Solving the Table Maker's Dilemma by reducing divergence on GPU

Pierre Fortin, Mourad Gouicem, Stef Graillat

PEQUAN Team, LIP6/UPMC

SCAN 2012, Novosibirsk, Russia
September 27th 2012

The IEEE 754-2008 standard

Aim

Ensure predictable and portable numerical software.

Basic Formats

- single-precision (binary32)
- double-precision (binary64)
- quadruple-precision (binary128)

Rounding Modes

- Rounding to nearest
- Directed rounding (towards 0, $-\infty$ and $+\infty$)

Correctly rounded operations

$+, -, \times, /, \sqrt{\ }$

And for elementary mathematical functions ?

exp, log, sin, cos, tan, $\cdots$

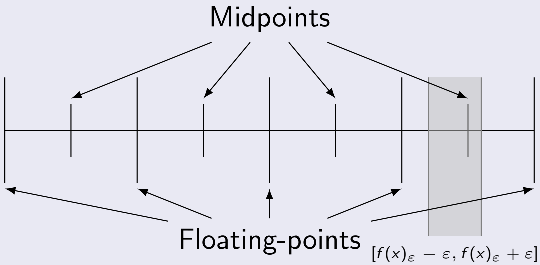
$\Rightarrow$ IEEE-754-2008 only recommends correct rounding because of the Table Maker's Dilemma

The Table Maker's Dilemma

Correct rounding

$$\circ_p(f(x)_\varepsilon) = \circ_p(f(x)_0)$$

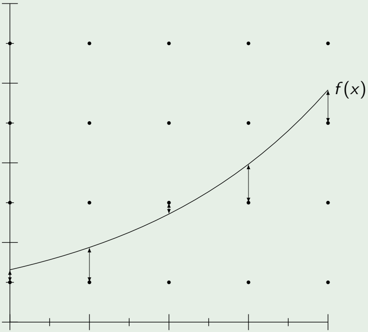
Hard-to-round case



The Table Maker's Dilemma

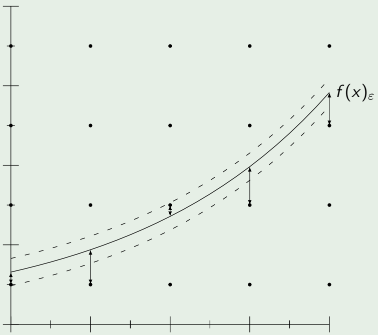
Given a function f defined over I and a rounding mode $\circ_p$, find ϵ such that $\forall x \in I$

$$\circ_p(f(x)_\epsilon - \epsilon) = \circ_p(f(x)_\epsilon + \epsilon).$$



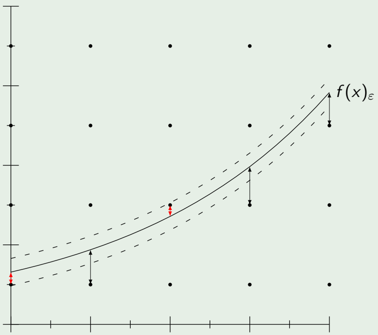
General Framework

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



General Framework

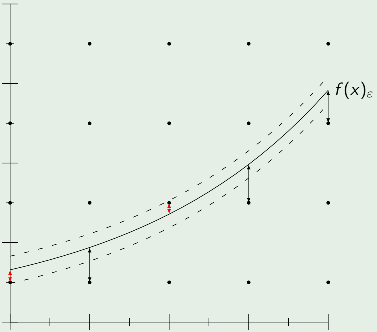
1. Split the domain and approximate the function on each sub-domain with error ε.
2. Search hard-to-round cases.



The Table Maker's Dilemma

General Framework

1. Split the domain and approximate the function on each sub-domain with error ε.
2. Search hard-to-round cases.
3. Find the hardness-to-round ϵ of f among the HR-cases.



High Performance Computing

Problem

- HR-cases search is very computationally intensive.
  $\Rightarrow$ Several years of computation on CPU.
- Time complexity is exponential in the number of bits of the targeted format.

Good news

- We focus on fixed size intances namely 64, 80 and 128-bit formats.
- We can search for HR-cases in each sub-domain independently.
  $\Rightarrow$ Embarrassingly parallel problem.

High Performance Computing

Single Instruction Multiple Data (SIMD)

Data parallelism implemented in almost all hardware :

- Intel X5650 CPU : 6 SIMD cores (SSE intructions : 4x32-bit data)
- NVIDIA C2070 GPU : 14 SIMD cores (32x32-bit data)

High Performance Computing

Single Instruction Multiple Data (SIMD)

Data parallelism implemented in almost all hardware :

- Intel X5650 CPU : 6 SIMD cores (SSE intructions : 4x32-bit data)
- NVIDIA C2070 GPU : 14 SIMD cores (32x32-bit data)

CUDA

- Language designed for NVIDIA GPU.

Single Instruction Multiple Data (SIMD)

Data parallelism implemented in almost all hardware :

- Intel X5650 CPU : 6 SIMD cores (SSE intructions : 4x32-bit data)
- NVIDIA C2070 GPU : 14 SIMD cores (32x32-bit data)

CUDA

- Language designed for NVIDIA GPU.
- Threads are grouped by *warps* and executed on SIMD Units.
  $\Rightarrow$ The threads of a warp must execute the same instructions at the same time.

Single Instruction Multiple Data (SIMD)

Data parallelism implemented in almost all hardware :

- Intel X5650 CPU : 6 SIMD cores (SSE intructions : 4x32-bit data)
- NVIDIA C2070 GPU : 14 SIMD cores (32x32-bit data)

CUDA

- Language designed for NVIDIA GPU.
- Threads are grouped by *warps* and executed on SIMD Units.
  $\Rightarrow$ The threads of a warp must execute the same instructions at the same time.
- If the treads of a warp do not follow the same execution path (conditionals and loops), they *diverge*.
  $\Rightarrow$ Their executions are serialized.

HR-case search

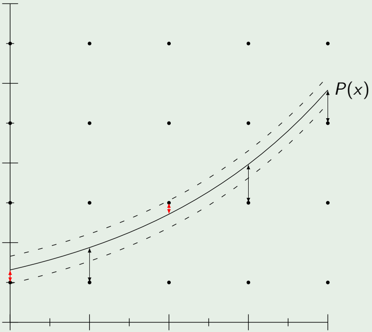
Problem

$$\text{Given } |P(x) - f(x)| < \varepsilon \text{ with } P \in \mathbb{R}[x]$$

$$\text{Find } x \in \mathbb{N}, \text{ if it exists, such that :}$$

$$\begin{cases} x < N \\ |P(x) \text{ cmod } d| < \varepsilon \end{cases}$$

$$\text{with } (d, \varepsilon, N) \in \mathbb{N}^3.$$



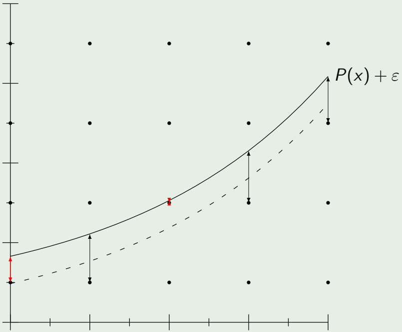
HR-case search

Problem

Given $|P(x) - f(x)| < \varepsilon$ with $P \in \mathbb{R}[x]$
Find $x \in \mathbb{N}$, if it exists, such that :

$$\begin{cases} x < N \\ P(x) + \varepsilon \bmod d < 2\varepsilon \end{cases}$$

with $(d, \varepsilon, N) \in \mathbb{N}^3$.



HR-case search

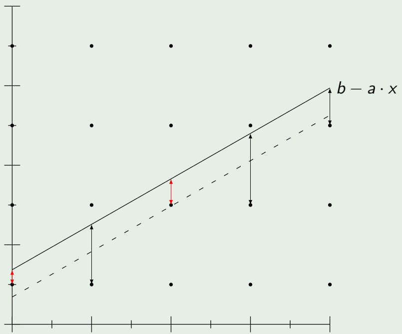
Problem

Given $|P(x) - f(x)| < \varepsilon$ with $P \in \mathbb{R}[x]$

Find $x \in \mathbb{N}$, if it exists, such that :

$$\begin{cases} x < N \\ b - a \cdot x \bmod d < 2\varepsilon \end{cases}$$

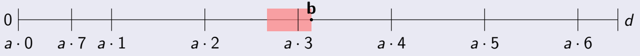
with $(d, \varepsilon, N) \in \mathbb{N}^3$.

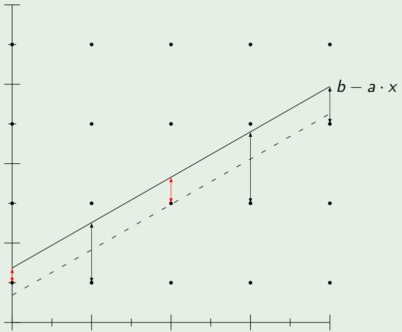


HR-case search

Strategy

- Place $a \cdot x$ modulo d.
- Test if there are points at distance 2ε at the left of b.



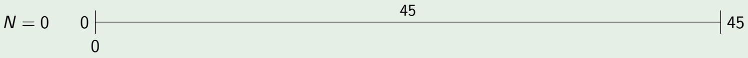


Position of the $a \cdot x \mod d$ on $[0, d[$

Three distance theorem [Slater 50]

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

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

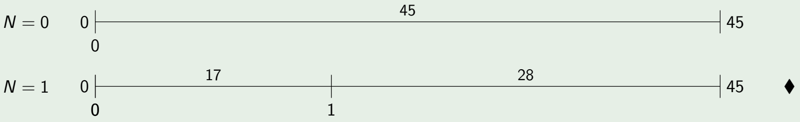


Position of the $a \cdot x \mod d$ on $[0, d[$

Three distance theorem [Slater 50]

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

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



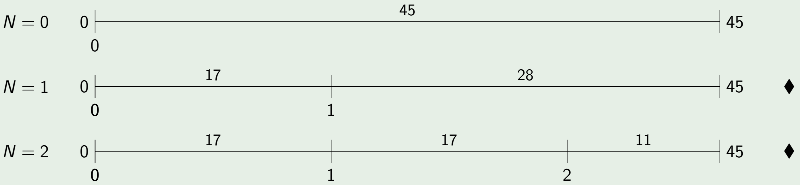
♦ : 2-length configurations

Position of the $a \cdot x \mod d$ on $[0, d[$

Three distance theorem `[Slater 50]`

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

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



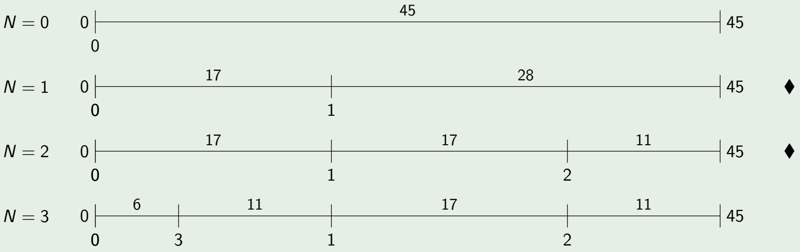
♦ : 2-length configurations

Position of the $a \cdot x \mod d$ on $[0, d[$

Three distance theorem [Slater 50]

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

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



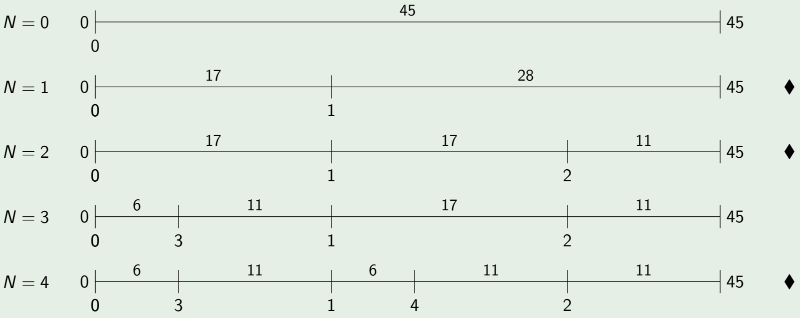
♦ : 2-length configurations

Position of the $a \cdot x \mod d$ on $[0, d[$

Three distance theorem `[Slater 50]`

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

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



♦ : 2-length configurations

Position of the $a \cdot x \mod d$ on $[0, d[$

Going from a 2-length configuration to the next

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

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

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

`[Slater 67]`.

Continued Fraction Expansion

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

At each step alternatively,

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

Computing a lower bound on $b - a \cdot x \mod d$

Objective

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

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.

Computing a lower bound on $b - a \cdot x \mod d$

Objective

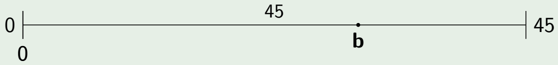
Compute iteratively b_i, the distance from b to the closest point "to its left" at step i.

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.

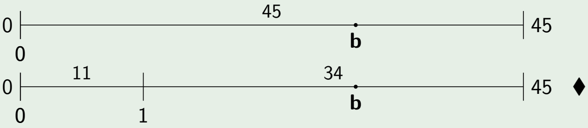
Case 3 : reduction of d_i

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



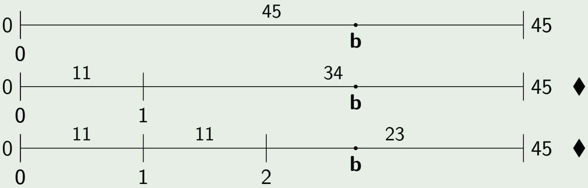
Case 3 : reduction of d_i

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



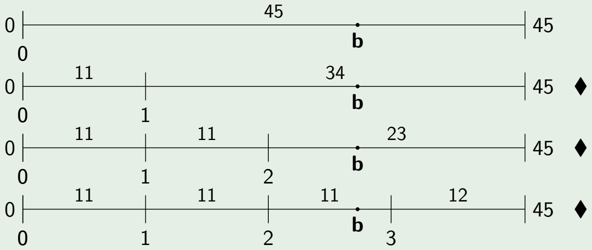
Case 3 : reduction of d_i

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



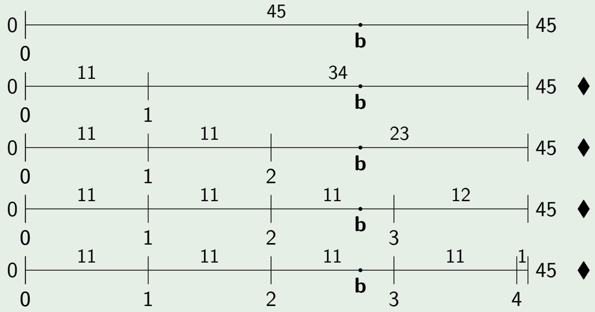
Case 3 : reduction of d_i

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



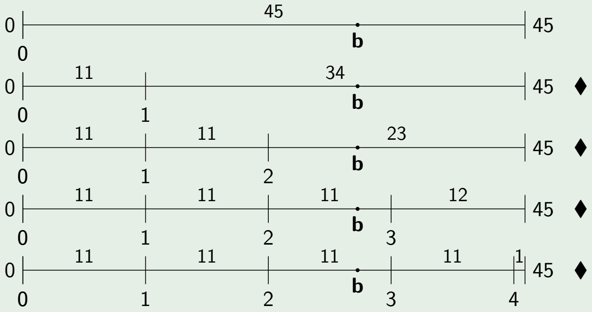
Case 3 : reduction of d_i

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



Case 3 : reduction of d_i

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



b reduction rule

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

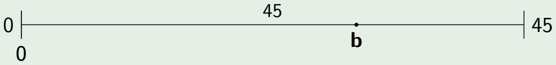
Computing a lower bound on $b - a \cdot x \mod d$

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.

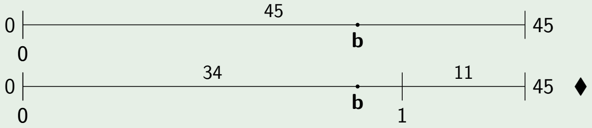
Case 4 : reduction of a_i

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



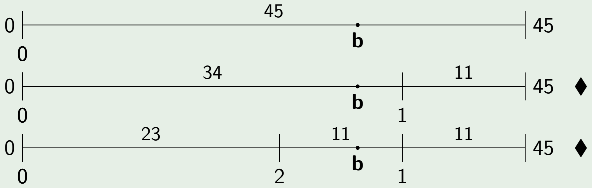
Case 4 : reduction of a_i

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



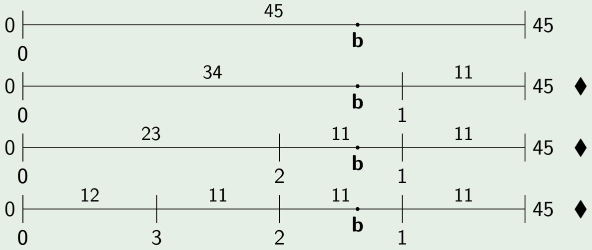
Case 4 : reduction of a_i

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



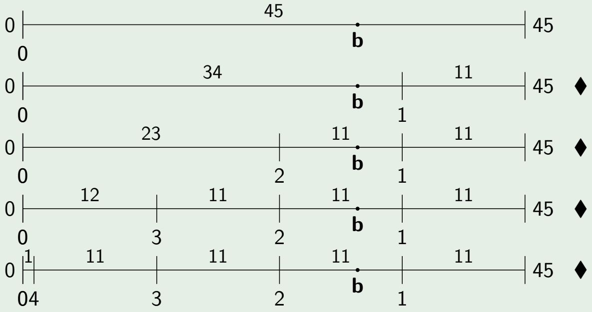
Case 4 : reduction of a_i

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

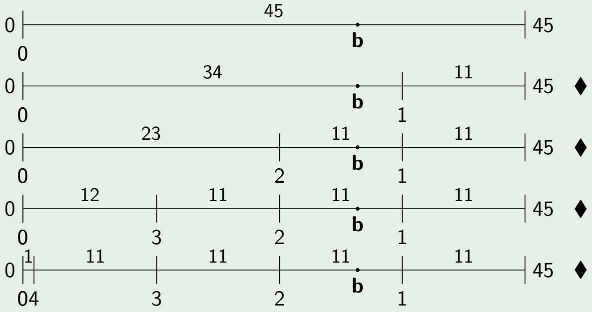


Case 4 : reduction of a_i

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



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



b reduction rule

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

Computing a lower bound on $b - a \cdot x \mod d$

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

Divergence in the two algorithms

Lefèvre 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.
$\Rightarrow$ Condition the reduction of d_i and a_i by the location of b.
$\Rightarrow$ From division-based to subtraction-based Euclidian algorithm when splitting the interval containing b.

New algorithm

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

Divergence in the two algorithms

Lefèvre 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 */`

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.

New algorithm unrolled

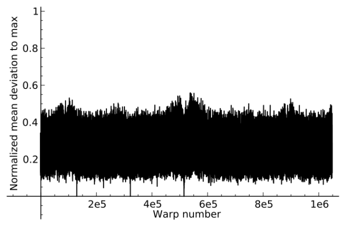
```
input          : P(x) = ax + b, ε, N
initialisation :  x ← {a};   y ← 1;   z ← {b};
                  u ← 0;     v ← 1;
while True do
    /* reduction of y                                  */
    q = y/x;
    y = y − q ∗ x;
    u = u + q ∗ v;
    z = z  mod x;
    if u + v ≥ N then return z > ε;
    /* reduction of x                                  */
    q = x/y;
    x = x − q ∗ y;
    v = v + q ∗ u;
    if z ≥ x then
        z = z − x;
        z = z  mod y;
    if u + v ≥ N then return z > ε;
```

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 Algorithm

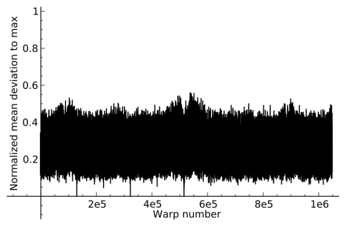


Divergence on the main loop (*exp*, interval $[1, 1 + 2^{-13}]$)

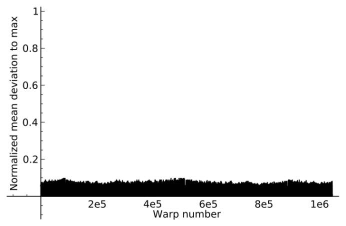
Normalized mean deviation to the maximum (NMDM)

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

Lefèvre Algorithm



New Algorithm

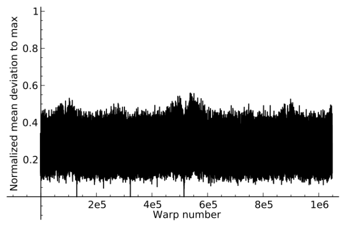


Divergence on the main loop (*exp*, interval $[1, 1 + 2^{-13}]$)

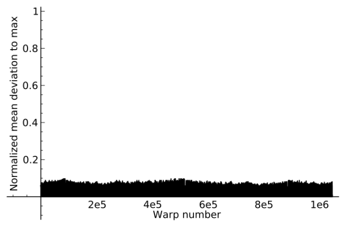
Normalized mean deviation to the maximum (NMDM)

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

Lefèvre Algorithm



New Algorithm



$\Rightarrow$ Lefèvre algorithm goes from division-based to subtraction-based Euclidian algorithm when splitting interval containing *b*.

Results

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

	CPU (X5650) No SIMD	GPU(C2070)	Speedup
Lefèvre algorithm	36816.10	2446.87	x15.0
New algorithm	34039.94	705.89	x48.2
Speedup	x1.08	x3.5	

Total speedup

Lefèvre on a CPU core $\rightarrow$ New algorithm on GPU : x52.2 .
Lefèvre on a hex-core CPU $\rightarrow$ New algorithm on GPU : x7.5 .

Conclusion and perspectives

Conclusion

- Implementation and algorithmic solutions to minimize :
  - loop divergence,
  - conditional divergence.
- Substancial speedups thanks to a more regular control flow.

Perspectives

- If the targeted function is not well approximated by a degree one polynomial
  $\Rightarrow$ Too many HR-cases !
  $\Rightarrow$ Exhaustive search of hardness-to-round becomes huge !
- Solution : using higher degree approximations
  $\Rightarrow$ SLZ algorithm, based on the LLL algorithm.
- Harness SIMD units on other hardware (SSE/AVX CPUs, Intel MIC, ...) with OpenCL, ISPC...