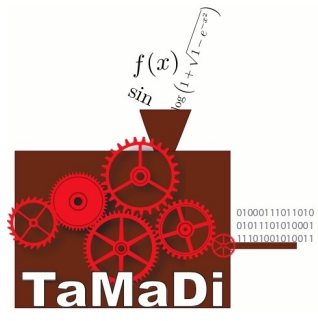
Conception et implantation d'algorithmes efficaces pour la résolution du dilemme du fabricant de tables sur architectures parallèles

Mourad Gouicem

PEQUAN Team, LIP6/UPMC

Lyon, France
7 Octobre 2013



Can we trust numerical computations?

"It makes me nervous to fly on airplanes since I know they are designed using floating-point arithmetic"

A. S. Householder

Can we trust numerical computations?

"It makes me nervous to fly on airplanes since I know they are designed using floating-point arithmetic"

A. S. Householder

$\sin(10^{22})$

- Google :
  0.46261304076

Can we trust numerical computations?

"It makes me nervous to fly on airplanes since I know they are designed using floating-point arithmetic"

A. S. Householder

$\sin(10^{22})$

- Google :
  0.46261304076
- OSX calculator :
  -0.029746237434271

Can we trust numerical computations?

"It makes me nervous to fly on airplanes since I know they are designed using floating-point arithmetic"

A. S. Householder

$\sin(10^{22})$

- Google :
  0.46261304076
- OSX calculator :
  -0.029746237434271
- Mac OSX libm :
  0.27090578830786904

Can we trust numerical computations?

"It makes me nervous to fly on airplanes since I know they are designed using floating-point arithmetic"

A. S. Householder

$\sin(10^{22})$

- Google :
  0.46261304076
- OSX calculator :
  -0.029746237434271
- Mac OSX libm :
  0.27090578830786904
- Sage :
  -0.85220084976718879

Can we trust numerical computations?

"It makes me nervous to fly on airplanes since I know they are designed using floating-point arithmetic"

A. S. Householder

$\sin(10^{22})$

- Google :
  0.46261304076
- OSX calculator :
  -0.029746237434271
- Mac OSX libm :
  0.27090578830786904
- Sage :
  -0.85220084976718879
- Sollya :
  $-0.852200849767188801772\ldots$

Can we trust numerical computations?

"It makes me nervous to fly on airplanes since I know they are designed using floating-point arithmetic"

A. S. Householder

$\sin(10^{22})$

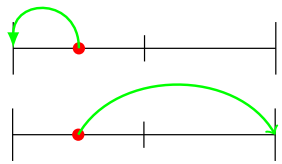
- Google :
  0.46261304076
- OSX calculator :
  -0.029746237434271
- Mac OSX libm :
  0.27090578830786904
- Sage :
  $-0.852200849767188$79
- Sollya :
  $-0.852200849767188$801772 . . .

The IEEE standard 754-2008 for floating-point arithmetic

Goal

Ensure predictable and portable numerical software.

1. Formats
   - A real x is approximated by $\widehat{x}$ using the scientific notation
   $$\widehat{x} = (-1)^s \times m \times \beta^e.$$
   - β can be 2 or 10,
   - $s \in \{0, 1\}$, and m and e have fixed, specified sizes.
     binary32, binary64, binary128
2. Rounding modes
   - Rounding to nearest
   - Directed rounding (towards 0, $-\infty$ and $+\infty$)



3. Correctly rounded operations :
   $+, -, \times, /, \sqrt{}$, *FMA* and conversions.

The case of elementary functions (exp, log, sin, cos, . . .)

$\Rightarrow$ IEEE 754-2008 only recommends correct rounding because of the Table Maker's Dilemma

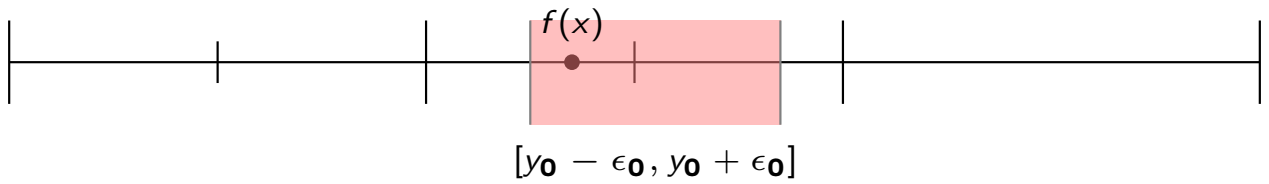
Examples of libm (correct rounding is 0.5 ULP)

- CUDA:
  maximum error is 2-8 ULP depending on the function
- AMD LibM:
  "the estimated accuracy is better than 1.0 ULP"
- Intel Vector Math Library (VML):
  "The design requirement for the HA functions is to have error less than 1.0 ulp"

The Table Maker's Dilemma

Problem

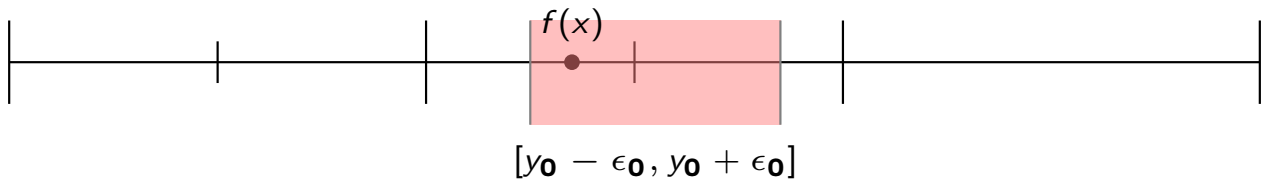
Generally, functions cannot be evaluated exactly. We have to approximate the results !

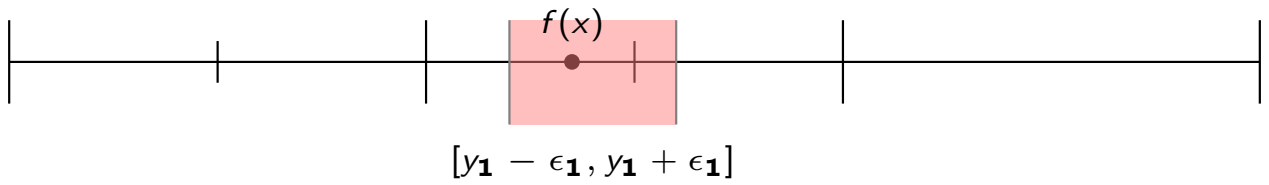


The Table Maker's Dilemma

Problem

Generally, functions cannot be evaluated exactly. We have to approximate the results !

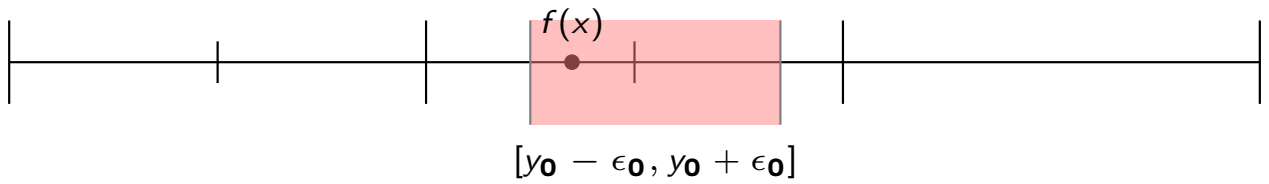


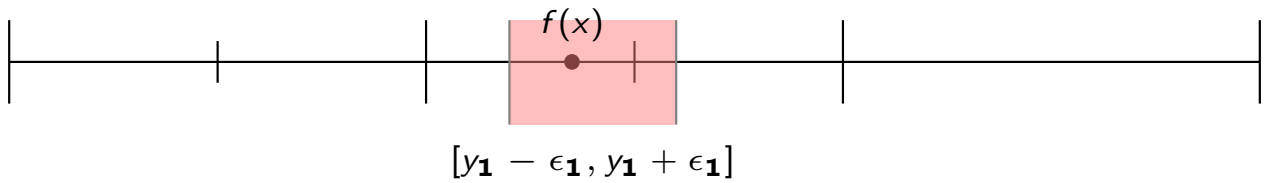


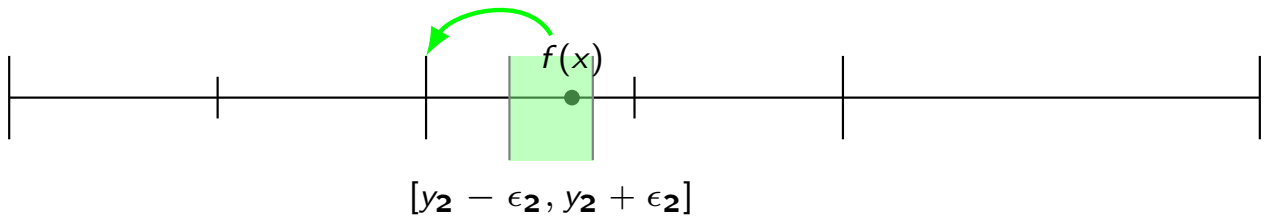
The Table Maker's Dilemma

Problem

Generally, functions cannot be evaluated exactly. We have to approximate the results !



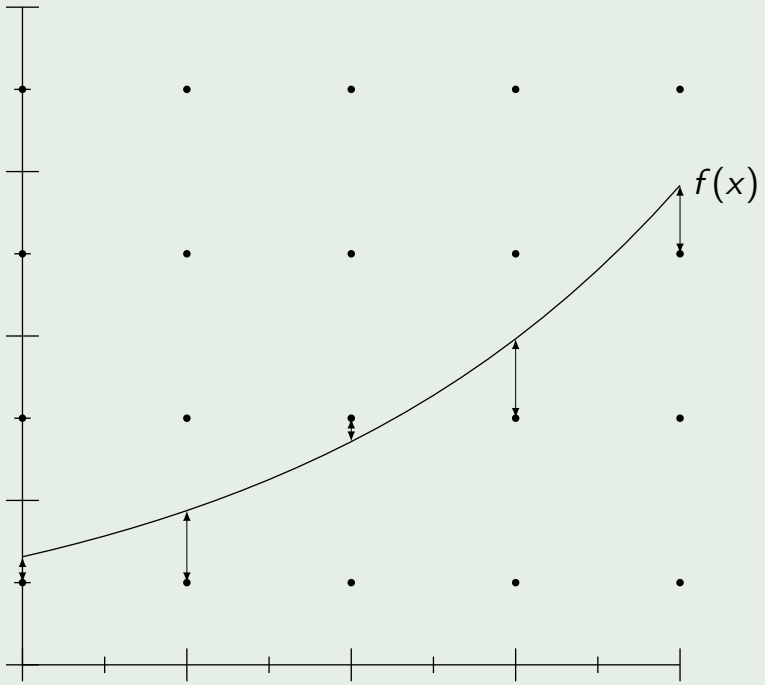




The Table Maker's Dilemma

Given a function f defined over I and a rounding mode $\circ_p$, find ε such that $\forall x \in I$

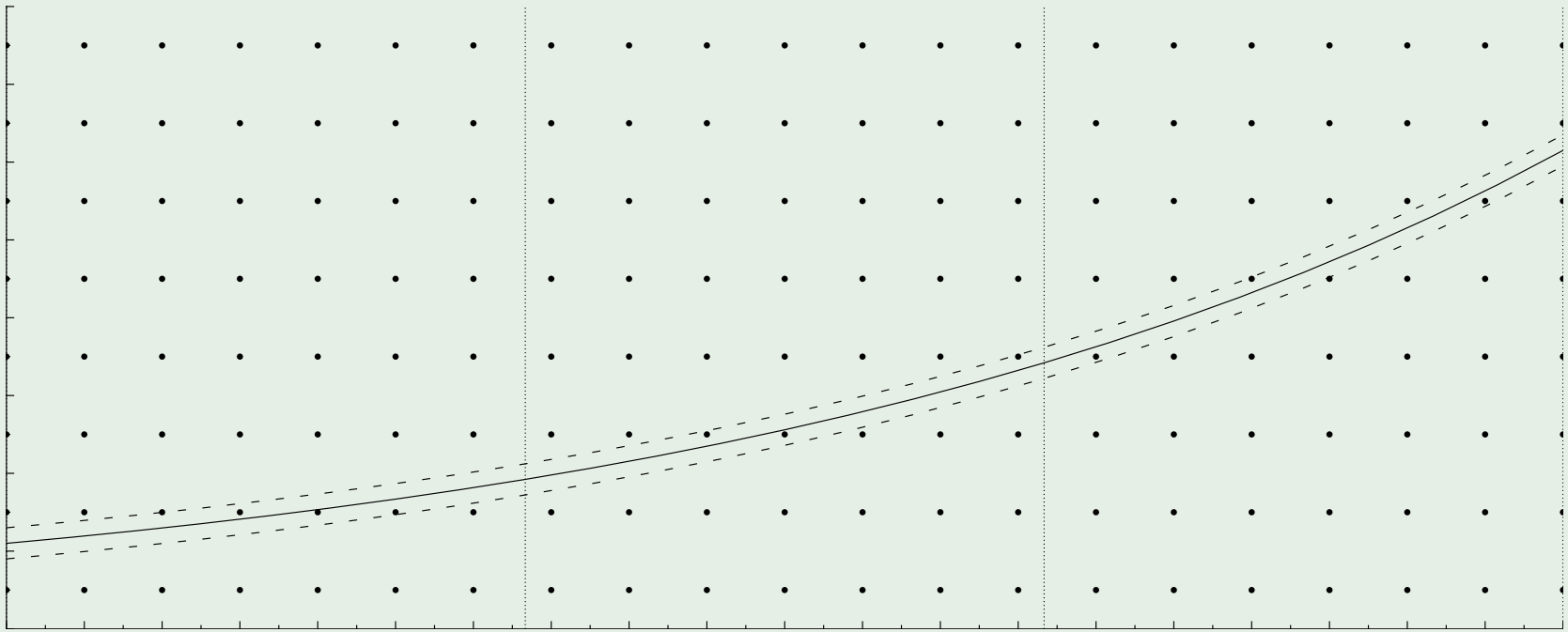
$$\circ_p(f(x) - \varepsilon) = \circ_p(f(x) + \varepsilon).$$



The Table Maker's Dilemma

Lefèvre-Muller-Tisserand framework

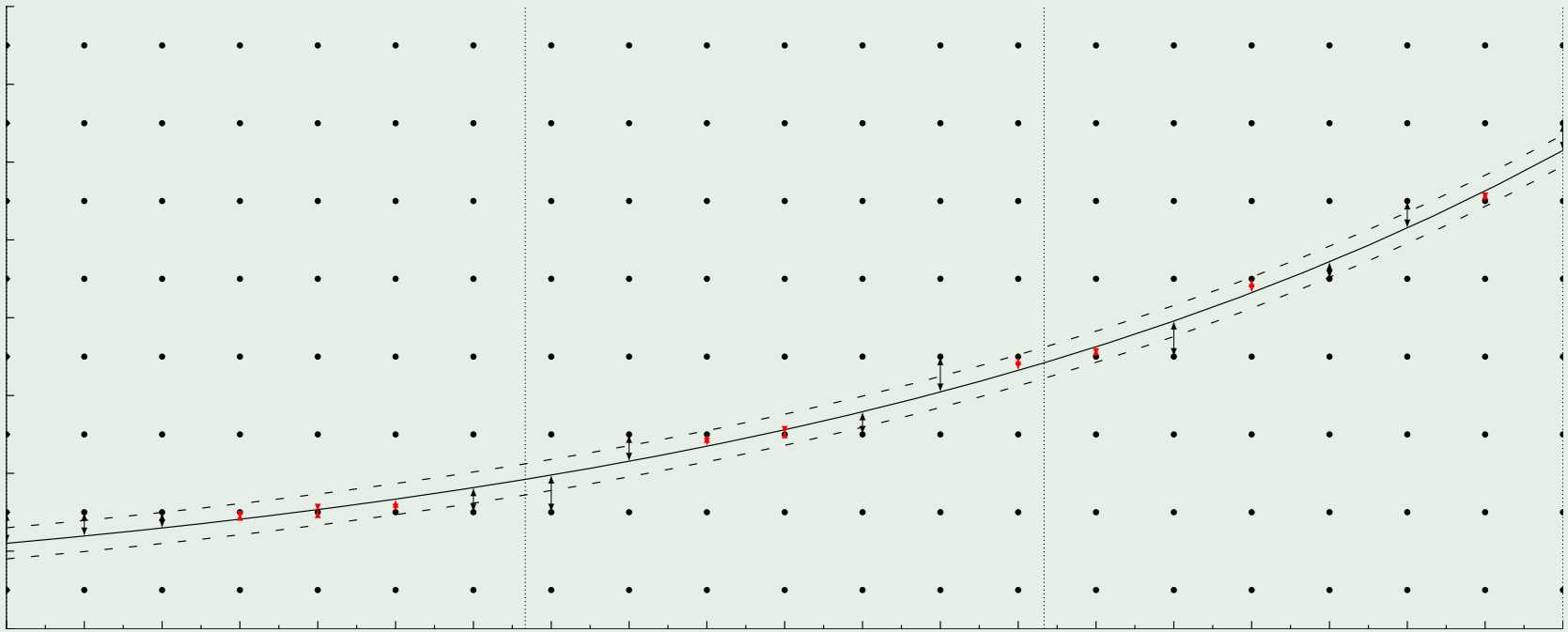
1. Split the function domain and approximate the function on each sub-domain by polynomials with error ϵ.



The Table Maker's Dilemma

Lefèvre-Muller-Tisserand framework

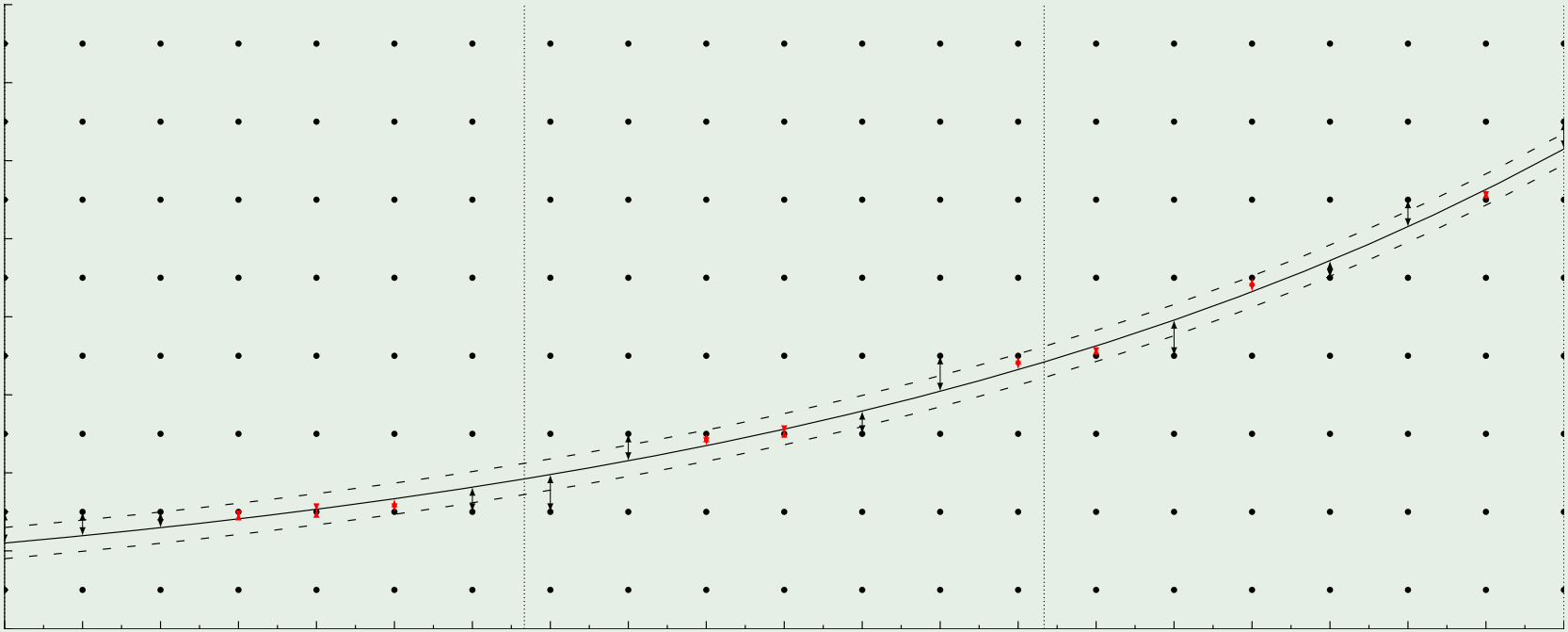
1. Split the function domain and approximate the function on each sub-domain by polynomials with error ϵ.
2. Search for ϵ hard-to-round cases.



The Table Maker's Dilemma

Lefèvre-Muller-Tisserand framework

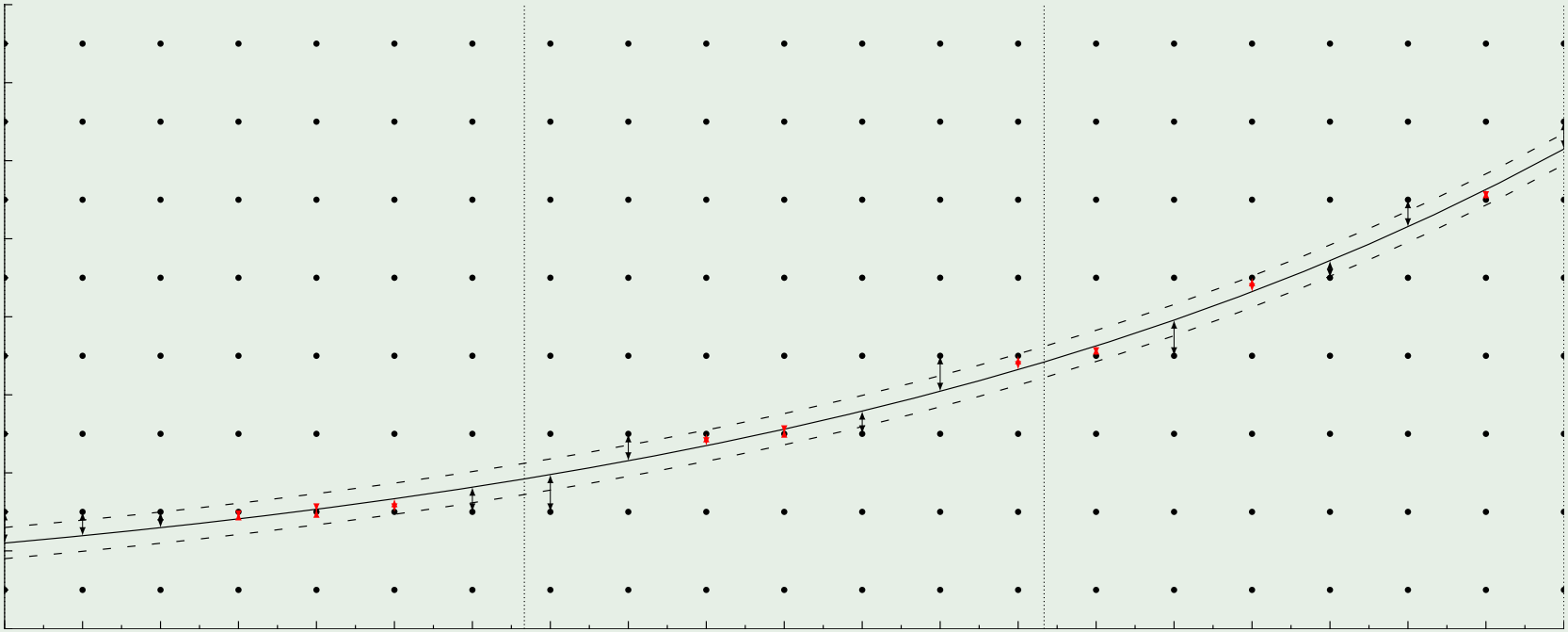
1. Split the function domain and approximate the function on each sub-domain by polynomials with error ϵ.
2. Search for ϵ hard-to-round cases.
3. Find the hardness-to-round ϵ of f among the HR-cases.
   Very negligible computation time as there are few HR-cases.



The Table Maker's Dilemma

Lefèvre-Muller-Tisserand framework

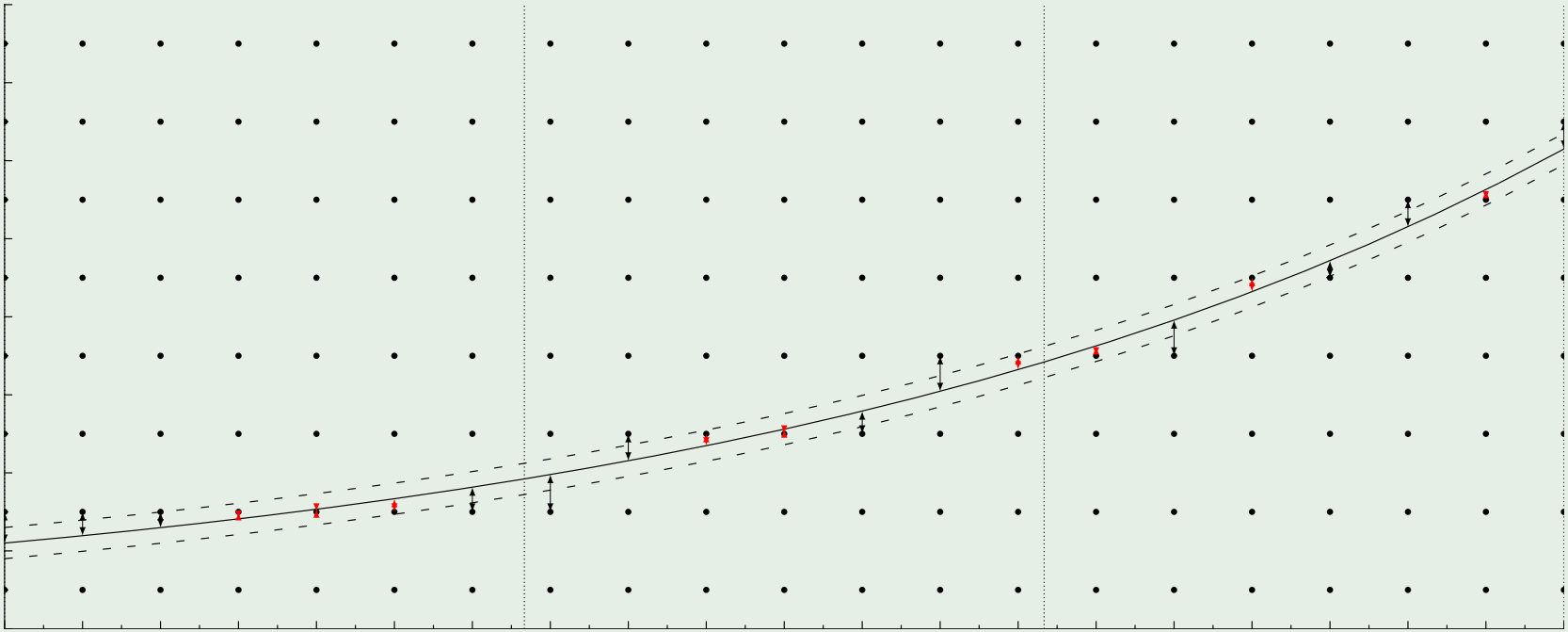
1. Split the function domain and approximate the function on each sub-domain by polynomials with error ϵ.
2. Search for ϵ hard-to-round cases. Most compute intensive step.
3. Find the hardness-to-round ϵ of f among the HR-cases. Very negligible computation time as there are few HR-cases.



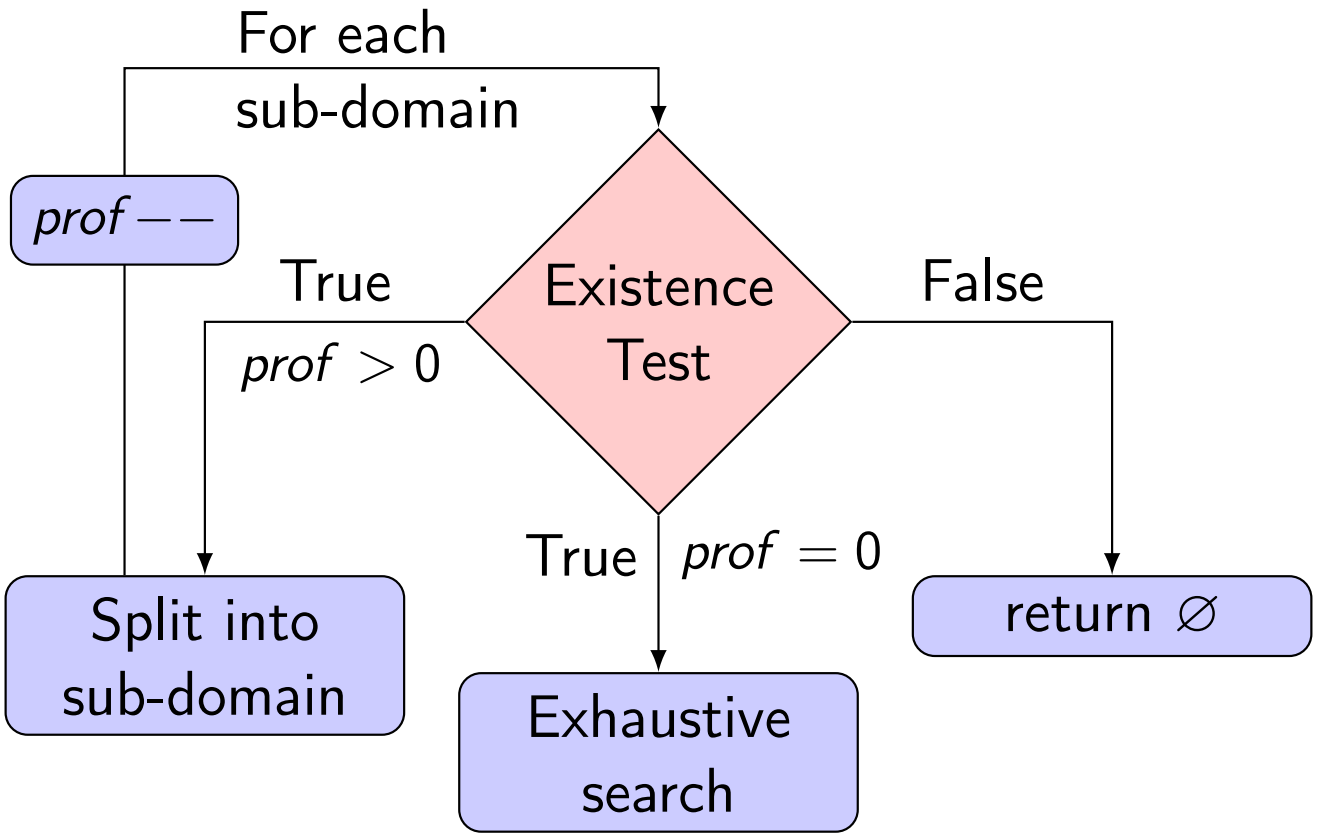
The Table Maker's Dilemma

Lefèvre-Muller-Tisserand framework

1. Split the function domain and approximate the function on each sub-domain by polynomials with error ϵ.
2. Search for ϵ hard-to-round cases. Embarrassingly parallel !
3. Find the hardness-to-round ϵ of f among the HR-cases. Very negligible computation time as there are few HR-cases.



HR-case search by isolation



HR-case existence test

Solutions (with p the floating-point precision)

- Exhaustive search: $O(2^p)$;
- Lefèvre when $\deg P = 1$: $O(2^{2p/3})$ intervals in $O(p^2)$;
- Stehlé-Lefèvre-Zimmermann (SLZ) when $\deg P > 1$: $O(2^{p/2})$ intervals in $O(\text{poly}(p, \deg P))$.

Example of computation times

- e^x in full domain and $p = 53$ with Lefèvre: 5 years of CPU time;
- 2^x in $[1/2, 1)$ and $p = 64$ with SLZ: 8 years of CPU time.

Parallelism levels

Multicore parallelism

Provides parallelism among cores and nodes.

Single Instruction Multiple Data (SIMD)

Provides parallelism within a core. Available both on CPU and GPU.

Parallelism levels

Multicore parallelism

Provides parallelism among cores and nodes. Is directly exploited.

Single Instruction Multiple Data (SIMD)

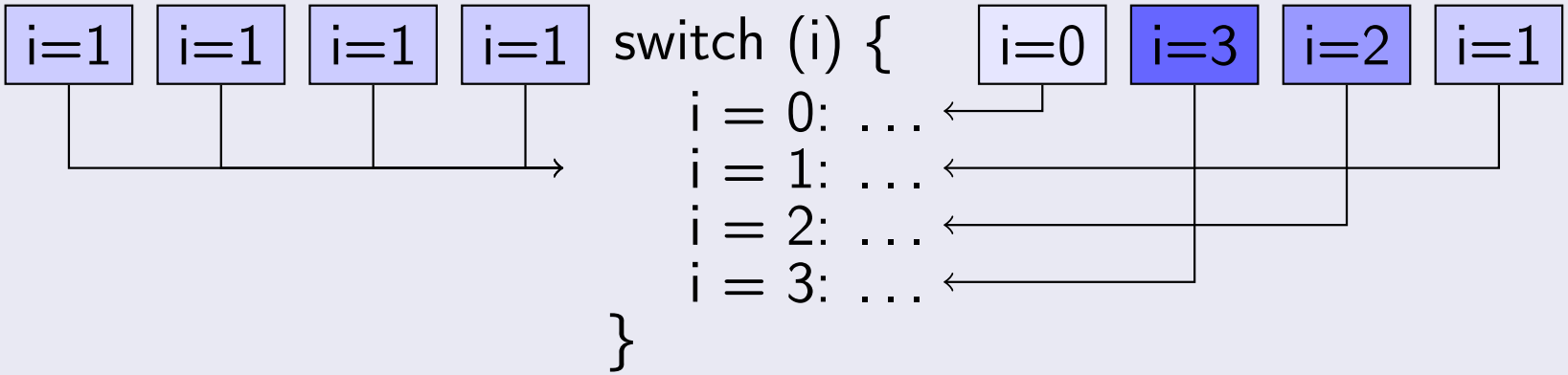
Provides parallelism within a core. Available both on CPU and GPU. Needs regular algorithms.

Programming the SIMD units

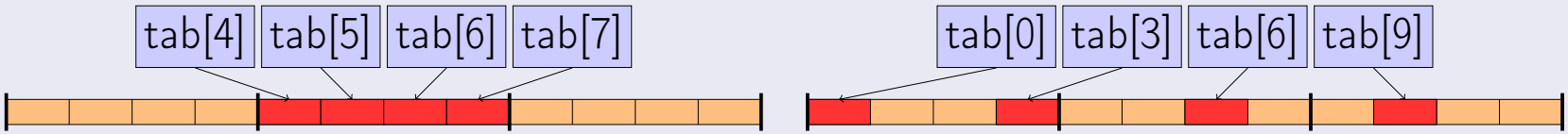
- Explicit vectorization: vector instructions (SSE, AVX, ...)
- Implicit vectorization: Single Program Multiple Data (SPMD) on SIMD (CUDA, OpenCL, ispc)
  - Launch multiple scalar threads running the same program
  - Group their execution on SIMD units

SIMD constraints

Control flow regularity



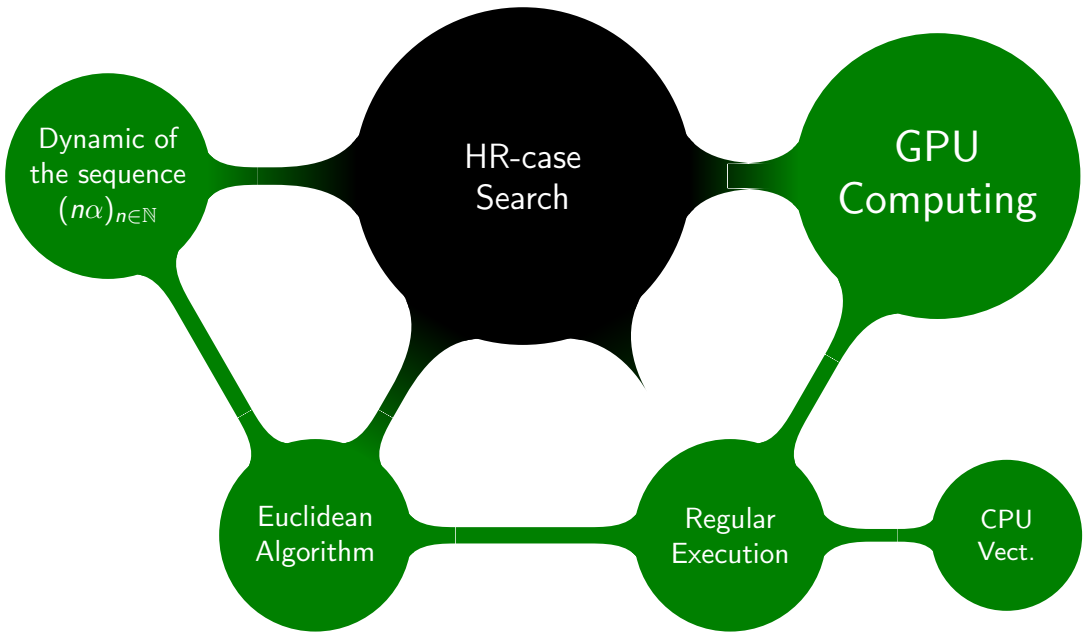
Memory regularity



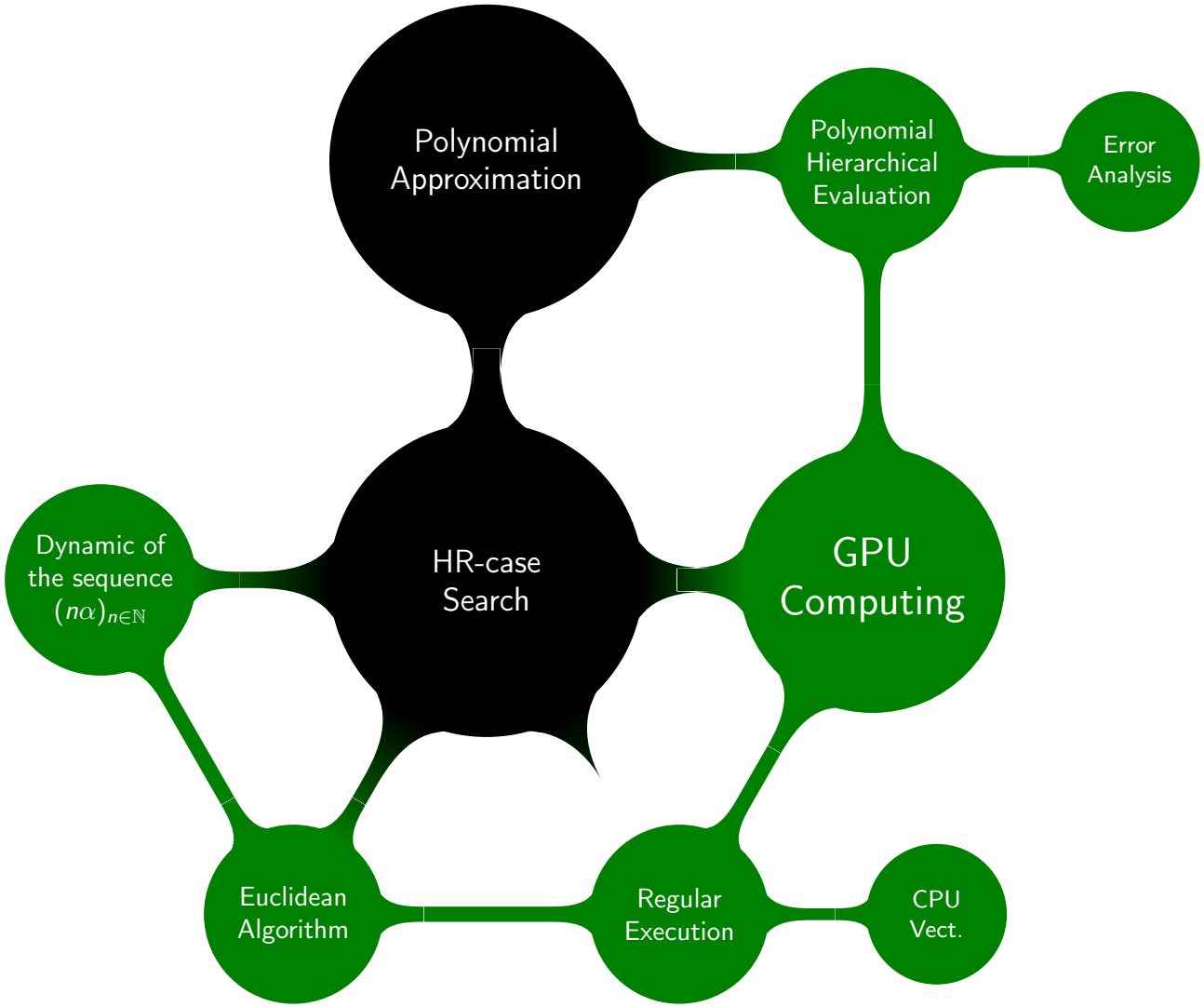
Challenges

- Binary128 is currently out of reach.
- Compute the hardest-to-round case for each of the 32 functions recommended by the IEEE std 754-2008 in binary64.
- Tackle any function in binary64 in reasonable time.
- Provide Regular algorithms to fully exploit SIMD units.

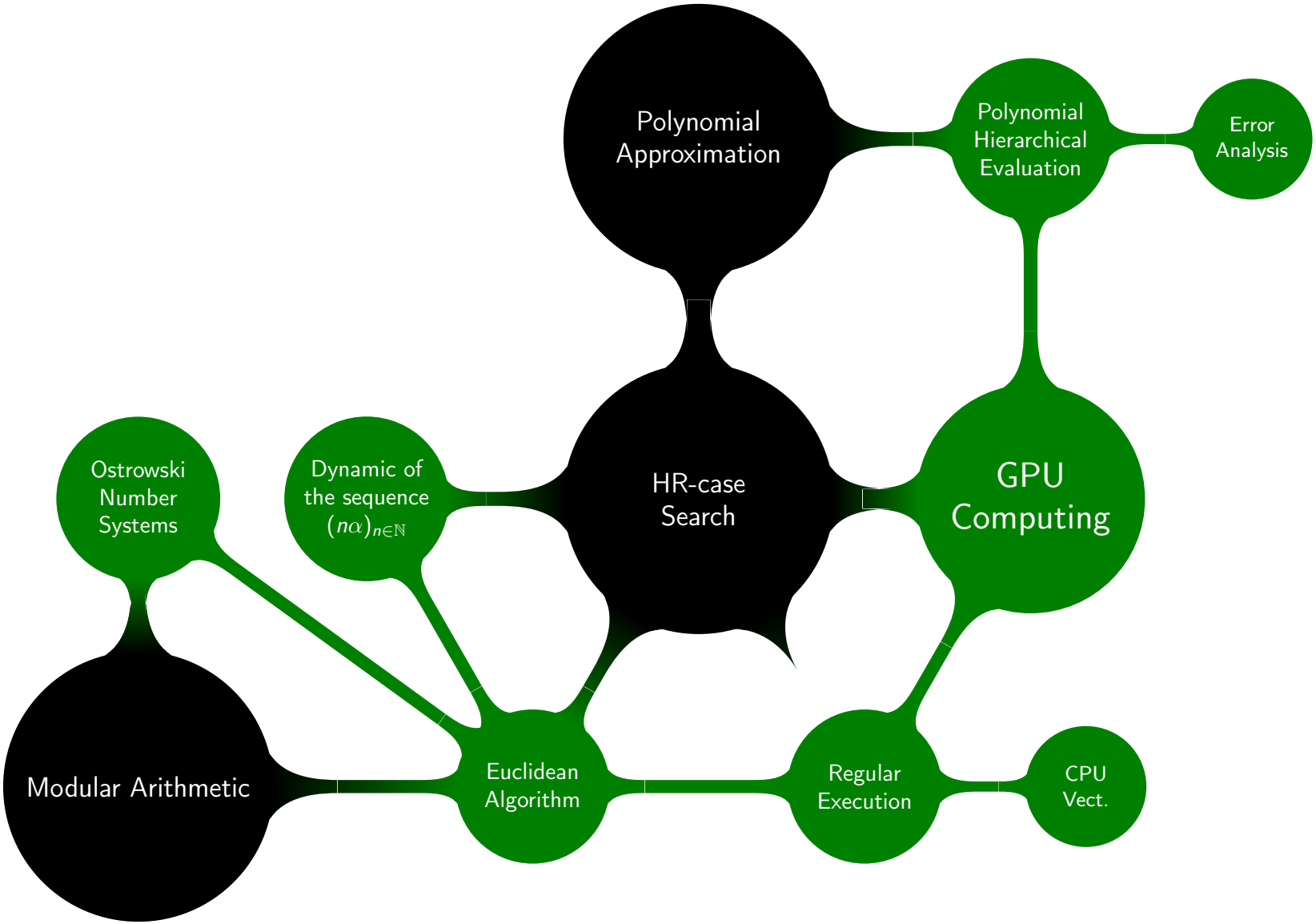
Overview of tools and contributions



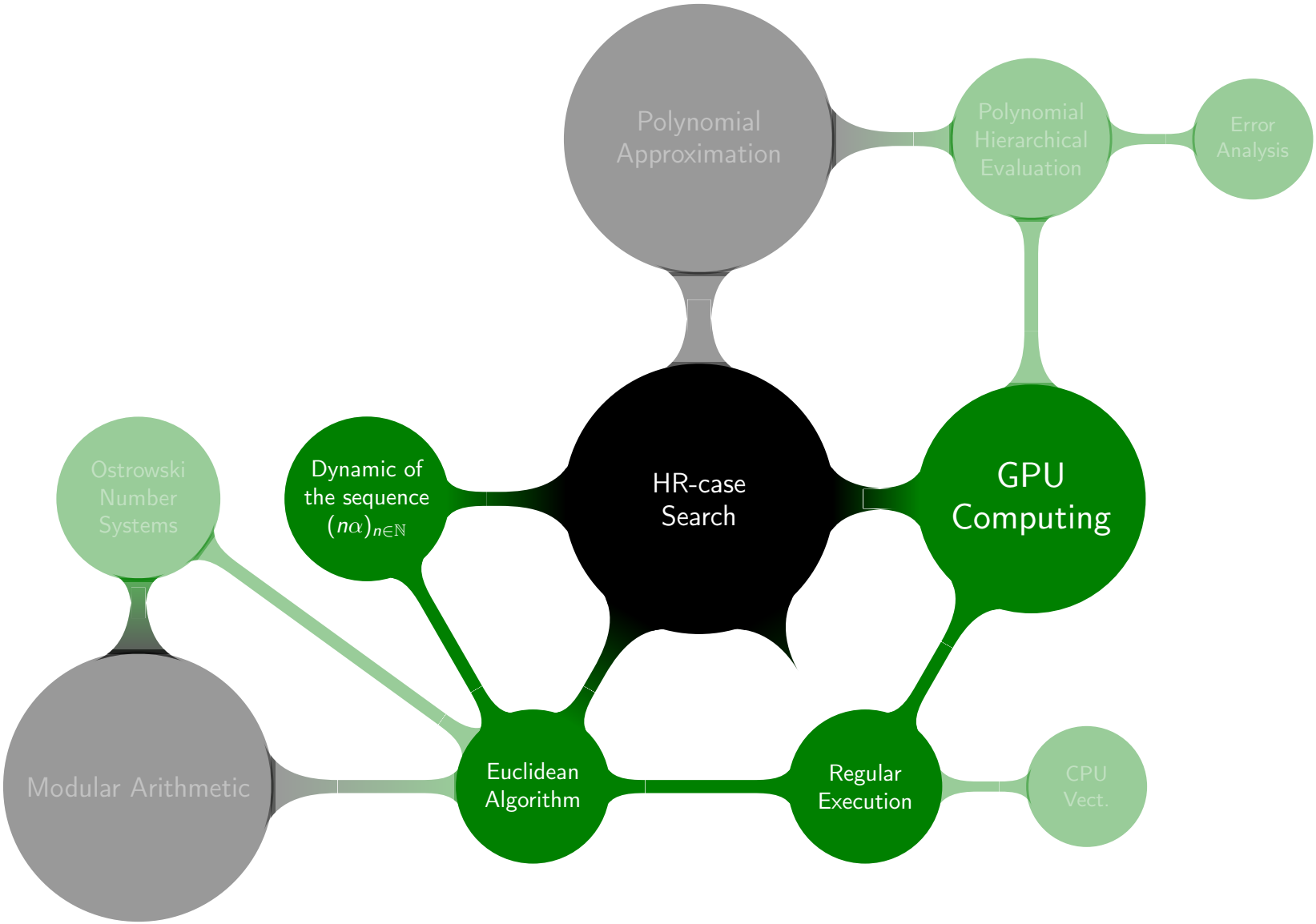
Overview of tools and contributions



Overview of tools and contributions



Overview of tools and contributions



Parallelization of the framework: HR-case search

Lefèvre HR-case search : Pros

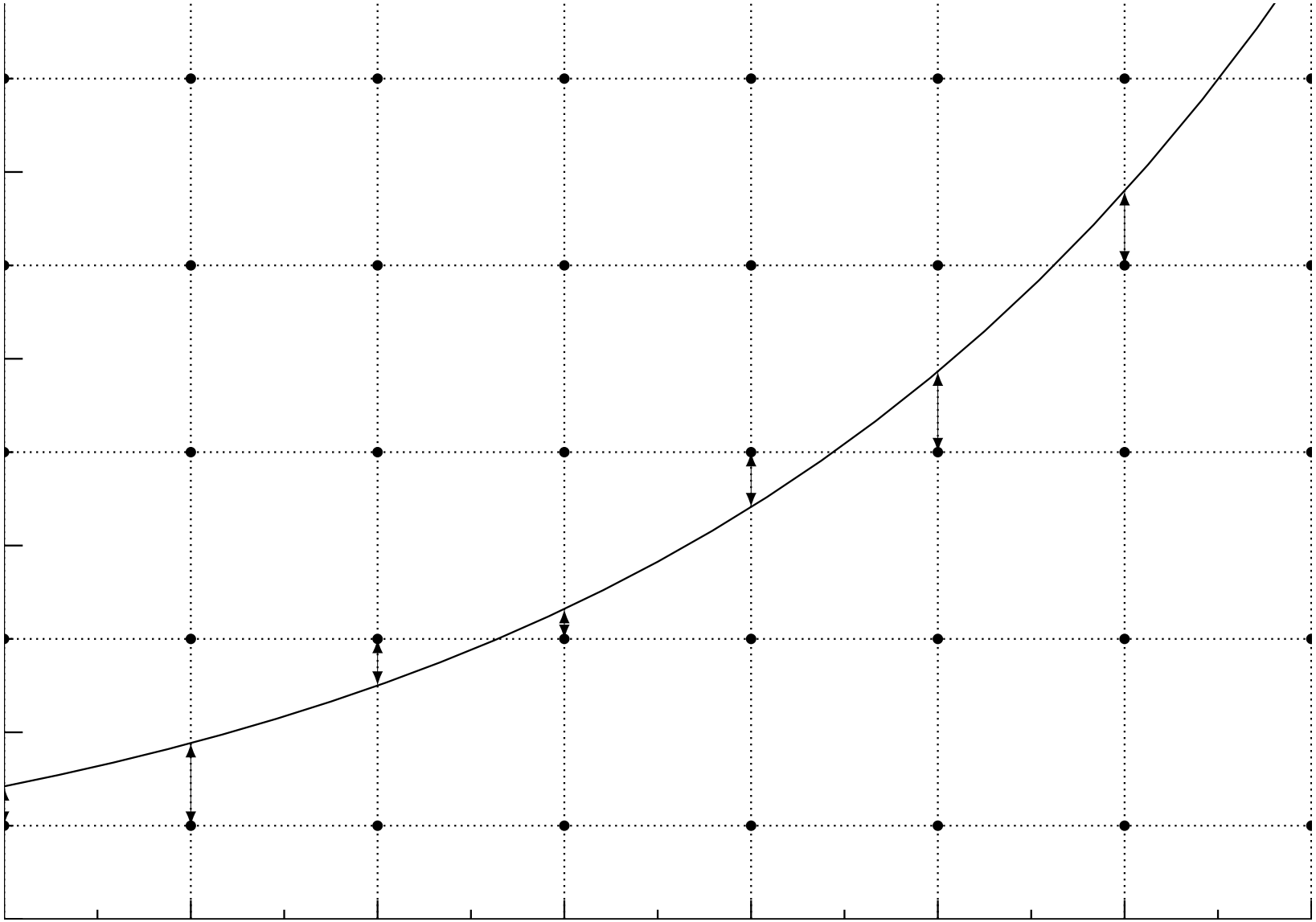
- Efficient for binary64 in practice: all known hardest-to-round in binary64 have been generated by Lefèvre;
- Embarrassingly parallel: computation on each sub-domain are independent
- Fine-grain parallelism: few data and computation per sub-domain (one polynomial and one error).

} Perfect for GPU!

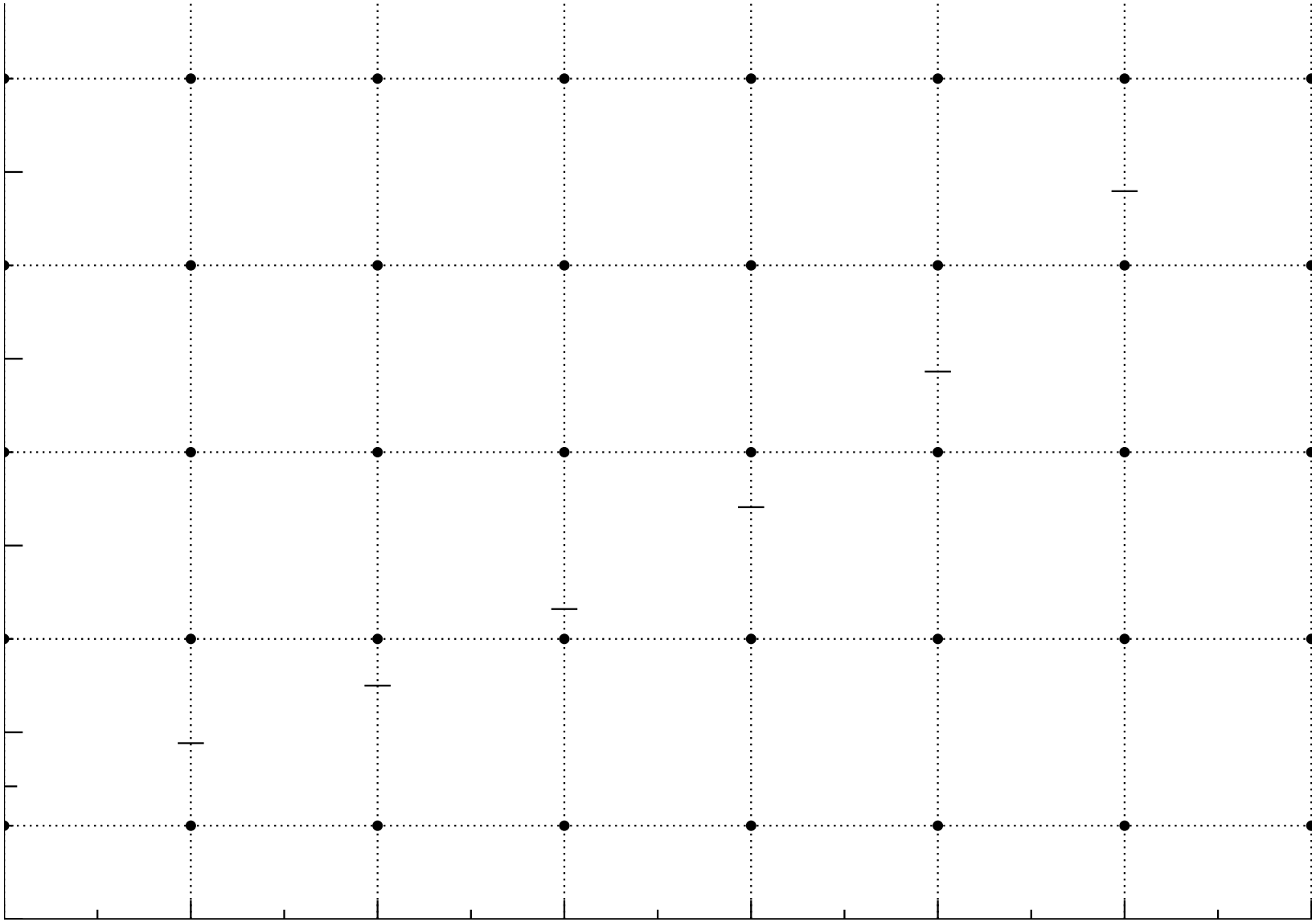
Cons

- Divergent control flow

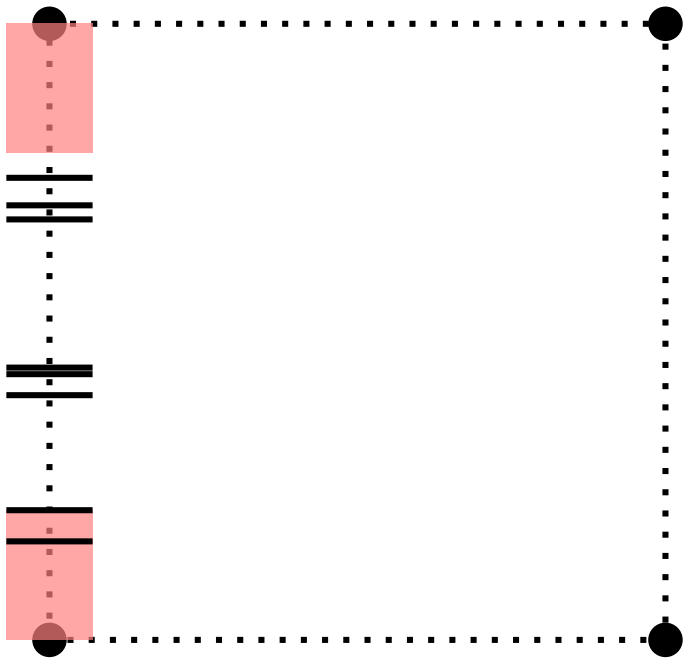
From the TMD to diophantine approximation



From the TMD to diophantine approximation



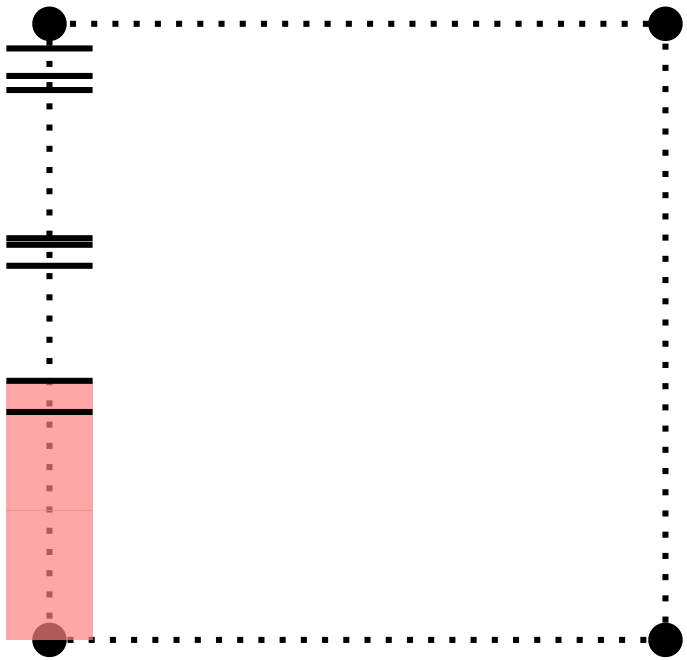
From the TMD to diophantine approximation



Given $P(x) \in \mathbb{R}[x]$, is there any $x \in \{0, \ldots, N\}$ such that

$$|P(x) \ cmod \ 1| < \epsilon?$$

From the TMD to diophantine approximation



Given $P(x) \in \mathbb{R}[x]$, is there any $x \in \{0, \ldots, N\}$ such that

$$P(x) + \epsilon \ mod \ 1 < 2\epsilon.$$

Lefèvre HR-case existence test

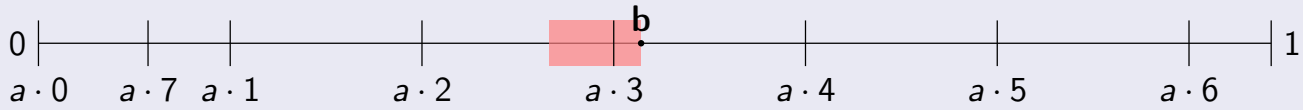
Problem

Given $b - ax \in \mathbb{R}[x]$, is there any $x \in \{0, \ldots, N\}$ such that

$$b - ax \mod 1 < \epsilon.$$

Geometrically

Is there any multiple of a in $\{ax \mod 1 \mid x \leq N\}$ at a distance less than ϵ to the left of b?

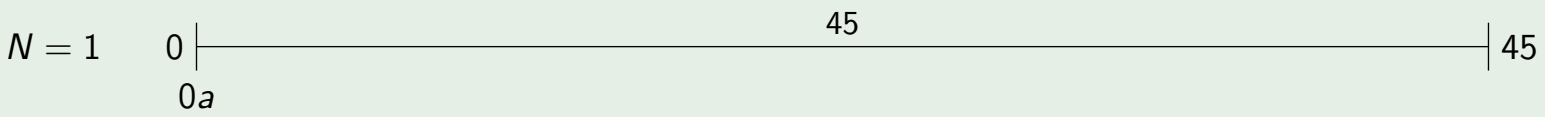


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

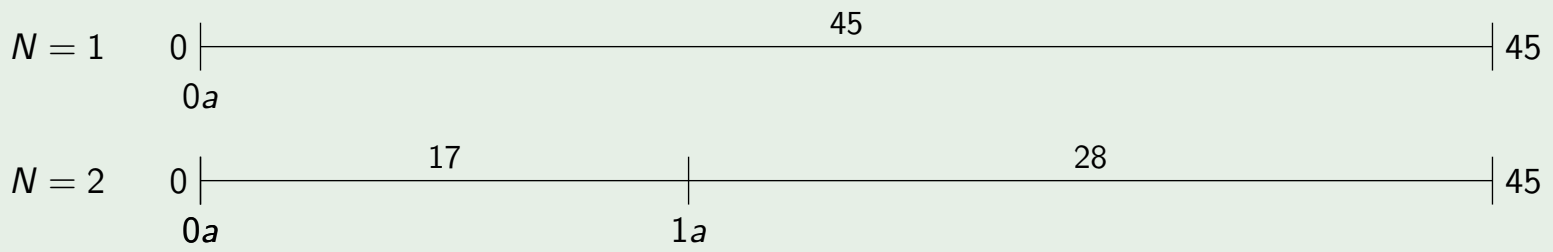


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

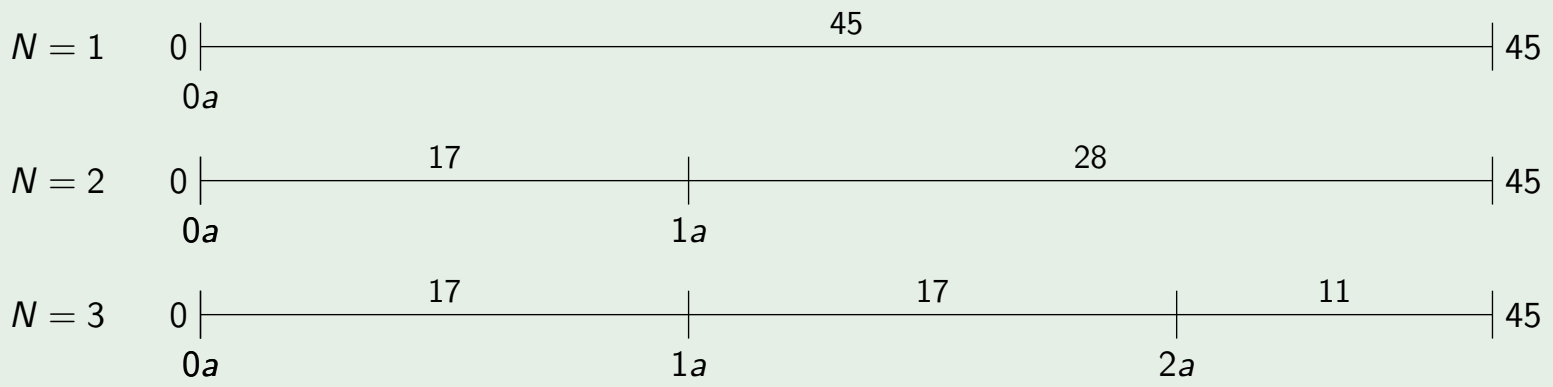


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

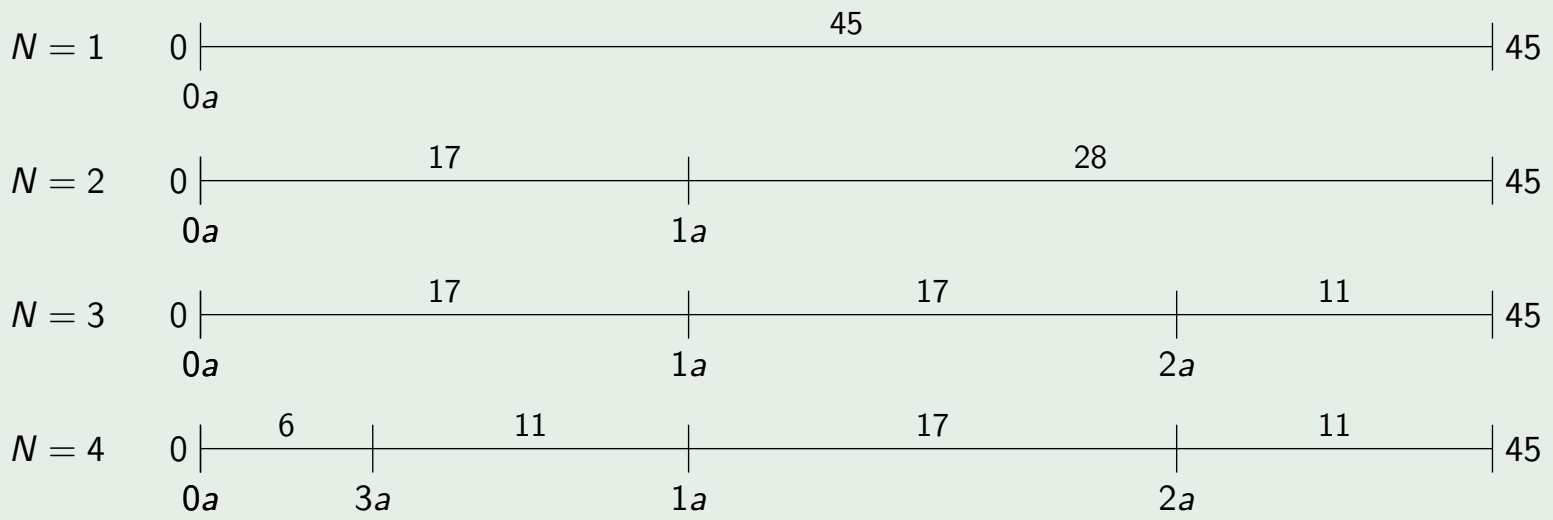


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

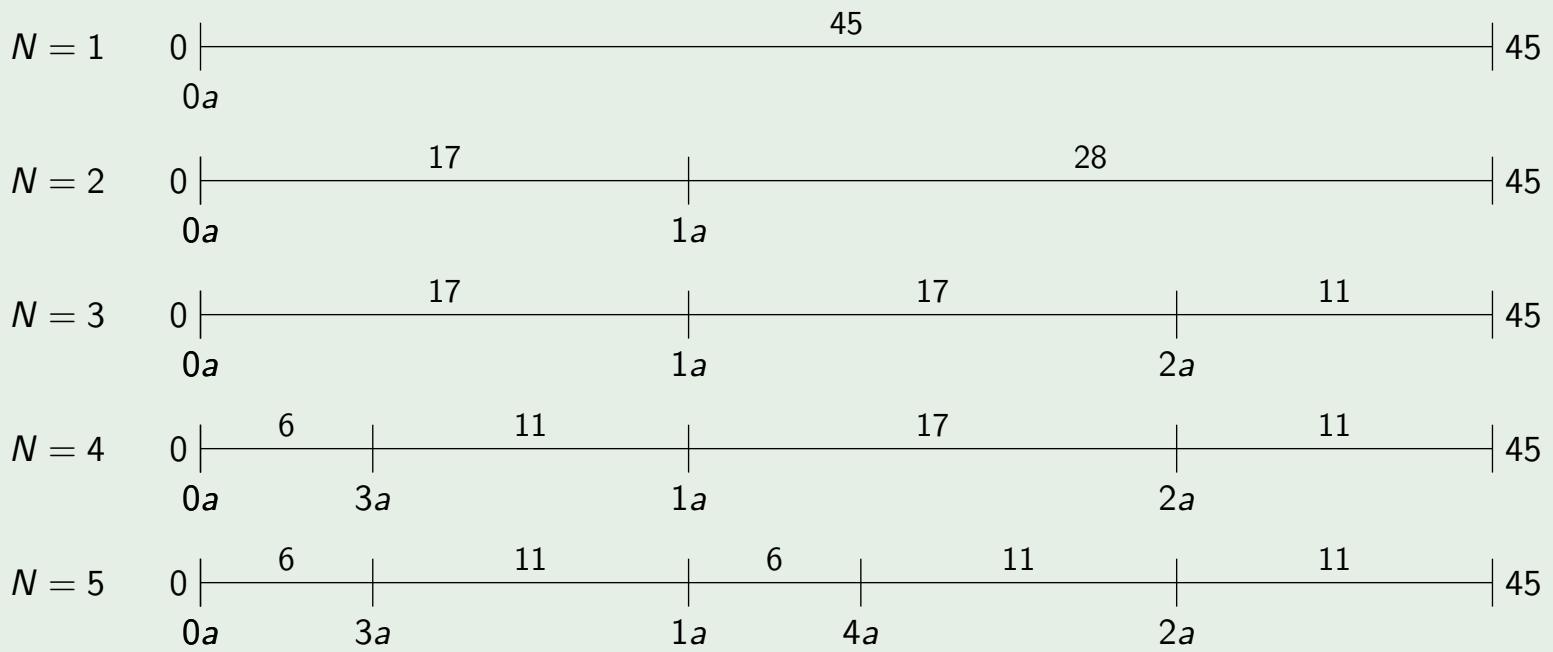


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

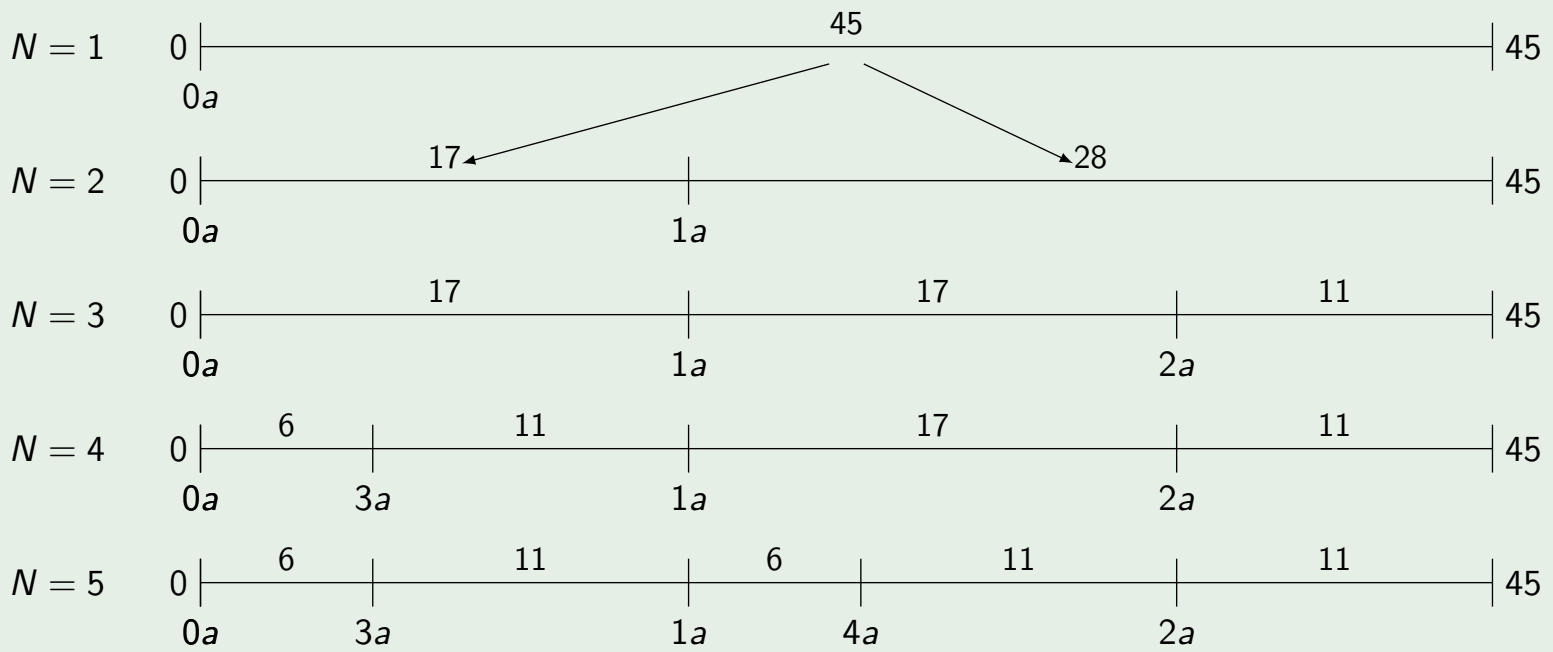


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

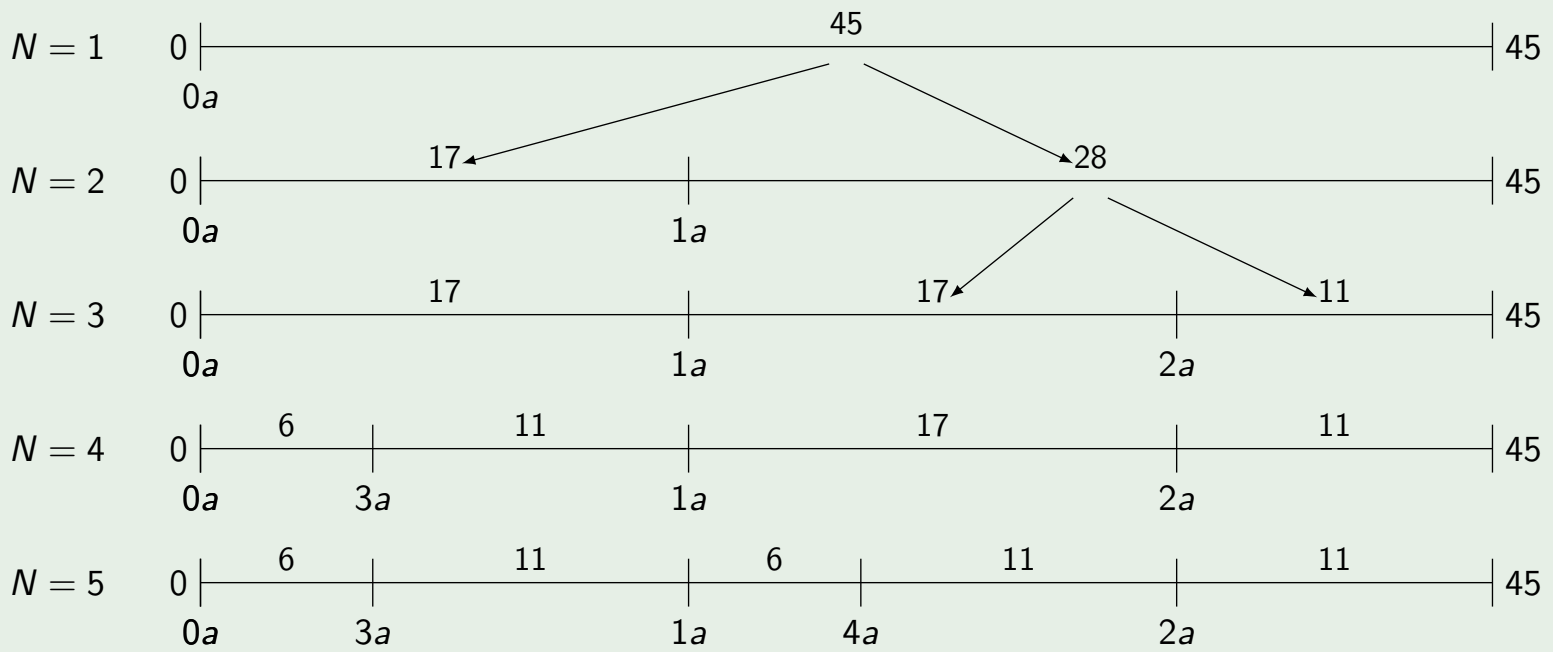


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

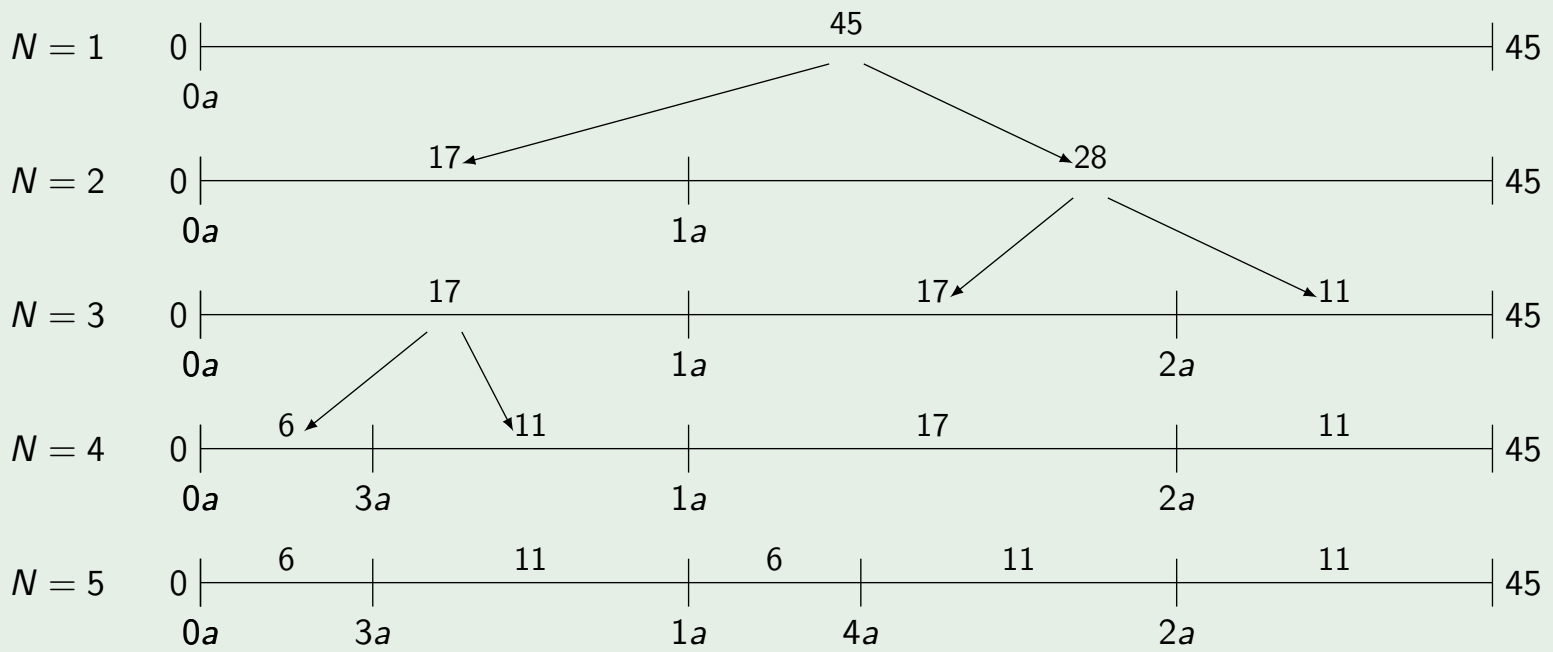


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

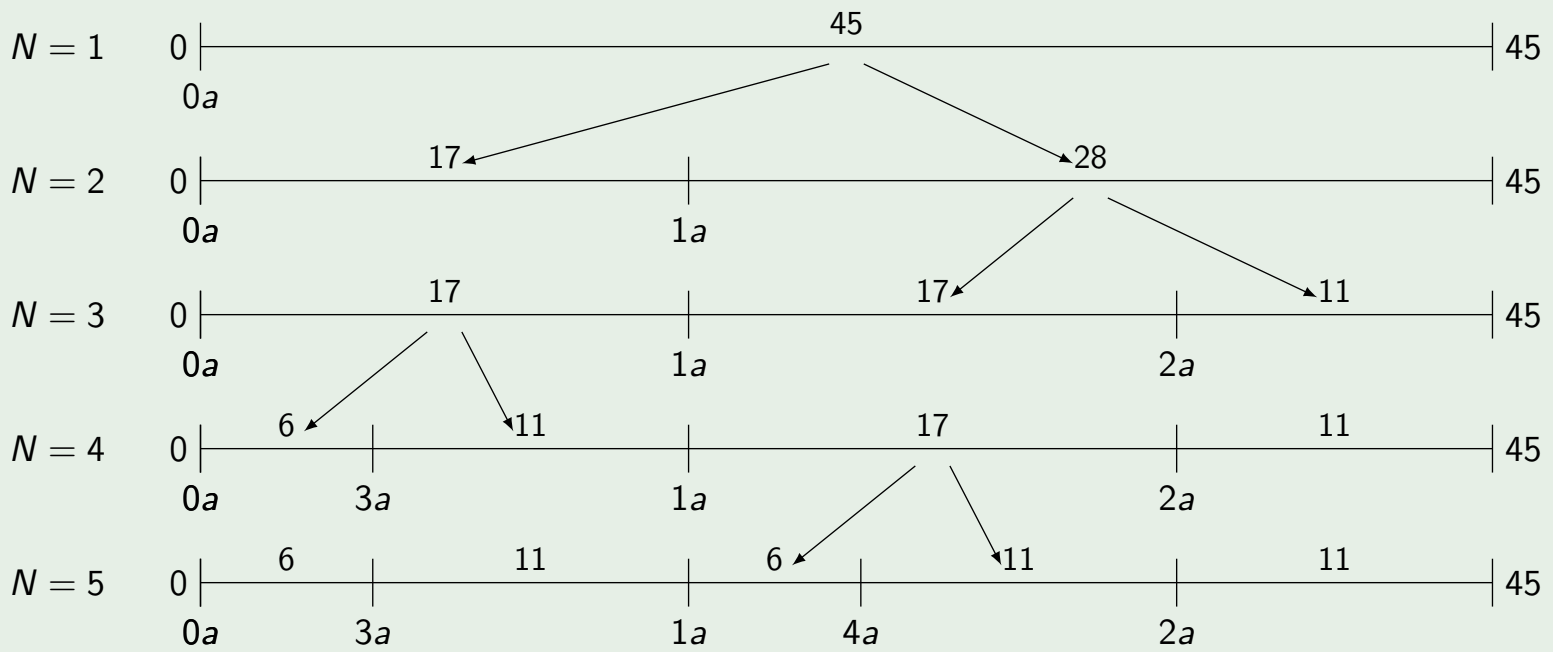


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45

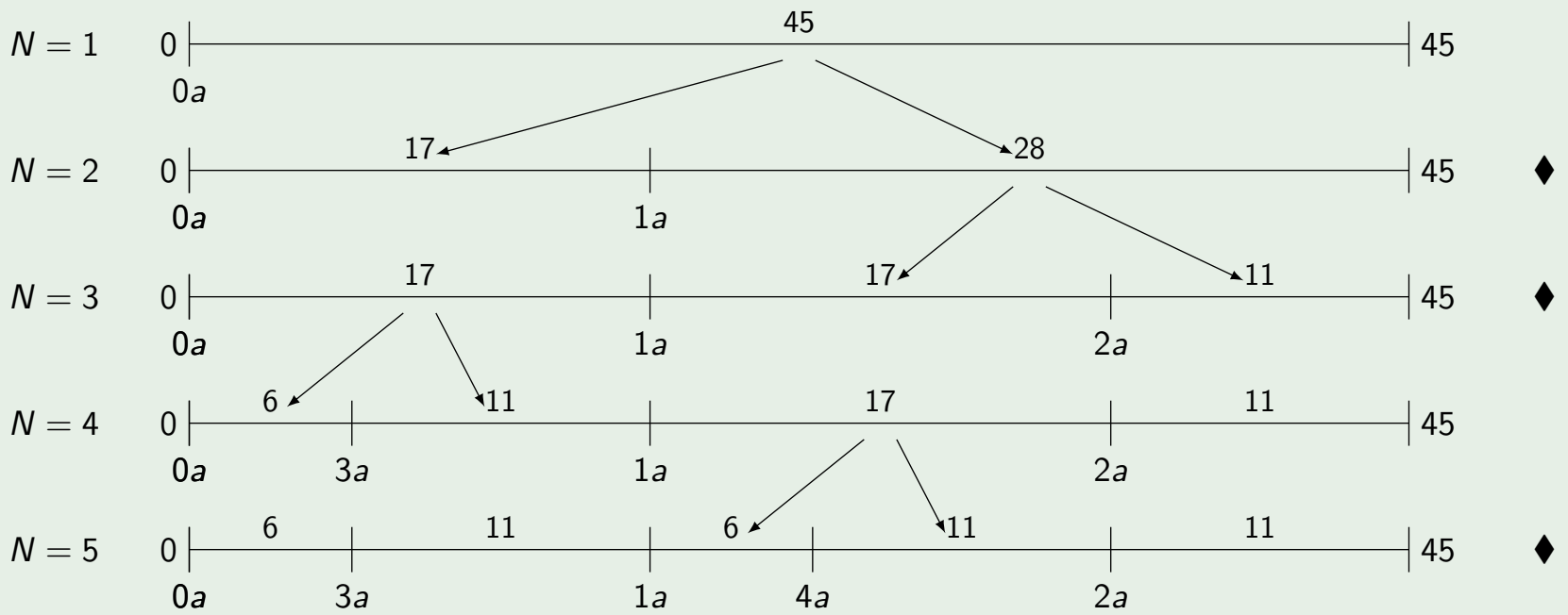


The three distance theorem

Three distance theorem (Slater)

The points $\{a \cdot x \mod 1 \mid x < N\}$ split the segment $[0, 1[$ into N segments. Their lengths take at most three different values, one being the sum of the two others.

Example: $a = 17/45$, lengths are scaled by 45



♦ : 2-length configurations

Euclidean algorithm and continued fractions

Euclidean Algorithm

Let $a \in \mathbb{R}$. It computes the sequence of remainders

$$\theta_{-1} = 1 \quad \theta_0 = a \quad \theta_{i+1} = \theta_{i-1} - k_{i+1}\theta_i$$

with $k_{i+1} = \lfloor \theta_{i-1}/\theta_i \rfloor$.

How to compute the remainders?

- By division, $\theta_{i+1} = \theta_{i-1} - k_{i+1} \cdot \theta_i$.
- By subtraction, $\theta_{i-1,t} = \theta_{i-1} - t \cdot \theta_i$ with $0 \leq t < k_{i+1}$.

Extended version

It also computes the sequences

$$\begin{aligned} p_{-1} = 1 \quad p_0 = 0 \quad p_i &= p_{i-2} + k_i p_{i-1}, \\ q_{-1} = 0 \quad q_0 = 1 \quad q_i &= q_{i-2} + k_i q_{i-1}, \end{aligned}$$

such that $\theta_i = (-1)^i(q_i a - p_i)$.

Three distance theorem and continued fraction expansion

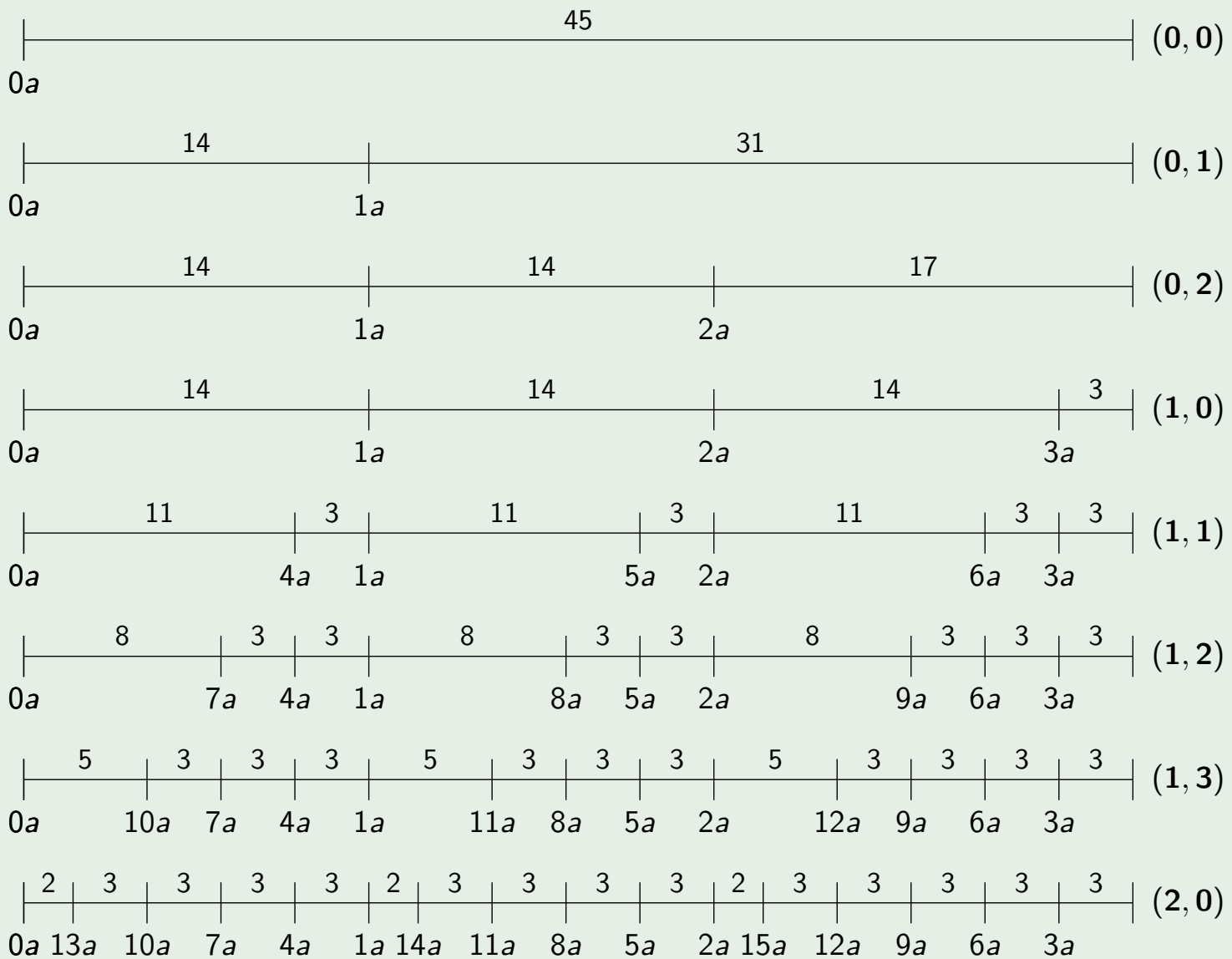
Given a and the associated sequences $(k_i)_{i\in\mathbb{N}}$, $(p_i)_{i\in\mathbb{N}}$ and $(q_i)_{i\in\mathbb{N}}$.

- The lengths are the $\theta_{i-1,t} = \theta_{i-1} - t \cdot \theta_i$, with $0 \leq t < k_{i+1}$.
  Their number are the $q_{i-1,t} = q_{i-1} + t \cdot q_i$, with $0 \leq t < k_{i+1}$.
- There are $O(\log N)$ two-length configurations and they verify

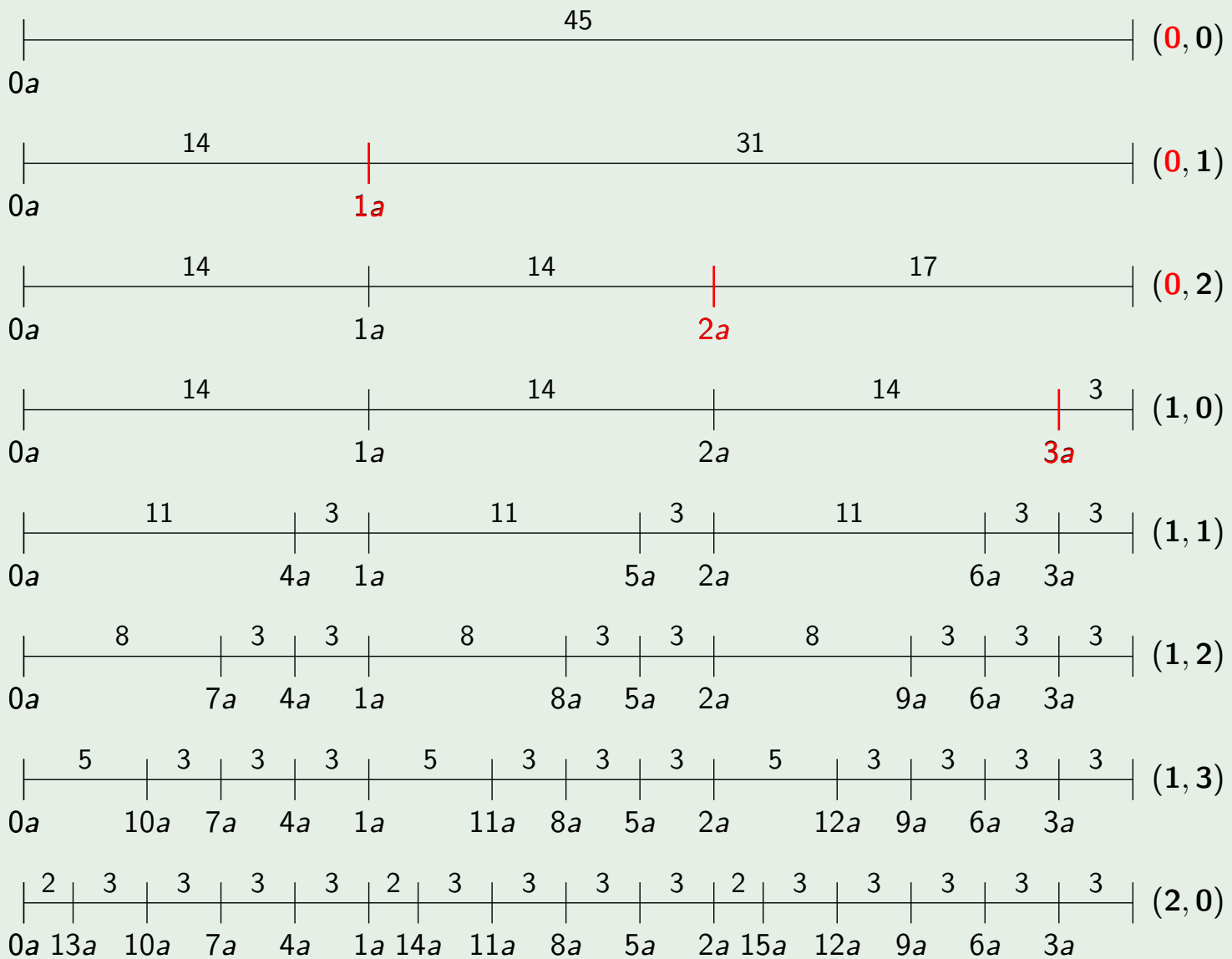
$$q_i\theta_{i-1,t} + q_{i-1,t}\theta_i = 1.$$

- Interpretation: if we place $N = q_i + q_{i-1,t}$ multiples of a,
  - there are q_i intervals of length $\theta_{i-1,t}$;
  - there are $q_{i-1,t}$ intervals of length θ_i.

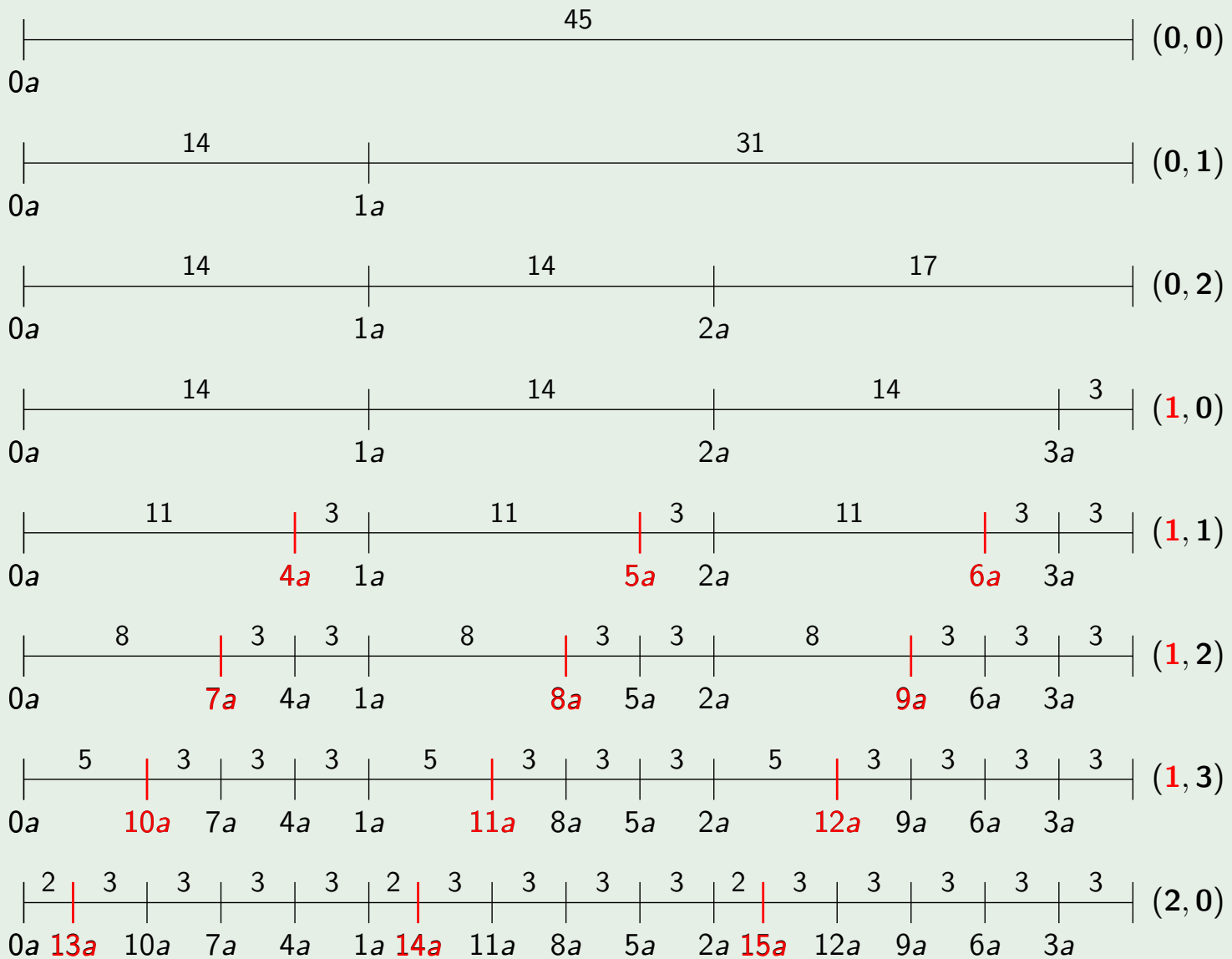
Example: $a = 14/45$, lengths are scaled by 45



Example: $a = 14/45$, lengths are scaled by 45



Example: $a = 14/45$, lengths are scaled by 45

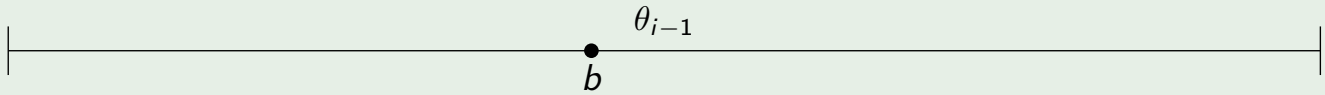


Lefèvre HR-case existence test

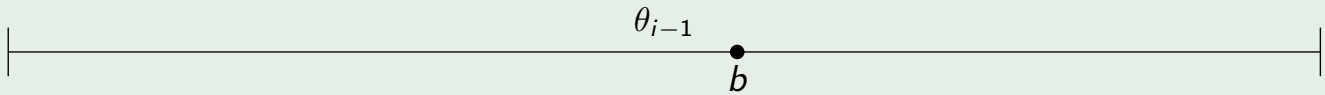
Idea

Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



If i is odd

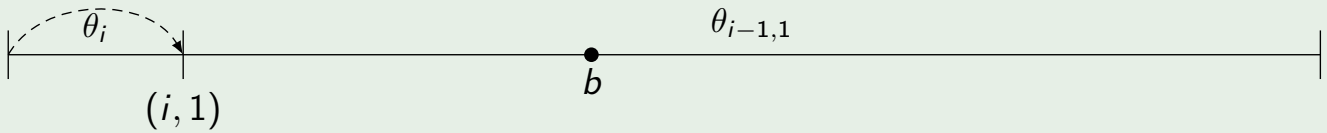


Lefèvre HR-case existence test

Idea

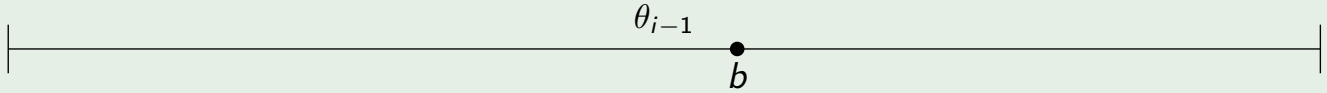
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b - \tilde{b}_{i,t+1}) = (b - \tilde{b}_{i,t}) - \theta_i$$

If i is odd

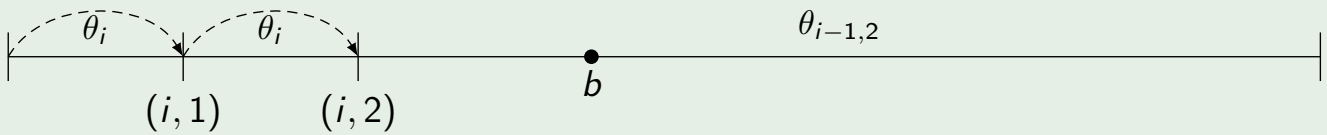


Lefèvre HR-case existence test

Idea

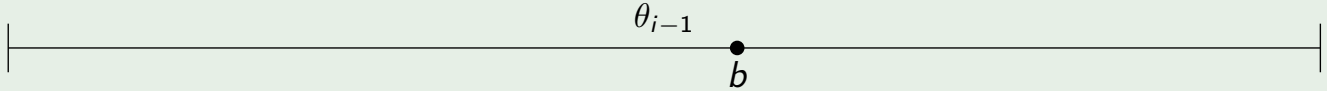
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b - \tilde{b}_{i,t+1}) = (b - \tilde{b}_{i,t}) - \theta_i$$

If i is odd

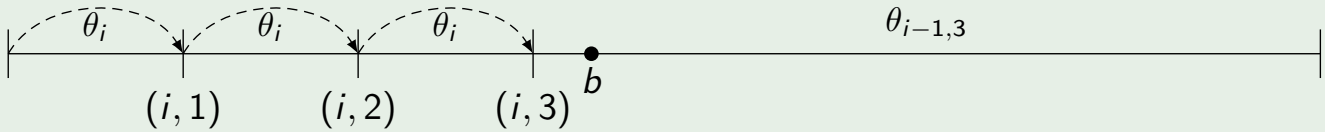


Lefèvre HR-case existence test

Idea

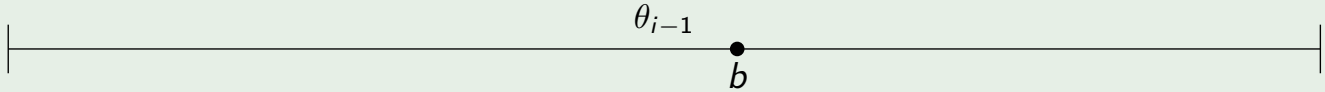
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b - \tilde{b}_{i,t+1}) = (b - \tilde{b}_{i,t}) - \theta_i$$

If i is odd

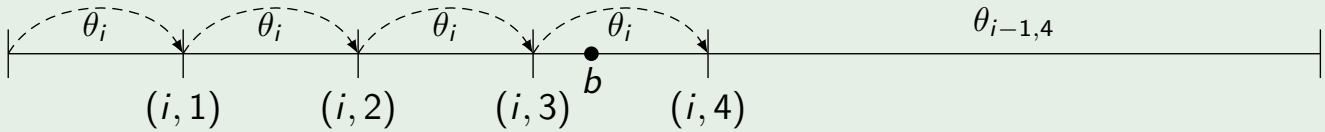


Lefèvre HR-case existence test

Idea

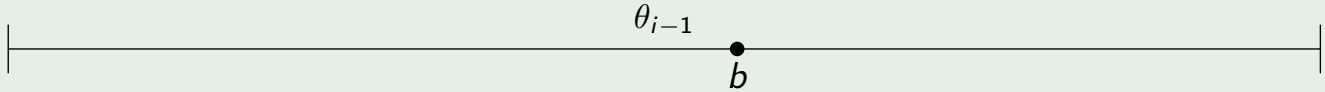
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b - \tilde{b}_{i,t+1}) = (b - \tilde{b}_{i,t}) - \theta_i$$

If i is odd

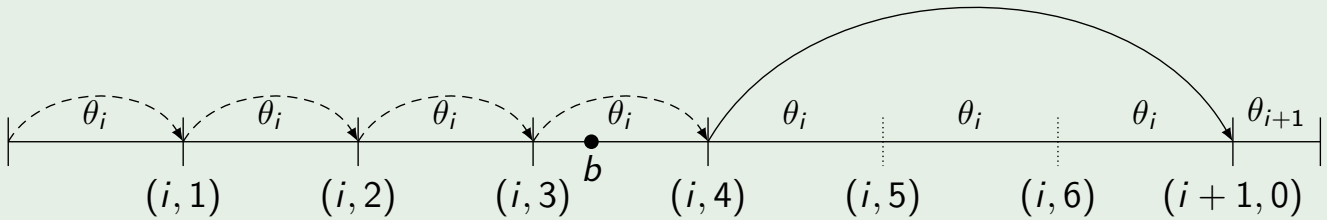


Lefèvre HR-case existence test

Idea

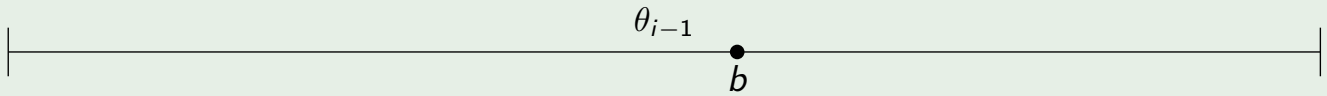
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t}) - \theta_i \quad \text{or} \quad (b-\tilde{b}_{i+1,0}) = (b-\tilde{b}_{i,t})$$

If i is odd

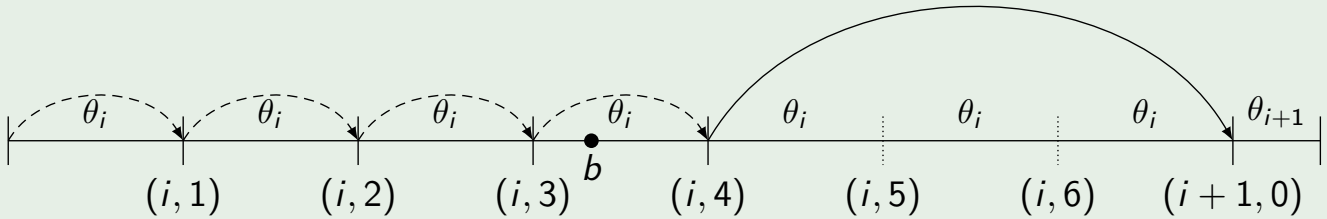


Lefèvre HR-case existence test

Idea

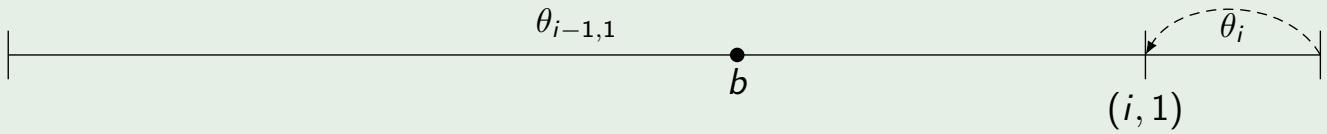
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t}) - \theta_i \quad \text{or} \quad (b-\tilde{b}_{i+1,0}) = (b-\tilde{b}_{i,t})$$

If i is odd



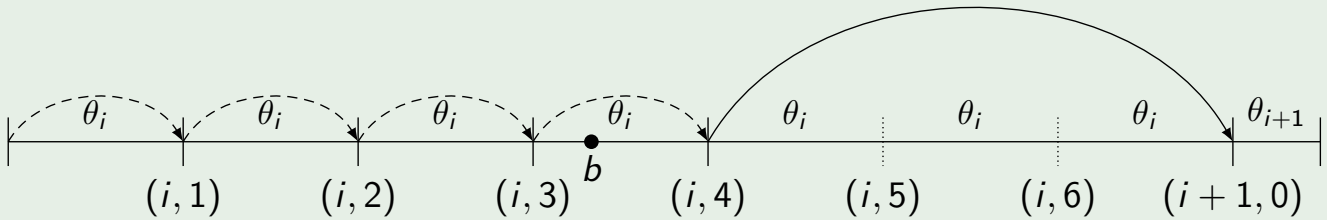
$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t})$$

Lefèvre HR-case existence test

Idea

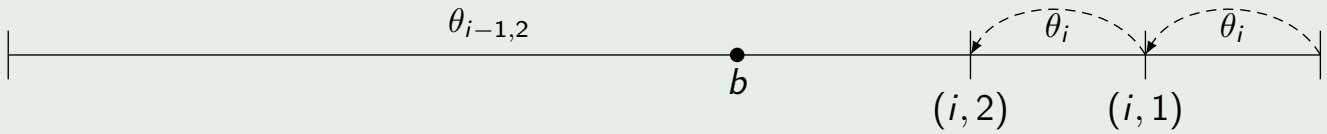
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t}) - \theta_i \quad \text{or} \quad (b-\tilde{b}_{i+1,0}) = (b-\tilde{b}_{i,t})$$

If i is odd



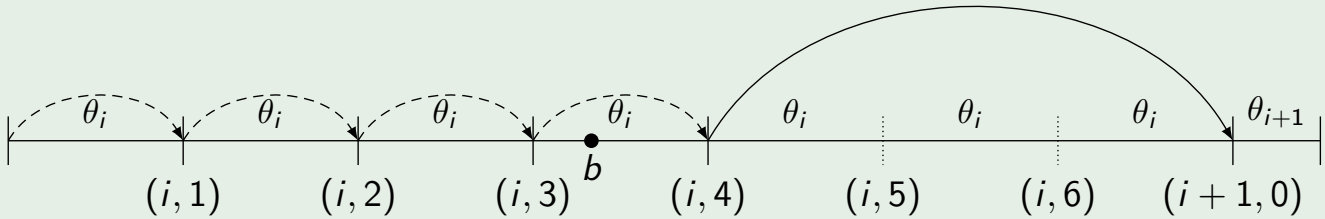
$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t})$$

Lefèvre HR-case existence test

Idea

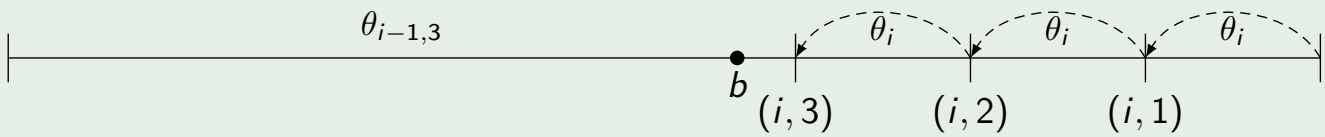
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t}) - \theta_i \quad \text{or} \quad (b-\tilde{b}_{i+1,0}) = (b-\tilde{b}_{i,t})$$

If i is odd



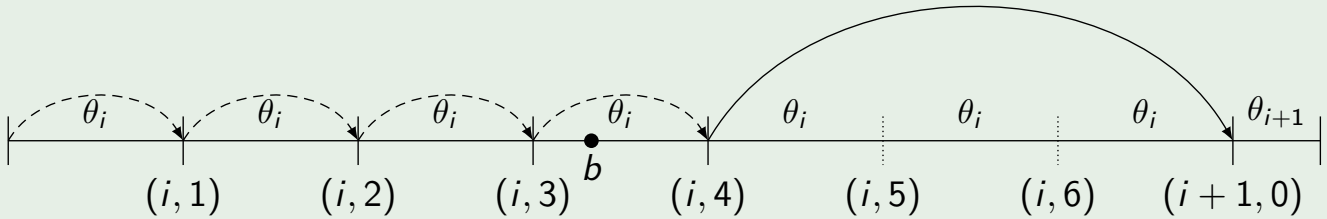
$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t})$$

Lefèvre HR-case existence test

Idea

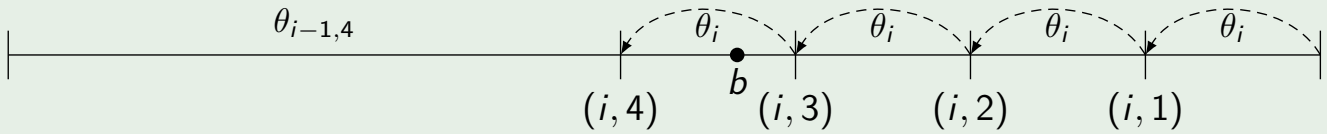
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t}) - \theta_i \quad \text{or} \quad (b-\tilde{b}_{i+1,0}) = (b-\tilde{b}_{i,t})$$

If i is odd



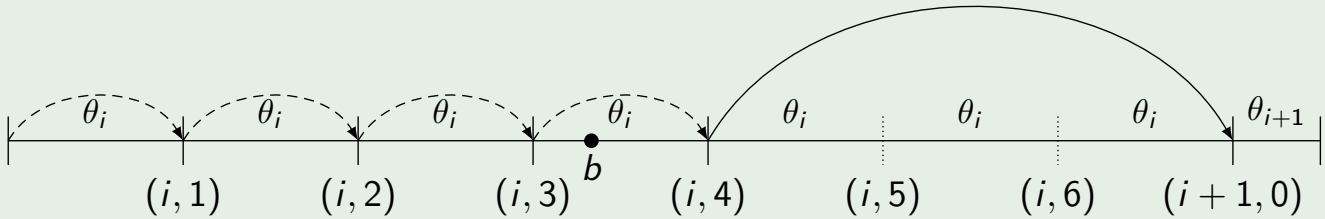
$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t})$$

Lefèvre HR-case existence test

Idea

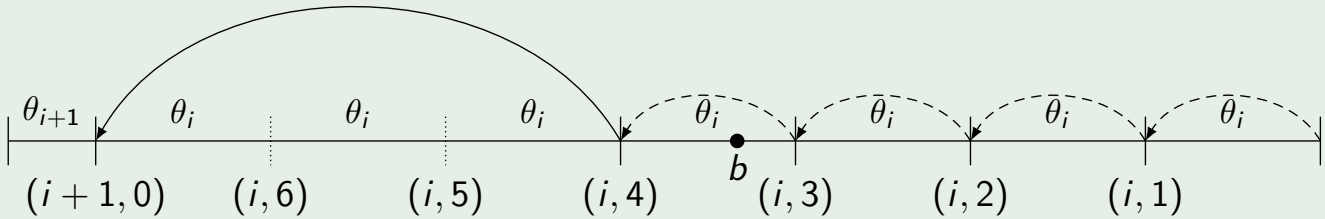
Write b in the basis $(\theta_{i,t})_{i\in\mathbb{N}}$ to get best approximations.

If i is even



$$(b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t}) - \theta_i \quad \text{or} \quad (b-\tilde{b}_{i+1,0}) = (b-\tilde{b}_{i,t})$$

If i is odd



$$(b-\tilde{b}_{i+1,0}) = (b-\tilde{b}_{i,t}) - \theta_{i-1,t+1} \quad \text{or} \quad (b-\tilde{b}_{i,t+1}) = (b-\tilde{b}_{i,t})$$

An irregular control flow: unsuitable for SIMD

Algorithm 1: Lefèvre HR-case existence test.

input : $b - a \cdot x$, ϵ, N

initialisation: $p \leftarrow \{a\}$; $q \leftarrow 1 - \{a\}$; $d \leftarrow \{b\}$;
$u \leftarrow 1$; $v \leftarrow 1$;

if $d < \epsilon$ **then return** Failure;

while *True* **do**

  if $d < p$ **then**

    $k = \lfloor q/p \rfloor$;

    $q \leftarrow q - k * p$; $u \leftarrow u + k * v$;

    if $u + v \geq N$ **then return** Success;

    $p \leftarrow p - q$; $v \leftarrow v + u$;

  else

    $d \leftarrow d - p$;

    if $d < \epsilon$ **then return** Failure;

    $k = \lfloor p/q \rfloor$;

    $p \leftarrow p - k * q$; $v \leftarrow v + k * u$;

    if $u + v \geq N$ **then return** Success;

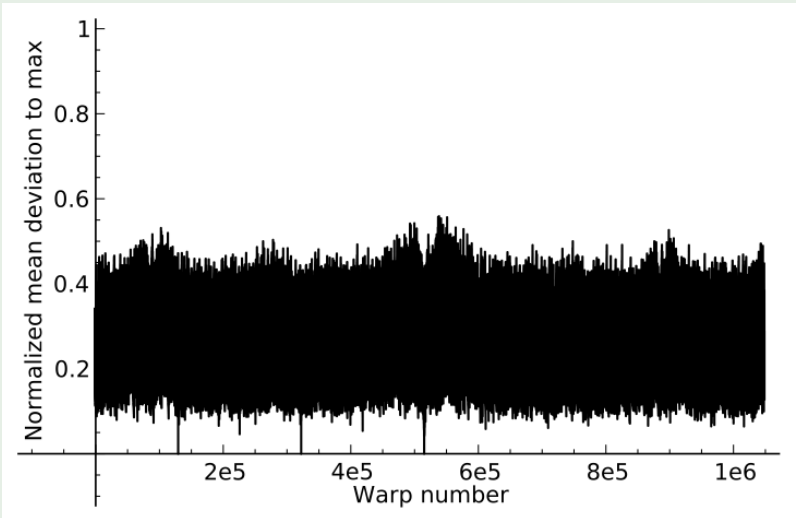
    $q \leftarrow q - p$; $u \leftarrow u + v$;

An irregular control flow: loop divergence

Normalized mean deviation to the maximum (NMDM)

$$1 - \frac{Mean(\{n_i, 0 \leq i < w\})}{Max(\{n_i, 0 \leq i < w\})}$$

Lefèvre existence test (e^x, $[1, 1 + 2^{-13}]$, $\epsilon = 2^{-32}$)



No implementation technique works!

- Re-organize data
  $\Rightarrow$ no *a priori* information
- Compute several sub-domains per thread without exiting the loop
  $\Rightarrow$ too few instructions to issue in the loop to offset the extra cost.

An irregular control flow

Why is Lefèvre HR-case existence test irregular?

Goes from subtraction-based to division-based Euclidean algorithm depending on the position of b.

$\Rightarrow$ The number of loop iterations is hence conditioned by:

- the position of b on the unit segment,
- the number of quotients to compute and
- the value of the quotients.

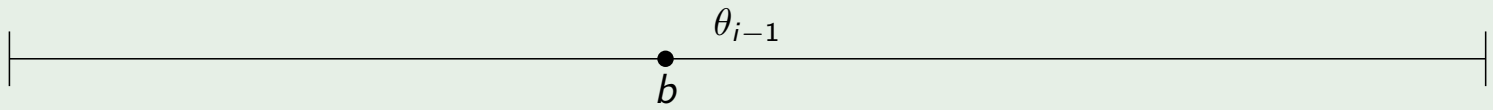
Goal

Break this dependency by considering only $(i, 0)$ configurations.

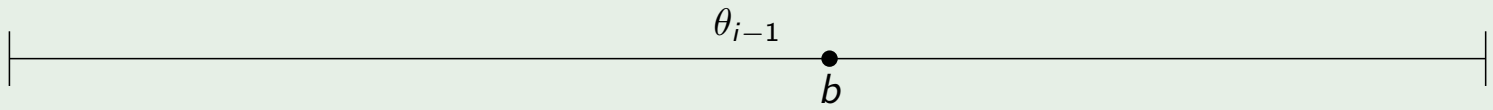
$\Rightarrow$ Write b in the basis $(\theta_i)_{i \in \mathbb{N}}$ to obtain the same sequence of best approximations.

New reduction rules

If i is even

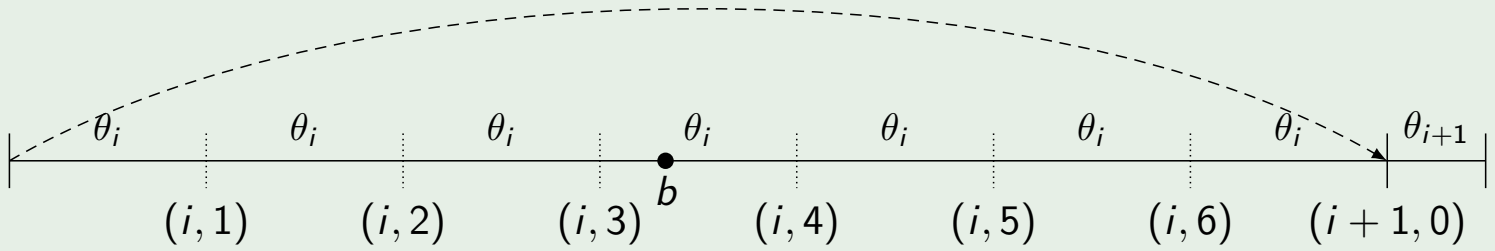


If i is odd



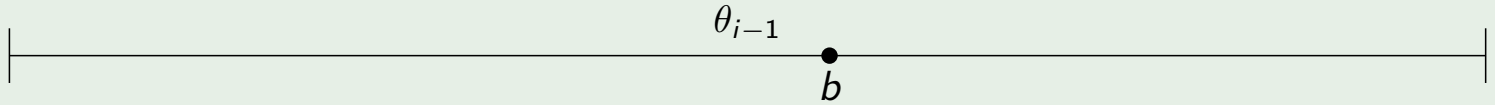
New reduction rules

If i is even



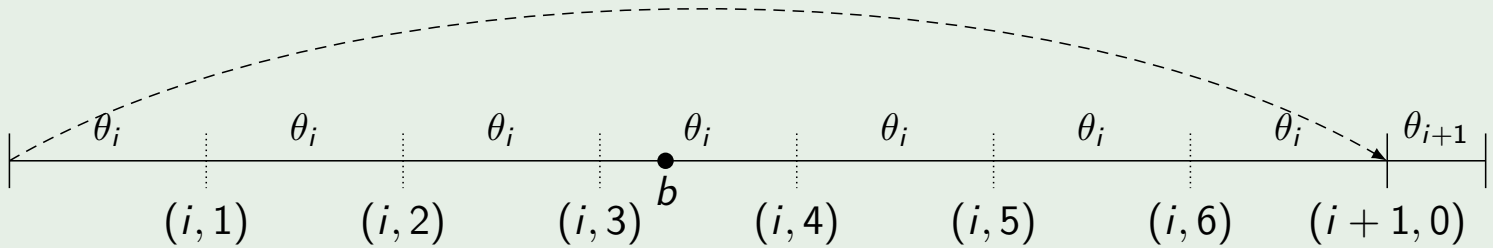
$$(b - \tilde{b}_{i+1}) = (b - \tilde{b}_i) \mod \theta_i$$

If i is odd



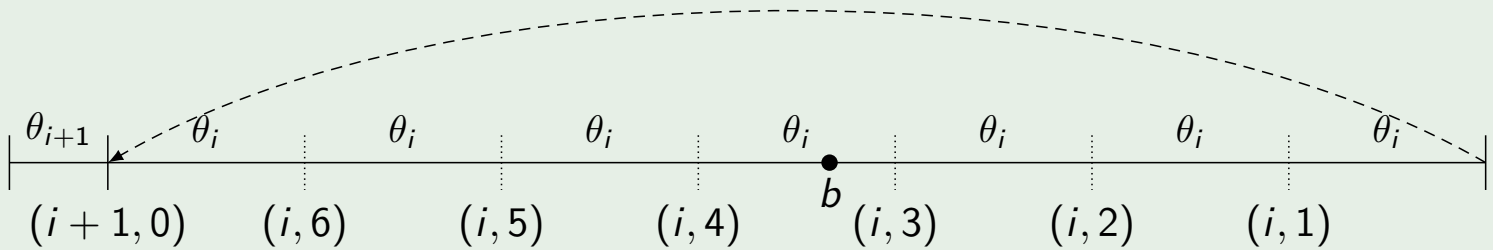
New reduction rules

If i is even



$$(b - \tilde{b}_{i+1}) = (b - \tilde{b}_i) \mod \theta_i$$

If i is odd



$$(b - \tilde{b}_{i+1}) = (b - \tilde{b}_i) - \theta_{i+1} \mod \theta_i$$

Algorithm 3: Regular HR-case existence test.

input : $b - a \cdot x$, ϵ, N

initialisation: $p \leftarrow \{a\}$; $q \leftarrow 1$; $d \leftarrow \{b\}$; $u \leftarrow 1$; $v \leftarrow 0$;

if $d < \epsilon$ **then return** Failure;

while *True* **do**

  if $p < q$ **then**

    $k = \lfloor q/p \rfloor$;

    $q = q - k * p$; $u = u + k * v$;

    $d = d \mod p$;

  else

    $k = \lfloor p/q \rfloor$;

    $p = p - k * q$; $v = v + k * u$;

    if $d \geq p$ **then**

      $d = (d - p) \mod q$;

  if $u + v \geq N$ **then return** $d > \epsilon$;

Divergence in the new algorithm

Algorithm 3: Regular HR-case existence test.

input : $b - a \cdot x$, ϵ, N

initialisation: $p \leftarrow \{a\}$; $q \leftarrow 1$; $d \leftarrow \{b\}$; $u \leftarrow 1$; $v \leftarrow 0$;

if $d < \epsilon$ **then return** Failure;

while *True* **do**

  if $p < q$ **then**

    $k = \lfloor q/p \rfloor$;

    $q = q - k * p$; $u = u + k * v$;

    $d = d \mod p$;

  else

    $k = \lfloor p/q \rfloor$;

    $p = p - k * q$; $v = v + k * u$;

    if $d \geq p$ **then**

      $d = (d - p) \mod q$;

  if $u + v \geq N$ **then return** $d > \epsilon$;

GPU branch predication

Divergence on the main conditional branch

A deterministic test

i is alternatively odd and even.

$\Rightarrow$ We can avoid divergence by unrolling 2 loop iterations.

Algorithm 4: Regular HR-case existence test unrolled.

input : $b - ax$, ϵ, N

initialisation: $p \leftarrow \{a\}$; $q \leftarrow 1$; $d \leftarrow \{b\}$; $u \leftarrow 1$; $v \leftarrow 0$;

while *True* **do**

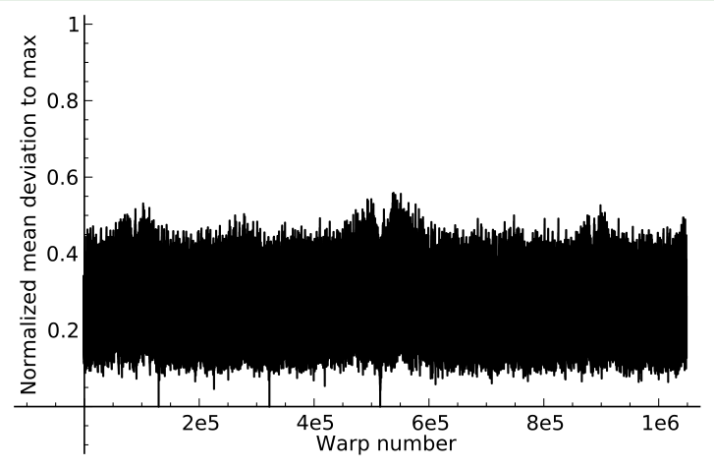
- $k = \lfloor q/p \rfloor$;
- $q = q - k * p$; $u = u + k * v$;
- $d = d \mod p$;
- **if** $u + v \geq N$ **then return** $d > \epsilon$;
- $k = \lfloor p/q \rfloor$;
- $p = p - k * q$; $v = v + k * u$;
- **if** $d \geq p$ **then**
  - $d = d - p \mod q$;
- **if** $u + v \geq N$ **then return** $d > \epsilon$;

Divergence on the main loop (e^x, subdomain $[1, 1 + 2^{-13}[$)

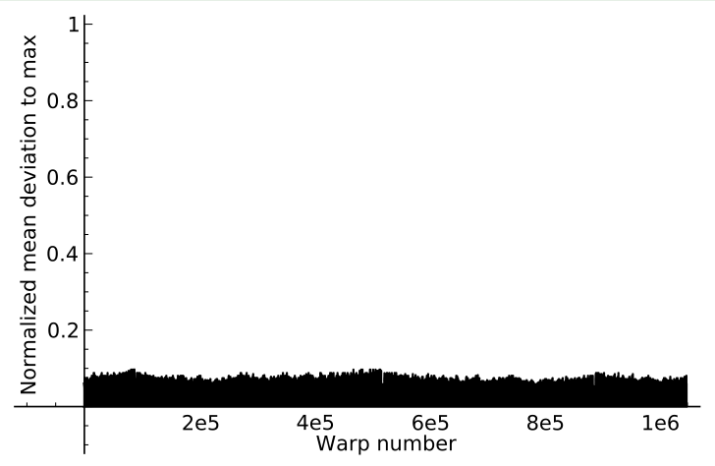
Normalized mean deviation to the maximum (NMDM)

$$1 - \frac{Mean(\{n_i, 0 \leq i < w\})}{Max(\{n_i, 0 \leq i < w\})}$$

Lefèvre Algorithm



New Algorithm



A regular control flow

Why is the new HR-case existence test regular?

It uses only division-based Euclidean algorithm.
$\Rightarrow$ The number of loop iterations only depends on the number of quotients to compute, which is very stable from one interval to the next.

A regular control flow

Why is the new HR-case existence test regular?

It uses only division-based Euclidean algorithm.
$\Rightarrow$ The number of loop iterations only depends on the number of quotients to compute, which is very stable from one interval to the next.

	Seq.	MPI	GPU	$\frac{\text{Seq.}}{\text{MPI}}$	$\frac{\text{MPI}}{\text{GPU}}$
Lefèvre	36816.10	5292.67	2446.27	6.96	2.16
New	34039.94	4716.97	711.92	7.22	6.63
Lef. / New	1.08	1.12	3.44	–	–

Table : Performance result on e^x in $[1, 2[$ for binary64.
Platform: Intel Xeon X5650 hexa-core and Nvidia C2070.

A regular control flow

Why is the new HR-case existence test regular?

It uses only division-based Euclidean algorithm.
$\Rightarrow$ The number of loop iterations only depends on the number of quotients to compute, which is very stable from one interval to the next.

	Seq.	MPI	GPU	$\frac{\text{Seq.}}{\text{MPI}}$	$\frac{\text{MPI}}{\text{GPU}}$
Lefèvre	36816.10	5292.67	2446.27	6.96	2.16
New	34039.94	4716.97	711.92	7.22	6.63
Lef. / New	1.08	1.12	3.44	–	–

Table : Performance result on e^x in $[1, 2[$ for binary64.
Platform: Intel Xeon X5650 hexa-core and Nvidia C2070.

A regular control flow

Why is the new HR-case existence test regular?

It uses only division-based Euclidean algorithm.
$\Rightarrow$ The number of loop iterations only depends on the number of quotients to compute, which is very stable from one interval to the next.

	Seq.	MPI	GPU	$\frac{\text{Seq.}}{\text{MPI}}$	$\frac{\text{MPI}}{\text{GPU}}$
Lefèvre	36816.10	5292.67	2446.27	6.96	2.16
New	34039.94	4716.97	711.92	7.22	6.63
Lef. / New	1.08	1.12	3.44	–	–

Table : Performance result on e^x in $[1, 2[$ for binary64.
Platform: Intel Xeon X5650 hexa-core and Nvidia C2070.

A regular control flow

Why is the new HR-case existence test regular?

It uses only division-based Euclidean algorithm.
$\Rightarrow$ The number of loop iterations only depends on the number of quotients to compute, which is very stable from one interval to the next.

	Seq.	MPI	GPU	$\frac{\text{Seq.}}{\text{MPI}}$	$\frac{\text{MPI}}{\text{GPU}}$
Lefèvre	36816.10	5292.67	2446.27	6.96	2.16
New	34039.94	4716.97	711.92	7.22	6.63
Lef. / New	1.08	1.12	3.44	–	–

Table : Performance result on e^x in $[1, 2[$ for binary64.
Platform: Intel Xeon X5650 hexa-core and Nvidia C2070.

A regular control flow

Why is the new HR-case existence test regular?

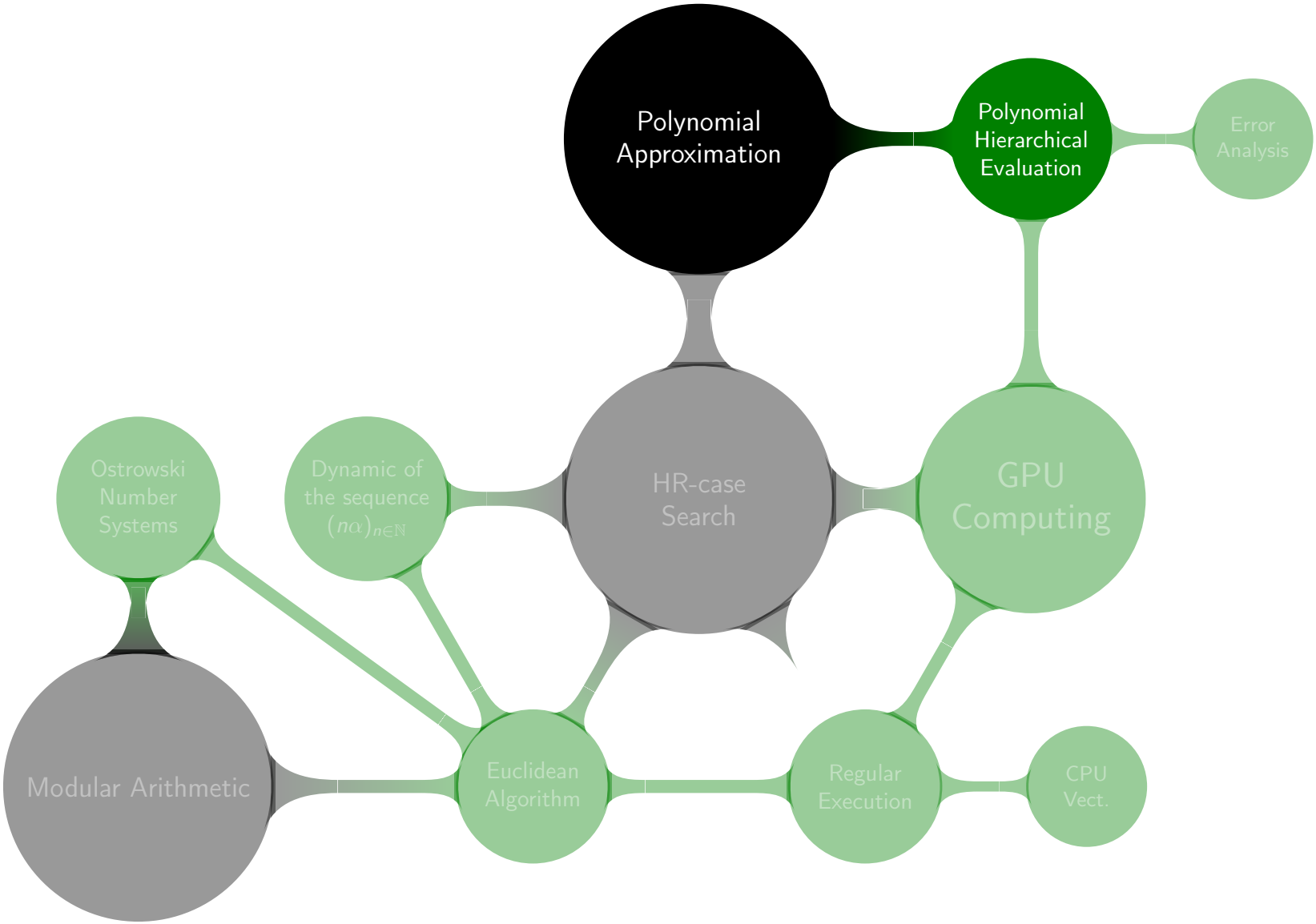
It uses only division-based Euclidean algorithm.
$\Rightarrow$ The number of loop iterations only depends on the number of quotients to compute, which is very stable from one interval to the next.

	Seq.	MPI	GPU	$\frac{\text{Seq.}}{\text{MPI}}$	$\frac{\text{MPI}}{\text{GPU}}$
Lefèvre	36816.10	5292.67	2446.27	6.96	2.16
New	34039.94	4716.97	711.92	7.22	6.63
Lef. / New	1.08	1.12	3.44	–	–

Table : Performance result on e^x in $[1, 2[$ for binary64.
Platform: Intel Xeon X5650 hexa-core and Nvidia C2070.

$\Rightarrow$ Lefèvre Seq. / New GPU: 51.7
Lefèvre MPI / New GPU: 7.4

Overview of tools and contributions



Parallelization of the framework: Pol. approx.

Pros

- Polynomial generation became prohibitive
- Avoids huge data transfers:
  - 2^{38} approximations by binade for double precision
  - Transfer time higher than HR-case seach with new HR-case exitence test !

Cons

- Best algorithm is inherently sequential
- Multiprecision

Polynomial approximation generation

Using Taylor shifts

- Generate an approximation P of degree δ of the function.
- Translate P such that $P_i(x) = P(x + iN)$ (Taylor shift).

Hierarchical Method

- We write $x = kN + m$,
- We interpolate P in the binomial basis w.r.t the variable m

$$P(kN + m) = \sum_{j=0}^{\delta} a_j(k) \binom{j}{m}$$

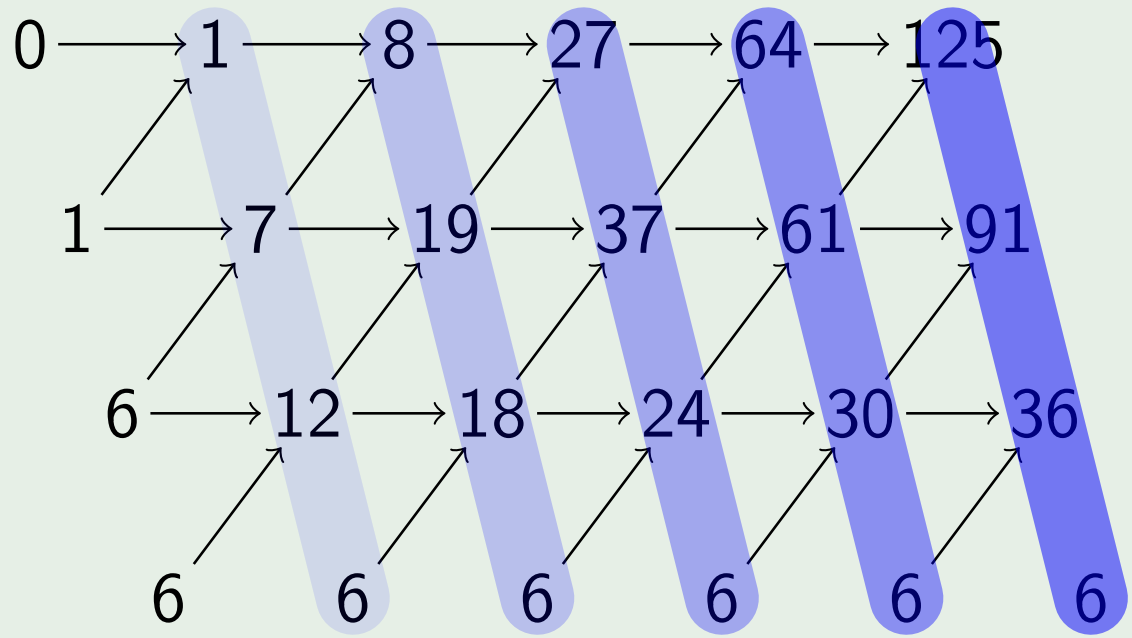
$\Rightarrow$ We can get $P_i(m)$ by evaluating the $a_j(k)$ in i.
$\Rightarrow$ A shift by N becomes $\delta + 1$ independent shifts by 1.

Polynomial approximation generation

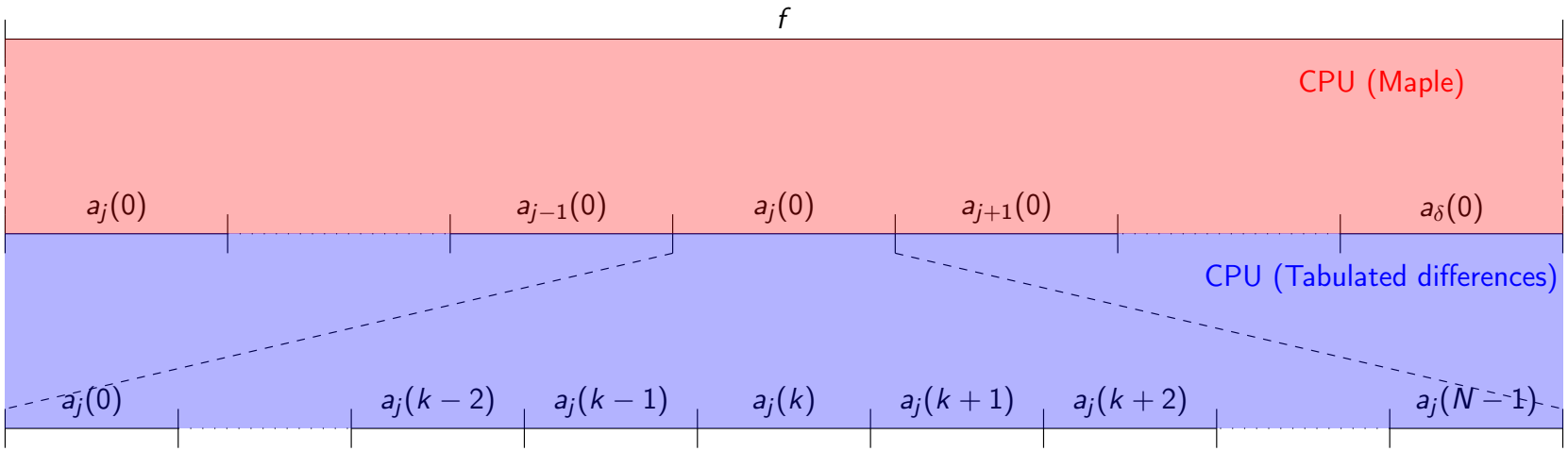
Difference table algorithm

- Needs only $deg(a_j)$ additions to shift by 1.
- Is inherently sequential.

Example with x^3



Lefèvre polynomial hierarchy



Remark

- One level of parallelism among several Taylor approximations $\Rightarrow$ too coarse for GPU
- It is inherently sequential for a given Taylor approximation.

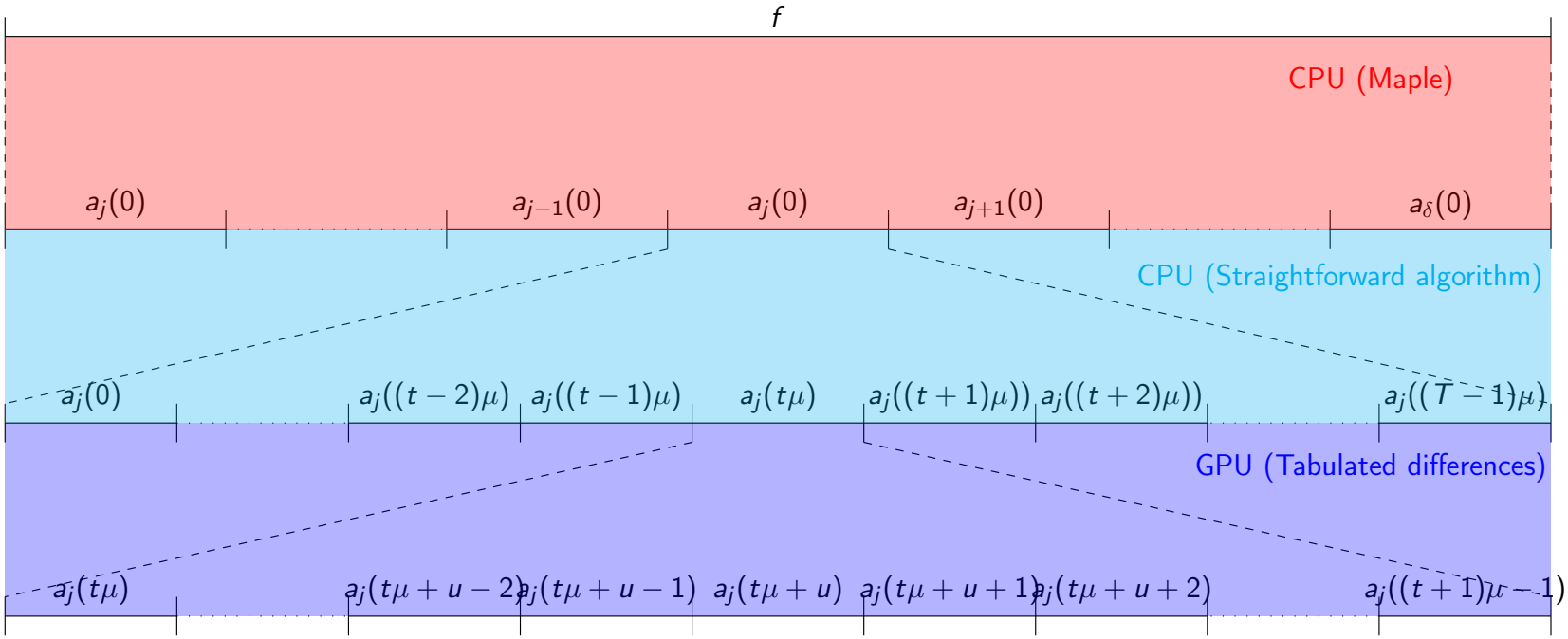
Straightforward algorithm

- Computes $a_j(k)$ by multiplying its coefficient vector by

$$\begin{pmatrix} \binom{k}{0} & \binom{k}{1} & \cdots & \binom{k}{deg(a_j)} \\ 0 & \binom{k}{0} & \cdots & \binom{k}{deg(a_j)-1} \\ 0 & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & \binom{k}{0} \end{pmatrix}$$

- All the evaluations can be done in parallel.
- Is exactly similar as computing k difference table step.
- Needs more operations than difference table algorithm.

Hybrid CPU-GPU Polynomial approximation generation



Remarks

- CPU and GPU computations are overlapped:
  while the $a_j(k)$ are being computed on GPU, the packets of a_{j+1} are simultaneously computed.
- Multiprecision is efficiently implemented in templated functions.

Overall performances

	Seq.	MPI	CPU-GPU	$\frac{\text{Seq.}}{\text{MPI}}$	$\frac{\text{MPI}}{\text{CPU-GPU}}$
Pol. approx.	**43300.81**	**5251.53**	**788.84**	**8.25**	**6.66**
Lefèvre	36816.10	5292.67	2446.27	6.96	2.16
Regular	34039.94	4716.97	711.92	7.22	6.63

Table : Performance result on e^x in $[1, 2[$ for binary64.
Platform: Intel Xeon X5650 hexa-core and Nvidia C2070.

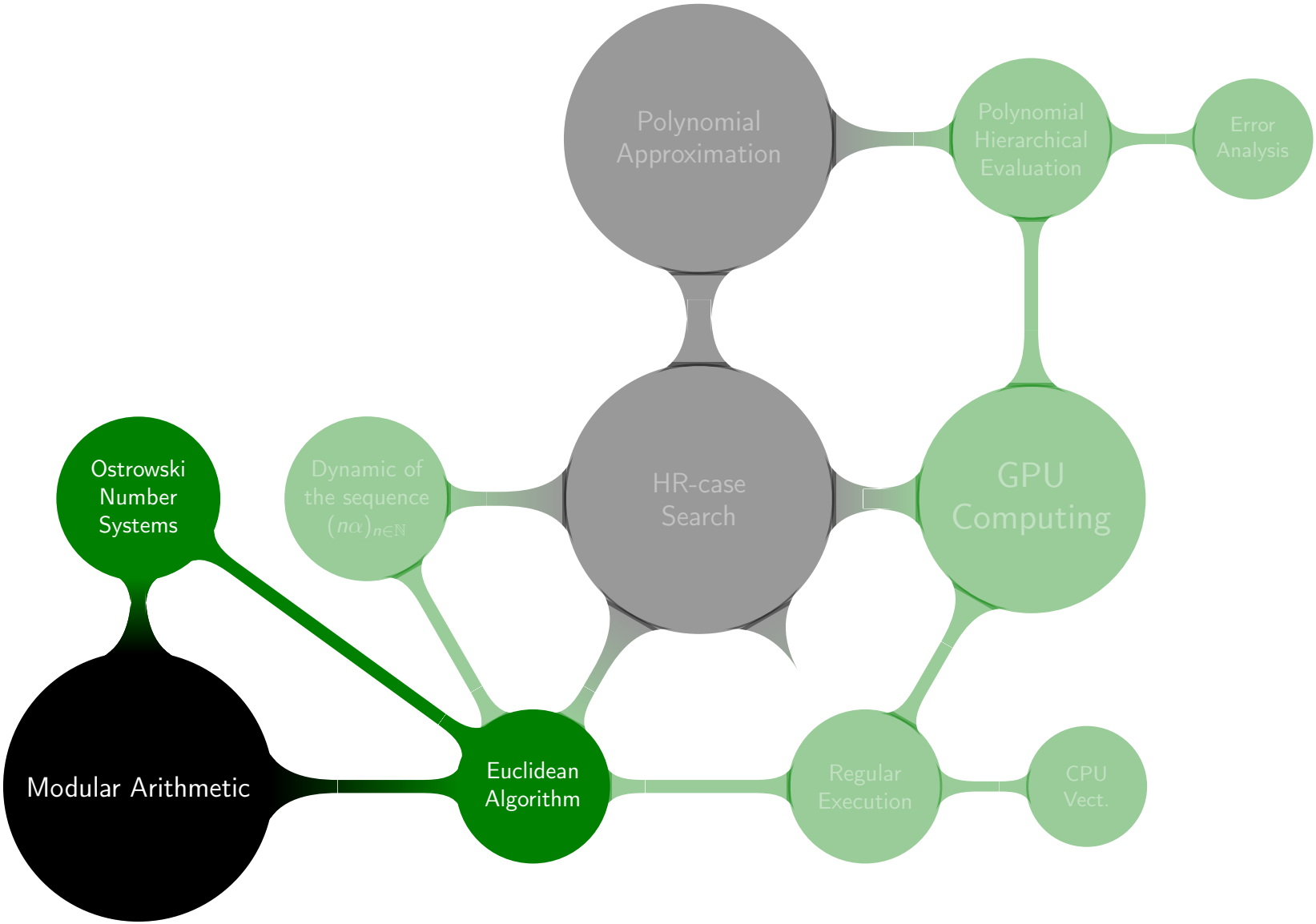
Overall performances

	Seq.	MPI	CPU-GPU	$\frac{\text{Seq.}}{\text{MPI}}$	$\frac{\text{MPI}}{\text{CPU-GPU}}$
Pol. approx.	**43300.81**	**5251.53**	**788.84**	**8.25**	**6.66**
Lefèvre	36816.10	5292.67	2446.27	6.96	2.16
Regular	34039.94	4716.97	711.92	7.22	6.63

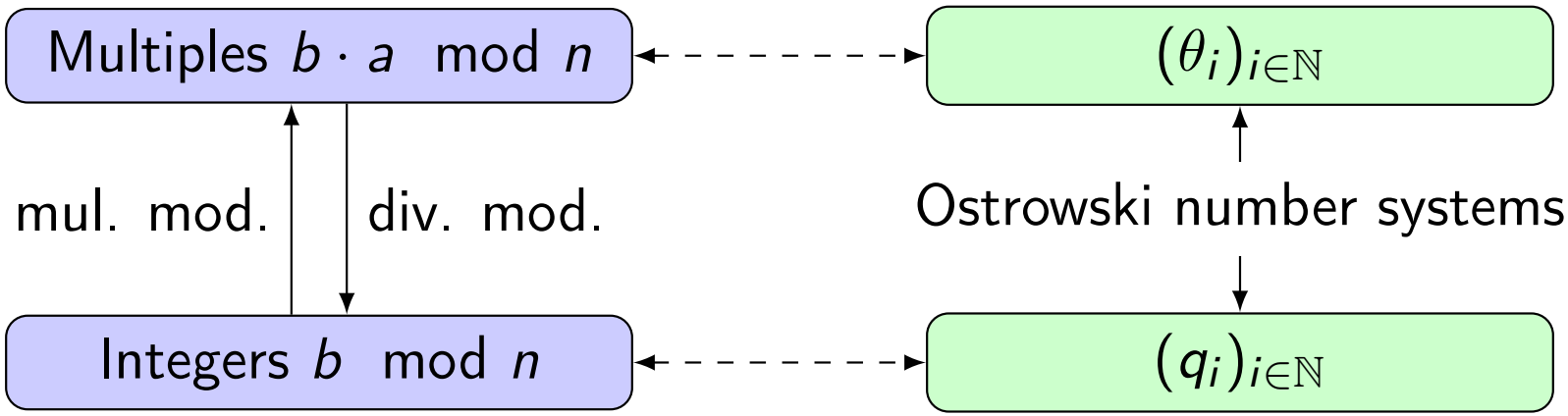
Table : Performance result on e^x in $[1, 2[$ for binary64.
Platform: Intel Xeon X5650 hexa-core and Nvidia C2070.

$$\Rightarrow \begin{array}{l} \text{Seq. / CPU-GPU:} \quad 55.2 \\ \text{MPI / CPU-GPU :} \quad 7.0 \end{array}$$

Overview of tools and contributions



Modular arithmetic



A bijection is given by Ostrowski number systems.

Modular multiplication ($a \cdot b \mod d$)

1. Compute the sequences $(\theta_i)_{i\in\mathbb{N}}$ and $(q_i)_{i\in\mathbb{N}}$ from extgcd(a, d)
2. Compute the sequence $(b_i)_{i\in\mathbb{N}}$ such that $b = 1 + \sum_{i=1}^{m} b_i q_{i-1}$
3. Return $a + \sum_{i=1}^{m} b_i (-1)^i \theta_{i-1}$

Modular division ($a^{-1} \cdot b \mod d$)

1. Compute the sequences $(\theta_i)_{i\in\mathbb{N}}$ and $(q_i)_{i\in\mathbb{N}}$ from extgcd(a, d)
2. Compute the sequence $(b_i)_{i\in\mathbb{N}}$ such that
$$b = a + \sum_{i=1}^{m} b_i (-1)^{i-1} \theta_{i-1}$$
3. Return $1 + \sum_{i=1}^{m} b_i q_{i-1}$

Conclusion

- Improving performances by improving regularity
  $\Rightarrow$ algorithm / architecture match
- HR-case search for binary64 becomes tractable in reasonable time $\Rightarrow$ expected time for exp : 1 week on 1 GPU

Perspectives

- Enter production phase: solve the TMD for all functions recommended by IEEE std 754-2008
- New and simple formalism based on Euclidean algorithm
  - enables formally proved algorithm in COQ
  - enables certified implementations

Perspectives: tackling binary128 using SLZ

- Exploit the structure of the Coppersmith lattices
  - Bloc structure
  - Coppersmith on translated polynomials
    $\Rightarrow$ Prove the heuristic "reduce one lattice suffices"
    $\Rightarrow$ Use polynomials in binomial basis to improve complexity?
  - Lattice reduction with truncated coefficients

Perspectives: modular arithmetic

- Consider sub-sequences for better complexities and computation times (e.g. half-gcd algorithm)
- Provide an efficient implementation
- Applications
  - Compute many mul./div. by the same value a in parallel (e.g. Gauss reduction)
  - The division is very efficient and is done in place (e.g. Exponentiation in Booth coding)
  - CRT with well chosen modulus
    $\Rightarrow$ what is "well-chosen" for these algorithms ?