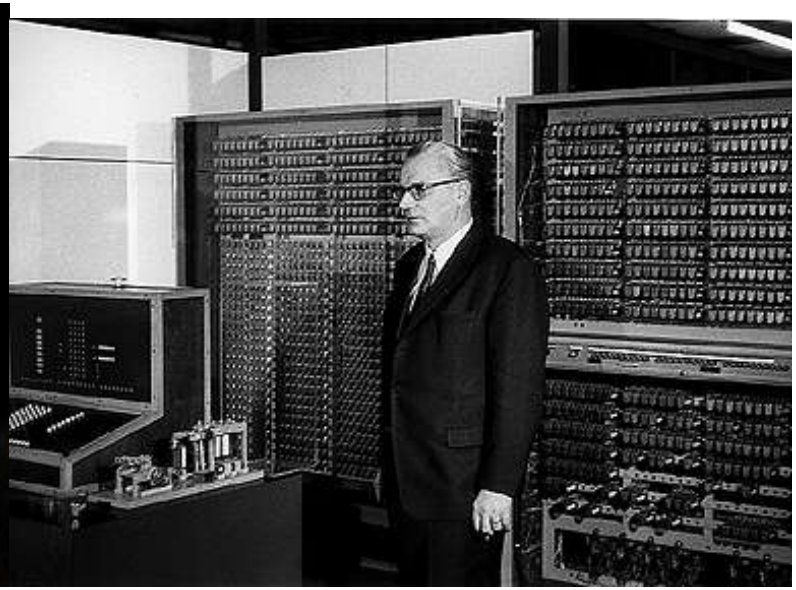
Calculs "exacts" avec une arithmétique approchée

Jean-Michel Muller
CNRS - Laboratoire LIP
Février 2008

http://perso.ens-lyon.fr/jean-michel.muller/

Virgule flottante ?

- babyloniens (absence de zéro, base 60, on n'écrit que les "mantisses") ;
- 1936-1938 : machine Z1 de Konrad Zuse (1910-1995) : base 2, mantisses de 16 bits et exposants de 7 bits. Mémoire de 16 nombres. Voir `http://www.epemag.com/zuse/`



Quelques chiffres

- besoins en dynamique ?

$$\frac{\text{Diamètre estimé de l'Univers}}{\text{Distance de Planck}} \approx 1.4 \times 10^{62}$$

Quelques chiffres

- besoins en dynamique ?

$$\frac{\text{Diamètre estimé de l'Univers}}{\text{Distance de Planck}} \approx 1.4 \times 10^{62}$$

- besoins en précision ? Certaines prédictions de la relativité générale et de la mécanique quantique vérifiées avec une précision relative d'environ 10^{-14}

Quelques chiffres

- besoins en dynamique ?

$$\frac{\text{Diamètre estimé de l'Univers}}{\text{Distance de Planck}} \approx 1.4 \times 10^{62}$$

- besoins en précision ? Certaines prédictions de la relativité générale et de la mécanique quantique vérifiées avec une précision relative d'environ 10^{-14}
- calculs intermédiaires : besoin de calculs en "quadruple précision", i.e., 128 bits (J. Laskar, Observatoire de Paris) pour stabilité à très long terme du système solaire ;

Quelques chiffres

- besoins en dynamique ?

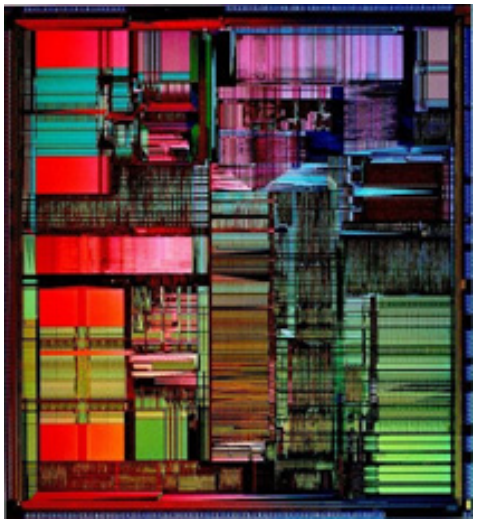
$$\frac{\text{Diamètre estimé de l'Univers}}{\text{Distance de Planck}} \approx 1.4 \times 10^{62}$$

- besoins en précision ? Certaines prédictions de la relativité générale et de la mécanique quantique vérifiées avec une précision relative d'environ 10^{-14}
- calculs intermédiaires : besoin de calculs en "quadruple précision", i.e., 128 bits (J. Laskar, Observatoire de Paris) pour stabilité à très long terme du système solaire ;
- record actuel : 1241 milliards de chiffres décimaux de π (Kanada, 2002), en utilisant les deux formules

$$\begin{aligned}\pi &= 48 \arctan \tfrac{1}{49} + 128 \arctan \tfrac{1}{57} - 20 \arctan \tfrac{1}{239} + 48 \arctan \tfrac{1}{110443} \\ &= 176 \arctan \tfrac{1}{57} + 28 \arctan \tfrac{1}{239} - 48 \arctan \tfrac{1}{682} + 96 \arctan \tfrac{1}{12943}\end{aligned}$$

On sait faire du très mauvais travail...

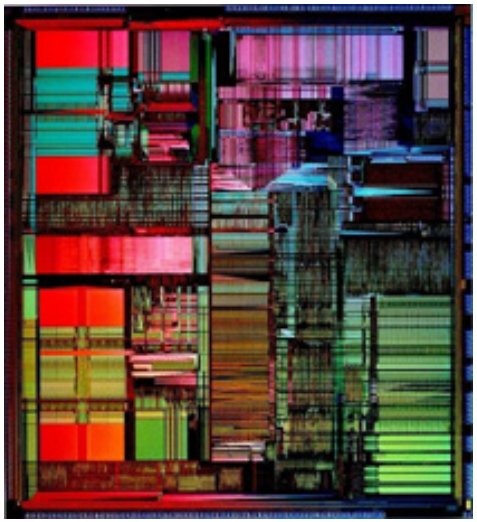
- Division du Pentium 1 : résultat faux dans 1 cas sur 4×10^{10} (en simple précision). Le calcul de 8391667/12582905 donnait $0.666869\cdots$ au lieu de $0.666910\cdots$. Erreur dans l'algorithme lui-même, pas dans son implantation ;



- Excel 2007, tapez $65536 - 2^{-37}$, vous obtiendrez

On sait faire du très mauvais travail...

- Division du Pentium 1 : résultat faux dans 1 cas sur 4×10^{10} (en simple précision). Le calcul de 8391667/12582905 donnait $0.666869\cdots$ au lieu de $0.666910\cdots$. Erreur dans l'algorithme lui-même, pas dans son implantation ;



- Excel 2007, tapez $65536 - 2^{-37}$, vous obtiendrez 100001.

On sait faire du très mauvais travail. . .

- dans la version 7.0 de Maple, si l'on calcule

$$\frac{5001!}{5000!}$$

on obtient 1 au lieu de 5001 ;

On sait faire du très mauvais travail. . .

- dans la version 7.0 de Maple, si l'on calcule

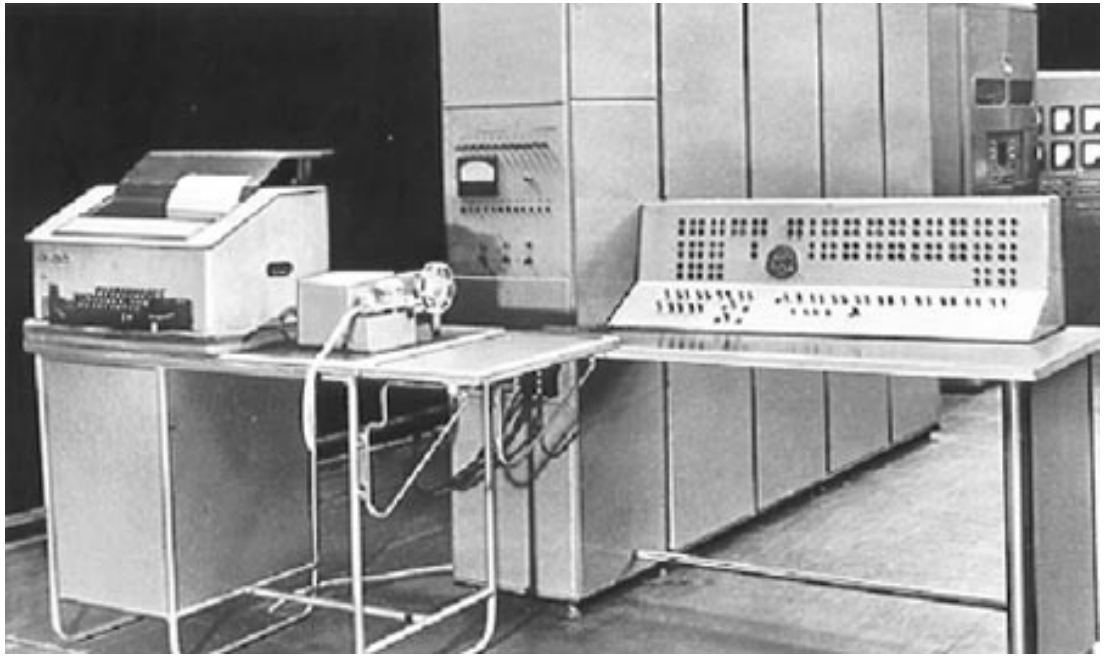
$$\frac{5001!}{5000!}$$

  on obtient 1 au lieu de 5001 ;
- dans la version 6.0, si on entre

$$214748364810$$

  on obtient 10.

Beaucoup de bizzareries



- Machine Setun, université de Moscou, 1958. 50 exemplaires ;
- base 3 et chiffres -1, 0 et 1. Nombres sur 18 « trits » ;
- idée : base β, nombre de chiffres n, $+$ grand nombre représenté M. Mesure du « coût » : $\beta \times n$.
- minimiser $\beta \times n$ sachant que $\beta^n \approx M$. Si variables réelles, optimum $\beta = e = 2.718\ldots \approx 3$.

La société cahotique de banque

- je dépose $e - 1$ euros sur mon compte ;

La société cahotique de banque

- je dépose $e - 1$ euros sur mon compte ;
- 1ère année : on multiplie mon avoir par 1, et on prélève 1 euro de frais de gestion ;

La société cahotique de banque

- je dépose $e - 1$ euros sur mon compte ;
- 1ère année : on multiplie mon avoir par 1, et on prélève 1 euro de frais de gestion ;
- 2ème année : on multiplie mon avoir par 2, et on prélève 1 euro de frais de gestion ;

La société cahotique de banque

- je dépose $e - 1$ euros sur mon compte ;
- 1ère année : on multiplie mon avoir par 1, et on prélève 1 euro de frais de gestion ;
- 2ème année : on multiplie mon avoir par 2, et on prélève 1 euro de frais de gestion ;
- 3ème année : on multiplie mon avoir par 3, et on prélève 1 euro de frais de gestion ; . . .

La société cahotique de banque

- je dépose $e - 1$ euros sur mon compte ;
- 1ère année : on multiplie mon avoir par 1, et on prélève 1 euro de frais de gestion ;
- 2ème année : on multiplie mon avoir par 2, et on prélève 1 euro de frais de gestion ;
- 3ème année : on multiplie mon avoir par 3, et on prélève 1 euro de frais de gestion ; . . .
- $n$ème année : on multiplie mon avoir par n, et on prélève 1 euro de frais de gestion.

La société cahotique de banque

- je dépose $e - 1$ euros sur mon compte ;
- 1ère année : on multiplie mon avoir par 1, et on prélève 1 euro de frais de gestion ;
- 2ème année : on multiplie mon avoir par 2, et on prélève 1 euro de frais de gestion ;
- 3ème année : on multiplie mon avoir par 3, et on prélève 1 euro de frais de gestion ; . . .
- $n$ème année : on multiplie mon avoir par n, et on prélève 1 euro de frais de gestion.
- *Quel est mon avoir au bout de* 25 *ans ?*

```
#include <stdio.h>

int main(void)
{
  double account = 1.71828182845904523536028747135;
  int i;
  for (i = 1; i <= 25; i++)
    {
      account = i*account - 1;
    }
  printf("Vous avez %1.17e euros\n", account);
}
```

- Processeur Intel Xeon et compilateur gcc sous Linux :

  `1.20180724741044855e+09`

- Bonne réponse :

```
#include <stdio.h>

int main(void)
{
  double account = 1.71828182845904523536028747135;
  int i;
  for (i = 1; i <= 25; i++)
    {
      account = i*account - 1;
    }
  printf("Vous avez %1.17e euros\n", account);
}
```

- Processeur Intel Xeon et compilateur gcc sous Linux :

  `1.20180724741044855e+09`

- Bonne réponse : environ 0.0399 euros.

Nombres Virgule Flottante

Etant donnés :

$$\begin{cases} \text{base} & \beta \geq 2 \\ \text{précision} & p \geq 1 \\ \text{plage d'exposants} & E_{\text{min}} \cdots E_{\text{max}} \end{cases}$$

Un nombre VF fini x est représenté par 2 entiers :

- mantisse entière : M, $|M| \leq \beta^p - 1$;
- exposant e, $E_{\text{min}} \leq e \leq E_{\text{max}}$.

tels que

$$x = M \times \beta^{e+1-p}.$$

On appelle mantisse réelle, ou mantisse de x le nombre

$$m = M \times \beta^{1-p},$$

de sorte que $x = m \times \beta^e$.

Représentation normalisée

Buts :

- représentation unique ;
- a priori la plus précise (3.142×10^0 vs. 0.003×10^3).

La représentation normalisée de x, si elle existe, est celle pour laquelle $1 \leq m < \beta$. C'est celle qui minimise l'exposant.
En base 2 le premier chiffre de mantisse d'un nombre normalisé est un "1" $\longrightarrow$ pas besoin de le mémoriser (convention du 1 implicite).
Un nombre sous-normal est un nombre de la forme

$$M \times \beta^{E_{\text{min}}+1-p}.$$

avec $|M| \leq \beta^{p-1} - 1$. Un tel nombre n'a pas de représentation normalisée.
Correspond à $\pm 0.xxxxxxxx \times \beta^{E_{\text{min}}}$.

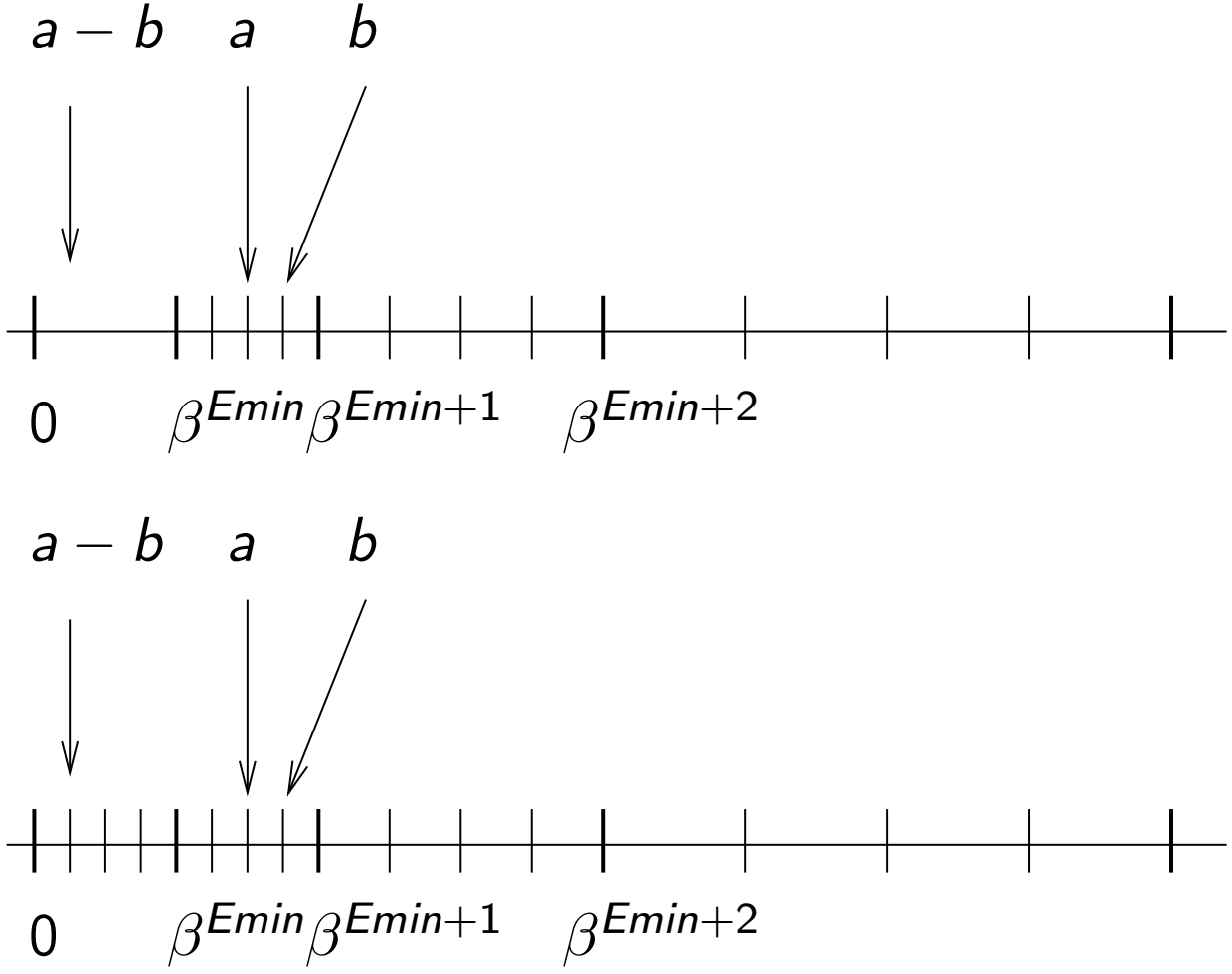


Fig.: *En haut : nombres VF normalisés. Dans cet ensemble, $a - b$ n'est pas représentable $\rightarrow$ le calcul $a - b$ donnera 0 en arrondi au plus près. En bas : on ajoute les sous-normaux.*

Système	β	p	$E_{\min}$	$E_{\max}$	+ grand représ.
DEC VAX	2	24	-128	126	$1.7\cdots \times 10^{38}$
(D format)	2	56	-128	126	$1.7\cdots \times 10^{38}$
HP 28, 48G	10	12	-500	498	$9.9\cdots \times 10^{498}$
IBM 370	16	6 (24 bits)	-65	62	$7.2\cdots \times 10^{75}$
et 3090	16	14 (56 bits)	-65	62	$7.2\cdots \times 10^{75}$
IEEE-754	2	23+1	-126	127	$3.4\cdots \times 10^{38}$
	2	52+1	-1022	1023	$1.8\cdots \times 10^{308}$
IEEE-754-R "binary 128"	2	112+1	-16382	16383	$1.2\cdots \times 10^{4932}$
IEEE-754-R "decimal 64"	10	16	-383	384	$9.999\cdots 9 \times 10^{384}$

Norme IEEE-754 (1985)

- mettre fin à un chaos (aucune portabilité, qualité très variable) ;
- moteur : W. Kahan (père de l'arithmétique de la HP35 et du coprocesseur Intel 8087) ;
- formats de représentation ;
- spécification des opérations, des conversions ;
- gestion des exceptions (max+1, 1/0, $\sqrt{-2}$, 0/0, etc.)

Arrondi correct

Definition (Arrondi correct)

L'utilisateur définit un *mode d'arrondi actif* parmi :

- arrondi au plus près (mode par défaut), càd vers le nombre VF le + proche. S'il y en a 2, on choisit celui dont la mantisse entière est paire.
- Arrondi vers $+\infty$, soit vers le plus petit nombre machine supérieur ou égal au résultat.
- Arrondi vers $-\infty$, soit vers le plus grand nombre machine inférieur ou égal au résultat.
- Arrondi vers zéro.

Une opération dont les entrées sont des nombres VF doit fournir ce qu'on obtiendrait si on avait d'abord calculé le résultat exactement, pour l'arrondir ensuite suivant le mode d'arrondi actif.

Arrondi correct

IEEE-754 (1985) : arrondi correct pour les 4 opérations arithmétiques ($+$, $-$, $\times$ et $\div$), $\sqrt{}$ et certaines conversions de format. Intérêts :

- si le résultat d'une opération portant sur 2 nombres VF est exactement représentable, c'est lui qu'on obtient ;
- tant qu'on se limite aux 4 opérations et à $\sqrt{}$ l'arithmétique est déterministe : $\rightarrow$ algorithmes et preuves qui utilisent les propriétés de cette "arithmétique machine" ;
- meilleures précision et portabilité des algorithmes numériques ;
- en jouant sur les modes d'arrondi vers $+\infty$ et vers $-\infty$, minorants et/ou majorants certains d'un résultat.

Approche "floue" $\rightarrow$ approche "exacte" : l'arithmétique VF est une structure qu'on peut étudier, et pas seulement une approximation du corps $\mathbb{R}$.

Un premier exemple

Lemme (Lemme de Sterbenz)

Soient a et b deux nombres VF positifs. Si

$$\frac{a}{2} \leq b \leq 2a$$

alors $a - b$ est exactement représentable en VF ($\rightarrow$ il est calculé sans erreur dans chacun des modes d'arrondi).

Preuve : élémentaire en se ramenant aux notations $x = M \times \beta^{e+1-p}$.

Erreur de l'addition flottante

Dans tout ce qui suit $\mathrm{RN}(x)$ désigne l'arrondi au plus près de x.

Lemme

Soient a et b deux nombres VF normalisés. On suppose que l'exposant de a est supérieur ou égal à celui de b. (est moins fort que d'imposer $|a| \geq |b|$). En l'absence de dépassement, si

$$s = RN(a + b)$$

et

$$r = (a + b) - s.$$

alors $|r| \leq |b|$ et r est un nombre VF.

Erreur de l'addition flottante

Démonstration :

1. s est le nombre VF le plus proche de $a+b$. Donc en particulier il est plus proche de $a+b$ que ne l'est a. Donc

$$|(a+b)-s| \leq |(a+b)-a|$$

donc

$$|r| \leq |b|.$$

2. notons $a = M_a \times \beta^{e_a-p+1}$ et $b = M_b \times \beta^{e_b-p+1}$, avec $|M_a|, |M_b| \leq \beta^p - 1$, et $e_a \geq e_b$. On a : $a+b$ est un multiple de β^{e_b-p+1}, donc s aussi, donc r aussi. On peut donc écrire

$$r = R \times \beta^{e_b-p+1}$$

or, $|r| \leq |b| \Rightarrow |R| \leq |M_b| \leq \beta^p - 1 \Rightarrow r$ est représentable exactement.

Obtenir r : Algorithme fast2sum (Dekker)

Base $\beta \leq 3$. A partir de 2 nombres VF normalisés a et b, avec $|a| \geq |b|$, $2a \leq MAX$, on effectue les calculs :

$$\begin{aligned} s &= \mathrm{RN}(a+b) \\ z &= \mathrm{RN}(s-a) \\ t &= \mathrm{RN}(b-z) \end{aligned}$$

qui correspondent au programme C :

```
s = a+b;
z = s-a;
t = b-z;
```

Bien entendu, s est le nombre VF le plus proche de $a+b$, mais de plus $s+t = a+b$ exactement. Donc $t = r$ contient l'erreur commise lors de l'addition de a et b.

Se méfier des compilateurs "optimisants".

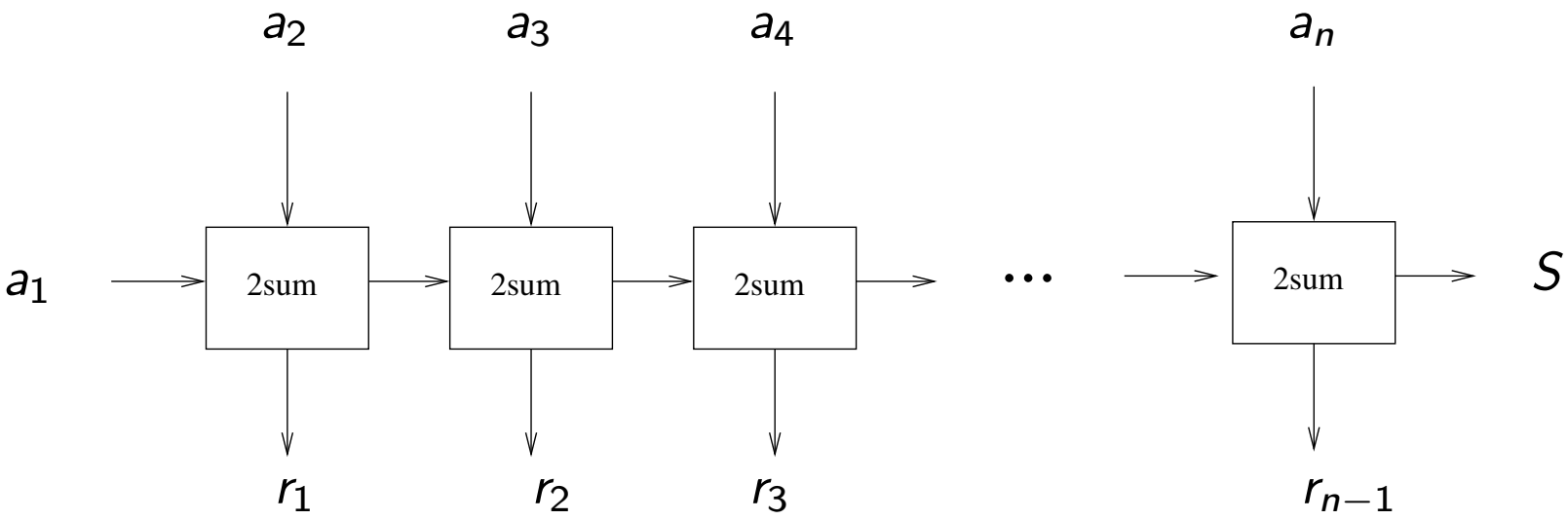
Généralisation

Si on n'a pas l'hypothèse exposant(a) $\geq$ exposant(b) on peut utiliser l'algorithme 2sum, dû à Knuth :

```
s = a + b;
b' = s - a;
a' = s - b';
deltab = b - b';
deltaa = a - a';
r = deltaa + deltab;
```

Même résultat que fast2sum. Trois opérations de plus, mais sur toutes les architectures modernes coûte moins cher que faire une comparaison.

Application : méthodes de "somme compensée"



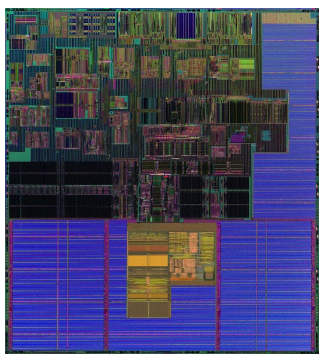
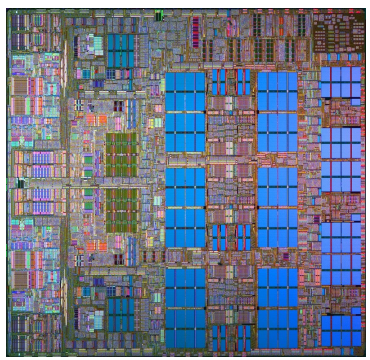
- $S + (r_1 + r_2 + \cdots + r_{n-1}) = a_1 + a_2 + \cdots + a_{n-1}$ exactement ;
- On calcule $S + (r_1 + r_2 + \cdots + r_{n-1})$ par additions VF usuelles. Si tous les a_i sont de même signe, et en double précision ($\beta = 2$, $p = 53$, arrondi au plus près) :

$$erreur \leq \left(\frac{1}{2} + n(n-1)2^{-54}\right) \text{ulp}(S)$$

$\longrightarrow$ additionner $\sqrt{2} \times 2^{26}$ termes avec une erreur $\leq$ poids du dernier bit.

Et avec des produits ?

- fma : *fused multiply-add*, calcule $\mathrm{RN}(ab+c)$. Sur Itanium et PowerPC. Evaluation de polynômes (Horner), produits scalaires : + rapide et en général + précis qu'avec $\pm$ et $\times$;
- si a et b sont des nombres VF, alors $r = ab - \mathrm{RN}(ab)$ est un nombre VF (élémentaire) ;
- s'obtient par $\begin{cases} p &= \mathrm{RN}(ab) \\ r &= \mathrm{RN}(ab-p) \end{cases}$
- sans fma, algorithme de Dekker : 17 opérations (7 $\times$, 10 $\pm$).

Itanium 2   PowerPC 5

Multiplication correctement arrondie par des constantes de précision arbitraire

- On veut calculer $\mathrm{RN}(Cx)$, où x est un nombre VF, et C une constante (i.e., connue à la compilation) réelle (qui n'est pas un nombre VF, sinon élémentaire).
- Valeurs typiques qui apparaissent dans des programmes numériques : π, $1/\pi$, $\ln(2)$, $\ln(10)$, e, $1/k!$, $B_k/k!$ (Euler-McLaurin), $\cos(k\pi/N)$ and $\sin(k\pi/N)$ (FFT), ...
- système de base 2 avec arrondi correct, précision p, disponibilité d'un fma

L'algorithme naïf

- on remplace C par $C_h = \mathrm{RN}(C)$;
- on calcule $\mathrm{RN}(C_h x)$ (instruction `y = Ch * x`).

p	Prop. de résultats correctement arrondis
5	0.93750
6	0.78125
7	0.59375
...	...
16	0.86765
17	0.73558
...	...
24	0.66805

Proportion de valeurs x pour lesquelles $\mathrm{RN}(C_h x) = \mathrm{RN}(Cx)$ pour $C = \pi$ et différentes valeurs de p.

L'algorithme proposé

On suppose pré-calculés les deux nombres VF

$$\begin{cases} C_h &= \mathrm{RN}(C), \\ C_\ell &= \mathrm{RN}(C - C_h), \end{cases} \tag{1}$$

Algorithme (MultC)

(Multiplication par C à l'aide d'un opérateur fma). *On calcule*

$$\begin{cases} u_1 &= RN(C_\ell x), \\ u_2 &= RN(C_h x + u_1). \end{cases} \tag{2}$$

Le résultat retourné est u_2.

Pour C et p donnés, on va chercher si l'algorithme MultC donne $\mathrm{RN}(Cx)$ pour tout nombre VF x.

Pour un C donné, l'algorithme est-il correct ?

Quelques remarques :

- Si l'algorithme MultC donne un résultat correct pour une constante C et un nombre VF x, il en sera de même pour $2^m C$ et $2^n x$.
- Si x est une puissance 2 ou si C est représentable exactement, ou si $C - C_h$ est une puissance de 2, alors $u_2 = \mathrm{RN}(Cx)$

$\Rightarrow$ Sans perte de généralité, on suppose que $1 < x < 2$ et $1 < C < 2$, que C n'est pas un nombre VF, et que $C - C_h$ n'est pas une puissance de 2.

Distance entre Cx et u_2

Propriété

Soient $x_{cut} = 2/C$ *et*

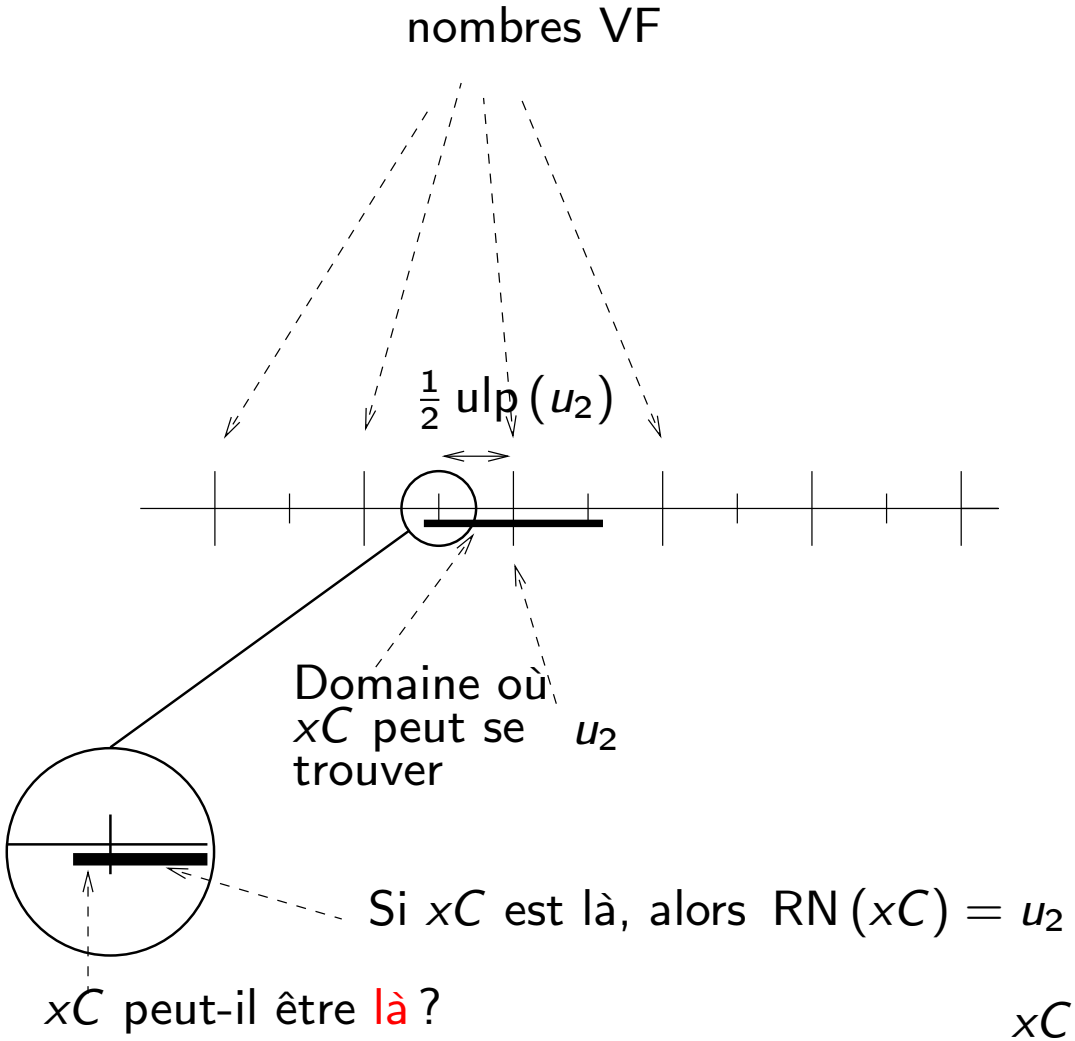
$$\epsilon_1 = |C - (C_h + C_\ell)| \tag{3}$$

- *Si* $x < x_{cut}$ *alors* $|u_2 - Cx| < 1/2\, ulp(u_2) + \eta$,
- *Si* $x \geq x_{cut}$ *alors* $|u_2 - Cx| < 1/2\, ulp(u_2) + \eta'$,

où

$$\begin{cases} \eta &= \frac{1}{2}\, ulp(C_\ell x_{cut}) + \epsilon_1 x_{cut}, \\ \eta' &= ulp(C_\ell) + 2\epsilon_1. \end{cases}$$

Analysons l'algorithme MultC



xC est à une distance $\leq 1/2\, ulp(u_2) + \eta$ de u_2. Si on montre que xC ne peut être à une distance $\leq \eta$ du milieu de 2 nombres VF consécutifs, alors $u_2 = RN(xC)$.

Remarques

- Si $x < x_{\text{cut}}$ alors $xC < 2$, donc le milieu de deux nombres VF consécutifs au voisinage de xC est de la forme $A/2^p$, où A est un entier impair compris entre $2^p + 1$ et $2^{p+1} - 1$.
- Si $x \geq x_{\text{cut}}$, alors le milieu de deux nombres VF consécutifs au voisinage de xC est de la forme $A/2^{p-1}$.

Petit rappel sur les fractions continues

Soit $\alpha \in \mathbb{R}$. On construit 2 suites (a_i) et (r_i) définies par :

$$\begin{cases} r_0 &= \alpha, \\ a_i &= \lfloor r_i \rfloor, \\ r_{i+1} &= 1/(r_i - a_i). \end{cases} \tag{4}$$

Si $\alpha \notin \mathbb{Q}$, ces suites sont définies $\forall i$, et le rationnel

$$\frac{p_i}{q_i} = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{a_3 + \cfrac{1}{\ddots + \cfrac{1}{a_i}}}}}$$

est la $i$ème réduite de α. Si $\alpha \in \mathbb{Q}$, s'arrête pour un i, et $p_i/q_i = \alpha$ exactement.

Petit rappel sur les fractions continues

Théorème (2)

Soient $(p_j/q_j)_{j\geq 1}$ les réduites de α. Pour tout (p,q), t.q. $q < q_{n+1}$, on a

$$|p - \alpha q| \geq |p_n - \alpha q_n|.$$

□

Théorème (3)

Soient p, q des entiers non nuls et premiers entre eux. Si

$$\left|\frac{p}{q} - \alpha\right| < \frac{1}{2q^2}$$

alors p/q est une réduite de α.

□

Méthode 1 : utilisation du théorème 1

Soient $X = 2^{p-1}x$ et $X_{\text{cut}} = \left\lfloor 2^{p-1}x_{\text{cut}} \right\rfloor$.
Cas $x < x_{\text{cut}}$: on cherche s'il existe A entre $2^p + 1$ et $2^{p+1} - 1$ t.q.

$$\left| Cx - \frac{A}{2^p} \right| < \eta.$$

Ceci équivaut à

$$|2CX - A| < 2^p \eta$$

Soient $(p_i/q_i)_{i \geq 1}$ les réduites de $2C$, et k le + petit entier t.q. $q_{k+1} > X_{\text{cut}}$, et soit $\delta = |p_k - 2Cq_k|$.
Théorème 1 $\Rightarrow \forall A, X \in \mathbb{Z}$, avec $0 < X \leq X_{\text{cut}}$, $|2CX - A| \geq \delta$.

Méthode 1 : utilisation du théorème 1

Par conséquent

1. Si $\delta \geq 2^p \eta$ alors $|Cx - A/2^p| < \eta$ est impossible $\Rightarrow$ l'algorithme MultC donne le bon résultat pour tout $x < x_{\text{cut}}$;
2. si $\delta < 2^p \eta$, on essaie MultC avec $x = q_k 2^{-p+1} \longrightarrow$ soit on obtient un contre-exemple, soit on ne peut conclure

Cas $x > x_{\text{cut}}$: étude similaire (avec les réduites de C)

Exemple 1 : C égal à π

```
> method1(Pi/2,53);
Ch = 884279719003555/562949953421312 = 1.570796327...
Cl = 4967757600021511/81129638414606681695789005144064
= .6123233996e-16...
xcut = 1.2732395447351626862, Xcut = 5734161139222658
eta = .8069505497e-32
pk/qk = 6134899525417045/1952799169684491
delta = .9495905771e-16
OK for X < 5734161139222658
etaprime = .1532072145e-31
pkprime/qkprime = 1205568675415943 8/767488855716784 7
deltaprime = .6943873667e-16
OK for 5734161139222658 <= X < 9007199254740992
```

$\Rightarrow$ On obtient toujours un résultat correct pour $C = 2^k\pi$, avec $C_h = 2^{k-48} \times 884279719003555$ et $C_\ell = 2^{k-105} \times 4967757600021511$.
Multiplier avec arrondi correct par π : une multiplication et un fma.

Autres exemples (double précision : $p = 53$)

Toujours en utilisant la même méthode :

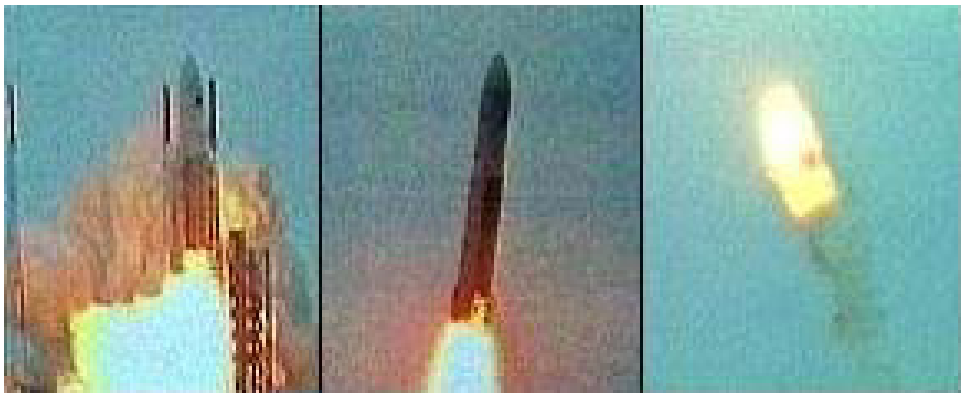
- pour $C = 2^k/\pi$, on trouve des contre-exemples, les nombres x de la forme
$$6081371451248382 \times 2^m.$$
- pour $C = \frac{2^k}{\ln 10}$, on ne sait pas conclure (une autre méthode permet de montrer que l'algorithme donne toujours un résultat correct) ;
- pour $C = 2^k \ln(2)$, l'algorithme donne toujours un résultat correct.

Programmes Maple et Pari implantant ces méthodes :
`http://perso.ens-lyon.fr/jean-michel.muller/MultConstant.html`

C	p	méthode 1	méthode 2	méthode 3
π	24	?	?	OK
π	53	OK	?	OK
π	64	?	OK	OK (c)
π	113	OK	OK	OK (c)
$1/\pi$	24	?	?	OK
$1/\pi$	53	contre-exemple 6081371451248382	?	seul contre-exemple $X = 6081371451248382$
$1/\pi$	64	OK	OK	OK (c)
$1/\pi$	113	?	?	OK
$\ln 2$	24	OK	OK	OK (c)
$\ln 2$	53	OK	?	OK (c)
$\ln 2$	64	OK	?	OK (c)
$\ln 2$	113	OK	OK	OK (c)
$\frac{1}{\ln 2}$	24	?	OK	OK (c)

Gestion des exceptions

- Nov. 1998, navire lance-missiles américain USS Yorktown, on a par erreur tapé un «zéro» sur un clavier → division par 0. Le programmeur n'avait pas songé que ce problème pourrait arriver → cascade d'erreurs → arrêt du système de propulsion.
- premier envol. . . et premier plongeon d'Ariane 5 (500 M$)



Vitesse horizontale de la fusée calculée sur des flottants 64 bits. Dans le programme du calculateur de bord, conversion vers un entier 16 bits. Il n'avait pas été envisagé que cette conversion puisse faire un dépassement. Fonctionnait très bien sur Ariane 4.

La norme IEEE 754 est en cours de révision

- les grandes lignes restent les mêmes ;
- ajouts fma, base 10, quelques considérations sur les fonctions élémentaires (sin, cos, exp, log, etc.) et leur arrondi correct ;
- voir `http://754r.ucbtest.org/`
- propositions de spécification de ces fonctions : `http://perso.ens-lyon.fr/jean-michel.muller/NumericalAlgorithms2004.pdf`

Les humains n'ont pas besoin des machines pour faire des bêtises

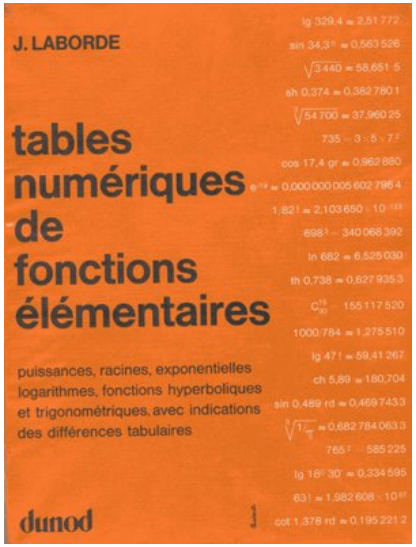
La sonde Mars Climate Orbiter s'est écrasée sur Mars en 1999.

Les humains n'ont pas besoin des machines pour faire des bêtises

La sonde Mars Climate Orbiter s'est écrasée sur Mars en 1999.

Une partie des concepteurs des logiciels supposait que l'unité de mesure était le mètre, et l'autre partie que c'était le pied.

Le dilemme du fabricant de tables



Considérons le nombre VF double précision ($p = 53$)

$$x = \frac{8520761231538509}{46116860184273879 04}$$

Nous avons

$$2^{53+x} = 9018742077413030.99999999999999999805240837303\cdots$$

A quoi ça sert ?

Pas de standardisation des fonctions élémentaires (Ng)

Système	$\sin 10^{22}$
résultat exact	$-0.852200849767188801772 7\cdots$
HP 48 GX	-0.852200849762
HP 700	0.0
HP 375, 425t (4.3 BSD)	$-0.65365288\cdots$
matlab V.4.2 c.1 pour Macintosh	0.8740
matlab V.4.2 c.1 pour SPARC	-0.8522
Silicon Graphics Indy	$0.87402806\cdots$
SPARC	$-0.8522008497671 8879$
IBM RS/6000 AIX 3005	$-0.852200849\cdots$
IBM 3090/600S-VF AIX 370	0.0
DECstation 3100	NaN
Casio fx-8100, fx180p, fx 6910 G	Error
TI 89	Trig. arg. too large

Arrondir correctement les fonctions élémentaires

- dans tout ce qui suit : base 2 ;
- Nombre VF x et entier m (avec $m > p$) $\rightarrow$ on peut calculer une approximation de $f(x)$ dont l'erreur sur la mantisse y est $\leq 2^{-m}$.
- Ce calcul peut être fait en employant un format plus large, ou en utilisant des algorithmes tels que Fast2Sum, Dekker, etc.
- obtenir un arrondi correct de $f(x)$ à partir de y ne sera pas possible si y est trop proche d'un point où l'arrondi change.

- en arrondi au plus près,

$$\underbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}1000000\cdots000000}_{m \text{ bits}}xxx\cdots$$

ou

$$\underbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}0111111\cdots111111}_{m \text{ bits}}xxx\cdots;$$

- avec les modes d'arrondi "dirigés",

$$\underbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}0000000\cdots000000}_{m \text{ bits}}xxx\cdots$$

ou

$$\underbrace{1.\underbrace{xxxxx\cdots xxx}_{p \text{ bits}}1111111\cdots111111}_{m \text{ bits}}xxx\cdots.$$

Trouver une valeur de m au-delà de laquelle le problème ne se pose pas ?

- fonction f : sin, cos, arcsin, arccos, tg, arctg, exp, log, sh, ch,
- Théorème de Lindemann (x algébrique non nul $\Rightarrow e^x$ transcendant) $\longrightarrow$ sauf pour les cas triviaux (e^0, $\ln(1)$, $\sin(0)$, $\arccos(1)$, etc.), lorsqu'on calcule $f(x)$ où x est un nombre VF, il existe une valeur de m, disons m_x, au delà de laquelle l'écriture binaire de $f(x)$ ne peut plus commencer comme au transparent précédent ;
- pour un format donné, nombre fini de nombres VF $\longrightarrow$ il existe un $m_{\max} = \max_x(m_x)$ pour lequel, pour tout x, arrondir l'approximation de $f(x)$ sur $m_{\max}$ bits est équivalent à arrondir $f(x)$;
- problème : ce raisonnement ne nous donne aucune idée de la valeur de $m_{\max}$. Pourrait être énorme, ce qui n'est pas le cas en pratique.

Une borne de Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, avec $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp \log p}$$

Application : Pour calculer ln et exp en double précision ($p = 53$), il suffit de calculer une approximation intermédiaire bonne sur environ

Une borne de Baker (1975)

- $\alpha = i/j$, $\beta = r/s$, avec $i, j, r, s < 2^p$;
- $C = 16^{200}$;

$$|\alpha - \log(\beta)| > (p2^p)^{-Cp \log p}$$

Application : Pour calculer ln et exp en double précision ($p = 53$), il suffit de calculer une approximation intermédiaire bonne sur environ

$$10^{244} \text{ bits}$$

Un peu de cuisine...

- la mantisse réelle y de $f(x)$ est de la forme :

$$y = y_0.y_1y_2\cdots y_{p-1}\overbrace{01111111\cdots11}^{kbits}xxxxx\cdots$$

ou

$$y = y_0.y_1y_2\cdots y_{p-1}\overbrace{10000000\cdots00}^{kbits}xxxxx\cdots$$

avec $k \geq 1$.

- En supposant qu'après la $p^{\text{ème}}$ position les "1" et les "0" apparaissent de manière équiprobable, la "probabilité" d'avoir $k \geq k_0$ est 2^{1-k_0} ;
- si on a N nombres virgule flottante en entrée, on observera environ $N \times 2^{1-k_0}$ valeurs pour lesquelles $k \geq k_0$;
- en pratique, le phénomène ne se produit donc plus dès que k_0 est significativement plus grand que $\log_2(N)$ (pour une seule valeur de l'exposant, dès que $k_0 >> p$).

Recherche des “pires cas” en double précision

- algorithmes de V. Lefèvre : basés sur une linéarisation des fonctions pour un préfiltrage ne laissant qu'un nombre raisonnable de cas à tester avec grande précision ;
- préfiltrage : variante de l'algorithme d'Euclide et utilisation du théorème des 3 distances ;
- double précision :
  - la plus utilisée ;
  - calculer tous les sinus des 2^{32} nombres simple-précision ne demande que quelques heures ;
  - plus hautes précisions : le nombre de cas à traiter (2^{128} en quadruple précision) est inatteignable dans un futur prévisible.

Tab.: *Pires cas pour l'exponentielle de nombres VF double précision.*

Intervalle	pire cas (binaire)
$[-\infty, -2^{-30}]$	$\exp(-1.11101101001100011000111011111011011000100111111101010 \times 2^{-27})$ $= 1.11111111111111111111111100\cdots0111000100\quad 1\quad 1^{59}0001\ldots \times 2^{-1}$
$[-2^{-30}, 0)$	$\exp(-1.00000000000000000000000000000000000000000000000000001 \times 2^{-51})$ $= 1.11111111111111\cdots1111111111111100\quad 0\quad 0^{100}1010\ldots \times 2^{-1}$
$(0, +2^{-30}]$	$\exp(1.11111111111111111111111111111111111111111111111111111 \times 2^{-53})$ $= 1.00000000000000000000000000000000000000000000000000000\quad 1\quad 1^{104}0101\ldots$
$[2^{-30}, +\infty]$	$\exp(1.01111111111111100111111111111111011100000000000100100 \times 2^{-32})$ $= 1.00000000000000000000000000000001011111111111111101000\quad 0\quad 0^{57}1101\ldots$ $\exp(1.10000000000000010111111111111111011011111111111011100 \times 2^{-32})$ $= 1.00000000000000000000000000000001100000000000000010111\quad 1\quad 1^{57}0010\ldots$ $\exp(1.10011110100111001011101111111110101100000100000001011 \times 2^{-31})$ $= 1.00000000000000000000000000000001100111101001110010111\quad 1\quad 0^{57}1010\ldots$ $\exp(110.00001111010100101111001101111010111011001111110100)$ $= 110101100.01010000101101000000100111001000101011101110\quad 0\quad 0^{57}1000\ldots$

Tab.: *Pires cas pour le logarithme népérien de nombres VF double précision.*

Interval	worst case (binary)
$[2^{-1074}, 1)$	$\log(1.1110101001110001110110000101110011101110000000100000 \times 2^{-509})$ $= -101100000.00101001011010100110011010110100001011111111 \quad 1 \quad 1^{60}0000\ldots$
	$\log(1.1001010001110110111000110000010011001101011111000111 \times 2^{-384})$ $= -100001001.10110110000011001010111101000111101100110101 \quad 1 \quad 0^{60}1010\ldots$
	$\log(1.0010011011101001110001001101001100100111100101100000 \times 2^{-232})$ $= -10100000.10101011001011000010010111100110100001000010 \quad 0 \quad 0^{60}1001\ldots$
	$\log(1.0110000100111001010101011101110010000000001011111000 \times 2^{-35})$ $= -10111.11110000001011111001101110101111011000000110101 \quad 0 \quad 1^{60}0011\ldots$
$(1, 2^{1024}]$	$\log(1.0110001010101000100001100001001101100010100110110110 \times 2^{678})$ $= 111010110.01000111100111101011101001111100100101110001 \quad 0 \quad 0^{64}1110\ldots$

Tab.: *Pire cas pour le cosinus de nombres VF double précision pris dans* $[1/64, 12867/8192]$. $12867/8192$ *est légèrement inférieur à* $\pi/2$.

Interval	worst case (binary)
$[1/64, 1]$	$\cos(1.1001011111100110011010011111010010110001000011100011111 \times 2^{-\mathbf{6}})$ $= 0.1111111111110101110110011\cdots 111000010101000 \quad 1 \quad 1^{\mathbf{55}}0111\ldots$
$[1, \frac{\mathbf{12867}}{\mathbf{8192}}]$	$\cos(1.0110101110001010011000100111001111010111110000100001)$ $= 1.00110011011111111100010\cdots 10110001010010 \quad 1 \quad 0^{\mathbf{54}}1011\ldots \times 2^{-\mathbf{3}}$

Un peu de multi-précision

Itération arithmético-géométrique (Gauss-Legendre)

$$\begin{cases} a_{n+1} & = & \dfrac{a_n + b_n}{2} \\ b_{n+1} & = & \sqrt{a_n b_n}. \end{cases}$$

convergence quadratique vers limite commune $A(a_0, b_0) =$ moyenne arithmético-géométrique de a_0 et b_0.
Gauss :

$$A(1, x) = \frac{\pi}{2F(x)},$$

où

$$F(x) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1 - (1 - x^2)\sin^2\theta}}.$$

A quoi ça sert ?

Donne un algorithme de calcul rapide de

$$F(x) = \int_0^{\pi/2} \frac{d\theta}{\sqrt{1-(1-x^2)\sin^2\theta}}.$$

or,

$$\begin{aligned} F(4/s) &= \ln(s) + \frac{4\ln(s)-4}{s^2} \\ &\quad + \frac{36\ln(s)-42}{s^4} + \frac{1200\ln(s)-1480}{3s^6} \\ &\quad + O\left(\frac{1}{s^8}\right), \end{aligned}$$

si s est suffisament grand, $F(4/s)$ est une bonne approximation de $\ln(s) \longrightarrow$ pour les très grandes précisions (milliers de chiffres), donne algorithme rapide de calcul du logarithme. Exponentielle : méthode de Newton. Contraire de ce qu'on fait pour les précisions moyennes (centaines de chiffres).

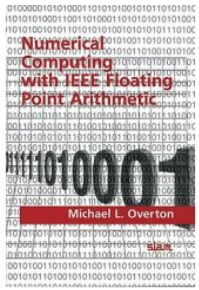
Quelques logiciels utiles

- CRLIBM : fonctions mathématiques avec arrondi correct.
  `http://lipforge.ens-lyon.fr/projects/crlibm/`
  La documentation qui explique les méthodes utilisées est à
  `http://lipforge.ens-lyon.fr/frs/download.php/41/crlibm-0.10.pdf`
- GAPPA : outil de vérifications de propriétés VF (p.ex. bornes) et de génération de preuves formelles.
  `http://lipforge.ens-lyon.fr/www/gappa/`
- MPFR : arithmétique multi-précision avec arrondi correct.
  `http://www.mpfr.org/`
- MPFI (basé sur MPFR) : arith. d'intervalles multi-précision.
  `http://perso.ens-lyon.fr/nathalie.revol/software.html`
- PARI/GP : calculs rapides en arithmétique (factorisations, théorie algébrique des nombres, courbes elliptiques...).
  `http://pari.math.u-bordeaux.fr/`

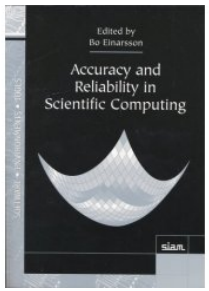
La virgule flottante sur le web

- le site de W. Kahan (père de la norme IEEE 754, de l'arithmétique du 8087 et de la HP35) :
  `http://http.cs.berkeley.edu/~wkahan/`
- le site de D. Hough sur la révision de la norme
  `http://www.validlab.com/754R/`
- l'article de Goldberg "What every computer scientist should know about Floating-Point arithmetic"
  `http://www.validlab.com/goldberg/paper.pdf`
- l'équipe Arénaire du LIP (ENS Lyon)
  `http://www.ens-lyon.fr/LIP/Arenaire/`
- l'équipe Algorithmique numérique et parallélisme du LIP6 (Paris 6)
  `http://www-anp.lip6.fr/`
- l'équipe SPACES du Loria (Nancy)
  `http://www.loria.fr/equipes/spaces/`
- ma propre page
  `http://perso.ens-lyon.fr/jean-michel.muller/`

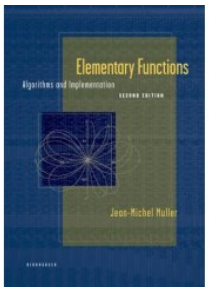
Quelques livres sur la virgule flottante



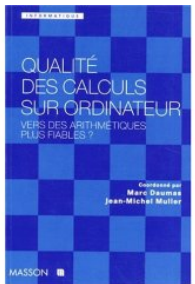
Michael Overton
Numerical Computing with IEEE Floating Point Arithmetic
Siam, 2001



Bo Einarsson
Accuracy and Reliability in Scientific Computing
Siam, 2005



Jean-Michel Muller
Elementary Functions, algorithms and implementation, 2ème édition
Birkhauser, 2006



Marc Daumas et al.
Qualité des calculs sur ordinateur
Masson, 1997.