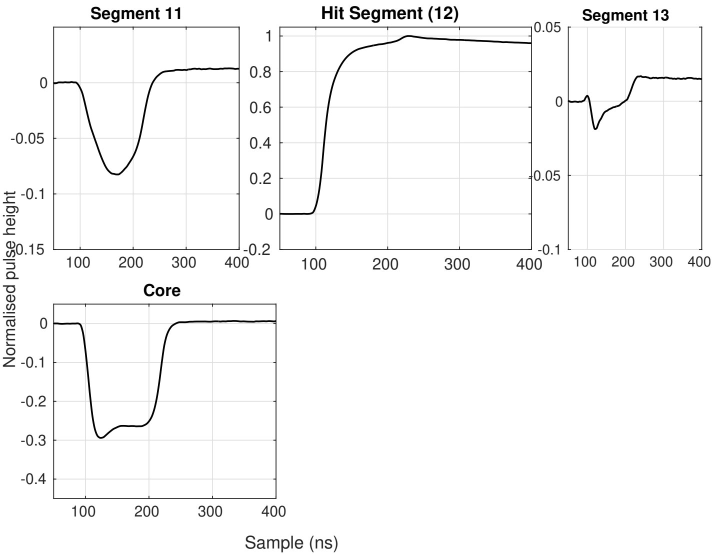
Vectorisation dans le PSA

Comment se vider un chargeur dans le pied avec une racine cubique ?

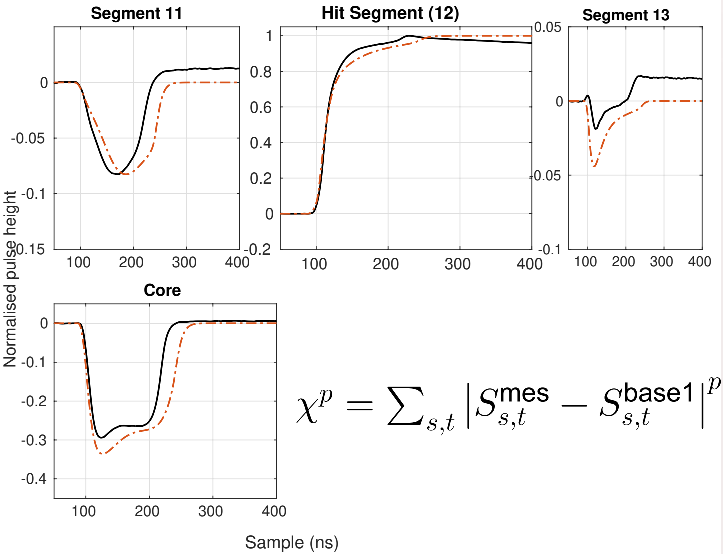
Roméo Molina Vincent Lafage

mardi 22 novembre 2022

PSA *Pulse Shape Analysis*

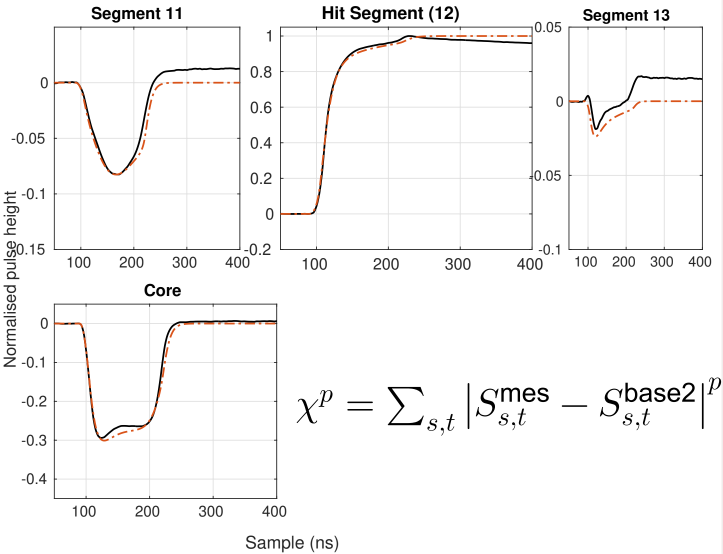


PSA *Pulse Shape Analysis*



$$\chi^p = \sum_{s,t} \left| S_{s,t}^{\text{mes}} - S_{s,t}^{\text{base1}} \right|^p$$

PSA *Pulse Shape Analysis*



$$\chi^p = \sum_{s,t} \left| S_{s,t}^{\text{mes}} - S_{s,t}^{\text{base2}} \right|^p$$

$p = 0.3$

$$x \mapsto x^p = x^{\frac{3}{10}} = \left(\sqrt[10]{x}\right)^3$$

$\Rightarrow$ fonction **algébrique** (racines de polynômes à coeff entiers)

- $x^{0.3} = \exp(0.3 \times \ln x)$ (on introduit une erreur sur 0.3)

  deux fonctions **transcendantes** élémentaires

  - … coûteuses
  - … dont l'**arrondi correct** n'est pas garanti

    The Table Maker's Dilemma

    pour avoir l'arrondi correct à n chiffres/bits… [1]

$$\exp(0.5091077534282133) = \underbrace{1.663806007261509}_{\text{16 digits}}\,\underbrace{5000000000000000}_{\text{16 digits}}\,49\ldots$$

$$\exp(0.7906867968553504) = \underbrace{2.204910231771509}_{\text{16 digits}}\,\underbrace{4999999999999999}_{\text{16 digits}}\,16\ldots$$

1. https://members.loria.fr/PZimmermann/wc/decimal32.html

$p = 0.3$

- Lookup Table
  - converti l'argument x en entier : $\mathtt{i} = \lfloor \mathtt{x} \rfloor$
  - renvoie la valeur de la table T indexée par cet entier : $x^p \sim \mathtt{T}[\lfloor \mathtt{x} \rfloor]$

$\Rightarrow$ perte de "continuité"$\Rightarrow$ perte de précision (surtout pour x petit)

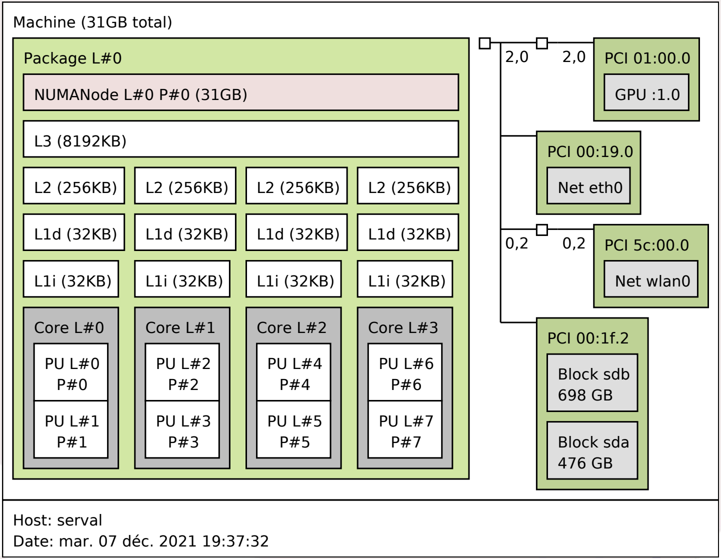
$\Rightarrow$ énorme empreinte mémoire : 2 Moctets

$\Rightarrow$ les valeurs pour un argument négatif sont stockées malgré la symétrie de la fonction : double l'empreinte mémoire !

$\Rightarrow$ indirection impossible à vectoriser (alors que le calcul pour $p = 2$ avait été vectorisé)

Know your tool

hwloc-ls



Alternatives pour l'évaluation

- `powf (x, p)`
- `expf (logf (x) * p)`
- `exp2f (log2f (x) * p)`
- `cbrtf (x)`
- cbrt maison

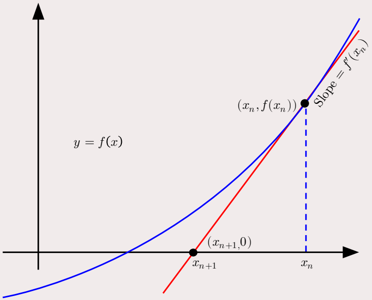
$$p = 0.3333333\cdots$$

$$a \mapsto z = a^p = a^{\frac{1}{3}} = (\sqrt[3]{a})$$

$\Rightarrow$ fonction **algébrique**

- `cbrt (a)` existe dans la `libm`
- ...mais au fait, comment extrait-on z=`cbrt (`a`)` ? $\Rightarrow$ zéro de $x \mapsto y = x^3 - a$

Extraire des racines



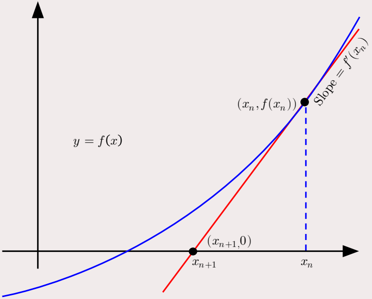
$$p = 0.3333333\cdots$$

$$a \mapsto z = a^p = a^{\frac{1}{3}} = (\sqrt[3]{a})$$

$\Rightarrow$ fonction **algébrique**

- `cbrt (a)` existe dans la `libm`
- ...mais au fait, comment extrait-on z=`cbrt` (a) ? $\Rightarrow$ zéro de $x \mapsto y = x^3 - a$

Extraire des racines



Newton $\qquad x_{n+1} = x_n - \dfrac{f(x_n)}{f'(x_n)}$

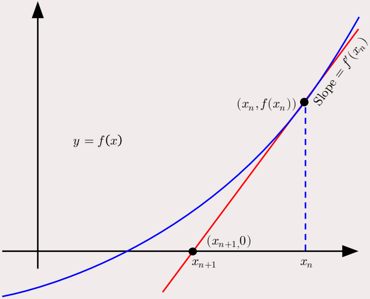
$$p = 0.3333333\cdots$$

$$a \mapsto z = a^p = a^{\frac{1}{3}} = (\sqrt[3]{a})$$

$\Rightarrow$ fonction **algébrique**

- `cbrt (a)` existe dans la `libm`
- ...mais au fait, comment extrait-on z=`cbrt` (a) ? $\Rightarrow$ zéro de $x \mapsto y = x^3 - a$

Extraire des racines



Newton	$x_{n+1} =$	$x_n - \dfrac{f(x_n)}{f'(x_n)}$
Halley	$x_{n+1} =$	$x_n - \dfrac{2f(x_n)f'(x_n)}{2[f'(x_n)]^2 - f(x_n)f''(x_n)}$

$p = 0.5$

- `sqrt (y)` existe dans la `libm` et dans le FPU
  garantie d'arrondi correct dans `IEEE-754`
- ...mais au fait, comment extrait-on x `=` `sqrt` (y) ?

$p = 0.5$

- `sqrt (y)` existe dans la `libm` et dans le FPU
  garantie d'arrondi correct dans `IEEE-754`
- …mais au fait, comment extrait-on x `=` `sqrt` (y) ?

  HÉRON $\qquad x_{n+1} = \frac{x_n + \frac{y}{x_n}}{2}$
- …et d'ailleurs, comment divise-t-on ?

$$p = 0.5$$

- `sqrt (y)` existe dans la `libm` et dans le FPU
  garantie d'arrondi correct dans `IEEE-754`
- ...mais au fait, comment extrait-on x `=` `sqrt` (y) ?

  HÉRON $\qquad x_{n+1} = \frac{x_n + \frac{y}{x_n}}{2}$
- ...et d'ailleurs, comment divise-t-on ?

inverse $\qquad x_{n+1} = \quad x_n \left[2 - y x_n\right]$

$$p = 0.5$$

- `sqrt (y)` existe dans la `libm` et dans le FPU
  garantie d'arrondi correct dans `IEEE-754`
- ...mais au fait, comment extrait-on x `= sqrt` (y) ?

  Héron $\quad x_{n+1} = \frac{x_n + \frac{y}{x_n}}{2}$
- ...et d'ailleurs, comment divise-t-on ?

inverse	$x_{n+1} = x_n \left[2 - y x_n\right]$
inverse `sqrt`	$x_{n+1} = \frac{x_n}{2} \left[3 - y x_n^2\right]$

$p = 0.5$

- `sqrt (y)` existe dans la `libm` et dans le FPU
  garantie d'arrondi correct dans `IEEE-754`
- ...mais au fait, comment extrait-on x `=` `sqrt` (y) ?

  HÉRON $\quad x_{n+1} = \frac{x_n + \frac{y}{x_n}}{2}$
- ...et d'ailleurs, comment divise-t-on ?

$$
\begin{array}{rll}
\text{inverse} & x_{n+1} = & x_n \left[2 - y x_n\right] \\
\text{inverse } \texttt{sqrt} & x_{n+1} = & \frac{x_n}{2} \left[3 - y x_n^2\right] \\
\texttt{cbrt}\ \text{NEWTON} & x_{n+1} = & \frac{1}{3}\left[2x_n - \frac{y}{x_n^2}\right] = x_n \frac{2x_n^3 - y}{3x_n^3}
\end{array}
$$

$$p = 0.5$$

- `sqrt (y)` existe dans la `libm` et dans le FPU
  garantie d'arrondi correct dans `IEEE-754`
- …mais au fait, comment extrait-on x `=` `sqrt` (y) ?

  HÉRON $\quad x_{n+1} = \frac{x_n + \frac{y}{x_n}}{2}$
- …et d'ailleurs, comment divise-t-on ?

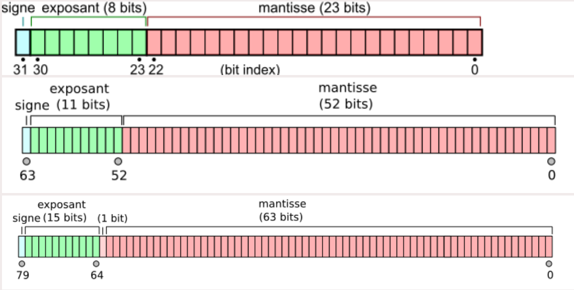
inverse	$x_{n+1} = x_n \left[2 - y x_n\right]$
inverse `sqrt`	$x_{n+1} = \frac{x_n}{2}\left[3 - y x_n^2\right]$
`cbrt` NEWTON	$x_{n+1} = \frac{1}{3}\left[2x_n - \frac{y}{x_n^2}\right] = x_n \frac{2x_n^3 - y}{3x_n^3}$
`cbrt` HALLEY	$x_{n+1} = x_n \frac{x_n^3 + 2y}{2x_n^3 + y}$

Flottants

Forme normalisée : mantisse, alias *normalized significand*

$\text{mantissa} \times \text{base}^{\text{exponent}}$ $\quad \text{mantissa} \in [1; \text{base}[, \text{exponent} \in \mathbb{Z}$

Astuce, en base 2 : le chiffre le plus significatif est forcément 1…



Racine maison

« Donnez-moi un point d'appui… »

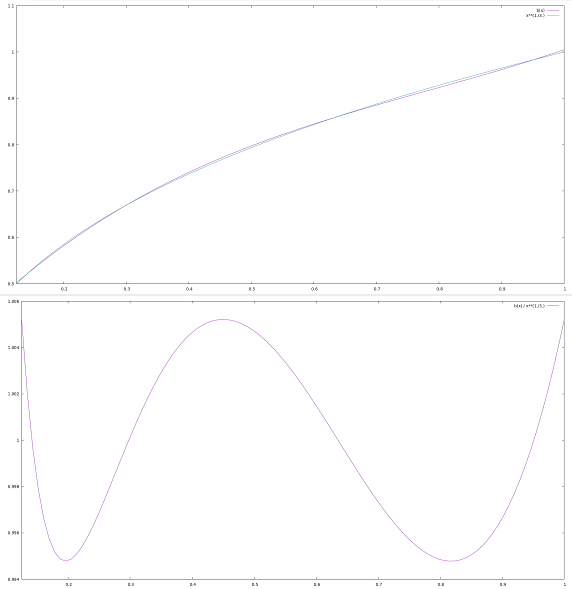
D'abord, on réduit l'intervalle d'étude :

$$\begin{array}{rcl}
\forall x \in \mathbb{F}, \exists (s = \pm 1, m \in [1, 2[, e \in \mathbb{Z})/x & = & s \times m \times 2^e \\
\forall x \in \mathbb{F}, \exists (s = \pm 1, m' \in [1/2, 1[, f \in \mathbb{Z}, g \in \{0, -1, -2\})/ \quad x & = & s \times m' \times 2^{(3f+g)} \\
\forall x \in \mathbb{F}, \exists (s = \pm 1, m' \in [1/2, 1[, f \in \mathbb{Z}, g \in \{0, -1, -2\})/ \quad \sqrt[3]{x} & = & s \times \sqrt[3]{m' \times 2^g} \times 2^f \\
m'' = m' \times 2^g & \in & [0.125, 1[
\end{array}$$

Meilleure approximation polynômiale d'ordre 3
($\textsc{Tchebychev}$ / algorithme de $\textsc{Remez}$)
Approximate cube root in $[0.125, 1)$ *with relative error 5.22e-3 i.e. 7.5 bits*

$$\begin{array}{rcl}
b_0 & = & 0.3408415318 \\
b_1 & = & 1.458112717 \\
b_2 & = & -1.385927796 \\
b_3 & = & 0.5921840668 \\
b(x) & = & b_0 + b_1 x + b_2 x^2 + b_3 x^3 \\
 & = & ((b_3 x + b_2)\, x + b_1)\, x + b_0
\end{array}$$

Racine maison



Vectorisation

Yapluka ?

```
gcc -Ofast -march=native -mtune=native -mavx2
    -fopenmp -fopenmp-simd -ftree-vectorize -fopt-info -ftree-vectorizer-verbose=1
    -mrecip --param=ssp-buffer-size=4
    -g -Wall -Wextra -Wshadow -Wconversion -Wpedantic
    speedcbrtf.c -o speedcbrtf -lm && taskset --cpu-list 0 ./speedcbrtf
```

Vectorisation

Est-ce que ça marche ?

```
objdump --disassemble=my_cbrtf speedcbrtf
```

```
0000000000401c30 <my_cbrtf>:
  401c30:    c5 f8 28 f0             vmovaps %xmm0,%xmm6
  401c34:    c5 fa 10 25 24 d5 00    vmovss  0xd524(%rip),%xmm4        # 40f160 <_IO_stdin_
  401c3b:    00
  401c3c:    c5 fa 10 15 1c d6 00    vmovss  0xd61c(%rip),%xmm2        # 40f260 <_IO_stdin_
  401c43:    00
  401c44:    c5 c8 54 fc             vandps %xmm4,%xmm6,%xmm7
  401c48:    c5 f9 7e f9             vmovd   %xmm7,%ecx
  401c4c:    c5 f9 7e f8             vmovd   %xmm7,%eax
  401c50:    c1 f9 17                sar     $0x17,%ecx
...
  401c92:    c4 e2 79 a9 15 c9 d5    vfmadd213ss 0xd5c9(%rip),%xmm0,%xmm2    # 40f264 <
  401c99:    00 00
  401c9b:    c4 e2 79 a9 15 c4 d5    vfmadd213ss 0xd5c4(%rip),%xmm0,%xmm2    # 40f268 <
  401ca2:    00 00
  401ca4:    c4 e2 79 a9 15 bf d5    vfmadd213ss 0xd5bf(%rip),%xmm0,%xmm2    # 40f26c <
  401cab:    00 00
  401cad:    c5 ea 59 ea             vmulss %xmm2,%xmm2,%xmm5
  401cb1:    c5 f8 28 ca             vmovaps %xmm2,%xmm1
  401cb5:    c4 e2 79 9d cd          vfnmadd132ss %xmm5,%xmm0,%xmm1
  401cba:    c5 f2 59 da             vmulss %xmm2,%xmm1,%xmm3
  401cbe:    c5 ea 58 ca             vaddss %xmm2,%xmm2,%xmm1
  401cc2:    c4 e2 79 99 e9          vfmadd132ss %xmm1,%xmm0,%xmm5
  401cc7:    c5 d2 53 cd             vrcpss %xmm5,%xmm5,%xmm1
  401ccb:    c5 f2 59 ed             vmulss %xmm5,%xmm1,%xmm5
  401ccf:    c5 f2 59 ed             vmulss %xmm5,%xmm1,%xmm5
  401cd3:    c5 f2 58 c9             vaddss %xmm1,%xmm1,%xmm1
```

MAQAO

« Abandonne tout espoir toi qui entre ici... »

http://maqao.org/

MAQAO (Modular Assembly Quality Analyzer and Optimizer) is a performance analysis and optimization framework operating at binary level with a focus on core performance.

MAQAO mixes both dynamic and static analyses based on its ability to reconstruct high level structures such as functions and loops from an application binary. [2]

2. Merci à Pierre AUBERT du LAPP, expert vectorisation du groupe Reprises

MAQAO

« La confiance n'exclut pas le contrôle »

```
maqao.intel64 cqa fct-loops=my_cbrtf conf=hint,expert --output-format=html --follow-calls=
```

```
Section 2.1.1: Binary loop #13
==============================

The loop is defined in:
 - /usr/local/lafage0/IJCLab/Reprises/my_float_cuberoot.c:52-70,93-121
 - /usr/local/lafage0/IJCLab/Reprises/speedcbrtf.c:64


The related source loop is multi-versionned.
43% of peak computational performance is used (13.95 out of 32.00 FLOP per cycle (GFLOPS @ 1GHz))

Vectorization
-------------
Your loop is fully vectorized, using full register length.

All SSE/AVX instructions are used in vector version (process two or more data elements in vector registers


Execution units bottlenecks
---------------------------
Performance is limited by:
 - execution of FP add operations (the FP add unit is a bottleneck)
 - execution of FP multiply or FMA (fused multiply-add) operations (the FP multiply/FMA unit is a bottlene

By removing all these bottlenecks, you can lower the cost of an iteration from 19.50 to 17.00 cycles (1.15

Workaround(s):
 - Reduce the number of FP add instructions
 - Reduce the number of FP multiply/FMA instructions
```

MAQAO

« La confiance n'exclut pas le contrôle »

```
...
VFMADD213PS %YMM1,%YMM4,%YMM7
VBROADCASTSS 0x2a9ea(%RIP),%YMM1
VFMADD213PS %YMM1,%YMM4,%YMM7
VBROADCASTSS 0x2a9e0(%RIP),%YMM1
VFMADD213PS %YMM1,%YMM4,%YMM7
VMULPS %YMM7,%YMM7,%YMM1
VMULPS %YMM7,%YMM1,%YMM1
VBROADCASTSS 0x2a9ce(%RIP),%YMM5
VMOVAPS %YMM5,%YMM12
VFMADD213PS %YMM4,%YMM1,%YMM12
VRCPPS %YMM12,%YMM6
VPADDD %YMM15,%YMM9,%YMM13
VSUBPS %YMM1,%YMM4,%YMM1
VMOVUPS 0x42d3a0,%YMM11
VFMSUB213PS %YMM11,%YMM6,%YMM12
VMULPS %YMM6,%YMM1,%YMM1^^I1^^I0.50
VBROADCASTSS 0x2a98c(%RIP),%YMM6
VBROADCASTSS 0x2a987(%RIP),%YMM3
VFMADD213PS %YMM3,%YMM13,%YMM6
VBROADCASTSS 0x2a97d(%RIP),%YMM9
VFMADD213PS %YMM9,%YMM13,%YMM6
...
```

- `V` Vector instruction...
- `SS` ...Scalar Single precision
- `PS` ...Packed Single precision

Vectorisation

Pourquoi ça a résisté ?

```
inline double my_cbrt (double a)
{
    double r;
    if ((a == 0.0) || isinf(a) || isnan(a)) {
        /* handle special cases */
        r = a + a;
    } else {
        /* strip off sign-bit */
        double b = fabs (a);
        /* compute exponent adjustments */
        int e;
        b = frexp (b, &e);
        int s = e - 3*342;
        int f = s / 3;
        s = s - 3 * f;
        f = f + 342;
        /* map argument into the primary approximation interval [0.125,1) */
        b = ldexp (b, s);
        const float bb = (float)b;
        /* approximate cube root in [0.125,1) with relative error 5.22e-3 */
        float uu =                0x1.2f32c0p-1f;
        uu = fmaf (uu, bb, -0x1.62cc2ap+0f);
        uu = fmaf (uu, bb,  0x1.7546e0p+0f);
        uu = fmaf (uu, bb,  0x1.5d0590p-2f);
...
```

Interface

Table – *leaky abstraction* : standardiser l'interface

ieee_arithmetic	C / C++	Fortran'90	Ada
ieee_copy_sign (x, y)	copysign (d x, d y)	sign (x, y)	F'Copy_Sign (value, sign)
ieee_logb (x)	frexp (d x, i *exp)	exponent (x) fraction (x)	F'Exponent (x) F'Fraction (x)
ieee_scalb (x, i)	ldexp (d x, i exp) scalbn (d x, i exp)	set_exponent (x, i)	F'Scaling (x, adjustment)
ieee_next_after (x, y)	nextafter(d x, d y)	nearest (x, s)	F'Adjacent (x, towards)
		radix (x)	F'Machine_Radix
		epsilon (x)	F'Model_Epsilon
		precision (x)	
		digits (x)	
		range (x)	
			F'Machine_Mantissa
	std::numeric_limits	minexponent (x)	F'Machine_Emin
	std::numeric_limits	maxexponent (x)	F'Machine_Emax
		spacing (x)	
		rrspacing (x)	
ieee_rint (x)	nearbyint (d x) rint(d x)	nint (x)	F'Rounding (x)
	floor (d x)	floor (x)	F'Floor (x)
	ceil (d x)	ceiling (x)	F'Ceiling (x)
ieee_rem (x, y)			F'Remainder (x, y)

Vectorisation

Résultats

cycles CPU par itération	gcc	gcc vect	icc
powf (x, p)	49.67	52.26	4.56
expf (logf (x) * p)	26.90	8.27	2.80
exp2f (log2f (x) * p)	24.54	28.52	2.54
cbrtf (x)	47.27	48.48	5.68
cbrt maison	35.62	6.87	3.89

```
gcc -Ofast -march=native -mtune=native -mavx2
    -fopenmp -fopenmp-simd -ftree-vectorize -fopt-info -ftree-vectorizer-verbose=1
    -mrecip --param=ssp-buffer-size=4
    -g -Wall -Wextra -Wshadow -Wconversion -Wpedantic
    speedcbrtf.c -o speedcbrtf -lm && taskset --cpu-list 0 ./speedcbrtf
```

(avec des `double` et pas des `float`)

	`libm`	$\mathtt{exp}(\mathtt{ln}\, x/3)$	`new_cbrt`
$\mathtt{cbrt}\ (x^3) \neq x$	57.45 %	22.87 %	0
$\mathtt{cbrt}\ (x)^3 \neq x$	82.94 %	65.31 %	63.61%

Conclusion

- optimisez vos fonctions !
- testez la vitesse et la précision !
- tentez la vectorisation contemporaine !
- testez que vos compilateurs vectorisent bien !
- les types flottants sont mieux utilisés en tant qu'objets
  que les objets que l'on construit :
  personne n'utilise les accesseurs, sauf à construire de nouvelles méthodes.
- appelez vos développeurs préférés
  quand il y a une question de calcul,
  mais aussi quand il y a une question d'outils.
- *« Attention, ces calculs ont été réalisés par des professionnels,*
  ESSAYEZ de les reproduire chez vous ! »