$, every subgroup of $\mathbb{Z}^d$.

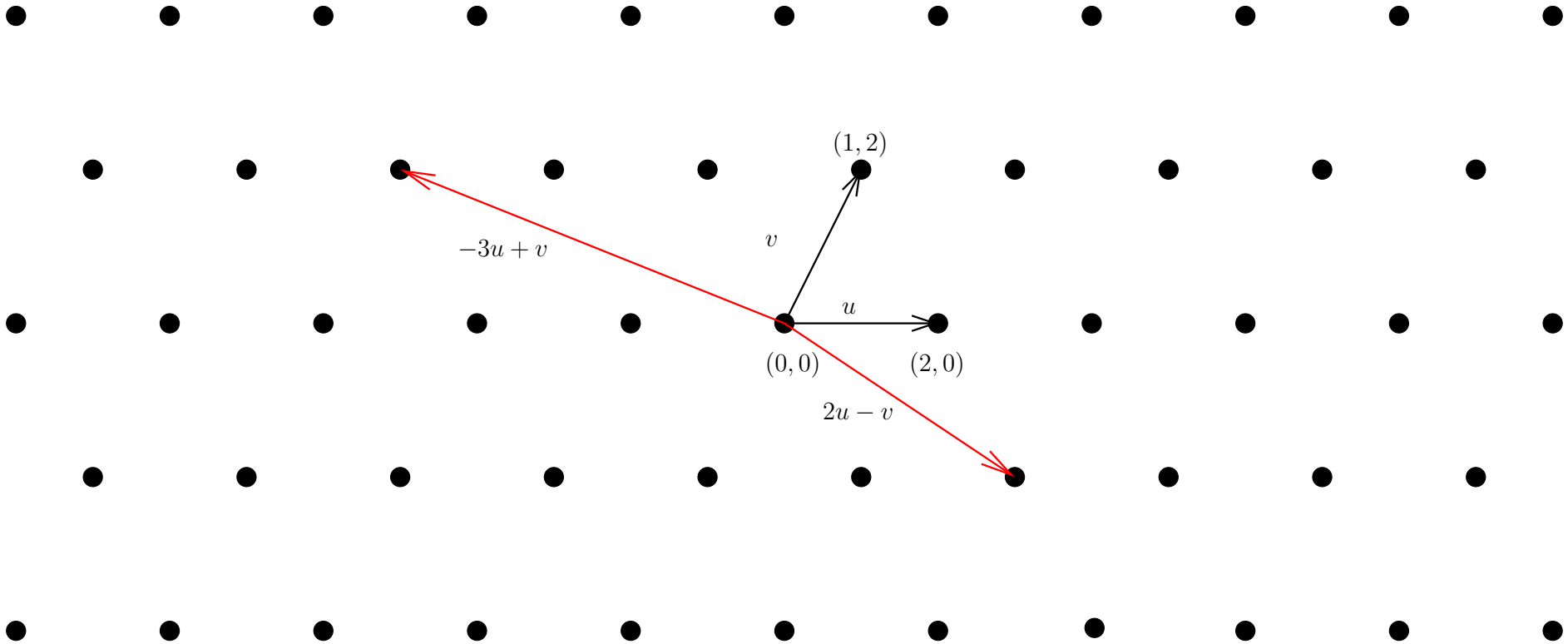
Example: The Lattice $\mathbb{Z}(2,0) \oplus \mathbb{Z}(1,2)$



Example: The Lattice $\mathbb{Z}(2,0) \oplus \mathbb{Z}(1,2)$

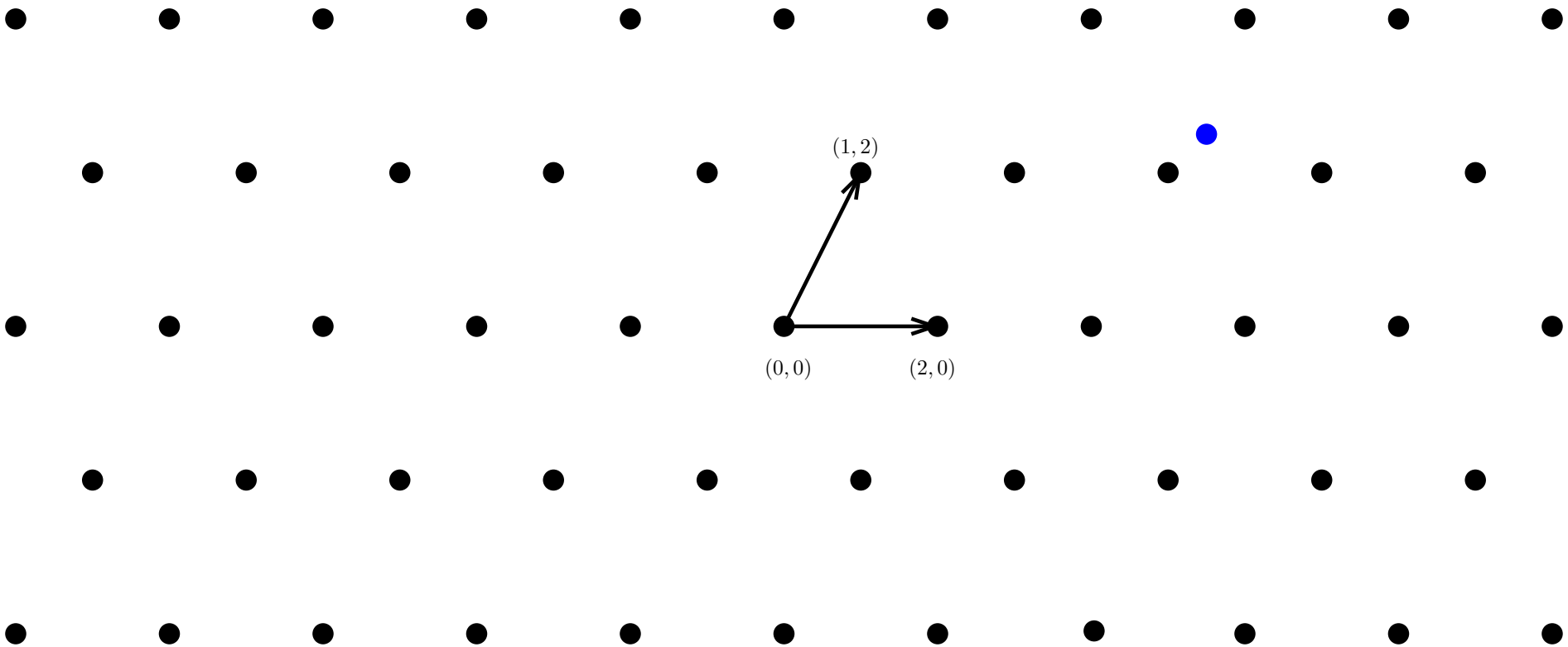


Example: The Lattice $\mathbb{Z}(2,0) \oplus \mathbb{Z}(1,2)$

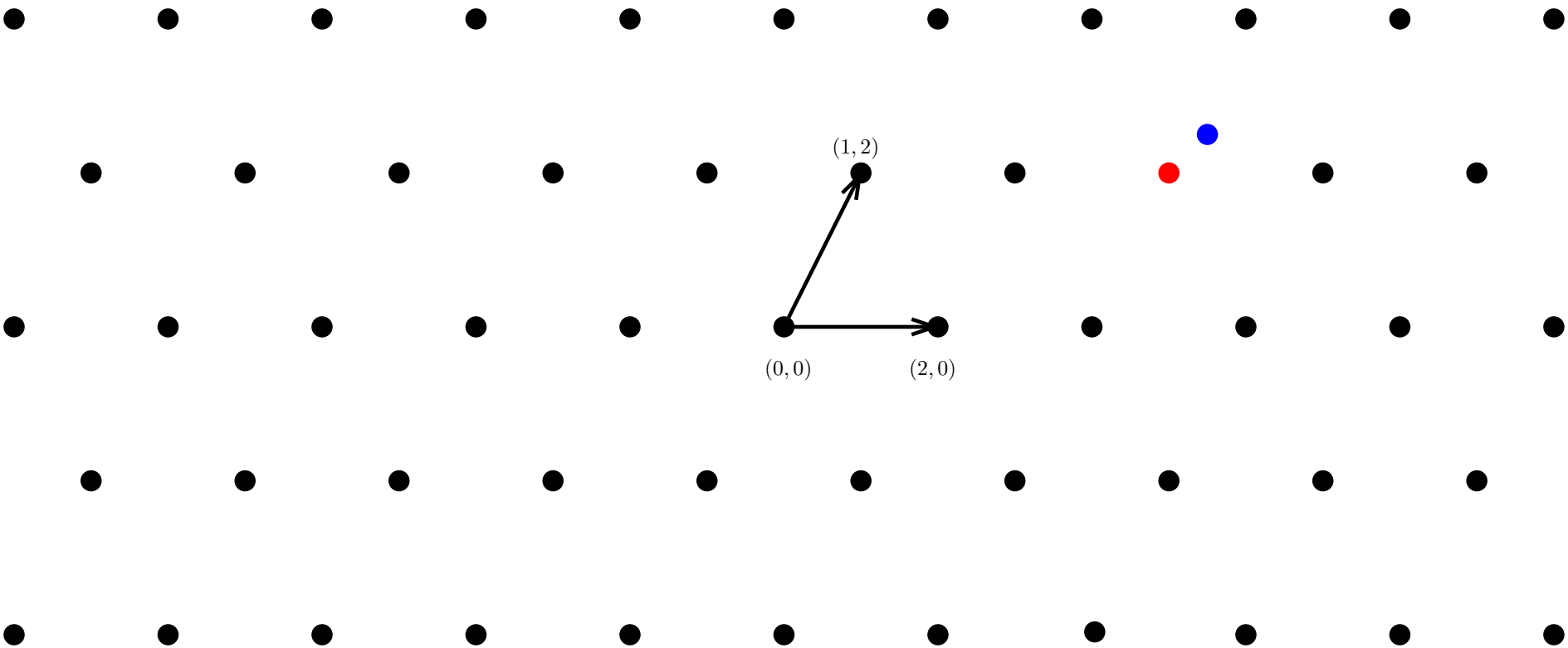


SVP (Shortest Vector Problem)

Closest Vector Problem



Closest Vector Problem



Lenstra-Lenstra-Lovász Algorithm

SVP (Shortest Vector Problem) and CVP (Closest Vector Problem) are NP-hard.

Lenstra-Lenstra-Lovász Algorithm

SVP (Shortest Vector Problem) and CVP (Closest Vector Problem) are NP-hard.

Factoring Polynomials with Rational Coefficients, A. K. Lenstra, H. W. Lenstra and L. Lovász, Math. Annalen **261**, 515-534, 1982.

The LLL algorithm gives an approximate solution to SVP in polynomial time.

Babai's algorithm (based on LLL) gives an approximate solution to CVP in polynomial time.

Absolute Error Problem

We search for (one of the) best(s) polynomial of the form

$$p^\star = \frac{a_0^\star}{2^{m_0}} + \frac{a_1^\star}{2^{m_1}} X + \cdots + \frac{a_n^\star}{2^{m_n}} X^n$$

(where $a_i^\star \in \mathbb{Z}$ and $m_i \in \mathbb{Z}$) that minimizes $\|f - p\|_{[a,\, b]}$.

Discretize the continuous problem: we choose $x_1, \cdots, x_d$ points in $[a,\, b]$ such that $\frac{a_0^\star}{2^{m_0}} + \frac{a_1^\star}{2^{m_1}} x_i + \cdots + \frac{a_n^\star}{2^{m_n}} x_i^n$ as close as possible to $f(x_i)$ for all $i = 1, \ldots, d$.

That is to say we want the vectors

$$\begin{pmatrix} \frac{a_0^\star}{2^{m_0}} + \frac{a_1^\star}{2^{m_1}} x_1 + \cdots + \frac{a_n^\star}{2^{m_n}} x_1^n \\ \frac{a_0^\star}{2^{m_0}} + \frac{a_1^\star}{2^{m_1}} x_2 + \cdots + \frac{a_n^\star}{2^{m_n}} x_2^n \\ \vdots \\ \frac{a_0^\star}{2^{m_0}} + \frac{a_1^\star}{2^{m_1}} x_d + \cdots + \frac{a_n^\star}{2^{m_n}} x_d^n \end{pmatrix} \text{ and } \begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_d) \end{pmatrix}$$

to be as close as possible, which can be rewritten as: we want the vectors

$$a_0^\star \underbrace{\begin{pmatrix} \frac{1}{2^{m_0}} \\ \frac{1}{2^{m_0}} \\ \vdots \\ \frac{1}{2^{m_0}} \end{pmatrix}}_{\overrightarrow{v_0}} + a_1^\star \underbrace{\begin{pmatrix} \frac{x_1}{2^{m_1}} \\ \frac{x_2}{2^{m_1}} \\ \vdots \\ \frac{x_d}{2^{m_1}} \end{pmatrix}}_{\overrightarrow{v_1}} + \cdots + a_n^\star \underbrace{\begin{pmatrix} \frac{x_1^n}{2^{m_n}} \\ \frac{x_2^n}{2^{m_n}} \\ \vdots \\ \frac{x_d^n}{2^{m_n}} \end{pmatrix}}_{\overrightarrow{v_n}} \text{ and } \underbrace{\begin{pmatrix} f(x_1) \\ f(x_2) \\ \vdots \\ f(x_d) \end{pmatrix}}_{\overrightarrow{y}}$$

to be as close as possible.
We have to minimize $\| a_0^\star \overrightarrow{v_0} + \cdots + a_n^\star \overrightarrow{v_n} - \overrightarrow{y} \|$.

We have to minimize $\|a_0^\star \overrightarrow{v_0} + \cdots + a_n^\star \overrightarrow{v_n} - \overrightarrow{y}\|$.

This is a closest vector problem in a lattice!

It is NP-hard: LLL algorithm gives an approximate solution.

Summary

We can use the method we developed in two directions

- it is able to give us a smaller (in term of degree and/or size of the coefficients) polynomial providing the same accuracy.
- we can also use it to find a much better polynomial (in term of accuracy) with same precision for the coefficients than the rounded minimax.

We illustrate the second item with an example taken from CRLibm.

An Example from CRlibm

- CRlibm is a library designed to compute correctly rounded functions in an efficient way (target : IEEE double precision).

  `http://lipforge.ens-lyon.fr/www/crlibm/`

- It uses specific formats such as double-double or triple-double.

- Example[1] for computing $\arcsin(x)$ on $[0.79; 1]$.

[1] N. Brisebarre, S. Chevillard, C. Lauter

Arcsine Function

After argument reduction we have the problem to approximate

$$g(z) = \frac{\arcsin(1 - (z + m)) - \frac{\pi}{2}}{\sqrt{2 \cdot (z + m)}}$$

where $\mathrm{0xBFBC28F800009107} \leq z \leq \mathrm{0x3FBC28F7FFFF6EF1}$ (i.e. approximately $-0.110 \leq z \leq 0.110$) and
$m = \mathrm{0x3FBC28F80000910F} \simeq 0.110$.

Data

Target accuracy to achieve correct rounding : 2^{-119}.
The minimax of degree 21 is sufficient (error $= 2^{-119.83}$).
Each approximant is of the form

$$\underbrace{p_0}_{t.d.} + \underbrace{p_1}_{t.d.} x + \underbrace{p_2}_{d.d.} x^2 + \underbrace{\cdots}_{\cdots} + \underbrace{p_9}_{d.d.} x^9 + \underbrace{p_{10}}_{d.} x^{10} + \underbrace{\cdots}_{\cdots} + \underbrace{p_{21}}_{d.} x^{21}$$

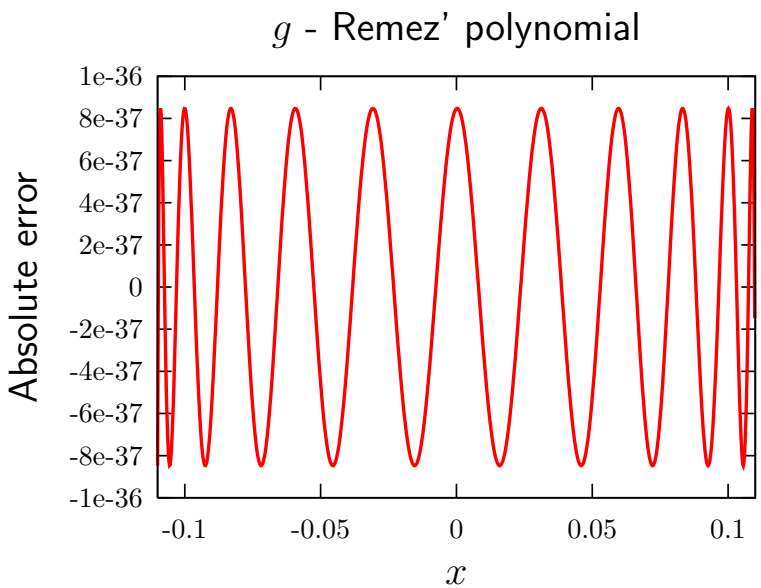
where the p_i are either double precision numbers (d.), a sum of two double precision numbers (d.d.), a sum of two double precision numbers (t.d.).

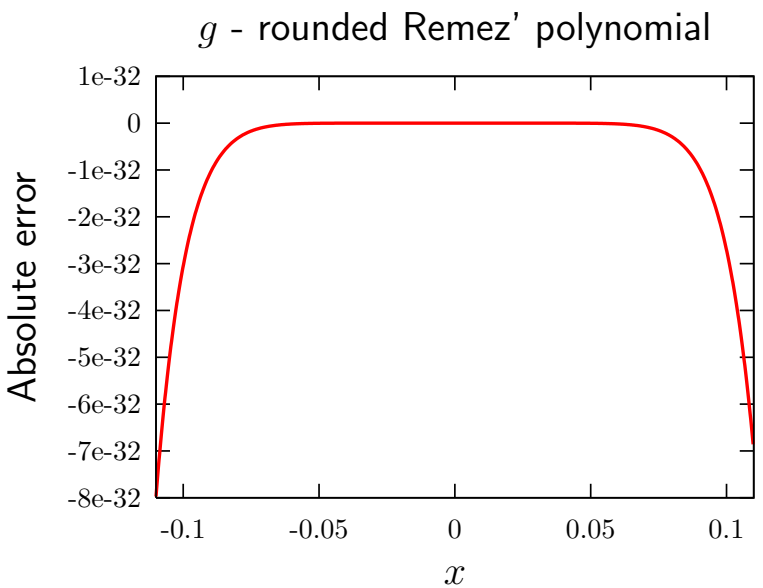
Figure: binary logarithm of the absolute error of several approximants

Target	-119
Minimax	-119.83
Rounded minimax	-103.31
Our polynomial	-119.77

Exact Minimax, Rounded Minimax, our Polynomial

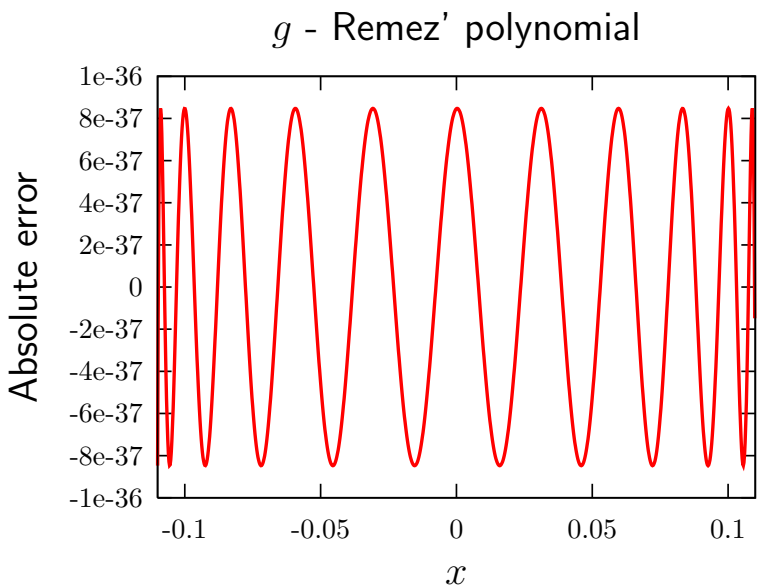
We save 16 bits with our method.

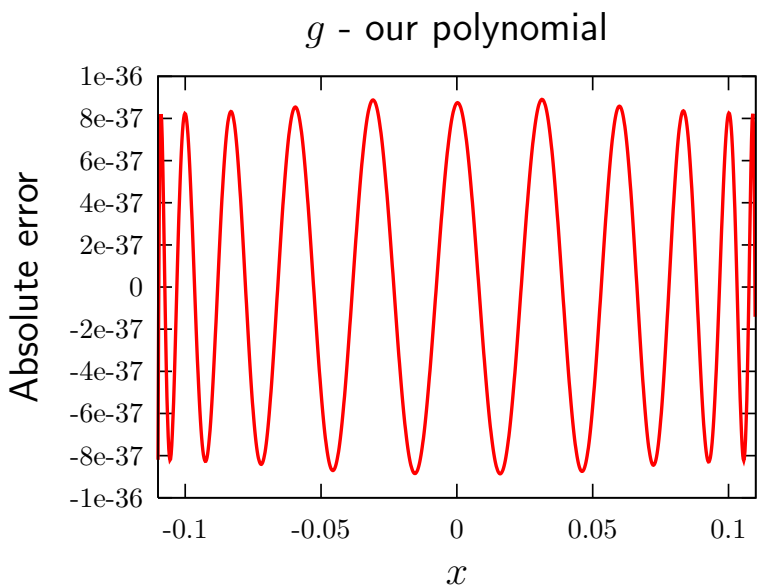




Exact Minimax, Rounded Minimax, our Polynomial

We save 16 bits with our method.





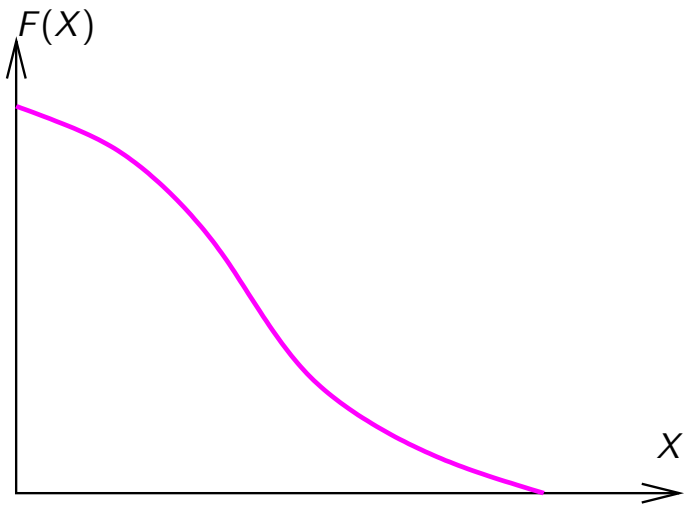
Machine-Efficient Polynomial Approximation: Summary

- Methods which improve the results provided by existing Remez' based method.
  All these tools are or shall be part of the Sollya software `http://sollya.gforge.inria.fr/`. Sollya is a tool environment for safe floating-point code development.

- Can be adapted to several kind of coefficients (fixed-point format, multi-double, classical floating point arithmetic with several precision formats).

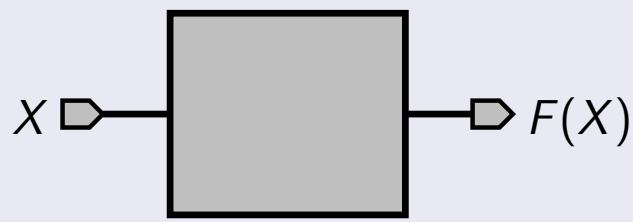
Hardware function evaluation

A function



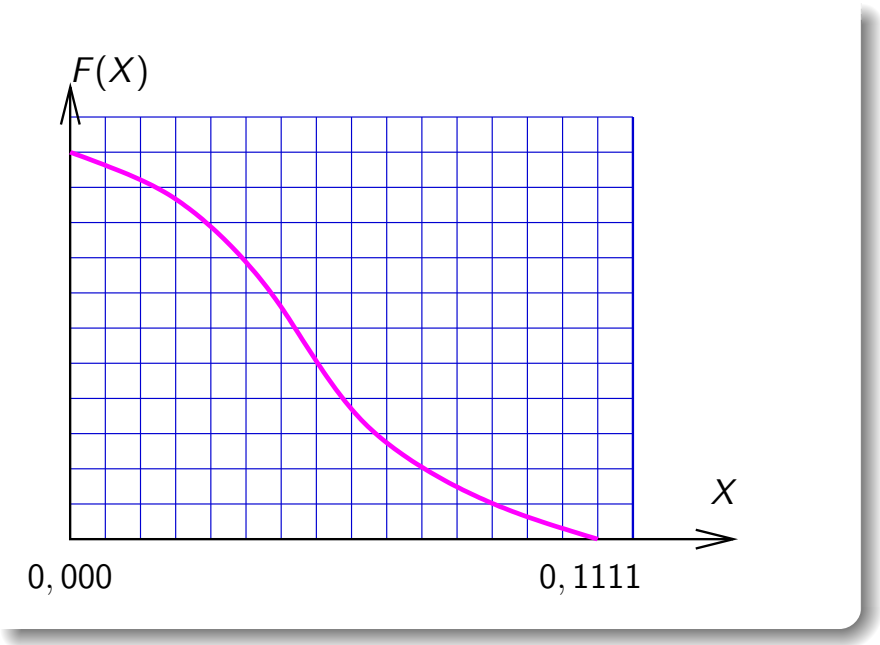
?

A hardware operator



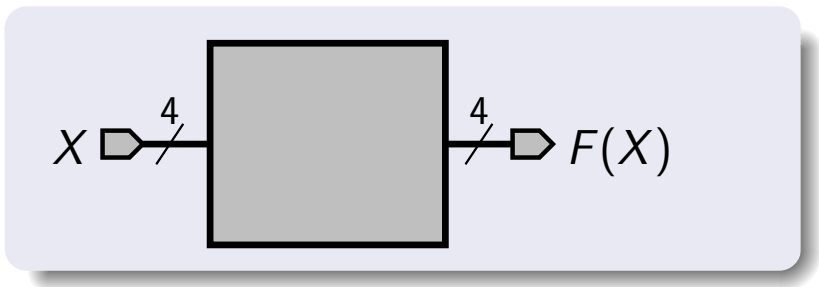
Hardware function evaluation

A function



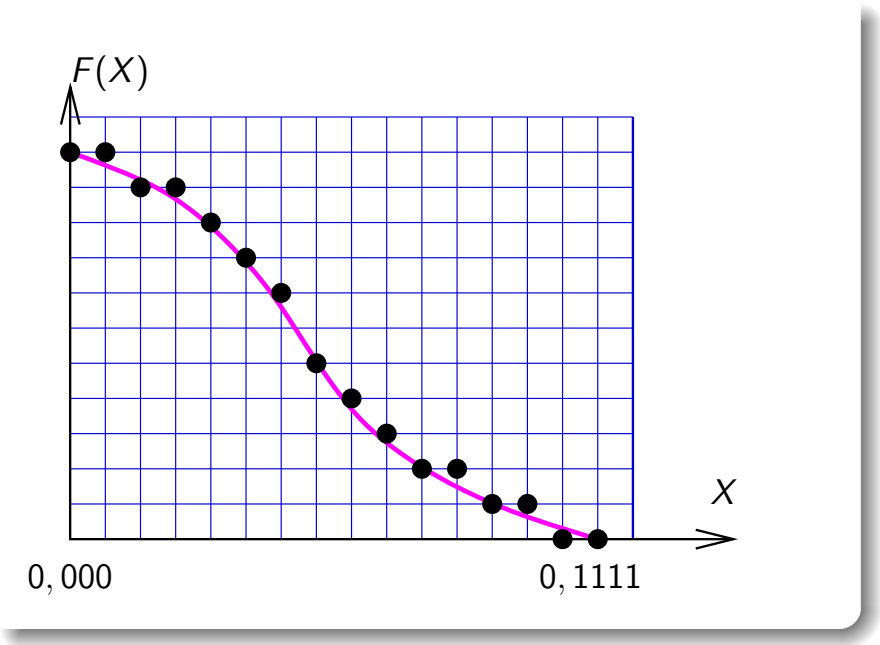
?

A hardware operator



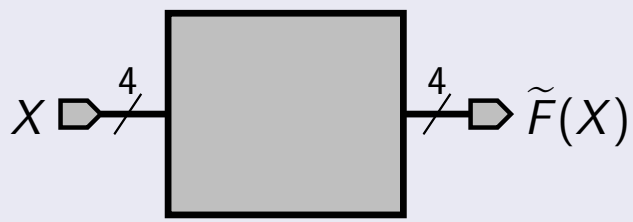
Hardware function evaluation

A function



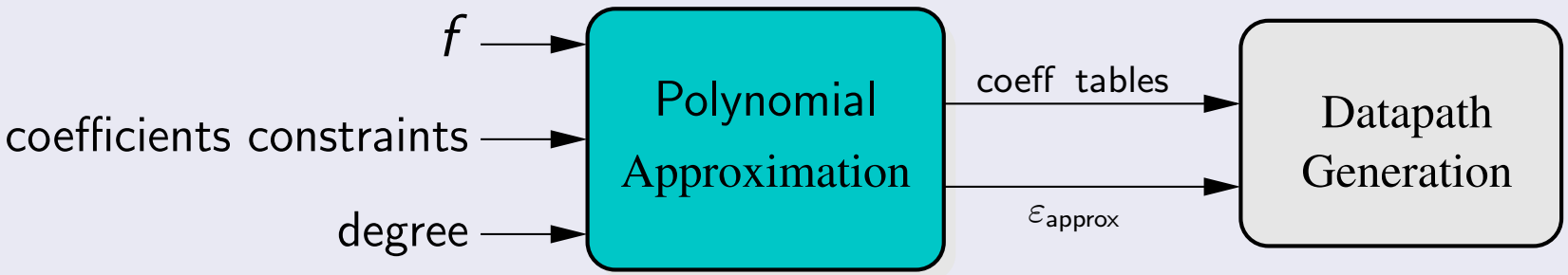
?

A hardware operator



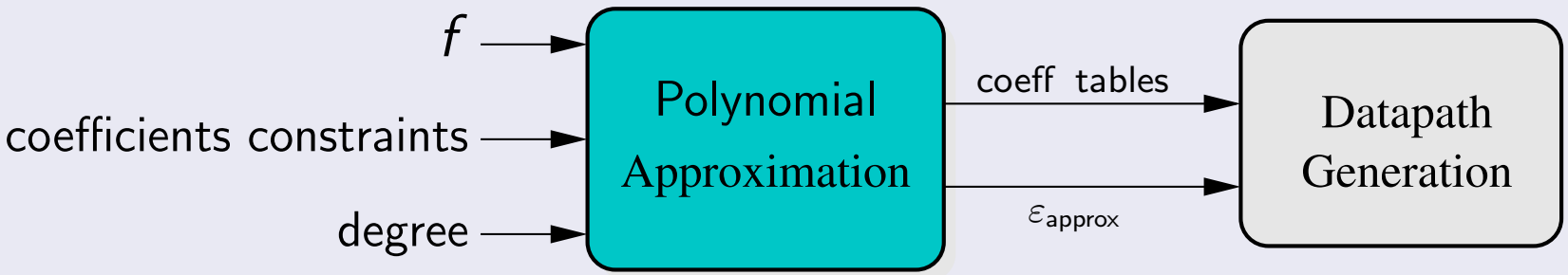
Evaluate f to an accuracy η.

Two fold process



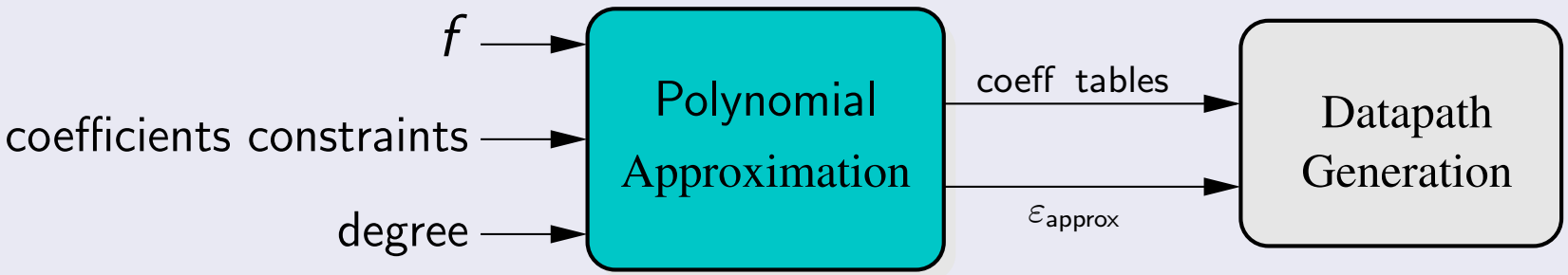
Evaluate f to an accuracy η.

Two fold process



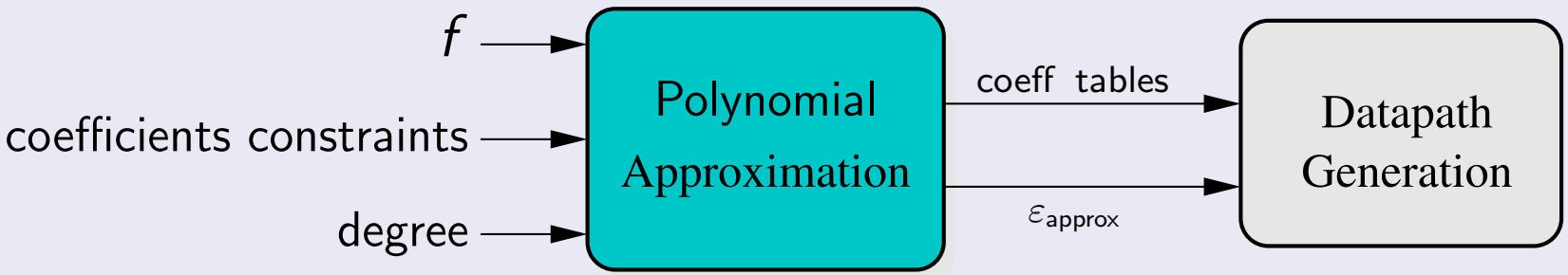
Evaluate f to an accuracy η.

Two fold process



Evaluate f to an accuracy η.

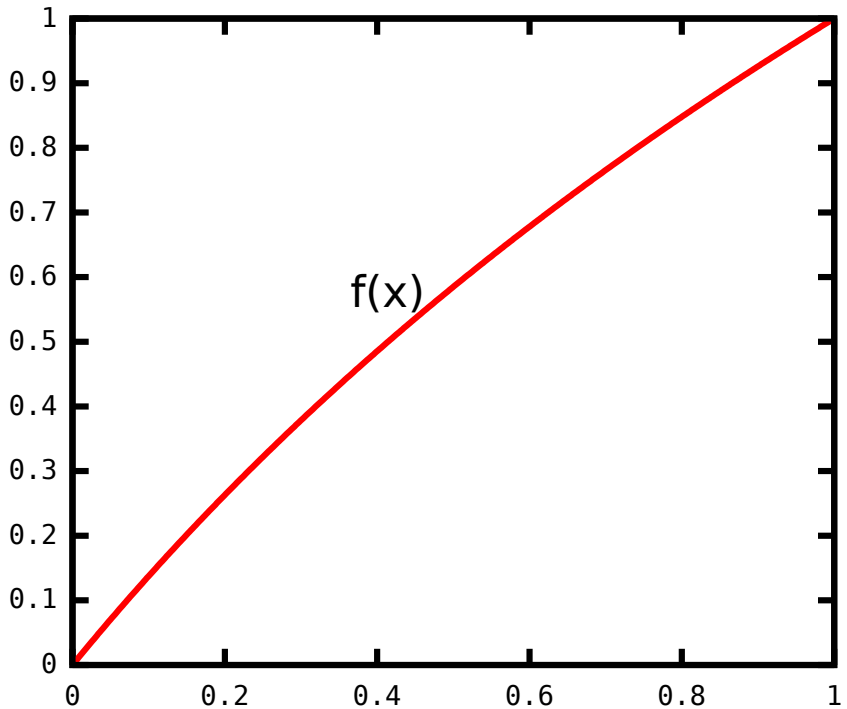
Two fold process

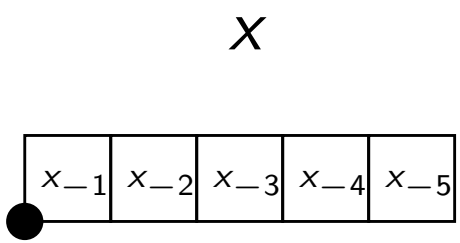


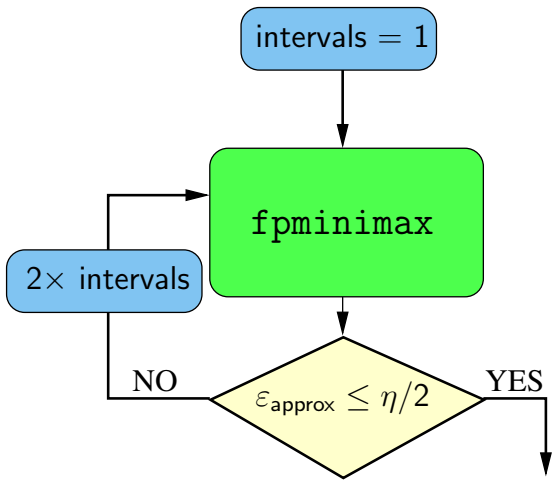
$$\varepsilon_{total} = \varepsilon_{approx} + \varepsilon_{eval} \leq \eta$$

Argument reduction

Example: $\varepsilon_{\text{approx}} \leqslant \eta/2$, degree of p fixed

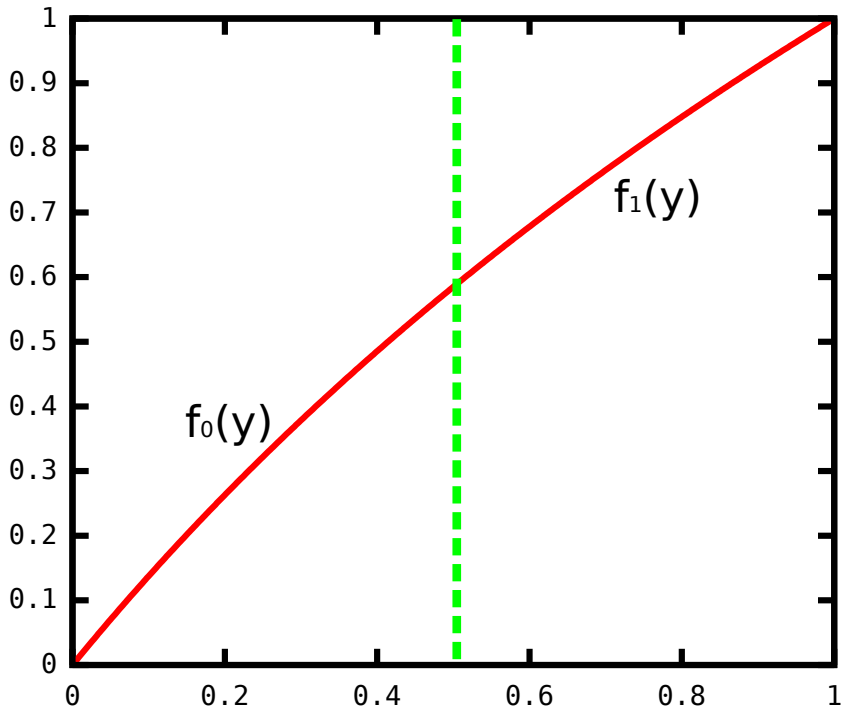




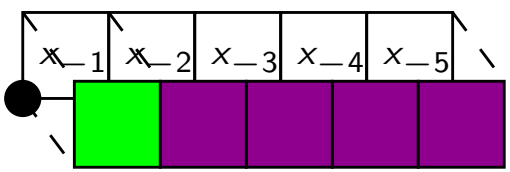


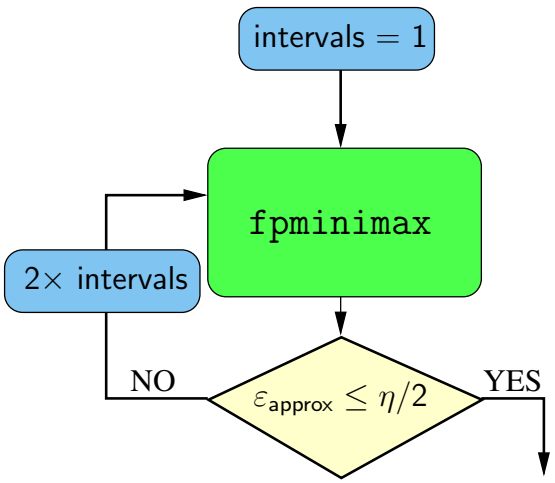
Argument reduction

Example: $\varepsilon_{\text{approx}} \leqslant \eta/2$, degree of p fixed



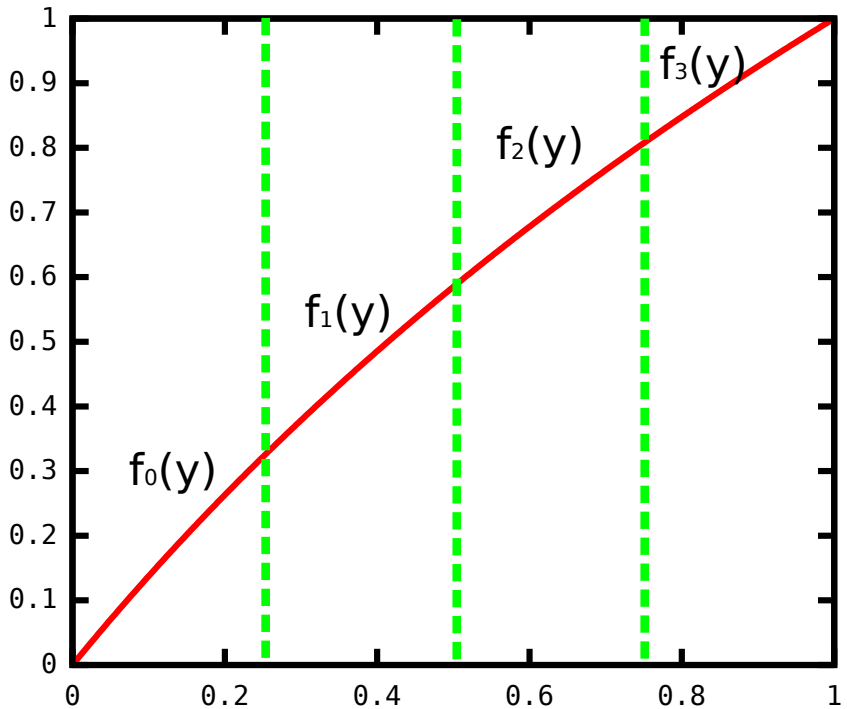
$k = 1$ y



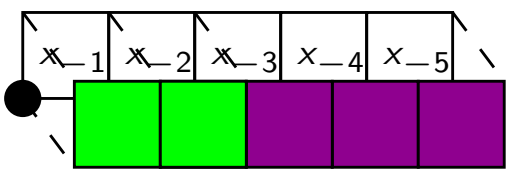


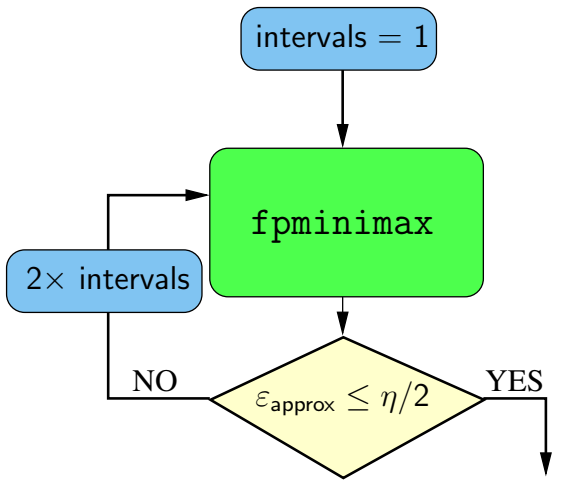
Argument reduction

Example: $\varepsilon_{\text{approx}} \leqslant \eta/2$, degree of p fixed



$k = 2$ y



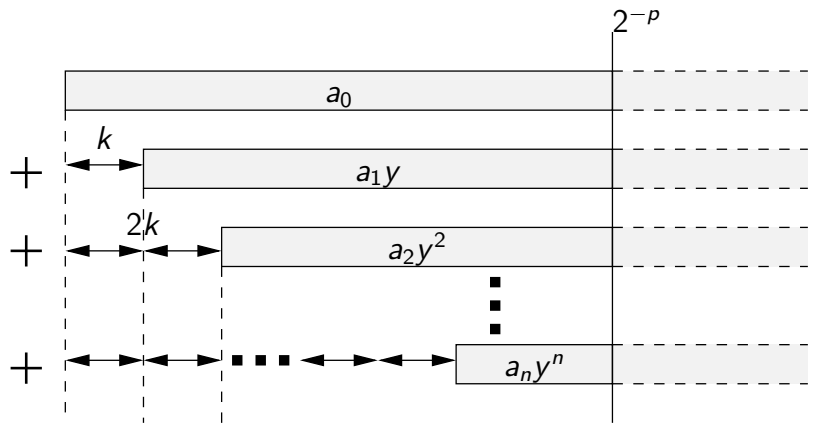


Determining coefficient constraints

Consider the polynomial:

$$p(y) = a_0 + a_1 y + a_2 y^2 + \ldots + a_d y^d$$

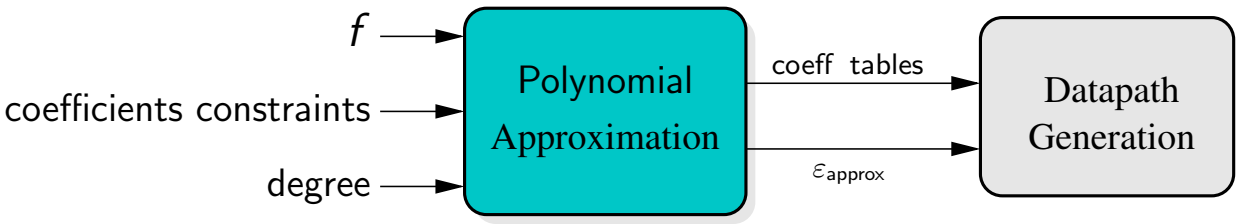
The fixed-point monomial alignment has **approximately** this form:



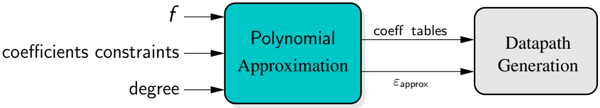
Rule of thumb

- no need for coefficient to have more precision than the monomial
- $a_j y^j$ with $y^j \in [0, 2^{-kj}]$ doesn't need much more precision than 2^{-p}
- LSB of a_j is approximatively 2^{-p+kj}

Polynomial Evaluation



Use `Horner` (trade latency for size)



Use `Horner` (trade latency for size)

$$p(y) = \underbrace{a_0 + \underbrace{y \times \underbrace{(a_1 + \underbrace{y \times \underbrace{(a_2 + \underbrace{y \times \underbrace{(a_3 + \underbrace{y \times \underbrace{a_4}_{\sigma_0}}_{\pi_1})}_{\sigma_1}}_{\pi_2})}_{\sigma_2}}_{\pi_3})}_{\sigma_3}}_{\pi_4}}_{\sigma_4}$$

Some orders of magnitude

Take as example $\log_2(1+x)$ for a precision of **53bits**.

The size of the multiplications for $d = 4$:

- $\pi_1 = y \times \sigma_0 \rightarrow (43 \times 15)$
- $\pi_2 = y \times \sigma_1 \rightarrow (43 \times 26)$
- $\pi_3 = y \times \sigma_2 \rightarrow (43 \times 37)$
- $\pi_4 = y \times \sigma_3 \rightarrow (43 \times 48)$

Some orders of magnitude

Take as example $\log_2(1+x)$ for a precision of **53bits**.

The size of the multiplications for $d = 4$:

- $\pi_1 = y \times \sigma_0 \rightarrow (43 \times 15)$
- $\pi_2 = y \times \sigma_1 \rightarrow (43 \times 26)$
- $\pi_3 = y \times \sigma_2 \rightarrow (43 \times 37)$
- $\pi_4 = y \times \sigma_3 \rightarrow (43 \times 48)$

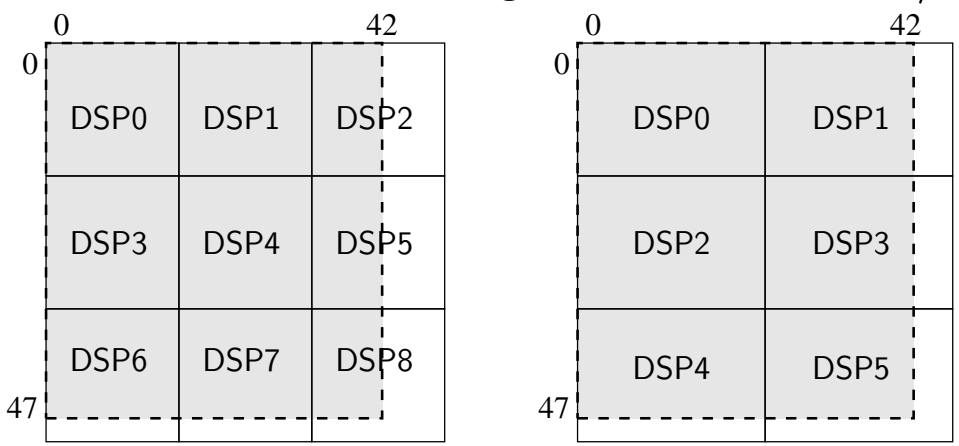
FPGAs have discrete-size multipliers

- 17×17 bit/DSP block for Xilinx VirtexIV
- 17×24 bit/DSP block for Xilinx Virtex5, Virtex6

Large multipliers using embedded multipliers

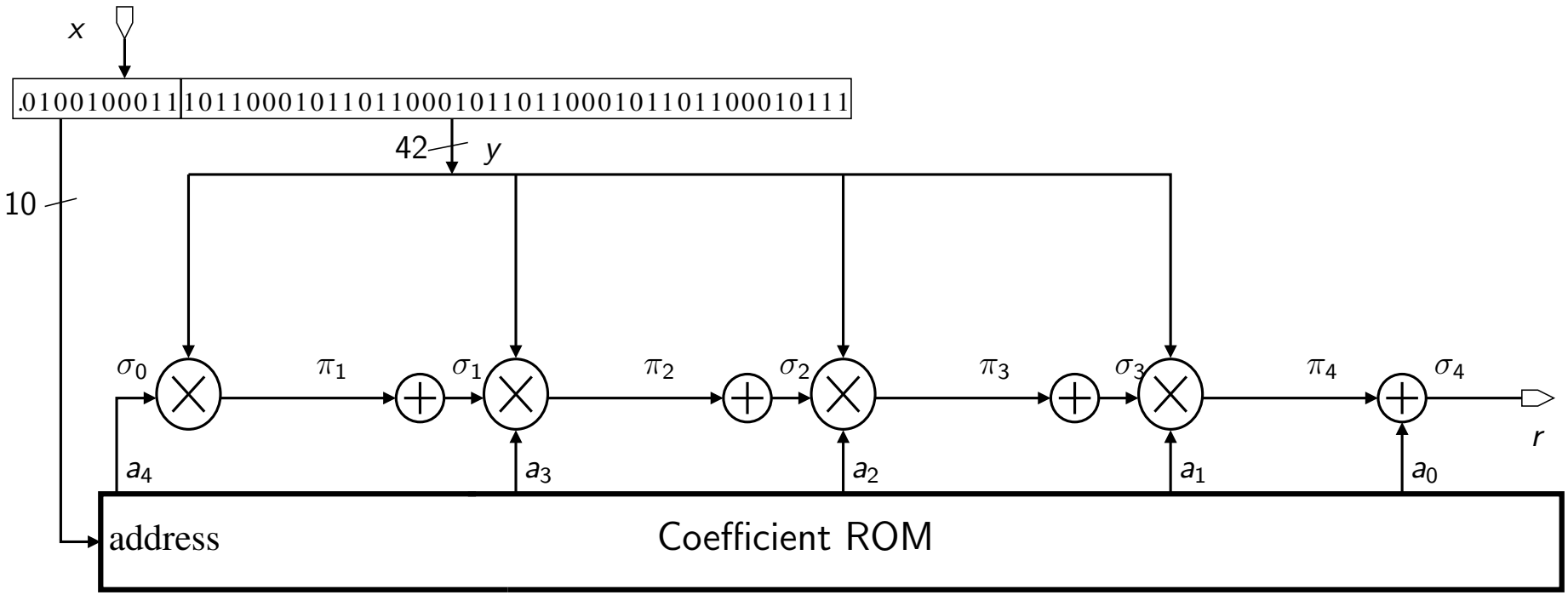
Take as example $\pi_4 = y \times \sigma_3 \rightarrow (43 \times 48)$:

- tile a rectangular board of size 43×48 with tiles[2] of:
  - 17×17 for VirtexIV
  - 17×25 and 25×17 for Virtex5 and Virtex6
- DSP cost = nb of tiles
  - (left 9 for VirtexIV, right 6 for Virtex5/6)

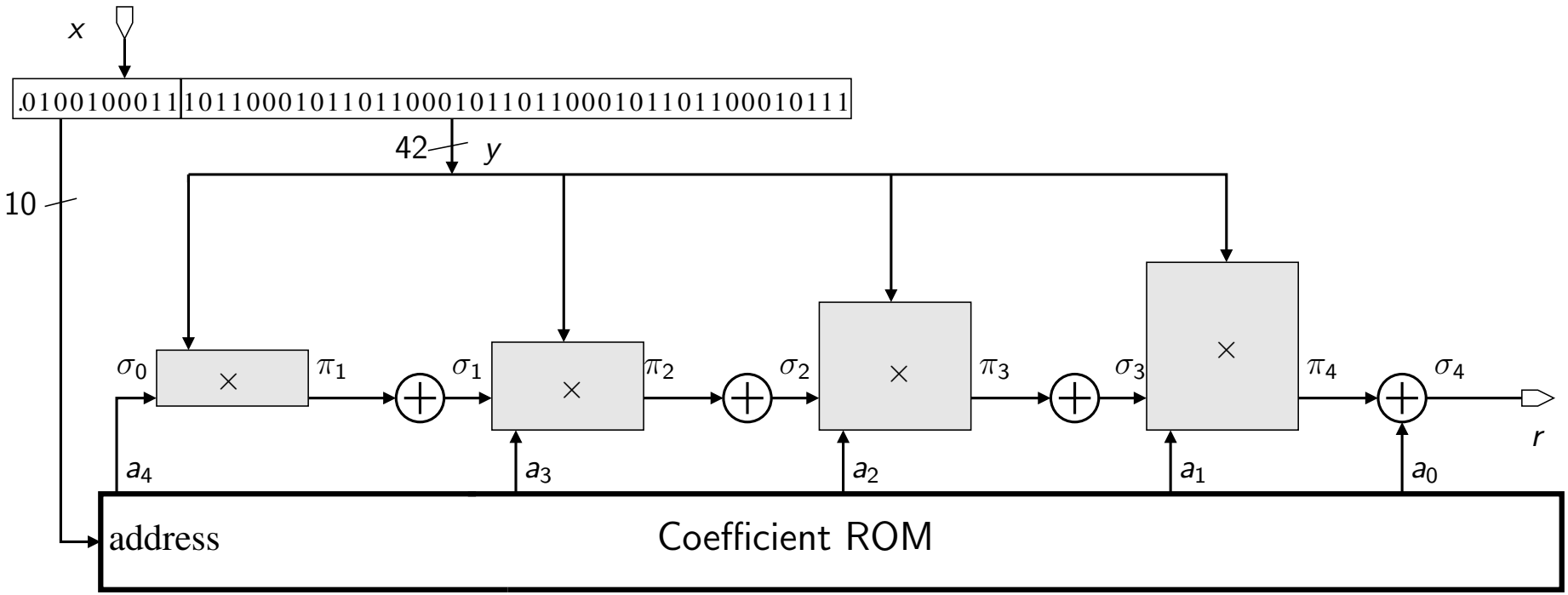


[2]Sebastian Banescu, Florent de Dinechin, Bogdan Pasca, Radu Tudoran. Multipliers for Floating-Point Double Precision and Beyond on FPGAs. HEART'10

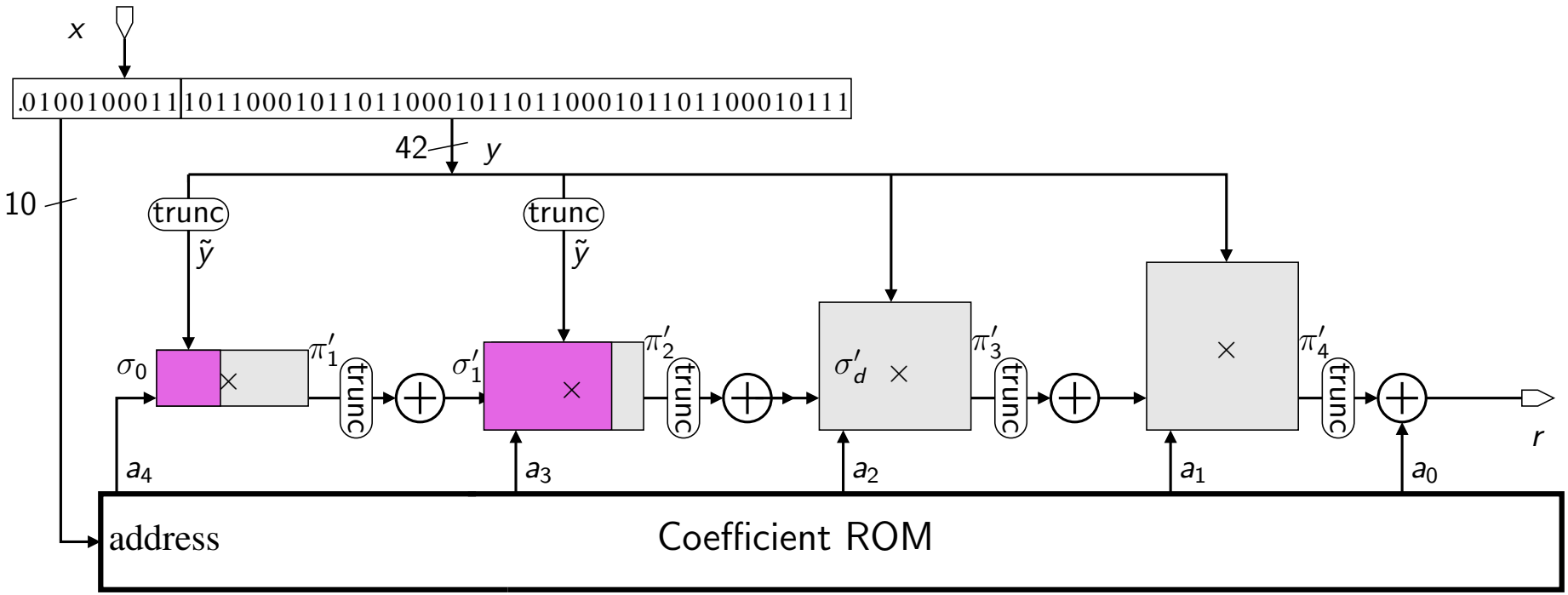
The architecture for $log_2(1+x), DP, d = 4$



The architecture for $log_2(1+x), DP, d = 4$



The architecture for $log_2(1+x), DP, d = 4$



truncation on y and on π_j

Parameter space exploration

FPGA

truncation of y ($\pi_j = \tilde{y} \times \sigma_{j-1}$)

- favor truncations to smallest multiple of multiplier input
- rule of thumb: truncate to get square multipliers

Parameter space exploration

FPGA

truncation of y ($\pi_j = \tilde{y} \times \sigma_{j-1}$)

- favor truncations to smallest multiple of multiplier input
- rule of thumb: truncate to get square multipliers

truncation on π_j ($\sigma_j = a_{d-j} + \pi_j, \pi_{j+1} = y \times \sigma_j$)

- truncation of π'_j defines the size of the sum σ'_j
- two upper bounds on g_j^{π}:
  - no bits to the right of the LSB of $\tilde{\pi}'_j$
  - guard bits should not increase the size of σ'_j over a multiple of the multiplier input size

Parameter exploration results

Take as example $\log_2(1+x)$ for double precision on Virtex4:
The size of the multiplications:

π_1	43×15	3 DSPs	π'_1	18×15	1 DSPs
π_2	43×26	6 DSPs	π'_2	35×26	4 DSPs
π_3	43×37	9 DSPs	π'_3	43×39	9 DSPs
π_4	43×48	9 DSPs	π'_4	43×52	9 DSPs

Total save of 4 DSPs

An in-depth view of our example

$f(x)$	36 bits			52 bits (double prec.)		
	d	k	Coeffs size	d	k	Coeffs size
$\log_2(1+x)$	3	256	$39, 31, 23, 15$	4	256	$55, 45, 35, 25, 15$
	2	4096	$39, 27, 15$	3	4096	$55, 43, 31, 19$

An in-depth view of our example

$f(x)$	36 bits			52 bits (double prec.)		
	d	k	Coeffs size	d	k	Coeffs size
$\log_2(1+x)$	3	256	$39, 31, 23, 15$	4	256	$55, 45, 35, 25, 15$
	2	4096	$39, 27, 15$	3	4096	$55, 43, 31, 19$

Synthesis Results using ISE 11.1 on VirtexIV.
Operators runs at over 320 MHz.
l is the latency of the operator in cycles.

$f(x)$	36 bits					52 bits (double prec.)				
	d.	l	slices	DSP	BRAM	d	l	slices	DSP	BRAM
$\log_2(1+x)$	3	20	425	11	4	4	33	1039	24	10
	2	11	214	5	22	3	26	722	17	38

Conclusion:

- An optimized fixed-point generator of functions.
- Uses state-of-the art polynomial approximation tool
- Scales up to **64bits and more**
- Finer **datapath optimization**
- **Pipelined** to a user-specified frequency
- **Fully automated** and integrated in open-source **FloPoCo**
- Basic brick in implementing **elementary floating-point functions** quickly in FloPoCo

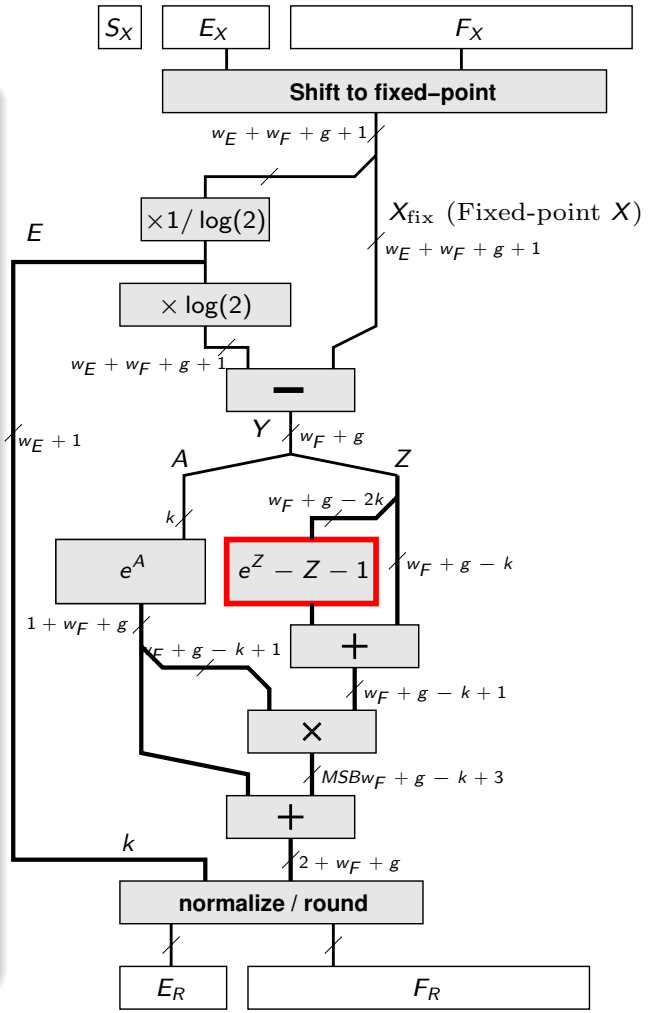
And above all, it's generic

```
Final report:
|---Entity TableGenerator_10_179:
|      Not pipelined
|   |---Entity SignedIntMultiplier_35_15:
|   |      Pipeline depth = 1
|   |---Entity IntAdder_26_f400_slice_SRL_noBUFFER_uid1_Alternative:
|   |      Pipeline depth = 1
|   |   |   |---Entity IntAdder_53_f325_slice_SRL_noBUFFER_uid2_Alternative:
|   |   |   |      Pipeline depth = 1
|   |   |---Entity IntCompressorTree_53_2:
|   |   |      Pipeline depth = 2
|   |---Entity SignedIntMultiplier_35_26:
|   |      Pipeline depth = 4
|   |---Entity IntAdder_39_f400_slice_SRL_noBUFFER_uid3_Classical:
|   |      Pipeline depth = 2
|   |   |   |---Entity IntAdder_87_f325_slice_SRL_noBUFFER_uid4_Classical:
|   |   |   |      Pipeline depth = 2
|   |   |---Entity IntCompressorTree_87_3:
|   |   |      Pipeline depth = 3
|   |---Entity SignedIntMultiplier_43_39:
|   |      Pipeline depth = 6
|   |---Entity IntAdder_51_f400_slice_SRL_noBUFFER_uid5_Classical:
|   |      Pipeline depth = 2
|   |   |   |---Entity IntAdder_87_f325_slice_SRL_noBUFFER_uid6_Classical:
|   |   |   |      Pipeline depth = 2
|   |   |---Entity IntCompressorTree_87_3:
|   |   |      Pipeline depth = 3
|   |---Entity SignedIntMultiplier_43_51:
|   |      Pipeline depth = 6
|   |---Entity IntAdder_103_f400_slice_SRL_noBUFFER_uid7_Classical:
|   |      Pipeline depth = 4
|---Entity PolynomialEvaluator_d4:
|      Pipeline depth = 29
|---Entity IntAdder_55_f400_slice_SRL_noBUFFER_uid8_Classical:
|      Pipeline depth = 2
```

Why put functions in hardware?

Acceleration of function evaluation

Double-precision floating-point exponential[1]:



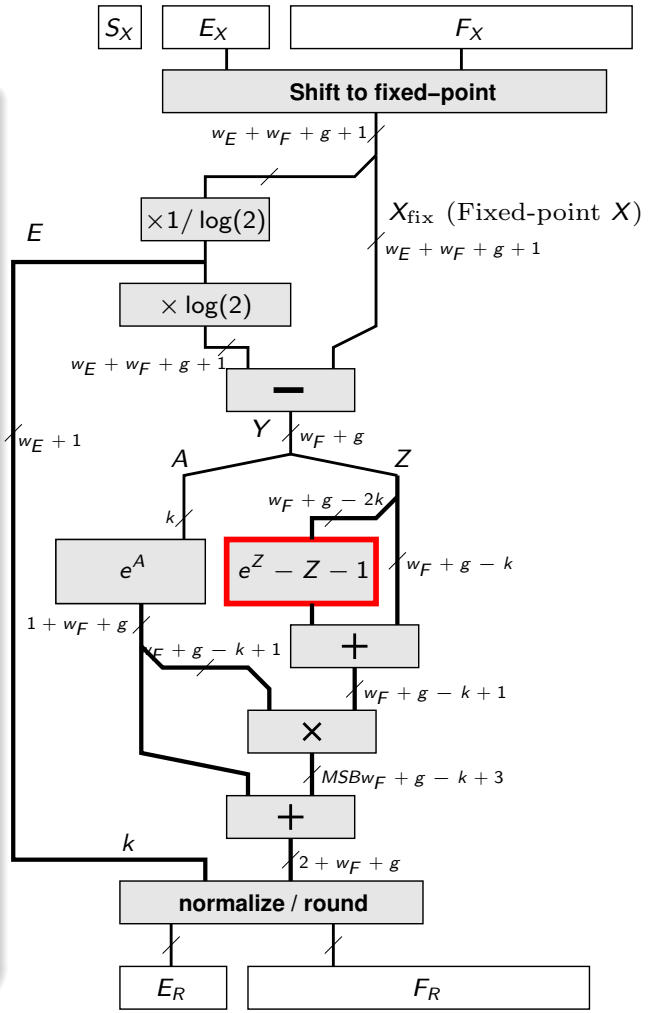
[1] F. de Dinechin, B. Pasca, *Floating-point exponential functions for DSP-enabled FPGAs*, FPT, 2010.

Why put functions in hardware?

Acceleration of function evaluation

Double-precision floating-point exponential[1]:

- Pentium Corei7:
  20 cycles / DPExp @ 3GHz: **150 MDPExp/s**
- Virtex6 FPGA:
  1 DPExp/cycle @ 400MHz: **400 MDPExp/s**



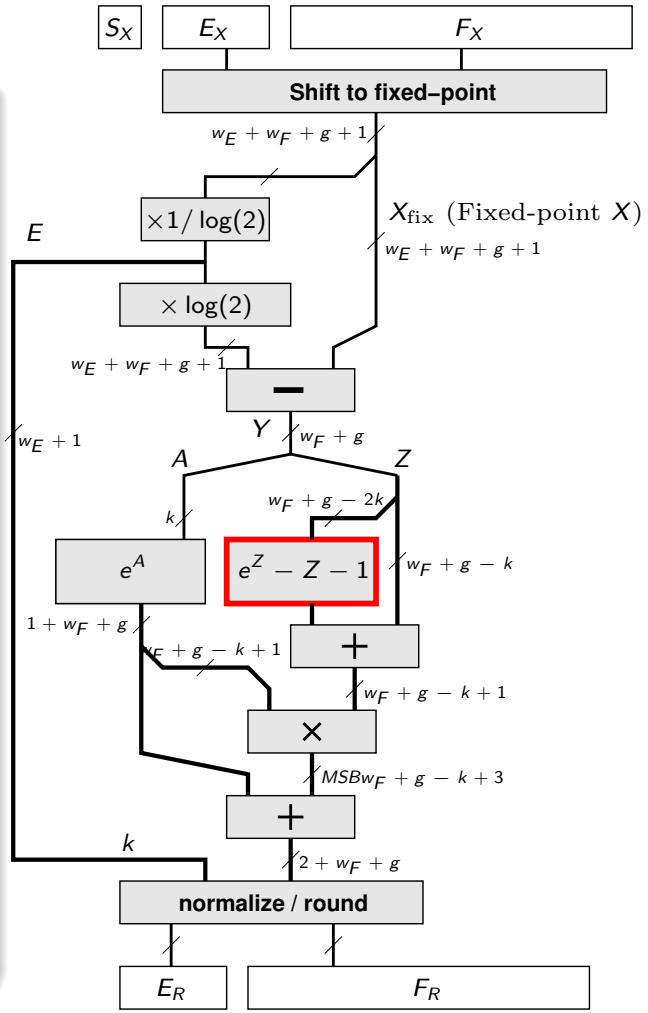
[1] F. de Dinechin, B. Pasca, *Floating-point exponential functions for DSP-enabled FPGAs*, FPT, 2010.

Why put functions in hardware?

Acceleration of function evaluation

Double-precision floating-point exponential[1]:

- Pentium Corei7:
  20 cycles / DPExp @ 3GHz: **150 MDPExp/s**
- Virtex6 FPGA:
  1 DPExp/cycle @ 400MHz: **400 MDPExp/s**
- Chip vs chip:
  8 Pentium cores vs 150 FPExp/FPGA
- ★ **Power consumption** also better
  (Intel MKL vector libm vs. FloPoCo v.2.0.0)



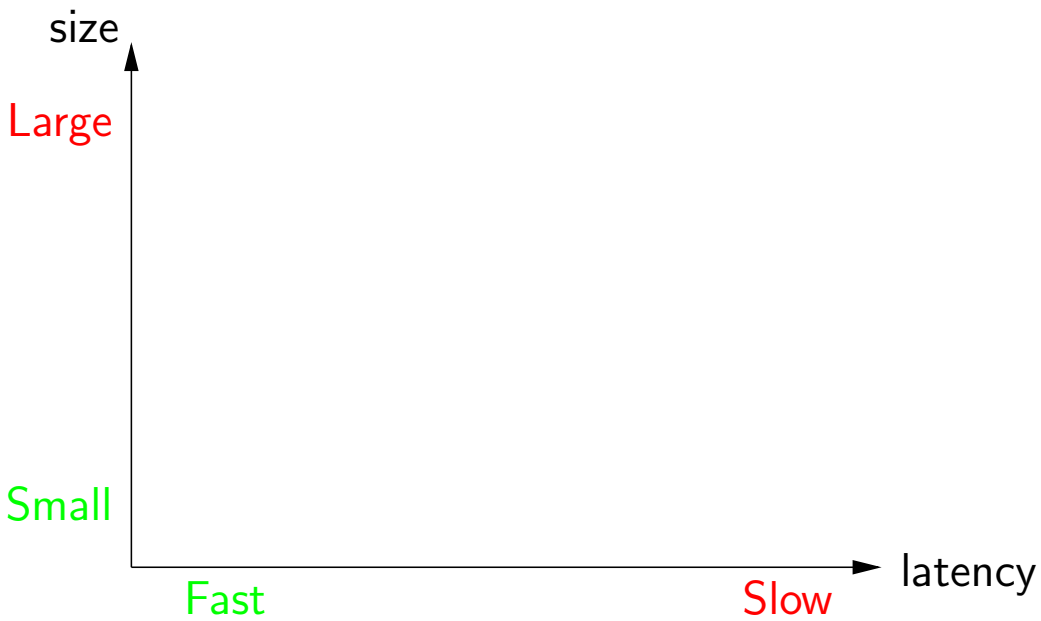
[1] F. de Dinechin, B. Pasca, *Floating-point exponential functions for DSP-enabled FPGAs*, FPT, 2010.

Thank you for your attention!

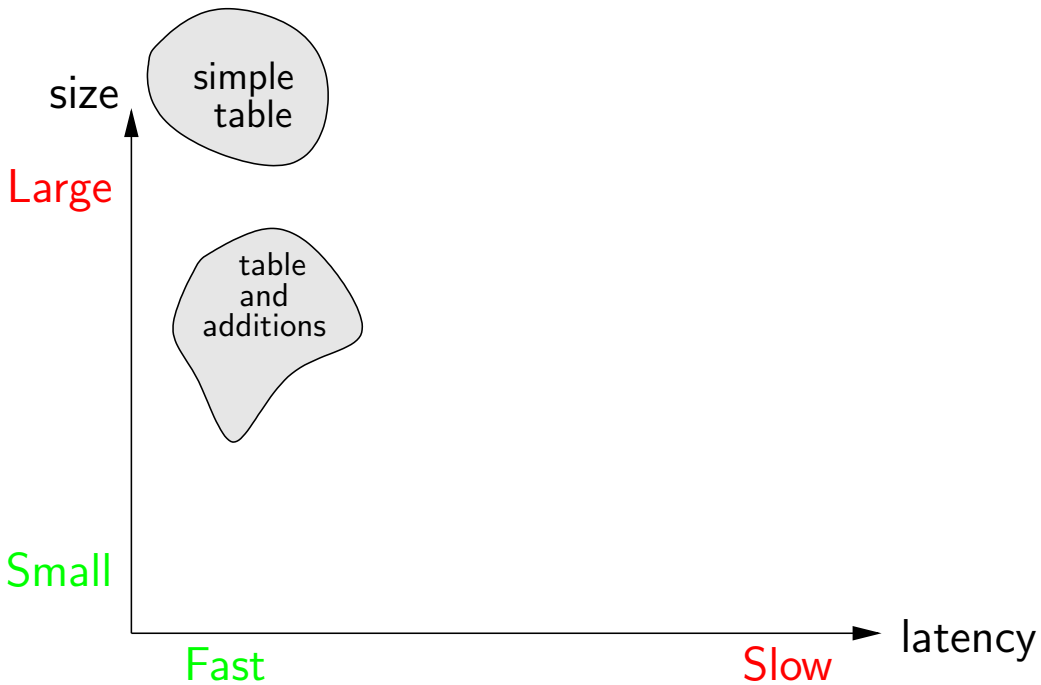
Try FloPoCo [3]

[3] http://www.ens-lyon.fr/LIP/Arenaire/Ware/FloPoCo/

Function evaluation methods

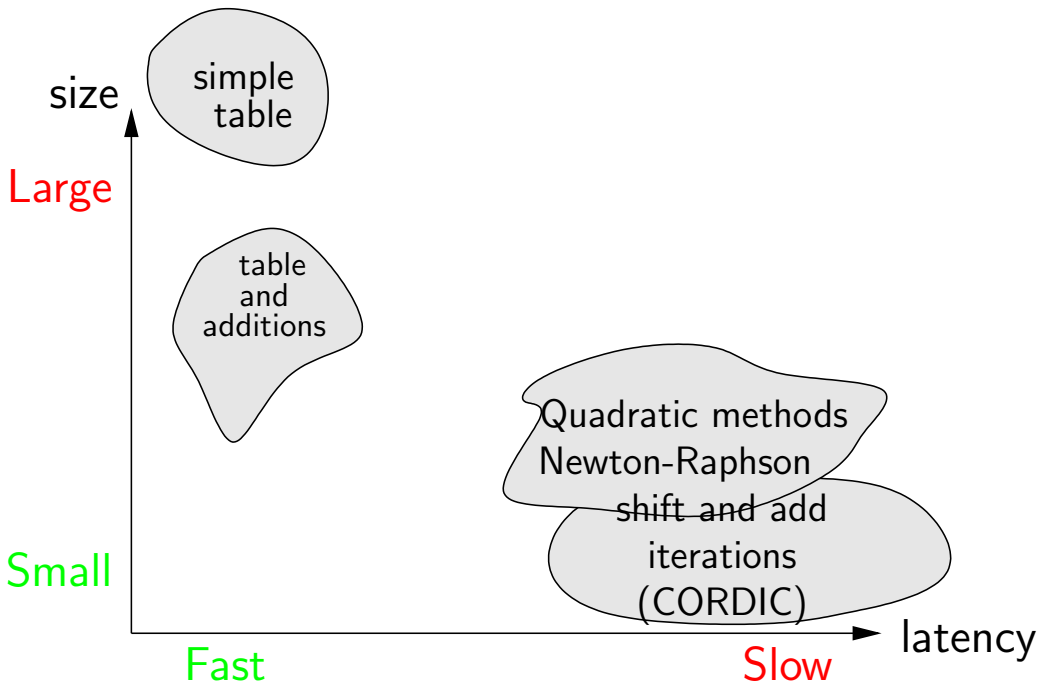


Function evaluation methods



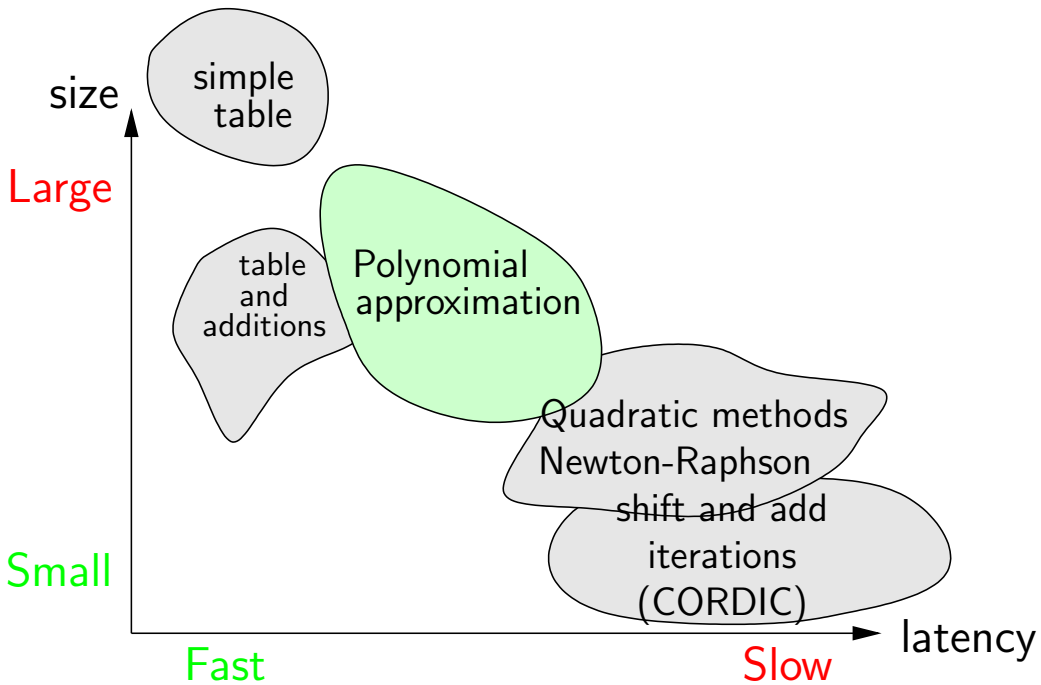
- simple table: bad scalability
- tables+adders: still bad scalability

Function evaluation methods



- simple table: bad scalability
- tables+adders: still bad scalability
- CORDIC: not generic enough
- Newton-Raphson: not generic

Function evaluation methods



- simple table: bad scalability
- tables+adders: still bad scalability
- CORDIC: not generic enough
- Newton-Raphson: not generic