Searching for Some Worst Cases for the Correct Rounding of the Power Functions in Double Precision

Vincent LEFÈVRE

LIP / INRIA, France

Dagstuhl Seminar 08021

January 6–11, 2008

Introduction / Outline

Worst cases for the correct rounding of x^y in double precision:
2 floating-point arguments → too many values to test.
But interesting partial results…

- My algorithm to search for *all* the worst cases of a numerically regular unary function. Sublinear time complexity.
- Application to the integer power functions x^n, where n is an integer (not too large).
  → Joint work with Peter Kornerup and Jean-Michel Muller.
- Application to the detection of the exact cases of x^y.
  → Joint work with Christoph Quirin Lauter.

Lefèvre's Algorithm: Introduction

1976: More general case (Hirschberg and Wong).

1997: Find a lower bound on the distance between a segment and $\mathbb{Z}^2$.
I presented a first *efficient* algorithm (with low-level operations).
Complex proof. In fact, *exact* distance on a larger domain.

2005 (Arith'17): 2 improvements:

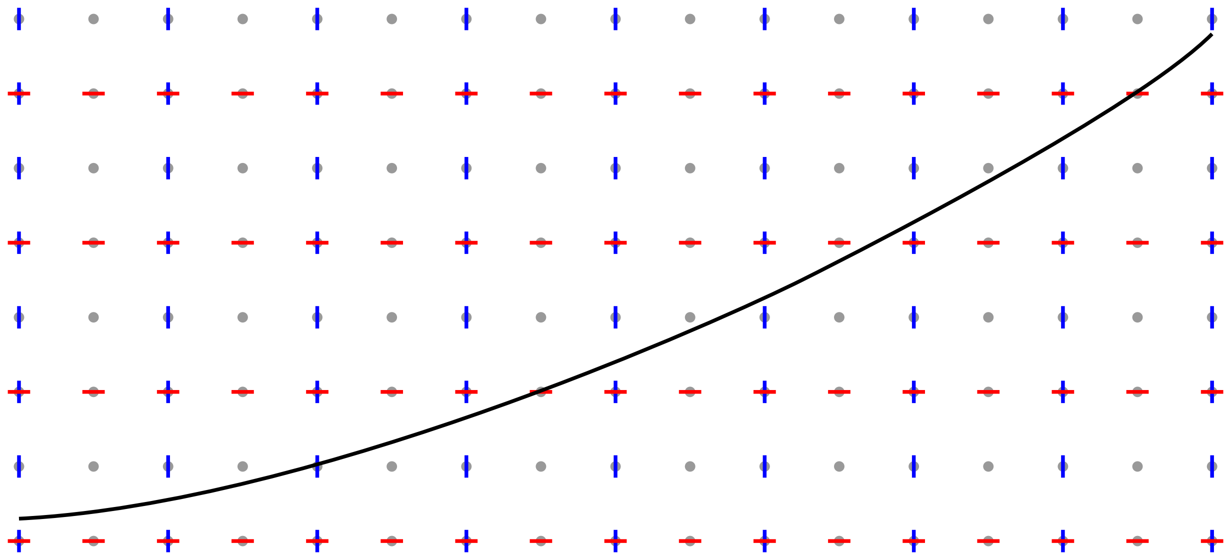
- A more geometrical and intuitive proof.
- A variant/improvement of the algorithm.

Today: simplified explanations…

The Problem (Without Details)

Goal: the exhaustive test of the elementary functions for the TMD in a fixed precision (e.g., in double precision), i.e. "find the breakpoint numbers x such that $f(x)$ is very close to a breakpoint number".

Breakpoint number: machine number or "half-machine number".
$\rightarrow$ Worst cases for f and the inverse function f^{-1}.

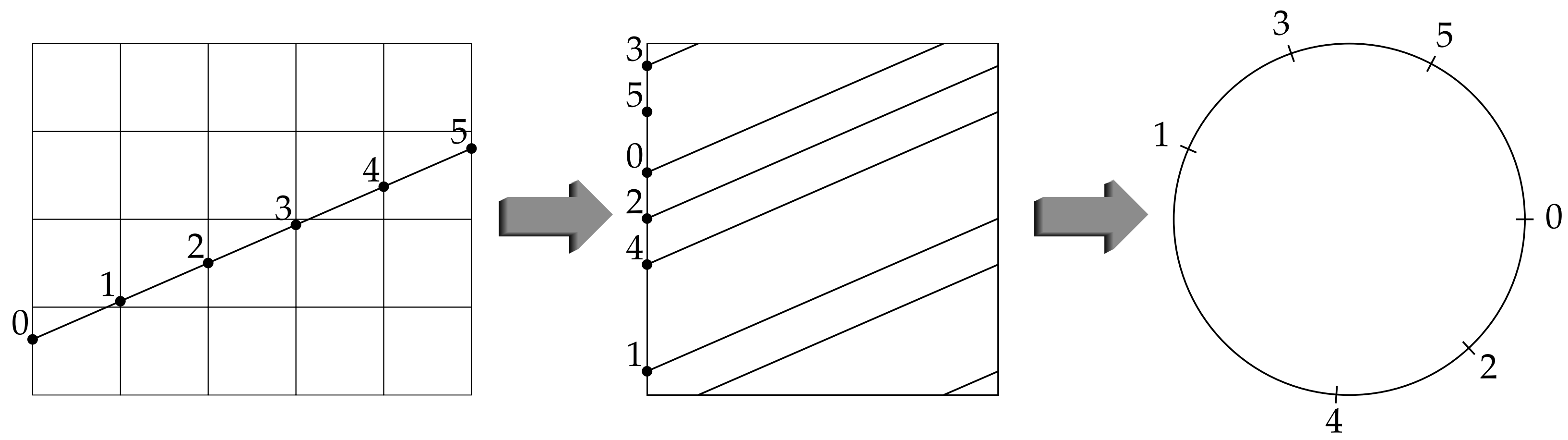


In each interval:

- f approached by a polynomial of degree $1 \to$ segment $y = b - ax$.
- Multiplication of the coordinates by powers of $2 \to$ grid $= \mathbb{Z}^2$.

One searches for the values n such that $\{b - n.a\} < d_0$,
where a, b and d_0 are real numbers and $n \in [\![0, N-1]\!]$.

$\{x\}$ denotes the positive fractional part of x.



- We chose a positive fractional part instead of centered.
  $\rightarrow$ An upward shift is taken into account in b and d_0.

- If a is rational, then the sequence $0.a$, $1.a$, $2.a$, $3.a$, $\ldots$ (modulo 1) is periodical.

  $\rightarrow$ This makes the theoretical analysis more difficult.

  $\rightarrow$ In the proof, one assumes a irrational, or equivalently, a rational number $+$ an arbitrary small irrational number.

  But in the implementation, a is rational.

  $\rightarrow$ Extension to rational numbers by continuity.

  $\rightarrow$ Care has been taken with the inequality tests since they are not continuous functions.

Notations / Properties of $k.a$ mod 1 ($0 \leq k < n$)

Configuration properties to be proved by induction, for some values of n (determined by induction):

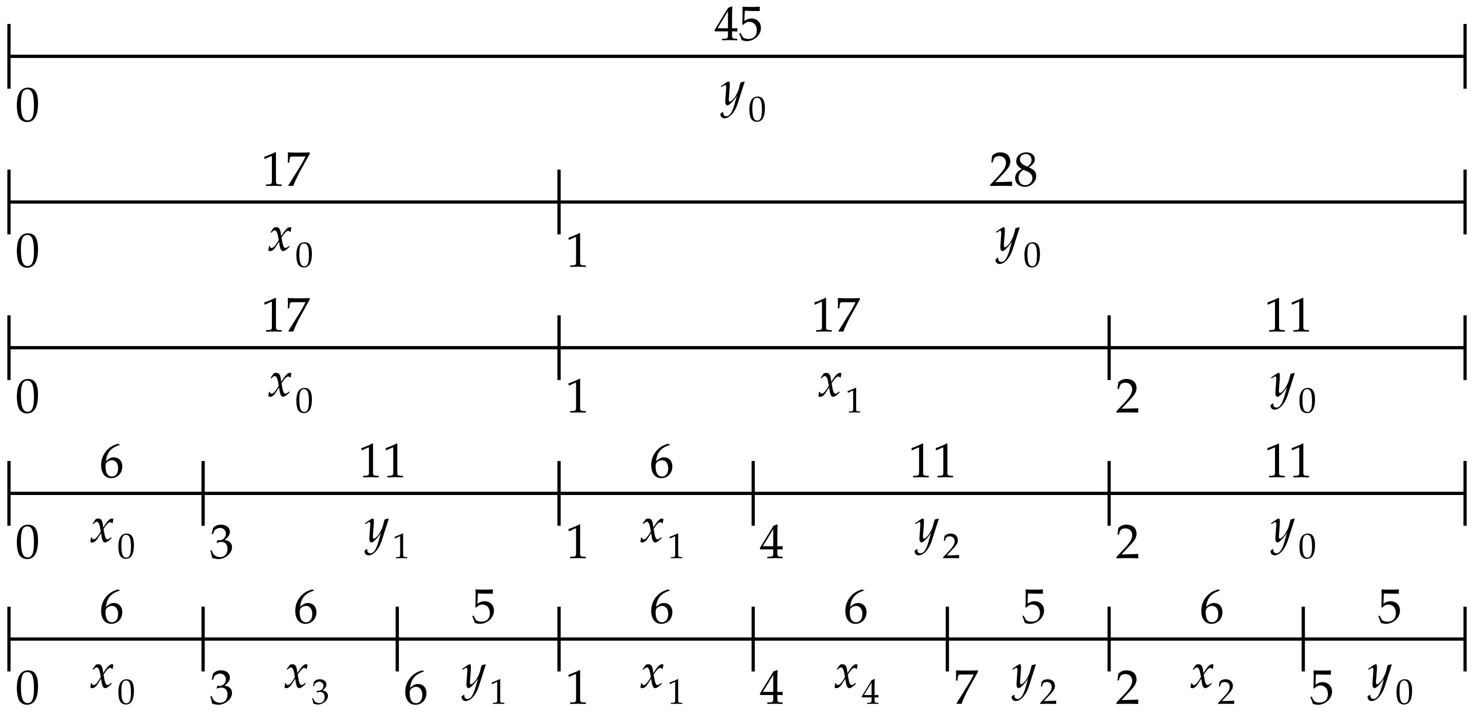
- Intervals $x_0, x_1, \ldots, x_{u-1}$ of length x, where x_0 is the left-most interval and $x_r = x_0 + r.a$ (translation by $r.a$ modulo 1).
- Intervals $y_0, y_1, \ldots, y_{v-1}$ of length y, where y_0 is the right-most interval and $y_r = y_0 + r.a$ (translation by $r.a$ modulo 1).
- Total number of points (or intervals): $n = u + v$.

In short: **2 primary intervals x_0 (left) and y_0 (right) + images.**

Initial configuration: $n = 2$, $u = v = 1$.

Example: The First Configurations

With $a = 17/45$. Note: scaling by 45.



From a Configuration to the Next One

- The main idea: when adding new points, one of the primary intervals is affected first, then all its images are affected in the same way.
  For instance, see both intervals of length 17 on the figure and how they are split on the following two configurations.
- We only need to focus on what occurs in the primary intervals.
- At the same time, we track the position of the point b:
  - whether it is in an interval x_k or in an interval y_k;
  - its distance to the left endpoint of the interval.

How a Primary Interval is Split Into Two Intervals

- The primary interval has an endpoint of index 0 (no inverse image). Let m be the index of the other endpoint.
- The new point $n.a$ splits the interval into $[m.a, n.a]$ and $[n.a, 0.a]$.
- The points of indices m and n are adjacent. $\rightarrow$ So are the points of indices $m-1$ and $n-1$, and their distance ℓ is either x or y. $\rightarrow$ Same distance between the points of indices m and n.
- Only possibility: the primary interval of length $h = \max(x, y)$ is split into 2 intervals of respective lengths $\ell = \min(x, y)$ and $h - \ell$ ($\rightarrow$ similar to the subtractive Euclidean algorithm).

$\rightarrow$ As a consequence, the point of index n is completely determined.

Algorithms

Basic algorithm (1997): returns a lower bound d on $\{b - n.a\}$ for $n \in [\![0, N-1]\!]$ (in fact, d is the exact distance for $n \in [\![0, N'-1]\!]$, where $N \leq N' < 2N$).

Here: parameters chosen so that $d \geqslant d_0$ in most intervals, allowing to immediately conclude that there are no worst cases in the interval.

New algorithm (mentioned in 1998): returns the index $n < N$ of the first point such that $\{b - n.a\} < d_0$, otherwise any value $\geqslant N$ if there are no such points.

Gives the information we need, but uses an additional variable, so that it is slower. Good replacement for the naive algorithm.

Another improvement: test with a shift (fast!) if it is interesting to replace a sequence of iterations by a single one with a division.

The necessary data:

- lengths x and y, numbers u and v of these intervals;
- a binary value saying whether b is in an interval of length x or y;
- the index r of this interval (new algorithm only);
- the distance d between b and the left endpoint of this interval.

Immediate consequence of the properties:

- The left endpoint of an interval x_r has index r.
- The left endpoint of an interval y_r has index $u + r$.

Algorithm (Subtractive Version)

In red: additional instructions for the new algorithm.

Initialization: $x = \{a\}\,;\ y = 1 - \{a\}\,;\ d = \{b\}\,;\ u = v = 1\,;\ r = 0\,;$

if $(d < d_0)$ **return** 0

Unconditional loop:

if $(d < x)$

- **while** $(x < y)$
  - **if** $(u + v \geqslant N)$ **return** N
  - $y = y - x;\ u = u + v;$
- **if** $(u + v \geqslant N)$ **return** N
- $x = x - y;$
- **if** $(d \geqslant x)$ $r = r + v;$
- $v = v + u;$

else

- $d = d - x;$
- **if** $(d < d_0)$ **return** $r + u$
- **while** $(y < x)$
  - **if** $(u + v \geqslant N)$ **return** N
  - $x = x - y;\ v = v + u;$
- **if** $(u + v \geqslant N)$ **return** N
- $y = y - x;$
- **if** $(d < x)$ $r = r + u;$
- $u = u + v;$

Application to the functions $f_n(x) = x^n$

- Only one exponent to test: $f_n(2x) = 2^n f_n(x)$ and $f_n(x)$ have the same significand.
- Input interval $[1, 2)$ decomposed into $2^{13} = 8192$ sub-intervals.
- For each sub-interval: the main test (see next slide).
- Second step to filter the spurious worst cases and the least interesting ones.

For each sub-interval:

- f_n approximated by a degree-d polynomial (for $n = 383$, $d = 11$ to 13, with coefficients on 128 to 576 bits – TODO: reduce the size of the longest coefficients since they are very small);
- code (C + mpn layer of GMP) is generated: my algorithm is applied on sub-intervals of $2^{15} = 32768$ points (64-bit integer arithmetic), and in case of failure, $2^{12} = 4096$ or (for large n) $2^{11} = 2048$ points, and if this still fails, the naive method;
- if supported, the code is compiled using `-fprofile-generate` and tested on the first $2^8 = 256$ sub-intervals;
- the code is recompiled using `-fprofile-use` and run.

$n = 383$: 140 to 250 seconds per sub-interval on a 2.2 GHz Opteron.

Current Results (to nearest, n from 3 to 388)

Values of n: $\exists\ x$ such that the significand of x^n has k identical bits after the rounding bit (exact cases excluded)	k
32	48
76, 81, 85, 200, 259, 314, 330, 381	49
9, 15, 16, 31, 37, 47, 54, 55, 63, 65, 74, 80, 83, 86, 105, 109, 126, 130, 148, 156, 165, 168, 172, 179, 180, 195, 213, 214, 218, 222, 242, 255, 257, 276, 303, 306, 317, 318, 319, 325, 329, 342, 345, 346, 353, 358, 362, 364, 377, 383, 384	50
10, 14, 17, 19, 20, 23, 25, 33, 34, 36, 39, 40, 43, 46, 52, 53, 72, 73, 75, 78, 79, 82, 88, 90, 95, 99, 104, 110, 113, 115, 117, 118, 119, 123, 125, 129, 132, 133, 136, 140, 146, 149, 150, 155, 157, 158, 162, 166, 170, 174, 185, 188, 189, 192, 193, 197, 199, 201, 205, 209, 210, 211, 212, 224, 232, 235, 238, 239, 240, 241, 246, 251, 258, 260, 262, 265, 267, 272, 283, 286, 293, 295, 296, 301, 302, 308, 309, 324, 334, 335, 343, 347, 352, 356, 357, 359, 363, 365, 371, 372, 385	51
3, 5, 7, 8, 22, 26, 27, 29, 38, 42, 45, 48, 57, 60, 62, 64, 68, 69, 71, 77, 92, 93, 94, 96, 98, 108, 111, 116, 120, 121, 124, 127, 128, 131, 134, 139, 141, 152, 154, 161, 163, 164, 173, 175, 181, 182, 183, 184, 186, 196, 202, 206, 207, 215, 216, 217, 219, 220, 221, 223, 225, 227, 229, 245, 253, 256, 263, 266, 271, 277, 288, 290, 291, 292, 294, 298, 299, 305, 307, 321, 322, 323, 326, 332, 349, 351, 354, 366, 367, 369, 370, 373, 375, 378, 379, 380, 382	52
6, 12, 13, 21, 58, 59, 61, 66, 70, 102, 107, 112, 114, 137, 138, 145, 151, 153, 169, 176, 177, 194, 198, 204, 228, 243, 244, 249, 250, 261, 268, 275, 280, 281, 285, 297, 313, 320, 331, 333, 340, 341, 344, 350, 361, 368, 386, 387	53
4, 18, 44, 49, 50, 97, 100, 101, 103, 142, 167, 178, 187, 191, 203, 226, 230, 231, 236, 273, 282, 284, 287, 304, 310, 311, 312, 328, 338, 355, 374, 388	54
24, 28, 30, 41, 56, 67, 87, 122, 135, 143, 147, 159, 160, 190, 208, 248, 252, 264, 269, 270, 279, 289, 300, 315, 339, 376	55
89, 106, 171, 247, 254, 278, 316, 327, 348, 360	56
11, 84, 91, 234, 237, 274	57
35, 144, 233, 337	58
51, 336	59

Fast Detection of the Exact Cases of x^y

On the *exact cases*, Ziv's iteration doesn't terminate.
$\rightarrow$ They need to be detected. With the knowledge of the worst case on some input subset $\mathbb{S}$, this can be done in a few cycles.

- Let $x = 2^E m$, $y = 2^F n$, $z = 2^G k$, where m, n, k are odd integers.
  $x^y = z \;\Leftrightarrow\; 2^{E \cdot 2^F n} m^{2^F n} = 2^G k \;\Leftrightarrow\; E \cdot 2^F n = G \;\wedge\; m^{2^F n} = k\,.$
  If $n < 0$, then $m = k = 1$ (trivial case). Now assume $m, n, k > 0$.

- If x^y is representable on 54 bits with $m \geq 3$, then either one has $y \in [\![1, 34]\!]$ or y is such that $F \in [\![-5, -1]\!] \;\wedge\; 1 \leq n \leq 33$.

- Worst case of $h_y(m) = m^y$ for these values of y and $m < 2^{53}$?
  Let $X = 1 + (m-1)/2^{53}$. We test $f_y(X) = \left(1 + (X-1) \cdot 2^{53}\right)^y$ in $[1, 2)$ split into sub-intervals (of small lengths for small X).

Search on the subset $\mathbb{S}$ ($y \notin \mathbb{Z}$, 80 values): 25 days on a small network.

Worst case:

$$x^y = 1110101111001110.0101001100001100010111001011000110001\,1\,\underbrace{0\ldots0}_{60\text{ zeros}}111\ldots$$

Algorithm for x^y (implemented before the worst case was known):

- Filter simple cases (e.g. $y = 2, 3, 4$). Ad-hoc computation.
- 1st approximation. Rounding OK with probability $\approx 1 - 2^{-7}$.
- 2nd approximation $z = x^y(1+\varepsilon)$ with $|\varepsilon| \leq 2^{-117}$.
  If $|\circ(z) - z| \geq 2^{-116}|z|$, rounding OK.
- $(x, y) \in \mathbb{S}$? If yes, exact case. If no, resume Ziv's iterations.

→ Ch. Lauter and V. Lefèvre, *An efficient rounding boundary test for* `pow(x,y)` *in double precision*, IEEE TC, 2008.