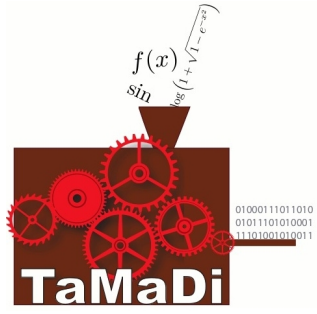
A new LSG test minimizing divergence on SIMD architectures

Pierre Fortin, Mourad Gouicem, Stef Graillat

PEQUAN Team, LIP6/UPMC

ANR TaMaDi Project Meeting
Lyon, December 13-14th 2011



Problem

Given $(a, b) \in \mathbb{R}^2$, $(d, \varepsilon) \in \mathbb{Q}^2$ and $N \in \mathbb{N}$.
Find $x \in \mathbb{N}$, if it exists, such that :

$$\begin{cases} x < N \\ |a \cdot x + b \ cmod \ d| < \varepsilon \end{cases}$$

Simplify the test

$$a \cdot x + (b + \varepsilon) \mod d < 2\varepsilon.$$

2 points of view

- add $k \cdot a$ to b modulo d, and search for closest point to 0
- add $k \cdot a$ to 0 modulo d, and search for closest point to b

Problem

Given $(a, b) \in \mathbb{R}^2$, $(d, \varepsilon) \in \mathbb{Q}^2$ and $N \in \mathbb{N}$.
Find $x \in \mathbb{N}$, if it exists, such that :

$$\begin{cases} x < N \\ |a \cdot x + b \ cmod \ d| < \varepsilon \end{cases}$$

Simplify the test

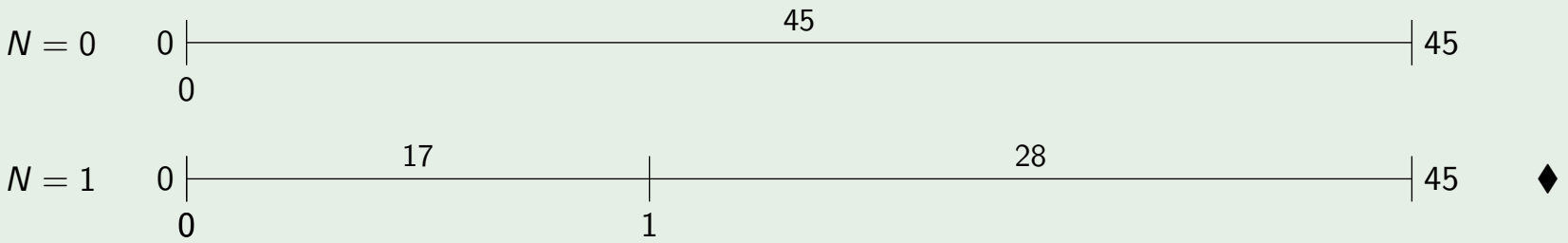
$$a \cdot x + (b + \varepsilon) \mod d < 2\varepsilon.$$

2 points of view

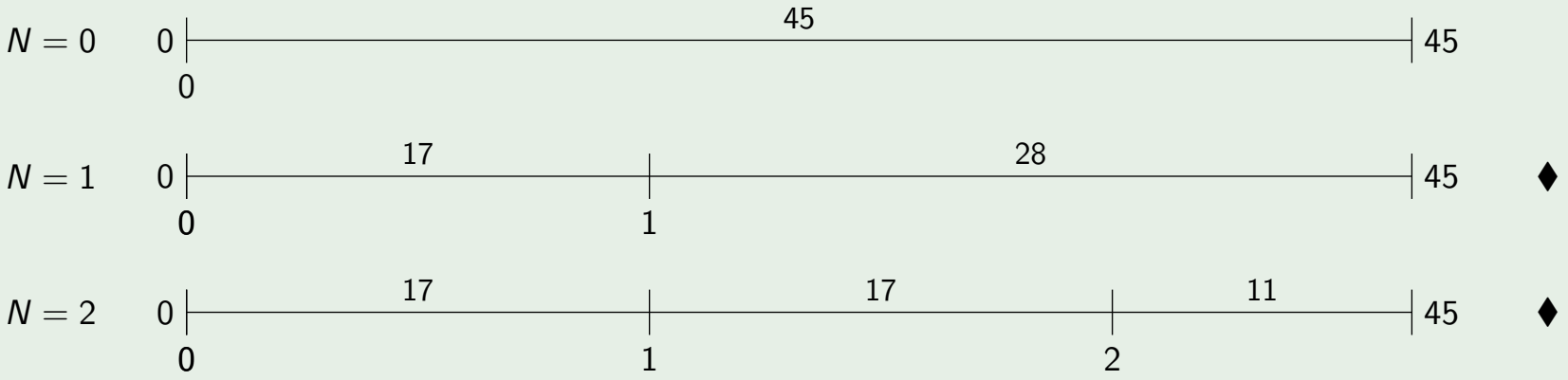
- add $k \cdot a$ to b modulo d, and search for closest point to 0
- add $k \cdot a$ to 0 modulo d, and search for closest point to b

Example : $a = 17/45$, $d = 1$

Example : $a = 17/45$, $d = 1$

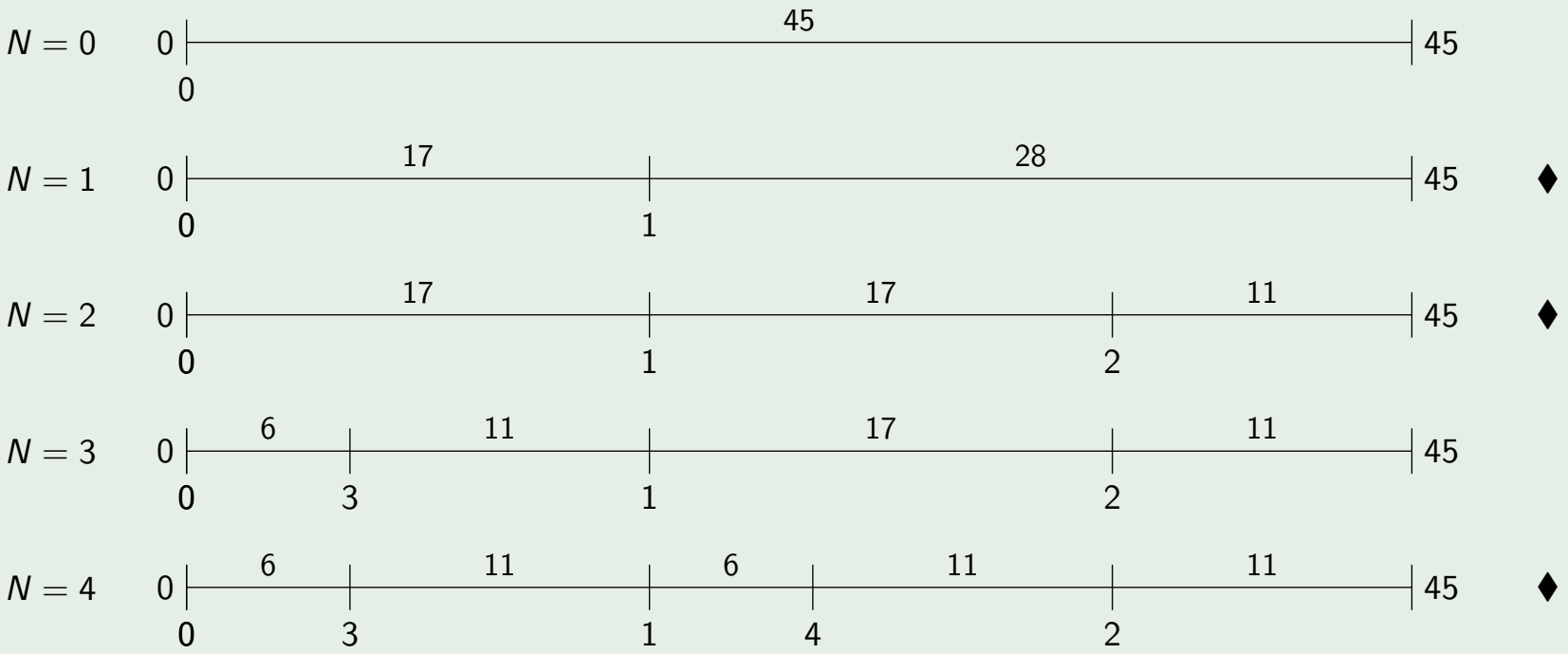


Example : $a = 17/45$, $d = 1$

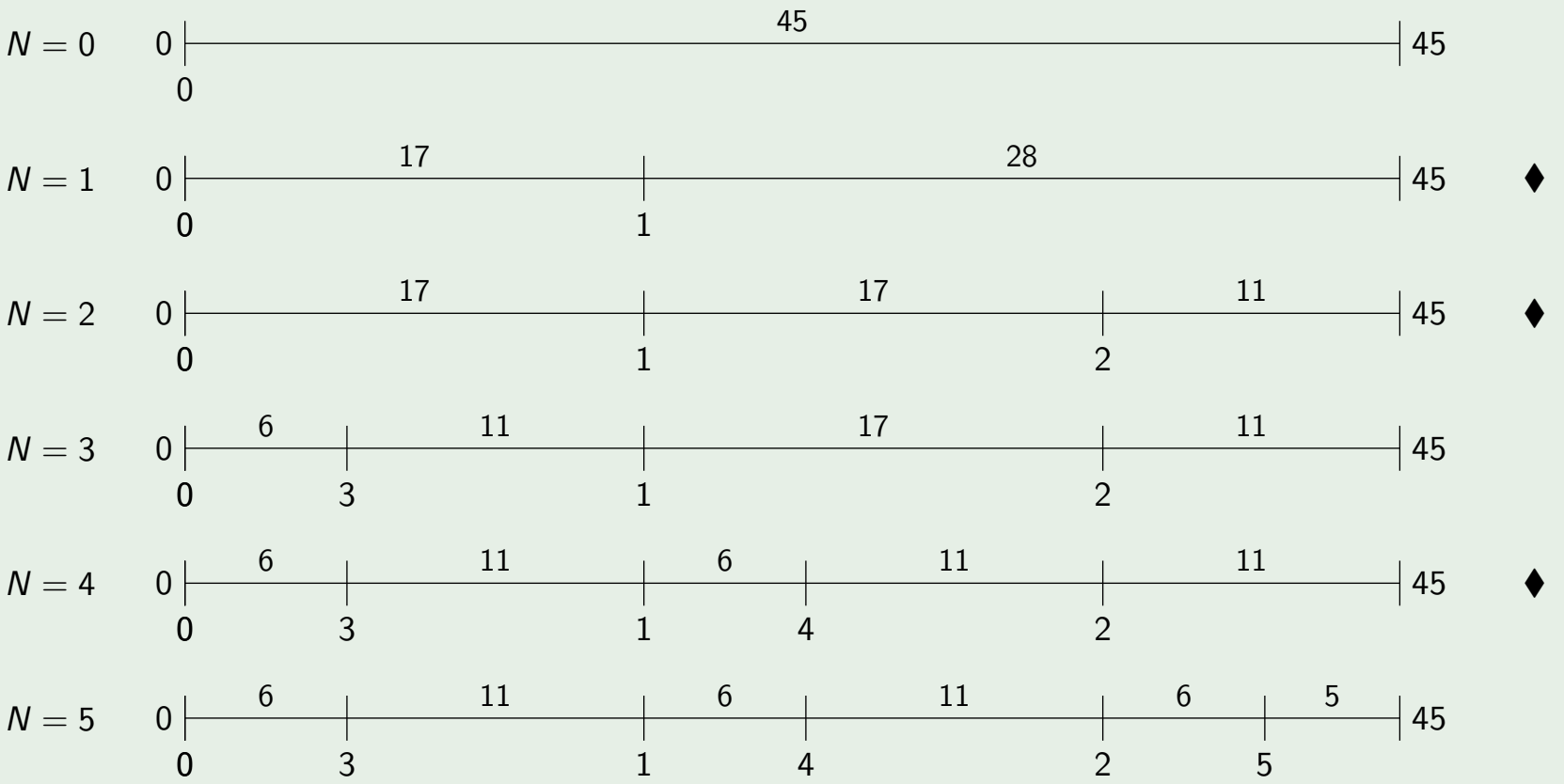


Example : $a = 17/45$, $d = 1$

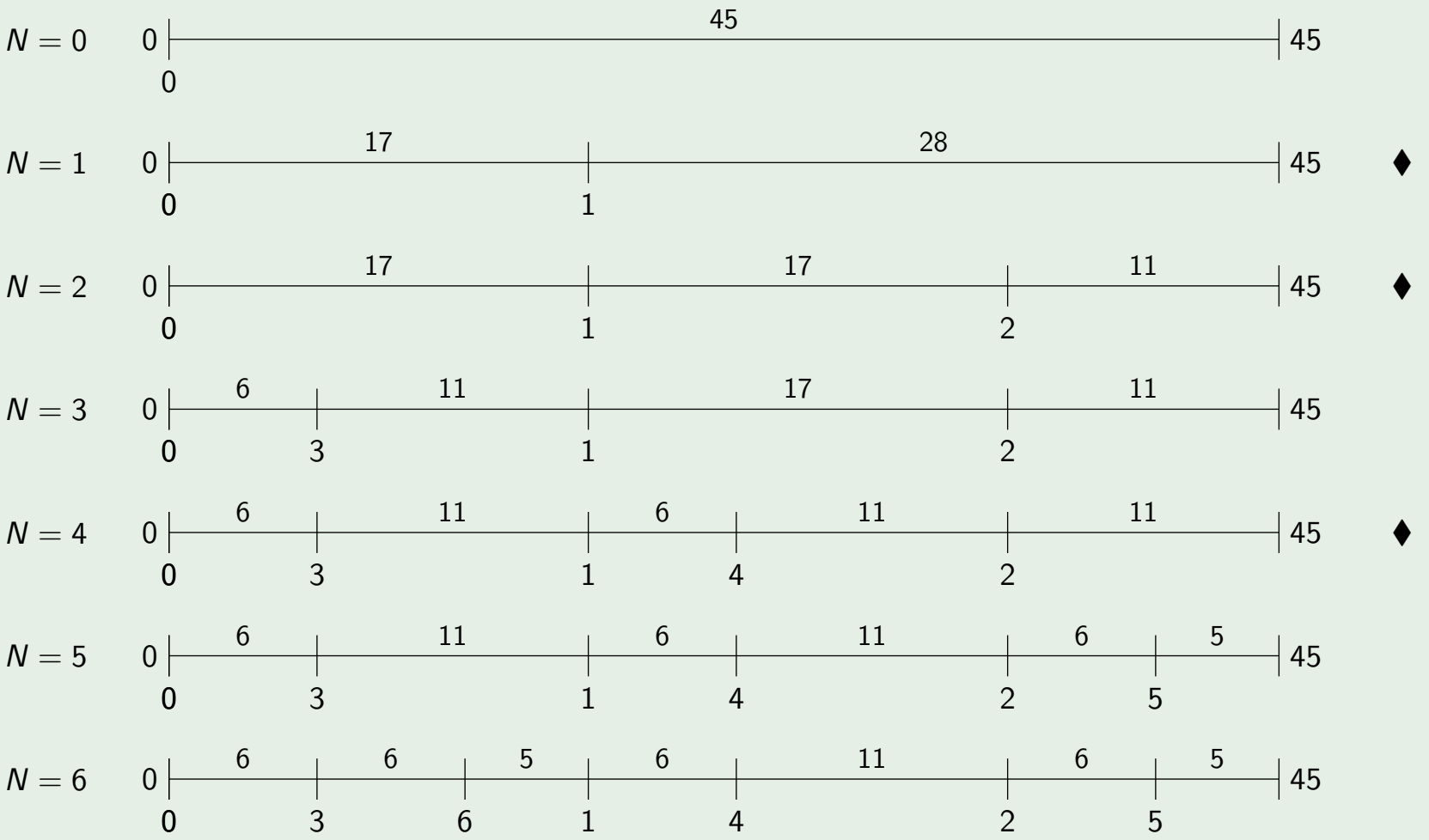
Example : $a = 17/45$, $d = 1$



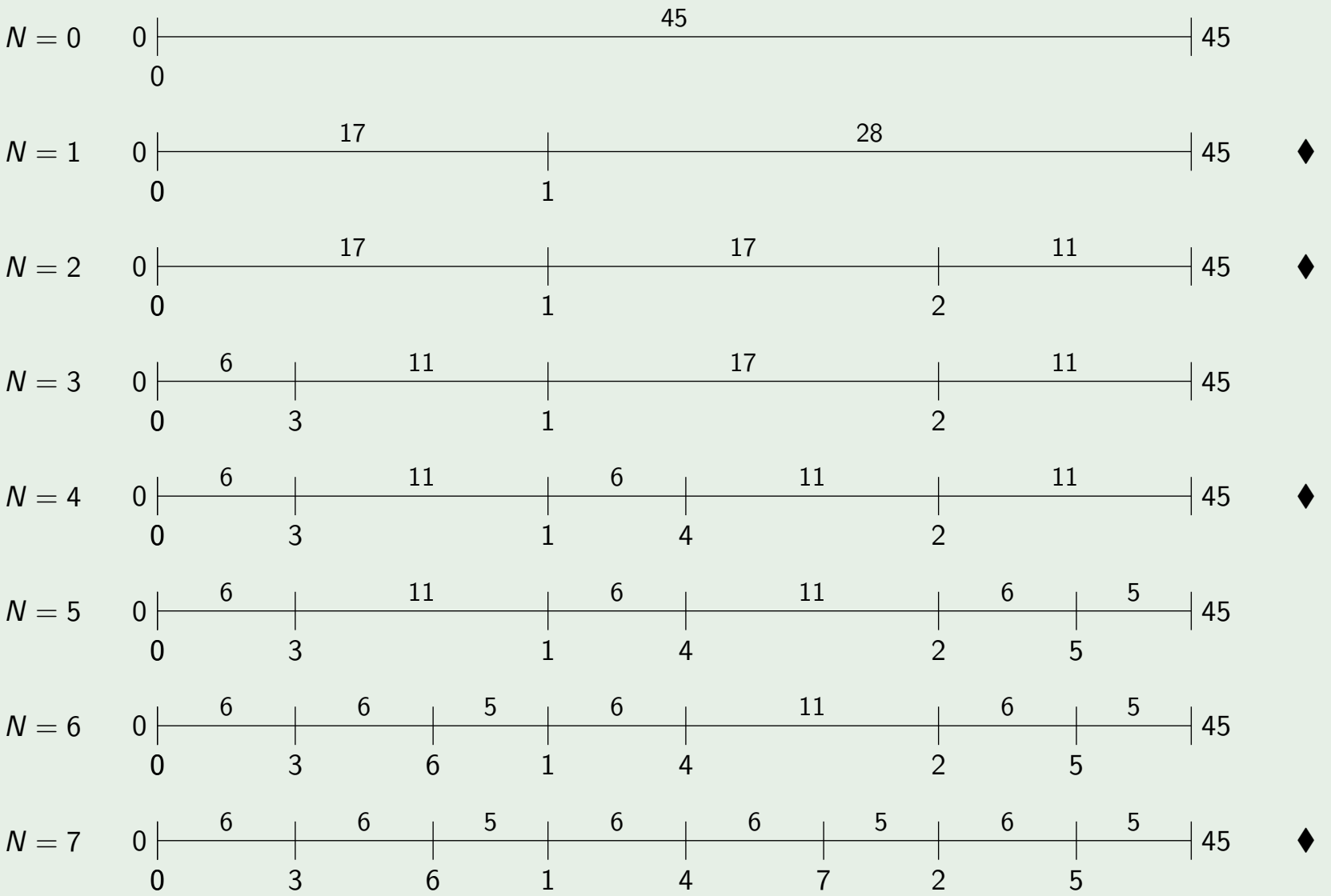
Example : $a = 17/45$, $d = 1$



Example : $a = 17/45$, $d = 1$



Example : $a = 17/45,\ d = 1$



Going from a 2-length configuration to the next

$$(h, l) \to (h - l, l), \text{ with } l < h.$$

$\Rightarrow$ Similar to the Euclidean algorithm for computing continued fraction.

$\Rightarrow$ In fact, this is the continued fraction of d/a.

Continued Fraction Expansion

$$\frac{d_0}{a_0} = q_0 + \frac{d_1}{a_1} = q_0 + \cfrac{1}{q_1 + \cfrac{a_2}{d_2}} = \cdots$$

At each step alternatively,

- $d_{2i} = q_{2i} \cdot a_{2i} + d_{2i+1}$; $\quad a_{2i+1} = a_{2i}$
- $a_{2i+1} = q_{2i+1} \cdot d_{2i+1} + a_{2i+2}$; $\quad d_{2i+2} = d_{2i+1}$

Objective

Compute iteratively b_i, the distance from b to the closest point "to its left".

4 cases

1. b is in an interval of length a_i and we reduce d_i,
2. b is in an interval of length d_i and we reduce a_i,
3. b is in an interval of length d_i and we reduce d_i,
4. b is in an interval of length a_i and we reduce a_i.

Objective

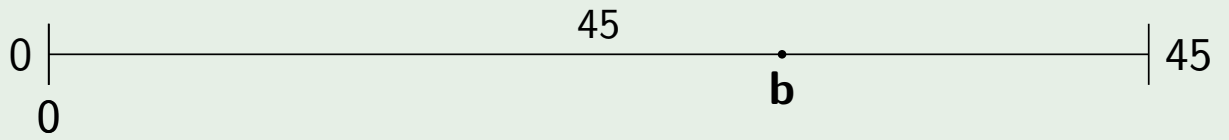
Compute iteratively b_i, the distance from b to the closest point "to its left".

4 cases

1. b is in an interval of length a_i and we reduce d_i,
   $\Rightarrow$ Nothing to do
2. b is in an interval of length d_i and we reduce a_i,
   $\Rightarrow$ Nothing to do
3. b is in an interval of length d_i and we reduce d_i,
4. b is in an interval of length a_i and we reduce a_i.

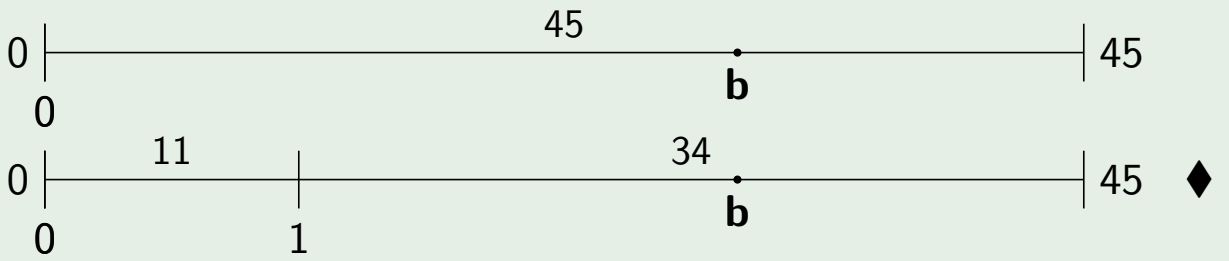
Case 3 : reduction of d_j

$a = 11/45$; $d = 1$; $b = 30/45$



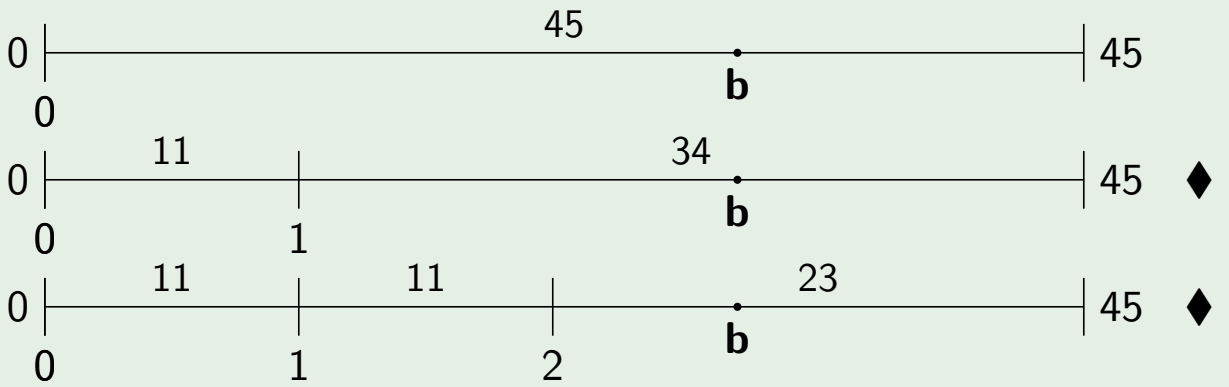
Case 3 : reduction of d_i

$a = 11/45$; $d = 1$; $b = 30/45$



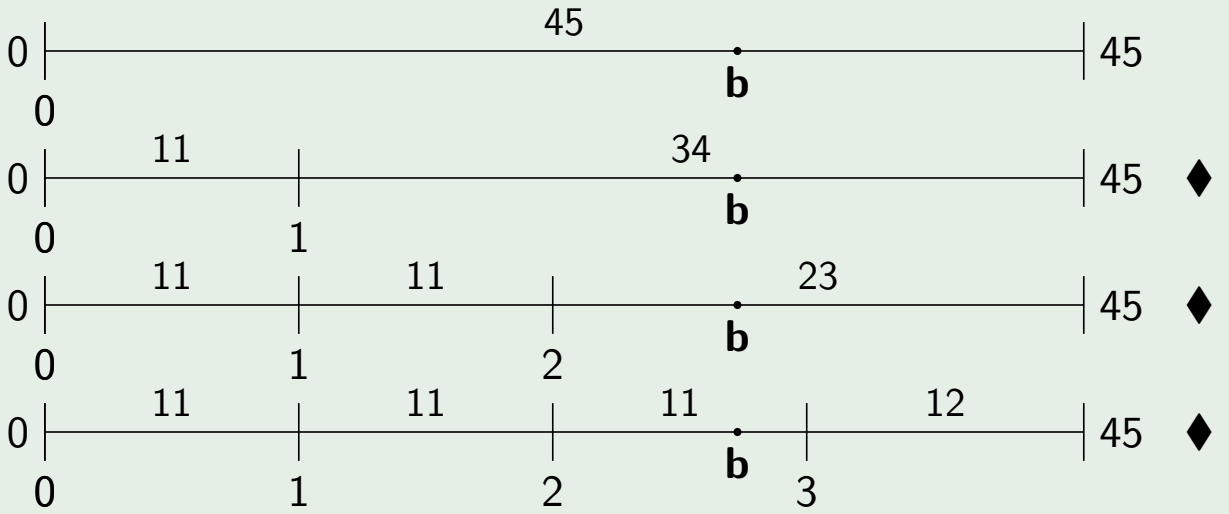
Case 3 : reduction of d_i

$a = 11/45\,;\ d = 1\,;\ b = 30/45$



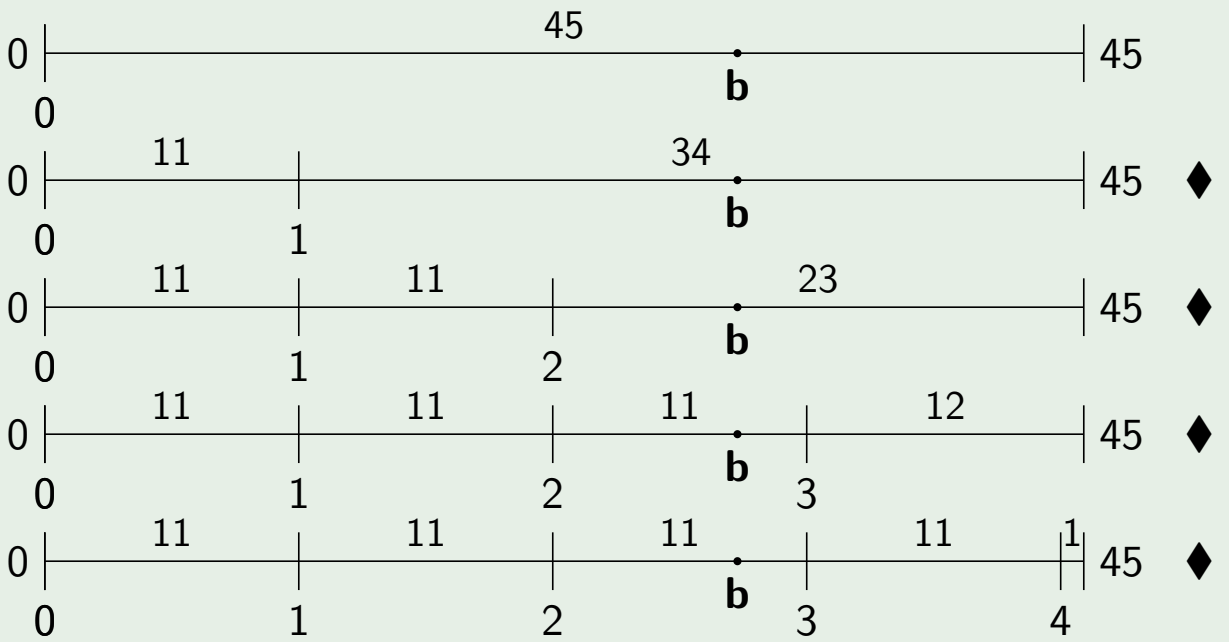
Case 3 : reduction of d_i

$a = 11/45\,;\ d = 1\,;\ b = 30/45$



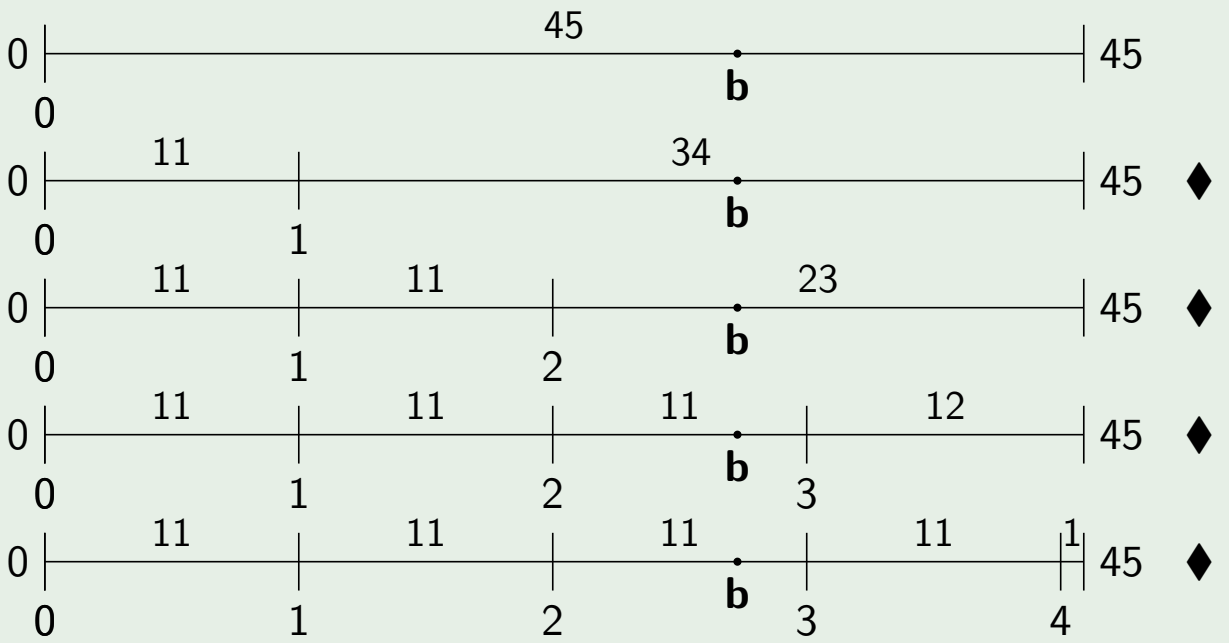
Case 3 : reduction of d_i

$a = 11/45\,;\ d = 1\,;\ b = 30/45$



Case 3 : reduction of d_i

$a = 11/45\,;\ d = 1\,;\ b = 30/45$



b reduction rule

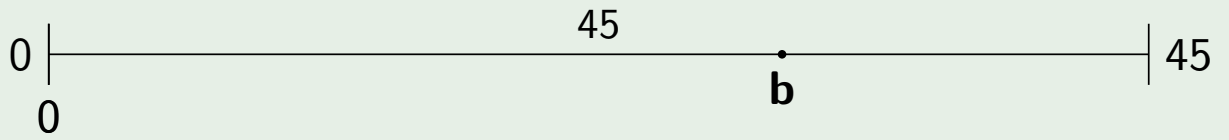
$$b_{i+1} = b_i \mod a_{i+1}$$

4 cases

- b is in an interval of length a_i and we reduce d_i,
  $\Rightarrow$ Nothing to do
- b is in an interval of length d_i and we reduce a_i,
  $\Rightarrow$ Nothing to do
- b is in an interval of length d_i and we reduce d_i,
  $\Rightarrow$ Reduction "from the left" : $b_{i+1} = b_i \mod a_{i+1}$
- b is in an interval of length a_i and we reduce a_i.

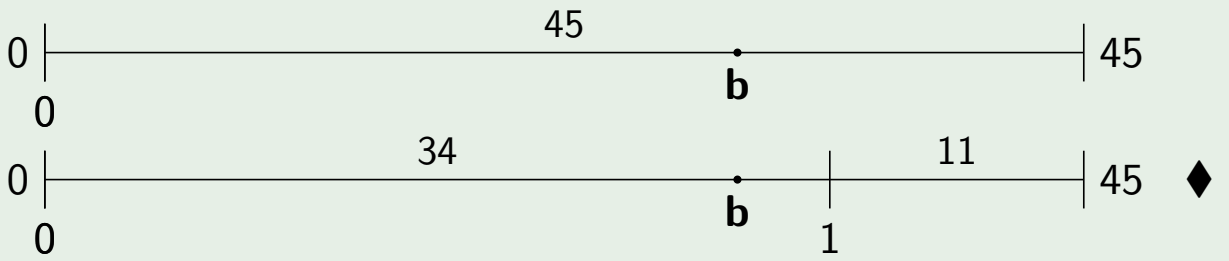
Case 4 : reduction of a_i

$a = 34/45\,;\ d = 1\,;\ b = 30/45$



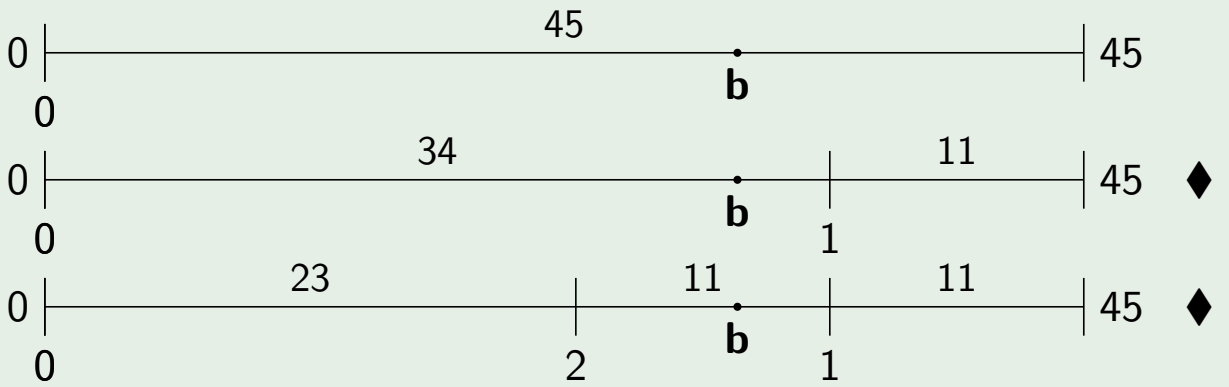
Case 4 : reduction of a_i

$a = 34/45$; $d = 1$; $b = 30/45$



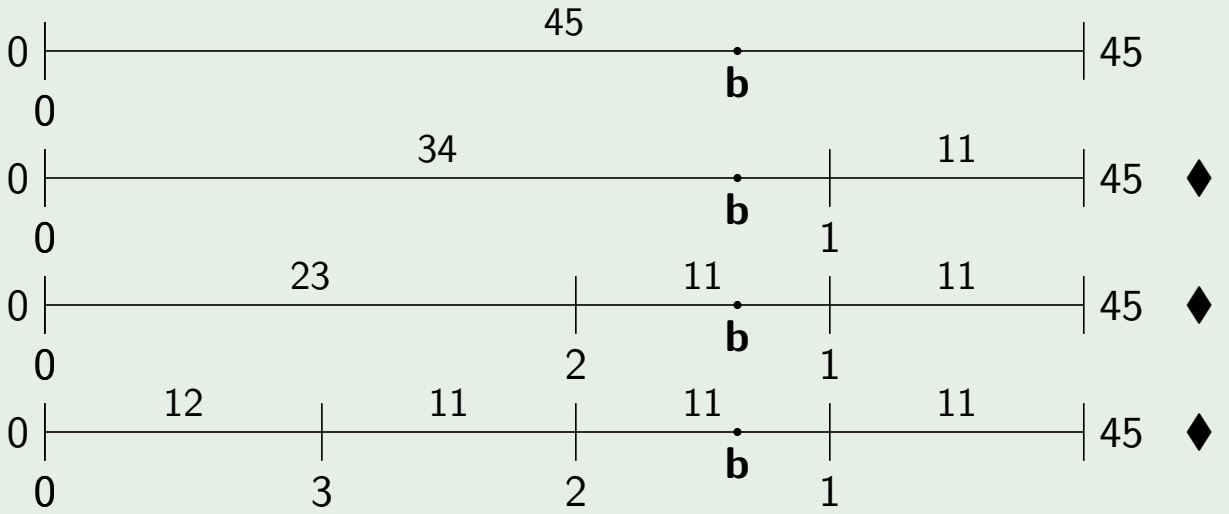
Case 4 : reduction of a_i

$a = 34/45$; $d = 1$; $b = 30/45$



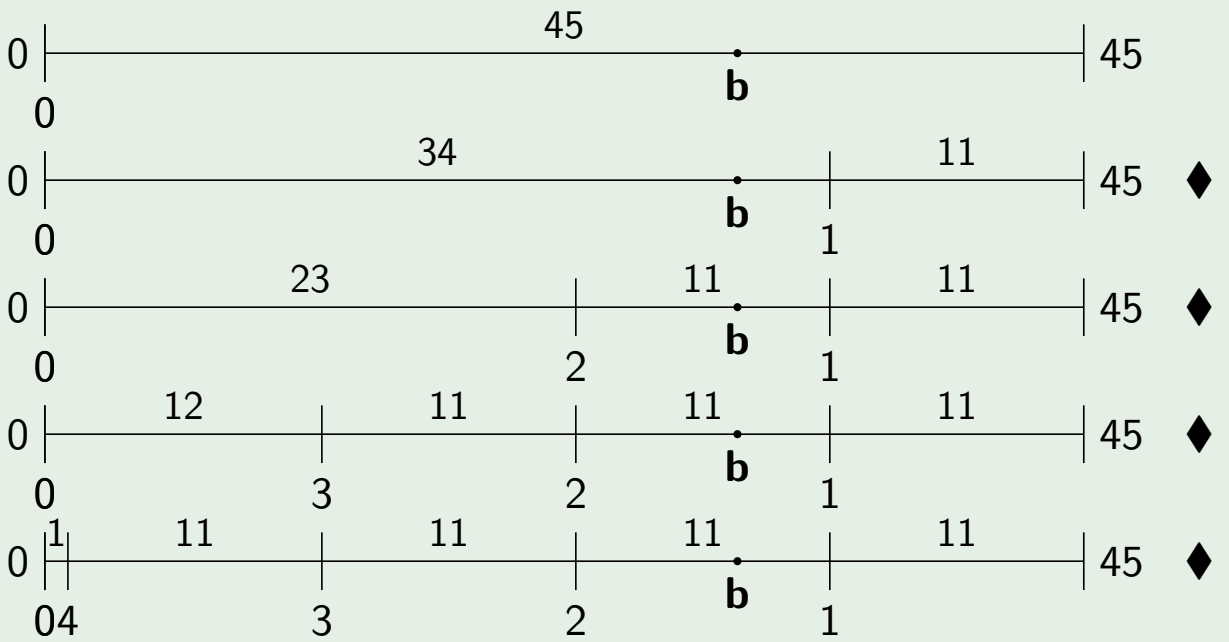
Case 4 : reduction of a_i

$a = 34/45\,;\ d = 1\,;\ b = 30/45$



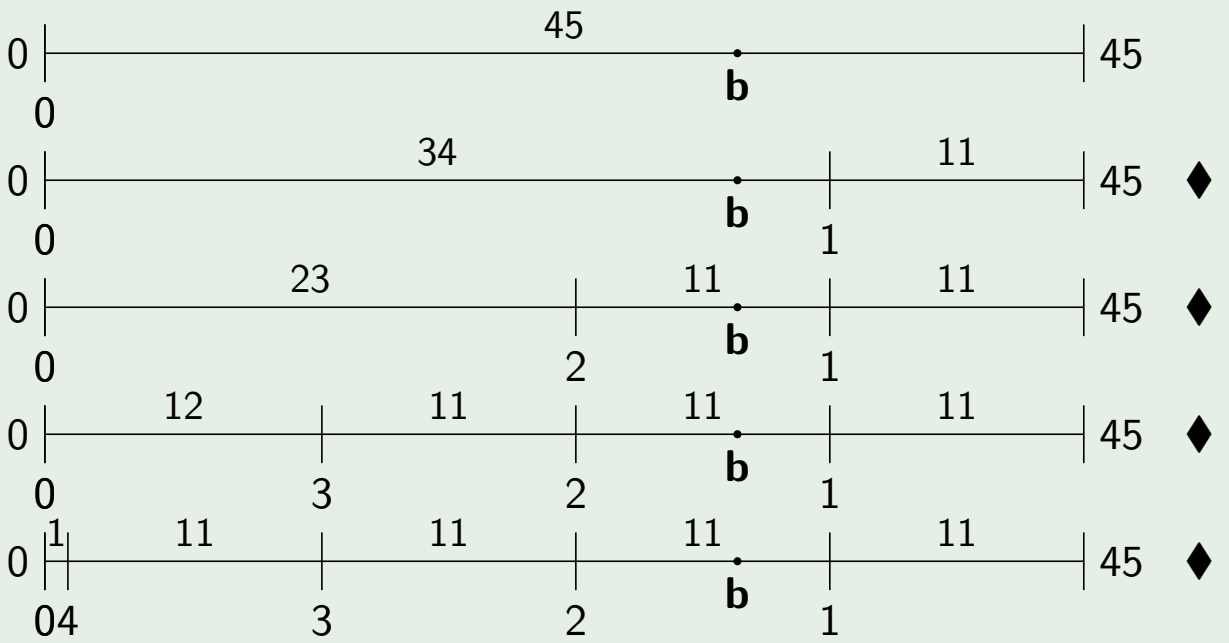
Case 4 : reduction of a_i

$a = 34/45\,;\ d = 1\,;\ b = 30/45$



Case 4 : reduction of a_i

$a = 34/45\,;\ d = 1\,;\ b = 30/45$



b reduction rule

$$b_{i+1} = (b_i - a_{i+1}) \mod d_{i+1}$$

4 cases

- b is in an interval of length a_i and we reduce d_i,
  $\Rightarrow$ Nothing to do
- b is in an interval of length d_i and we reduce a_i,
  $\Rightarrow$ Nothing to do
- b is in an interval of length d_i and we reduce d_i,
  $\Rightarrow$ Reduction "from the left" : $b_{i+1} = b_i \mod a_{i+1}$
- b is in an interval of length a_i and we reduce a_i.
  $\Rightarrow$ Reduction "from the right" : $b_{i+1} = (b_i - a_{i+1}) \mod d_{i+1}$

Lefèvre's algorithm

Update the distance from b to the closest point "to its left" as soon as we add a point to the left of b.

New algorithm

Update the distance from b to the closest point "to its left" at each step of the continued fraction expansion.

Lefèvre's algorithm

input : $P(x) = ax + b$, ε, N

initialisation : $x \leftarrow \{a\}$; $y \leftarrow 1 - \{a\}$; $z \leftarrow \{b\}$;
$u \leftarrow 1$; $v \leftarrow 1$;

if $z < \varepsilon$ **then return** Fail;
while *True* **do**
  if $z < x$ **then**
    $q \leftarrow \lfloor x/y \rfloor$; /* b is in a_i */
    $y \leftarrow y - q \times x$; /* reduction of d_i */
    $u \leftarrow u + q \times v$;
    if $u + v \geq N$ **then return** Success;
    $x \leftarrow x - y$; $v \leftarrow u + v$; /* reduction of a_i by one d_i */
  else
    $z \leftarrow z - x$; /* b changed from a_i to d_i */
    if $z < \varepsilon$ **then return** Fail; /* update distance to b */
    $q \leftarrow \lfloor y/x \rfloor$; /* reduction of a_i */
    $x \leftarrow x - q \times y$;
    $v \leftarrow v + q \times u$;
    if $u + v \geq N$ **then return** Success;
    $y \leftarrow y - x$; $u \leftarrow u + v$; /* reduction of a_i by one d_i */

Lefèvre's algorithm

input : $P(x) = ax + b$, ε, N

initialisation : $x \leftarrow \{a\}$; $y \leftarrow 1 - \{a\}$; $z \leftarrow \{b\}$;
$u \leftarrow 1$; $v \leftarrow 1$;

if $z < \varepsilon$ **then return** Fail;
while *True* **do**
  if $z < x$ **then**
    $q \leftarrow \lfloor x/y \rfloor$; /* b is in a_i */
    $y \leftarrow y - q \times x$; /* reduction of d_i */
    $u \leftarrow u + q \times v$;
    if $u + v \geq N$ **then return** Success;
    $x \leftarrow x - y$; $v \leftarrow u + v$; /* reduction of a_i by one d_i */
  else
    $z \leftarrow z - x$; /* b changed from a_i to d_i */
    if $z < \varepsilon$ **then return** Fail; /* update distance to b */
    $q \leftarrow \lfloor y/x \rfloor$; /* reduction of a_i */
    $x \leftarrow x - q \times y$;
    $v \leftarrow v + q \times u$;
    if $u + v \geq N$ **then return** Success;
    $y \leftarrow y - x$; $u \leftarrow u + v$; /* reduction of a_i by one d_i */

Bottleneck

$\Rightarrow$ From division-based to subtraction-based Euclidian algorithm when splitting interval containing b.

Lefèvre's algorithm

input : $P(x) = ax + b$, ε, N

initialisation : $x \leftarrow \{a\}$; $y \leftarrow 1 - \{a\}$; $z \leftarrow \{b\}$;
$u \leftarrow 1$; $v \leftarrow 1$;

if $z < \varepsilon$ **then return** Fail;
while *True* **do**
  if $z < x$ **then**
    $q \leftarrow \lfloor x/y \rfloor$; /* b is in a_i */
    $y \leftarrow y - q \times x$; /* reduction of d_i */
    $u \leftarrow u + q \times v$;
    if $u + v \geq N$ **then return** Success;
    $x \leftarrow x - y$; $v \leftarrow u + v$; /* reduction of a_i by one d_i */
  else
    $z \leftarrow z - x$; /* b changed from a_i to d_i */
    if $z < \varepsilon$ **then return** Fail; /* update distance to b */
    $q \leftarrow \lfloor y/x \rfloor$; /* reduction of a_i */
    $x \leftarrow x - q \times y$;
    $v \leftarrow v + q \times u$;
    if $u + v \geq N$ **then return** Success;
    $y \leftarrow y - x$; $u \leftarrow u + v$; /* reduction of a_i by one d_i */

Bottleneck

$\Rightarrow$ In Lefèvre's implementation, special codes to partly compute partial quotients with division in cases 3 and 4.

New algorithm

input : $P(x) = ax + b$, ε, N

initialisation : $x \leftarrow \{a\}$; $y \leftarrow 1$; $z \leftarrow \{b\}$;
$u \leftarrow 0$; $v \leftarrow 1$;

if $z < \varepsilon$ **then return** Fail;
while *True* **do**
  if $x < y$ **then**
    $q = y/x$; /* reduction of a_i */
    $y = y - q * x$;
    $u = u + q * v$;
    $z = z \mod x$; /* update distance to b */
  else
    $q = x/y$; /* reduction of d_i */
    $x = x - q * y$;
    $v = v + q * u$;
    if $z \geq x$ **then**
      $z = z - x$; /* update distance to b */
      $z = z \mod y$;
  if $u + v \geq N$ **then return** $z > \varepsilon$;

Divergence within the main loop

A deterministic test

a_i and d_i are reduced alternatively

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

Divergence within the main loop

New algorithm unrolled

input : $P(x) = ax + b$, ε, N

initialisation : $x \leftarrow \{a\}$; $y \leftarrow 1$; $z \leftarrow \{b\}$;
$u \leftarrow 0$; $v \leftarrow 1$;

while *True* **do**

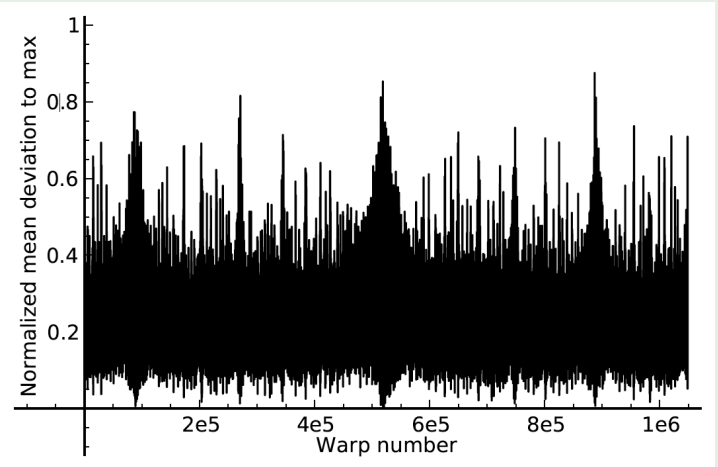
/* reduction of y */
$q = y/x$;
$y = y - q * x$;
$u = u + q * v$;
$z = z \mod x$;
if $u + v \geq N$ **then return** $z > \varepsilon$;
/* reduction of x */
$q = x/y$;
$x = x - q * y$;
$v = v + q * u$;
if $z \geq x$ **then**
$z = z - x$;
$z = z \mod y$;
if $u + v \geq N$ **then return** $z > \varepsilon$;

Divergence on the main loop (*exp*, interval $[1, 1 + 2^{-13}]$)

Normalized mean deviation to the maximum (NMDM)

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

Lefèvre's Algorithm

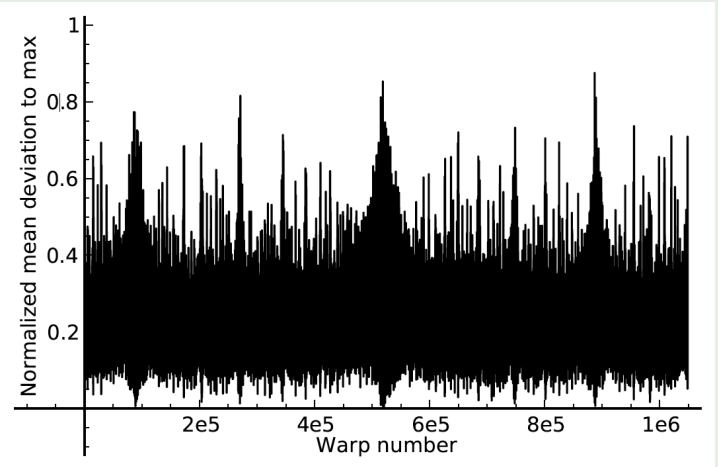


Divergence on the main loop (*exp*, interval $[1, 1 + 2^{-13}]$)

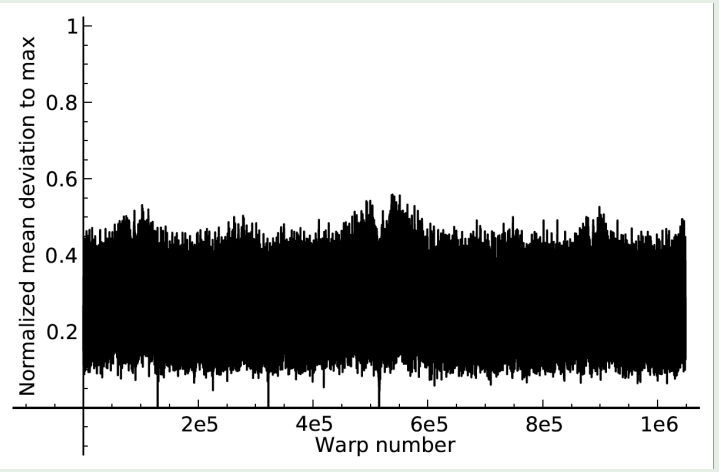
Normalized mean deviation to the maximum (NMDM)

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

Lefèvre's Algorithm



L-algorithm with special codes

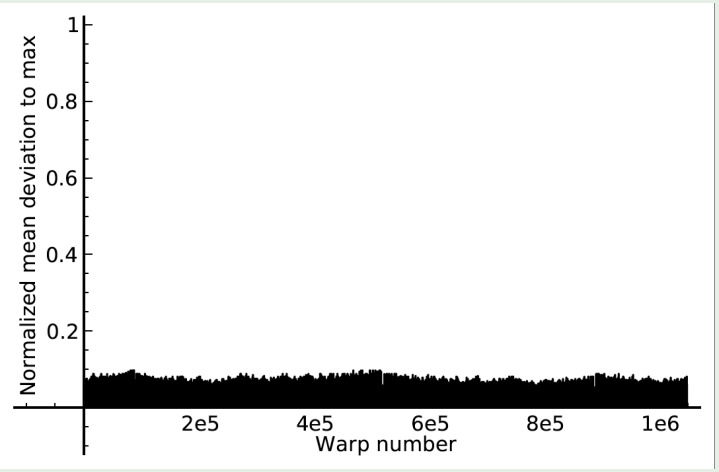


Divergence on the main loop (*exp*, interval $[1, 1 + 2^{-13}]$)

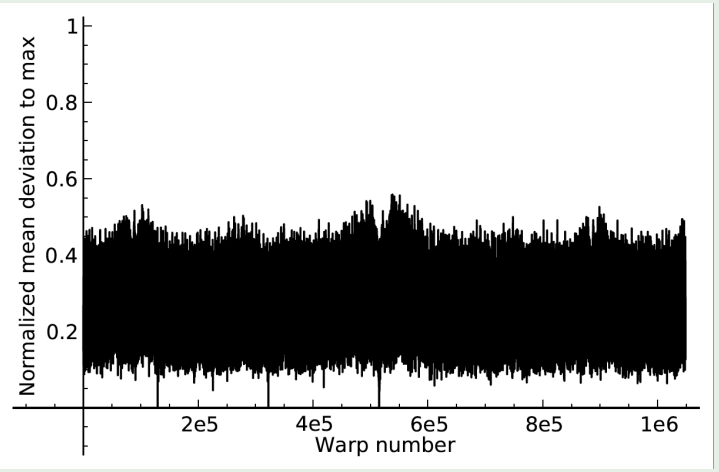
Normalized mean deviation to the maximum (NMDM)

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

New LSG



L-algorithm with special codes



Divergence on the main loop (*exp*, interval $[1, 1 + 2^{-13}]$)

Implementations Comparisons

Implementation	min iterations	max iterations	mean iterations	mean NMDM
Lefèvre's	5	328	24	25.6%
With special code	5	31	16	25.7%
New LSG	8	19	12	0.1%

Results

Times in seconds for HR-case search in $[1; 2]$ (2^{53} *doubles*, $\varepsilon = 2^{-96}$)

	CPU(X5650)	GPU(C2070)	Speedup
Lefèvre's algorithm	36816.10	2446.87	15.0
New algorithm	34039.94	705.89	48.2
Speedup	8.2%	3.5	

Results by phase

Number of intervals by phase for HR-case search in $[1; 1 + 2^{-13}]$ (2^{40} *doubles*, $\varepsilon = 2^{-96}$)

	Phase 1	Phase 2	Phase 3
Lefèvre's algorithm	33554432	109048	2182
New algorithm	33554432	558289	14340
Overhead	1.0	5.1	6.6

Times by phase for HR-case search in $[1; 1 + 2^{-13}]$ (2^{40} *doubles*, $\varepsilon = 2^{-96}$)

	Phase 1	Phase 2	Phase 3	Total
Lefèvre's algorithm	0.242	0.005	0.001	0.249
New algorithm	0.076	0.010	0.002	0.089
Overhead	3.2	1/1.8	1/2.1	2.8

Conclusion

Conclusion

- Algorithmic solutions to minimize :
  - loop divergence,
  - conditional divergence.
- Speedup of 8.2% on CPU and 52.2 on GPU over Lefèvre's algorithm on CPU.

Future works

- Deployment of the algorithm with OpenCL :
  - Portable version $\Rightarrow$ test on AMD GPUs.
  - Automatic Vectorization on CPU (SSE, AVX) with Intel's OpenCL SDK.
  - Target incoming Intel's MIC (50 x86 cores on Knight's Corner).
- Further optimizations and tests :
  - Unroll manually some loop iterations to fill pipeline.
  - Find the most suited division algorithm.
- Rigorous proof of the algorithm :
  - Proof $k \cdot a \mod d = k_1 a_1 - k_2 d_2 + \cdots + k_{n-1} a_{n-1} - k_n d_n$