Le mauvais résultat tout de suite, ou le bon résultat trop tard ?

Jean-Michel Muller
CNRS - Laboratoire LIP (CNRS, ENSL, INRIA, Univ. Claude Bernard)

11 mars 2004

Arithmétique des ordinateurs...

- systèmes de numération et algorithmes arithmétiques ;
- implantations logicielles et matérielles ;
- mots-clés : vitesse, précision, fiabilité, surface des circuits, consommation/dissipation d'énergie

$\Rightarrow$ solutions de compromis, qui diffèreront suivant les applications.

Le mot “ordinateur” dans “Arithmétique des ordinateurs” est peut-être malheureux

- Babylone : système de numération de base 60 dont on se sert encore ;
- algorithme sumérien de racine carrée (Fowler et Robson, 1998) : itération

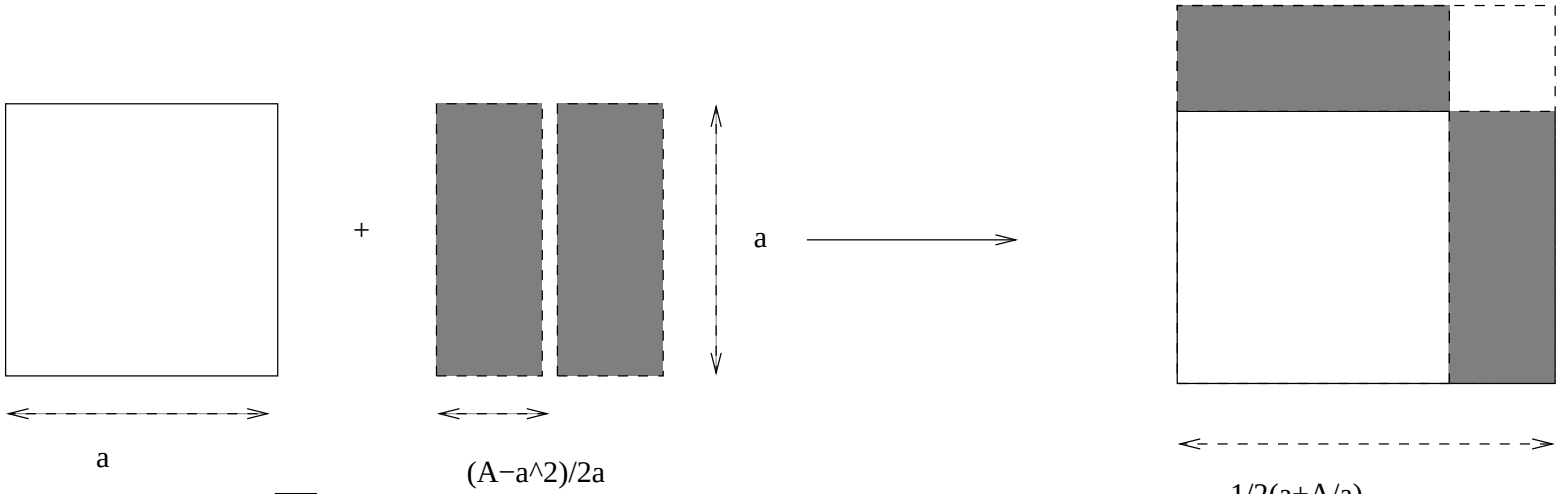
$$x_{n+1} = \frac{1}{2}\left(x_n + \frac{A}{x_n}\right)$$

de calcul de $\sqrt{A}$, communément attribuée à Héron d’Alexandrie (env. 10 – env. 75), voire à Newton, puisqu’il s’agit de l’itération de Newton-Raphson pour trouver les zéros de $f(x) = x^2 - A$.

La tablette de Yale

- entre -2000 et -1600 ;
- $\sqrt{2}$ en base 60 ;
- précision de 4 chiffres de base $60 \approx 7$ chiffres de base 10.

Racine carrée babylonienne



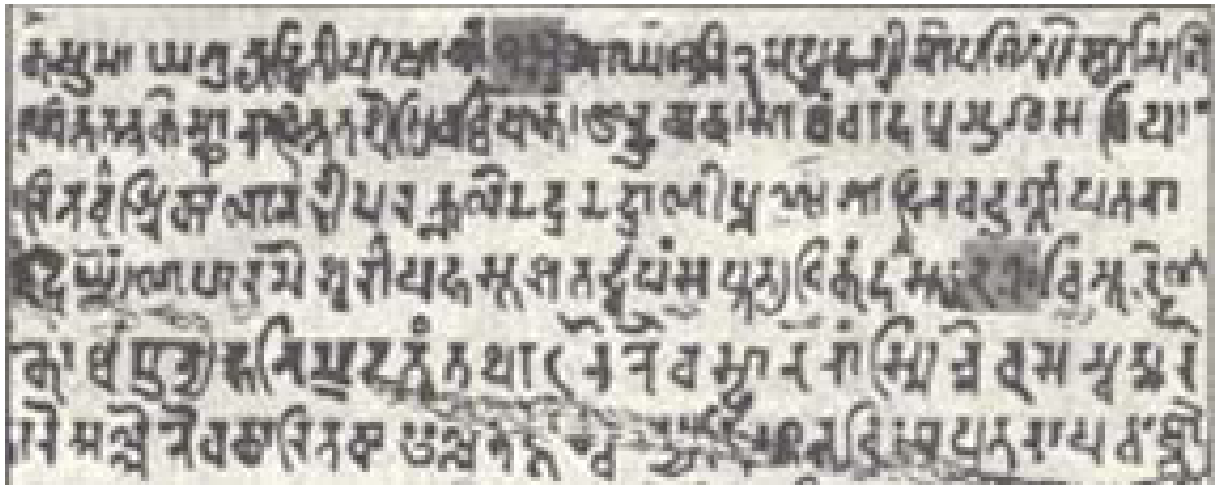
On cherche $\sqrt{A}$. Approximation a, telle que $a^2 < A$. Premier carré, de côté a et surface $a^2 < A$. Nouveau carré, de surface plus proche de A : adjonction de 2 rectangles de longueur a et de largeur $(A - a^2)/(2a)$.

Côté du nouveau carré :

$$\frac{1}{2}\left(a + \frac{A}{a}\right).$$

Notre numération de base 10 avec zéro explicite

- Inde, 8ème siècle ;
- Utilisation certaine et datable : stèle de Gwalior, $+876$;
- traité d'arithmétique d'Al-Khwarizmi.



En europe… il faut attendre la renaissance

- Gerbert d'Aurillac : 990, mais pas de réelle « percée » avant 1200 ;
- « Algorisme » ;
- Gravure de Gregor Reisch (1503) : Dame arithmétique préfère Boèce (qui calcule avec des chiffres) à Pythagore (qui calcule avec des jetons).

On sait faire du très mauvais travail...

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

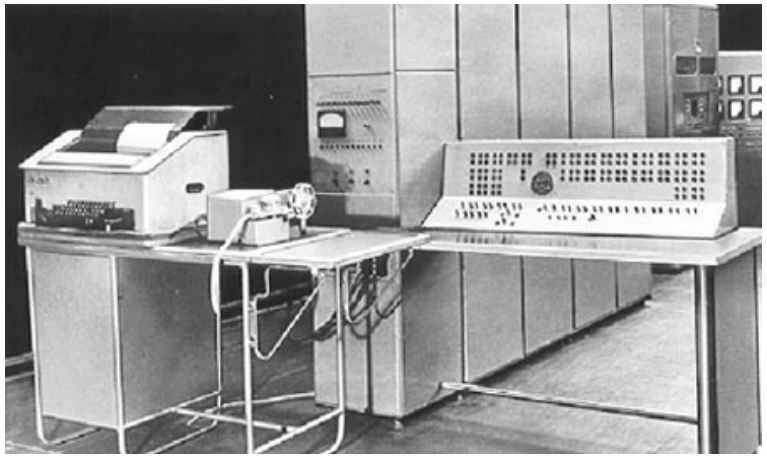
$$\frac{5001!}{5000!}$$

on obtient 1 au lieu de 5001. Dans la version 6.0, si on entrait :

$$21474836480413647819643794$$

la « quantité » affichée et mémorisée était `413647819643790)+'-.(-.(.`

. . . ou du travail bizarre



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

Arithmétique virgule flottante

- presque toujours utilisée pour représenter des réels ;
- principales caractéristiques : base b, taille n de la mantisse, plage des exposants

$$x = x_0.x_1x_2 \cdots x_{n-1} \times b^{e_x}$$

- a longtemps été de la « cuisine » :
  - valeur calculée de x op y : pas trop éloignée de la valeur exacte, mais que veut dire ce « pas trop éloignée » ?
  - premiers IBM 360 : multiplication assez approximative et fonctions soignées $\Rightarrow$ x ** 2 en général $+$ précis que x * x ;
  - choix de la base un peu arbitraire (2, 3, 4, 8, 10, 16, ...) ;
  - certains Cray : 1 * x $\Rightarrow$ overflow, divisions épouvantables ($x * (1/y)$) ;
  - formats très différents d'une machine à l'autre $\Rightarrow$ problèmes de portage.

À l'autre extrémité

Y. Kanada, de l'université de Tokyo, a calculé les $1.241.100.000.000$ premiers chiffres décimaux de π, en utilisant les relations

$$\pi = 48\arctan\frac{1}{49} + 128\arctan\frac{1}{57} - 20\arctan\frac{1}{239} + 48\arctan\frac{1}{110443}$$

$$\pi = 176\arctan\frac{1}{57} + 28\arctan\frac{1}{239} - 48\arctan\frac{1}{682} + 96\arctan\frac{1}{12943}.$$

qui donnent un algorithme à convergence linéaire (on a mieux en complexité en utilisant la moyenne arithmético-géométrique).

A demandé 600 heures de calcul sur un calculateur parallèle Hitachi à 64 processeurs. Mémoire nécessaire : 10^{12} octets.

Le mauvais résultat tout de suite, ou le bon résultat trop tard ?

- « vieille » arithmétique virgule flottante : très rapide, mais on ne sait pas borner (finement) les erreurs, ni prouver de propriétés ;
- arithmétiques « exactes », multi-précision, etc. : certains problèmes ne passeront jamais, soit pour des raisons de « fond », soit pour des raisons de rapidité.

Essayer de rendre rigoureuse l'arithmétique virgule flottante

Modes d'arrondi

- nombre machine : exactement représentable dans le système utilisé ;
- en général, la somme, le produit, le quotient de 2 nombres machines *ne sont pas* des nombres machine : il faut les arrondir. Modes d'arrondi usuels :

  - vers $-\infty$: $\nabla(x)$ est le plus grand nombre machine $\leq x$;
  - vers $+\infty$: $\Delta(x)$ est le plus petit nombre machine $\geq x$;
  - vers 0 : $\mathcal{Z}(x)$ vaut $\nabla(x)$ si $x \geq 0$, et $\Delta(x)$ si $x < 0$;
  - au plus près : $\mathcal{N}(x) =$ nombre machine le plus proche de x.

Norme IEEE-754 (1985)

- caractéristiques les plus connues : *formats* des représentations ;
- gestion des exceptions ($1/0$, $\sqrt{-5}$, over/underflows, etc.) ;
- caractéristique la plus intéressante : « arrondi correct » :

  – mode d'arrondi *actif* : $\diamond$
  – x et y : nombres machine

Quand on calcule $x \star y$ ($\star$ étant $+$, $-$, $\times$ ou $\div$), le résultat obtenu doit *toujours* être $\diamond(x \star y)$, i.e. *l'arrondi du résultat exact*.

- meilleure portabilité des programmes numériques ;
- arithmétique complètement spécifiée $\Rightarrow$ on peut élaborer des algorithmes et des preuves ;
- pas exigé pour les autres fonctions que $\pm$, $\times$, $\div$, $\sqrt{}$: dilemme du fabricant de tables.

Exemples de propriétés et algorithmes

- Lemme de Sterbenz : si $x/2 \leq y \leq 2x$ alors $x - y$ est calculé sans erreur ;
- avec ajout des nombres « dénormalisés », $x \neq y \Leftrightarrow x - y \neq 0$;
- arrondi au $+$ près : la valeur calculée de

$$\frac{x}{\sqrt{x^2 + y^2}}$$

  est toujours comprise entre -1 et $+1$;
- algorithmes de division (Harrison, Markstein), de racine carrée (Magaud) ;
- toutes sortes d'algorithmes géométriques ;
- `fast2sum(a,b) : s := a + b; z := s - a; r := b - z`

S'il ne doit y avoir qu'une chose à retenir concernant la division

Ce qu'autrefois vous conseilliez aux étudiants :

$$y := x/3 \rightarrow y := x * 0.333333333333$$

```
for j := i+1 to n do a[i,j] := a[i,j]/a[i,i]
```

$\rightarrow$

```
inv := 1/a[i,i];
for j := i+1 to n do a[i,j] := a[i,j]*inv
```

Demain :

. . . Exactement le contraire !

La Division

- en moyenne, 10 fois moins appelée que la multiplication ou l'addition dans les programmes numériques (dépend des applications) ;
- partiellement dû à une « censure » (ex. Elimination de Gauss) ;
- bien plus lente que la multiplication ou l'addition.

$\longrightarrow$ sur beaucoup d'applications, nos machines passent plus de temps à faire des divisions qu'à effectuer les autres opérations.

Processeur	Add. flottante	Mult. Flottante	Div. Flottante
Pentium IV	5	7	38
PowerPC 750	3	4	31
UltraSPARC III	4	4	24
Alpha21264	4	4	15
Athlon K6-III	3	3	20

Situation réelle pire que ce tableau : pas de pipe-line.

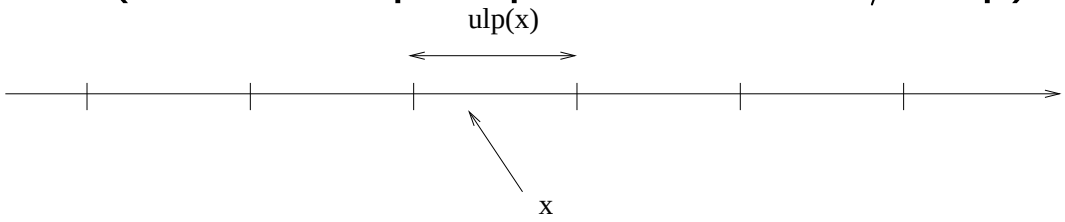
Exemple 1 : Division par des « constantes »

Calcul de x/y, où on suppose avoir le temps de faire du précalcul sur y :

- soit parce que y est connu à la compilation (« vraie » division par une constante) ;
- soit parce que y est connu nettement avant x (on peut alors faire le précalcul dans les « trous » du pipeline) ;
- soit parce qu'on va diviser de très nombreuses fois par le même y (le coût du précalcul sera alors amorti). Exemples typiques : élimination de Gauss, division par un pas de discrétisation.

Règle du jeu :

- complètement transparent : on doit avoir exactement le même résultat que par une vraie division (i.e. arrondi correct) ;
- opérateur utilisé : fused mac : $ax + b$ avec un seul arrondi (RS 6000, Power PC, Itanium) ;
- nombres machine : $\mathbb{M}_n = \left\{M \times 2^E; 2^{n-1} \leq |M| \leq 2^n - 1; M, E \in \mathbb{Z}\right\} \cup \{0\}$
- arrondi au plus près : $\circ(x) =$ élément de $\mathbb{M}_n$ le plus proche de x. S'il y en a deux, celui pour lequel M est pair.
- mesure : ulp(x) = distance entre les deux nombres machine qui encadrent x (arrondi au plus près = erreur $1/2$ ulp).



Solution naïve

- calculer l'inverse (en fait, $z = \circ(1/y)$) ;
- dès que x est connu, faire la multiplication $\circ(xz)$.

Quelques propriétés :

- erreur max < 1.5 ulps. Mieux, quand $n \to +\infty$, l'erreur max $\to 1.5$ ulps. Dans le format IEEE-754 double précision ($n = 53$), la division de
$$x = \frac{268435449}{134217728}$$
par
$$y = \frac{9007199120523265}{4503599627370496}$$
à l'aide de la solution naïve conduit à une erreur de $1.4999999739\cdots$ ulps ;
- quand $n \to \infty$, la proportion de couples (x, y) pour laquelle on obtient un arrondi incorrect tend vers $13/48 = 0.2708\cdots$.

Un premier exemple d'algorithme

On précalcule $z = \circ(1/y)$

Algorithme 1. [Division utilisant une multiplication et deux fused-macs]

$$\begin{aligned} q &= \circ(xz) \\ r &= \circ(x - qy) \\ q' &= \circ(q + rz) \end{aligned}$$

Donne toujours l'arrondi correct de x/y.

Un deuxième exemple

On précalcule

$$\begin{aligned} z_h &= \circ(1/y) \\ \rho &= \circ(1 - yz_h) \\ z_\ell &= \circ(\rho/y), \end{aligned}$$

ce qui assure que $z_\ell = \circ(1/y - z_h)$.

Algorithme 2. [Division utilisant une multiplication et un fused-mac]

$$\begin{aligned} q_1 &= \circ(xz_\ell) \\ q_2 &= \circ(xz_h + q_1). \end{aligned}$$

Quand donne-t-il un résultat correct ?

Expérimentalement : pour environ 98.7% des valeurs de y, marche $\forall x$.

Théorème 1. *L'algorithme précédent donne un résultat correct (i.e., $q_2 = \circ(x/y), \forall x$), dès que l'une au moins des conditions suivantes est satisfaite :*

1. *le dernier bit de mantisse de y est nul ;*
2. *$|z_\ell| < 2^{-n-2-e}$, où e est l'exposant de y (i.e., $2^e \leq |y| < 2^{e+1}$) ;*
3. *les calculs, sauf le dernier, sont faits sur au moins $n+1$ bits (i.e., une plus grande arithmétique interne est disponible – cas des processeurs Intel).*

La condition 1 permet de simplifier 50% des cas, la condition 2 simplifie 50% des cas restants.

Division par matériel : exemple d'outils de base

- Très petites tables (moins de 10 bits d'adresse) ;
- additions/soustractions à l'aide d'un système de numération redondant ;
- multiplication par de tous petits (en nombre de bits) nombres.

Un exemple : addition redondante « borrow-Save »

- base 2, chiffres -1, 0 et $+1$ (système BS) ;
- système redondant : $0.01\overline{1} = 0.1\overline{11} = 0.001 = 1/8$;
- Chaque chiffre d est représenté par deux bits d^+ et d^- tels que $d^+ - d^- = d$;
- entrée : deux « nombres BS » $a = a_{n-1}a_{n-2}\cdots a_0$ et $b = b_{n-1}b_{n-2}\cdots b_0$, où a_i, $b_i \in \{-1, 0, 1\}$;
- sortie : $s = s_n s_{n-1}\cdots s_0$ tel que $s = a + b$.

Algorithme

$a = a_{n-1}a_{n-2}\cdots a_0$ et $b = b_{n-1}b_{n-2}\cdots b_0$, $a_i = a_i^+ - a_i^-$, $b_i = b_i^+ - b_i^-$.

- Pour $i = 0, \ldots, n-1$ calculer deux bits c_{i+1}^+ et c_i^- t.q. $2c_{i+1}^+ - c_i^- = a_i^+ + b_i^+ - a_i^-$;
- Pour $i = 0, \ldots, n-1$ calculer s_{i+1}^- et s_i^+ t.q. $2s_{i+1}^- - s_i^+ = c_i^- + b_i^- - c_i^+$ (convention $c_0^+ = c_n^- = 0$, et $s_n^+ = c_n^+$).

les $s_i = s_i^+ - s_i^-$ sont les chiffres d'une écriture de $a + b$.

$$\begin{aligned} s &= \sum s_i 2^i = \sum (s_i^+ - s_i^-)2^i = \sum (-2s_{i+1}^- + s_i^+)2^i \\ &= \sum (c_i^+ - b_i^- - c_i^-)2^i = \sum (2c_{i+1}^+ - b_i^- - c_i^-)2^i \\ &= \sum (a_i^+ + b_i^+ - a_i^- - b_i^-)2^i = \sum (a_i + b_i)2^i \\ &= a + b \end{aligned}$$

Implantation matérielle

- les 2 pas de l'algorithme : à partir de 3 bits x, y, et z trouver 2 bits t et u tels que $2t - u = x + y - z$;
- cellule PPM (« Plus Plus Minus ») : très similaire à cellule élémentaire d'addition.

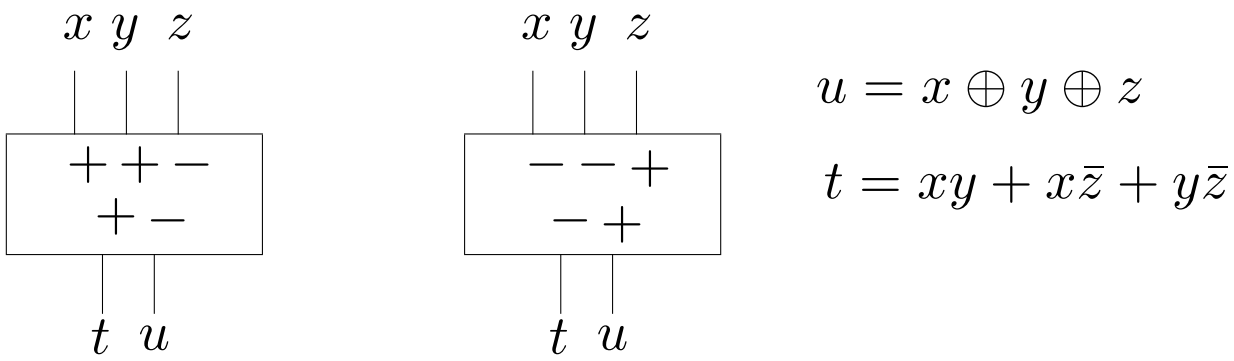
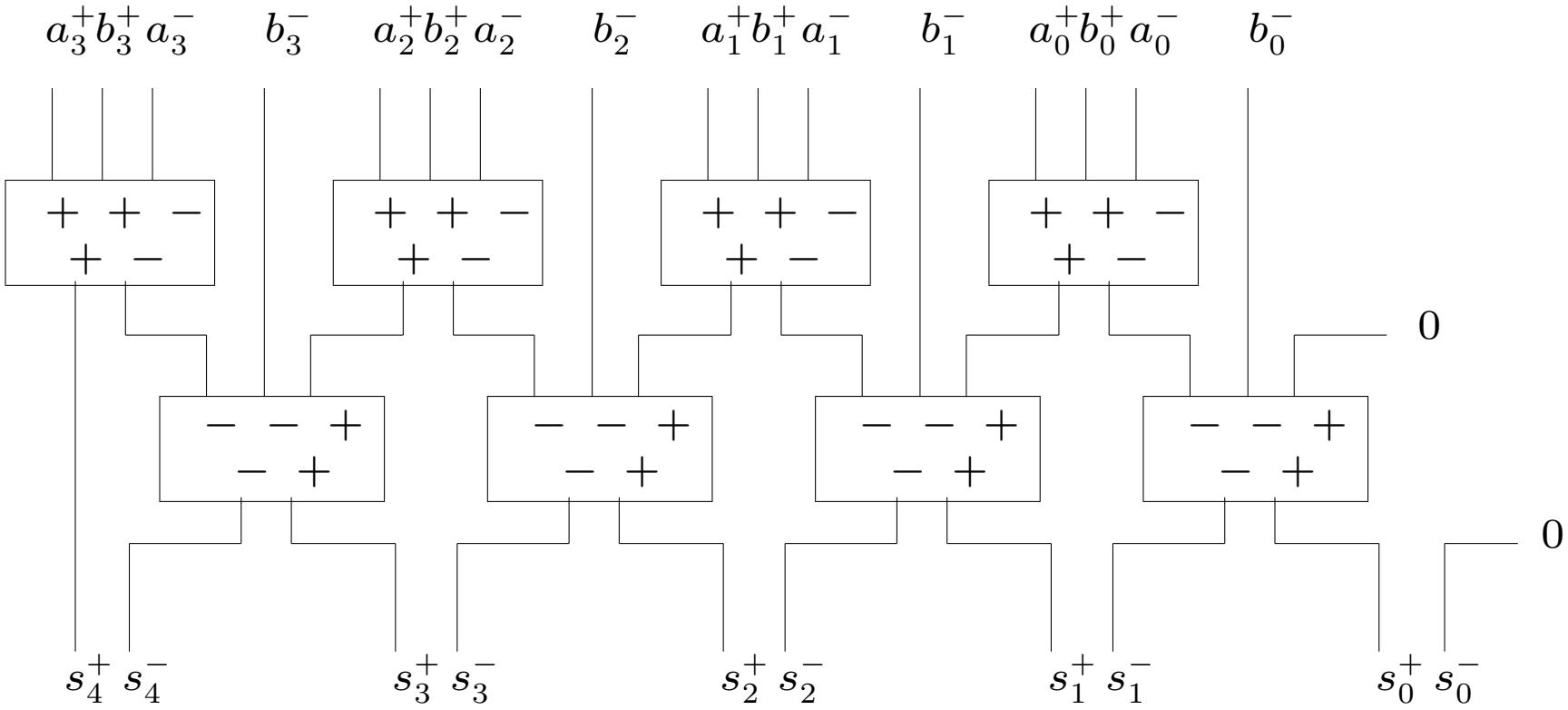
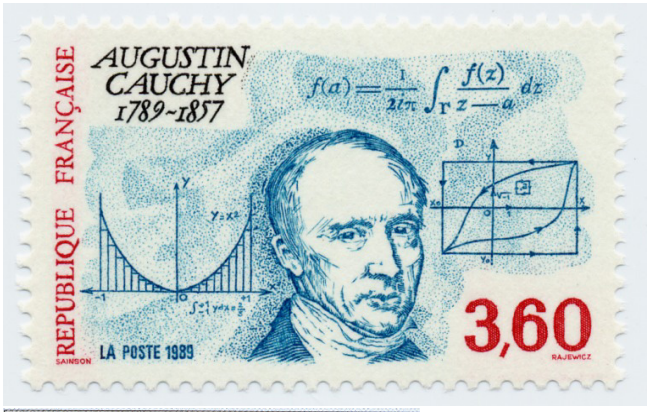


FIG. 1: *Une cellule PPM. Elle calcule t et u tels que $x + y - z = 2t - u$.*



- aucun signal ne traverse $+$ de 2 cellules entre l'entrée et la sortie ;
- temps d'addition indépendant de la taille des opérandes.

Systèmes de numération redondants



- Cauchy (1840) : utiliser, en base 10, les chiffres allant de -5 à $+5$ (but : simplifier les multiplications) ;
- Avizienis (1961) : base b et chiffres dans $\{-a, -a+1, \ldots, a-1, +a\}$. Algorithme parallèle d'addition lorsque $a \leq b-1$ et $2a \geq b+1$.

Division complexe

- Apparaît de manière naturelle dans de nombreux problèmes (astronomie, traitement d'images, de signaux, etc.) ;
- beaucoup de « censure » : même dans les langages qui offrent la division complexe, on ne l'utilise que peu ;
- formule conventionnelle :

$$\frac{a+ib}{c+id} = \frac{ac+bd+i(bc-ad)}{c^2+d^2}. \qquad (1)$$

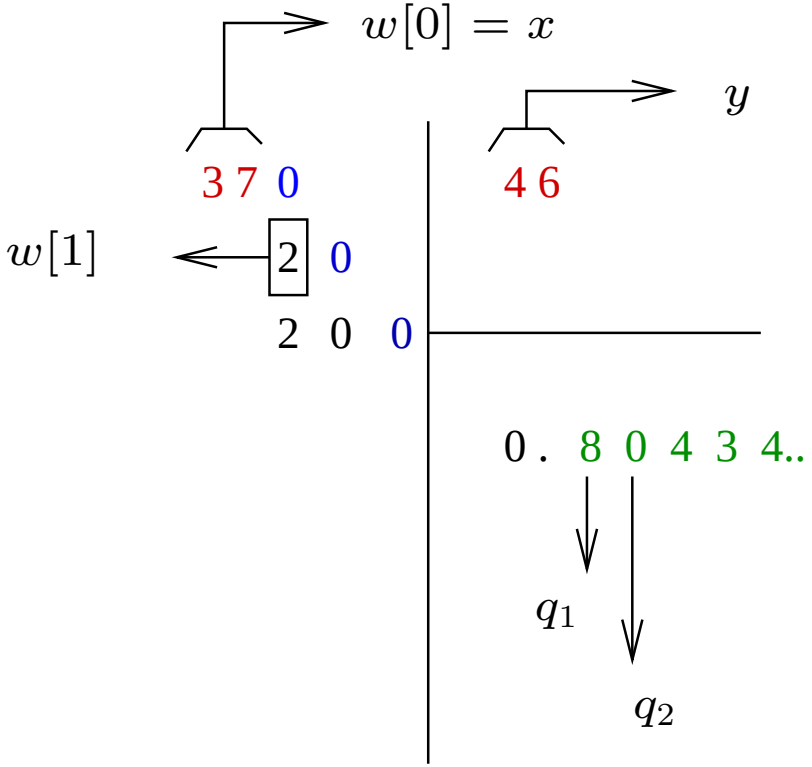
  possibles dépassements lors des calculs intermédiaires + précision parfois mauvaise.
- reste implanté en logiciel (ex. compilateur Fortran Linux d'Intel) ;
- Peu d'implantations matérielles (McIlhenny, thèse UCLA, 2002).

Formule de Smith (1962 + amélioration en 85)

$$\frac{a+ib}{c+id} = \begin{cases} \dfrac{a+b(d/c)}{c+d(d/c)} + i\dfrac{b-a(d/c)}{c+d(d/c)} & (\text{ si } |c| \geq |d|) \\ \dfrac{b+a(c/d)}{d+c(c/d)} + i\dfrac{a-b(c/d)}{d+c(c/d)} & (\text{ si } |c| \leq |d|) \end{cases} \tag{2}$$

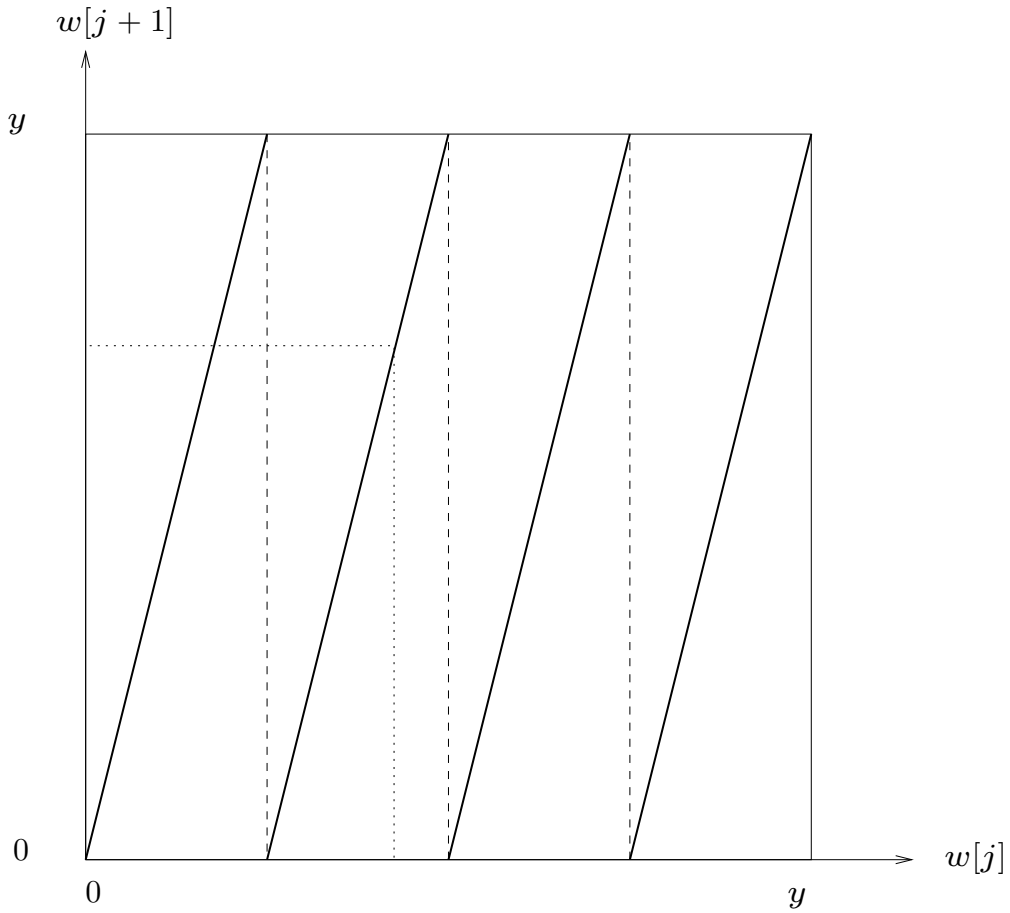
- élimine la plupart des dépassements intermédiaires ;
- Pas de garantie sur l'arrondi des parties réelles et imaginaires ;
- « usine à gaz ».

Retournons à l'École



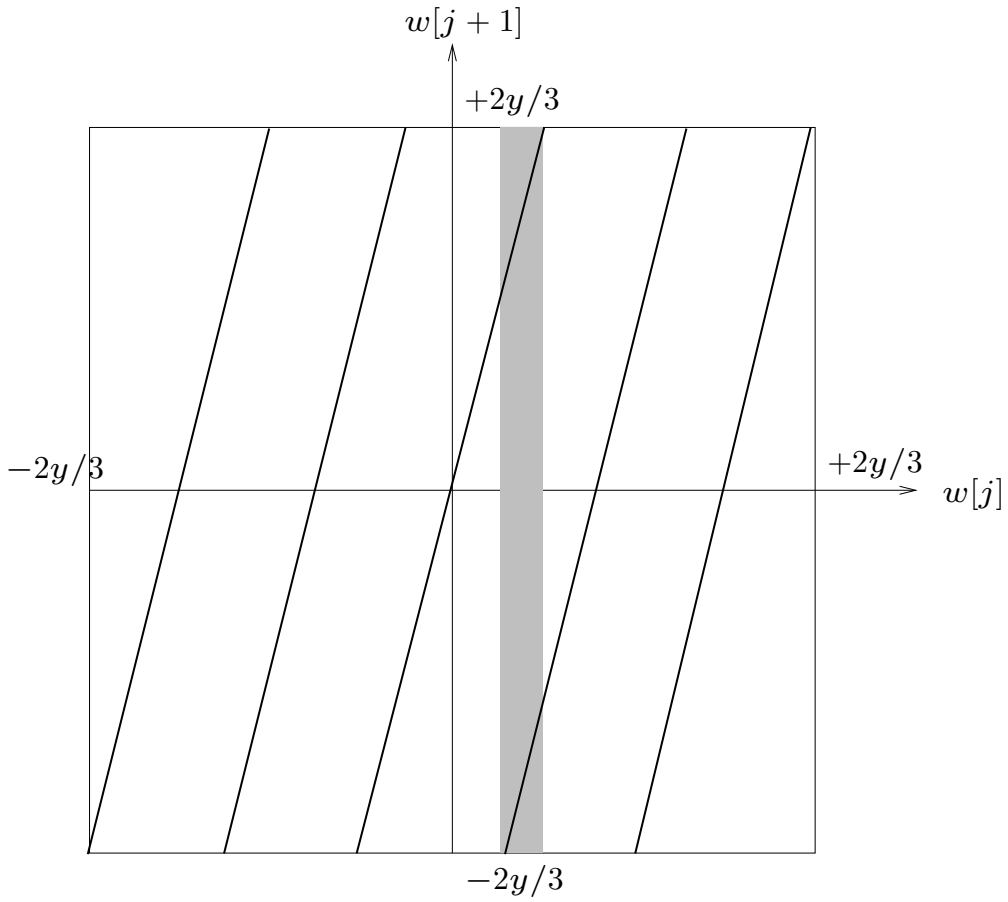
$$w[j+1] = 10w[j] - q_{j+1}y$$

Division « non redondante » de base 4.



$$w[j+1] = 4w[j] - q_{j+1}y, \text{ with } q_{j+1} \in \{0, 1, 2, 3\}$$

Division « redondante » de base 4



$$w[j+1] = 4w[j] - q_{j+1}y, \text{ with } q_{j+1} \in \{-2,-1,0,1,2\}$$

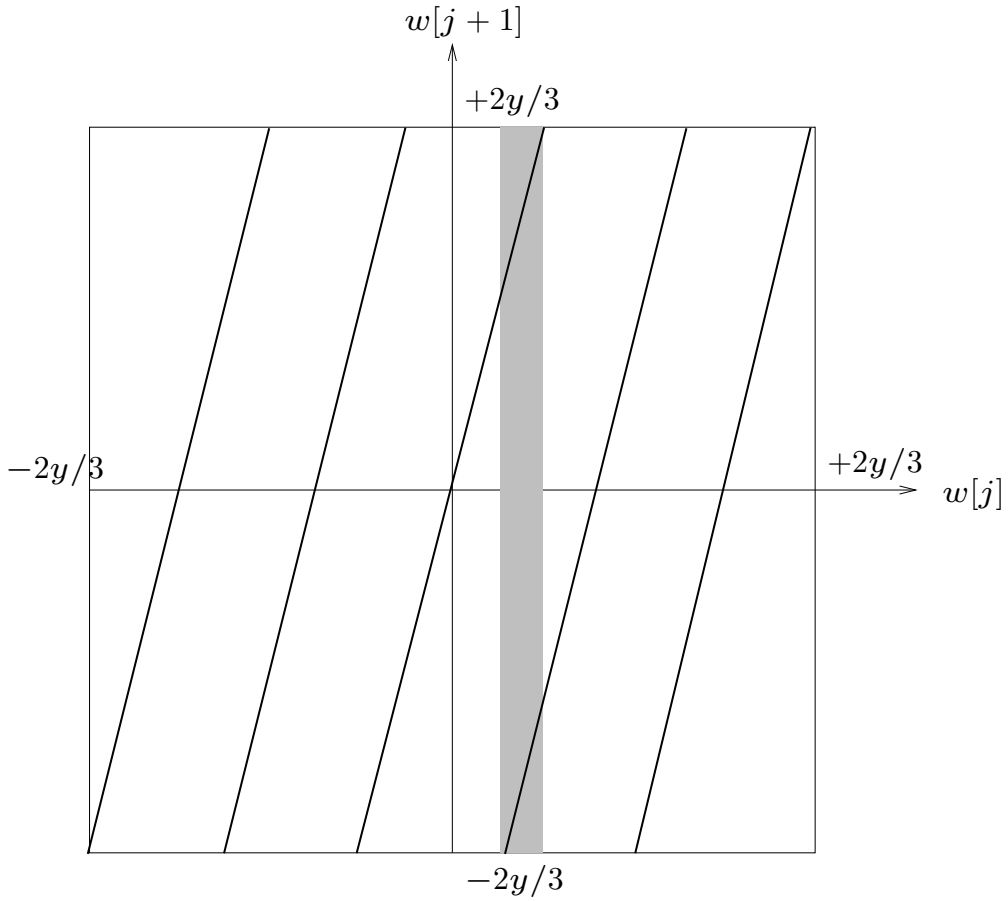
Division réelle de base 4

- Pour calculer x/y, l'algorithme de base 4 utilise la récurrence suivante sur les restes $w[j]$

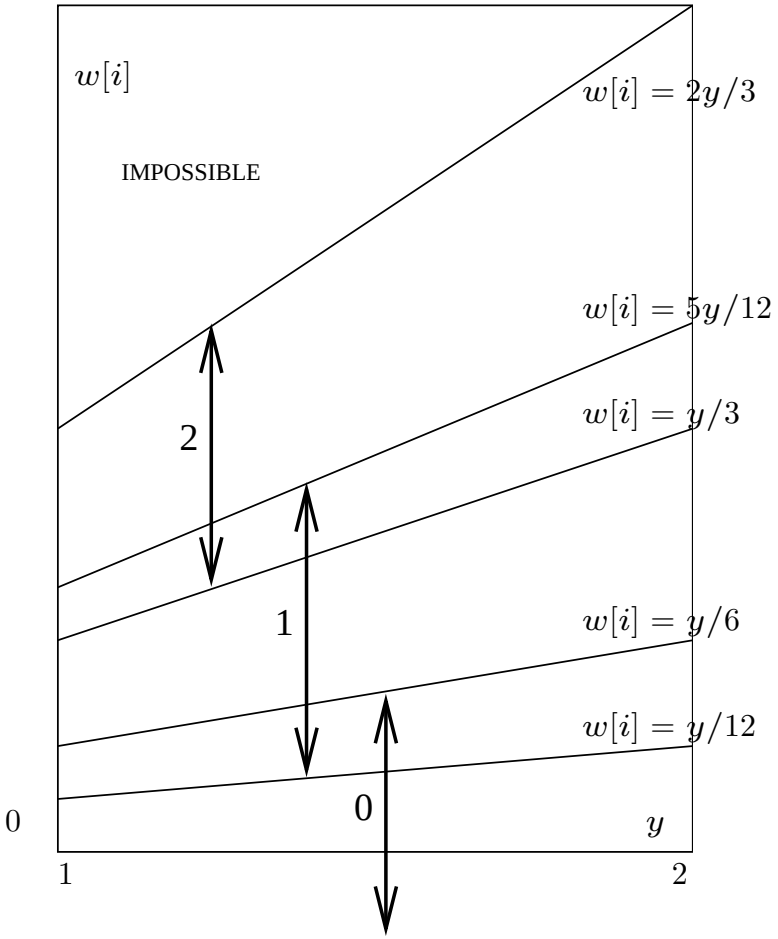
$$w[j+1] = 4w[j] - q_{j+1}y \tag{3}$$

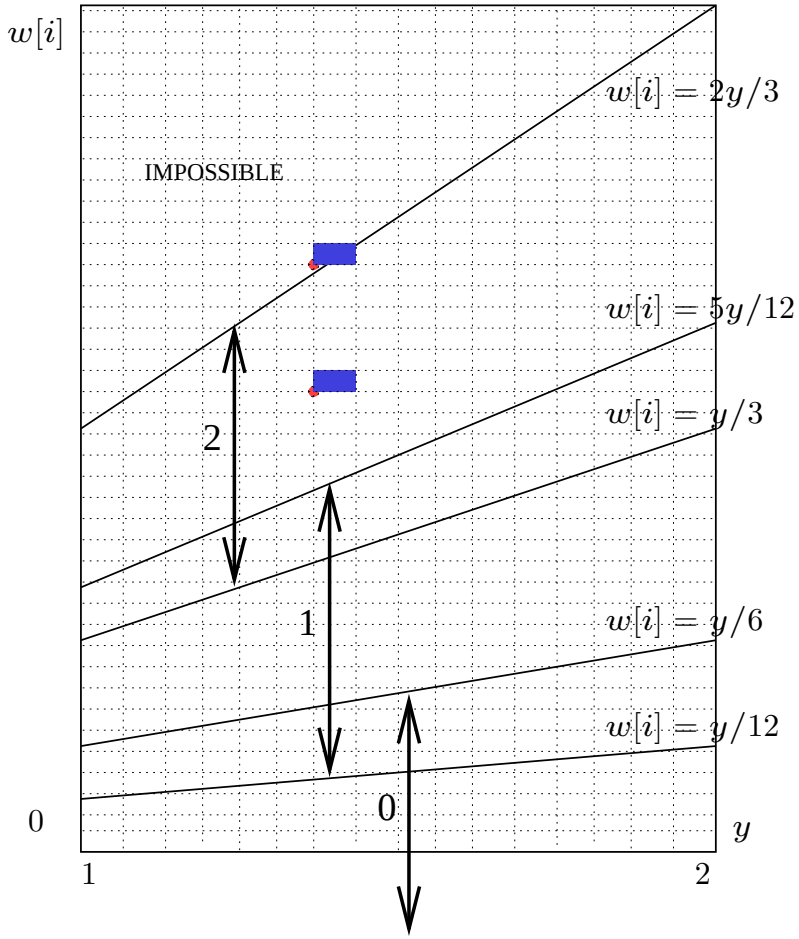
où $w[0] = x$. Les restes w sont dans un système redondant.
- Chiffres du quotient $q_j \in \{-2, \ldots, +2\}$ (ensemble redondant)
- Sélectionnés de sorte que les $w[j]$ restent bornés
- Par récurrence : $\frac{w[j]}{y} = 4^j \left[\frac{w[0]}{y} - \left(q_1 4^{-1} + q_2 4^{-2} + \cdots + q_j 4^{-j}\right)\right]$.

Division redondante de base 4

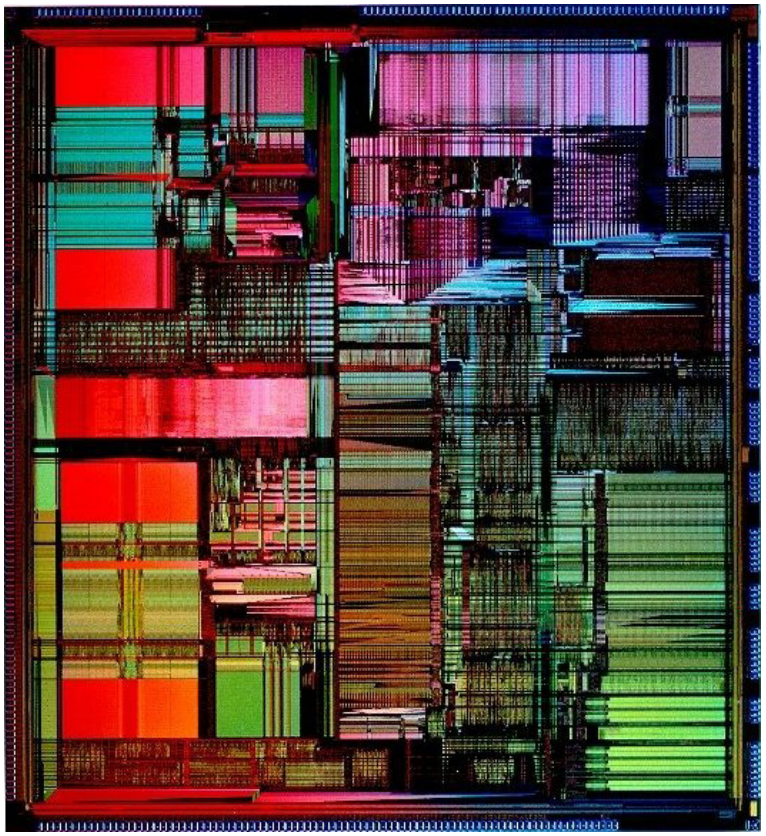


$$w[j+1] = 4w[j] - q_{j+1}y, \text{ with } q_{j+1} \in \{-2,-1,0,1,2\}$$

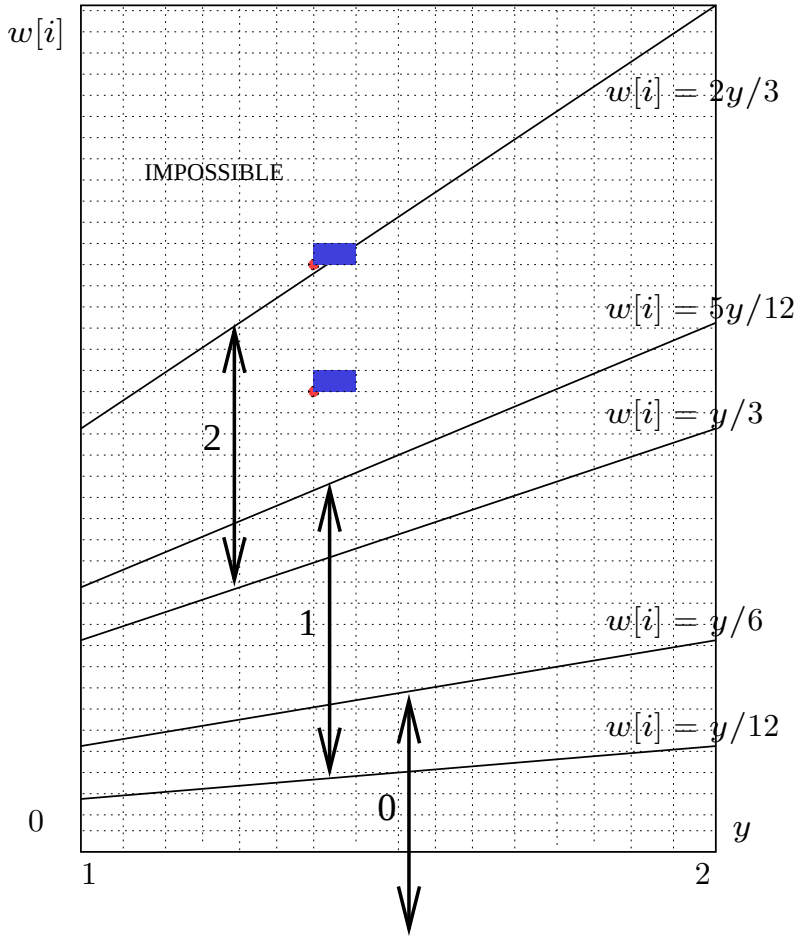

$w[i]$
$w[i] = 2y/3$
IMPOSSIBLE
$w[i] = 5y/12$
$w[i] = y/3$
2
1
$w[i] = y/6$
$w[i] = y/12$
0
0
y
1
2


$w[i]$
$w[i] = 2y/3$
IMPOSSIBLE
$w[i] = 5y/12$
$w[i] = y/3$
2
1
$w[i] = y/6$
$w[i] = y/12$
0
0
y
1
2

Celui par qui le scandale arrive



- Processeur Pentium 1 (1994) ;
- 5 erreurs dans la table. . .
- . . . mais elles ne sont pas « aléatoires »


$w[i]$
$w[i] = 2y/3$
IMPOSSIBLE
$w[i] = 5y/12$
$w[i] = y/3$
2
1
$w[i] = y/6$
$w[i] = y/12$
0
0
y
1
2

Division complexe : principes généraux

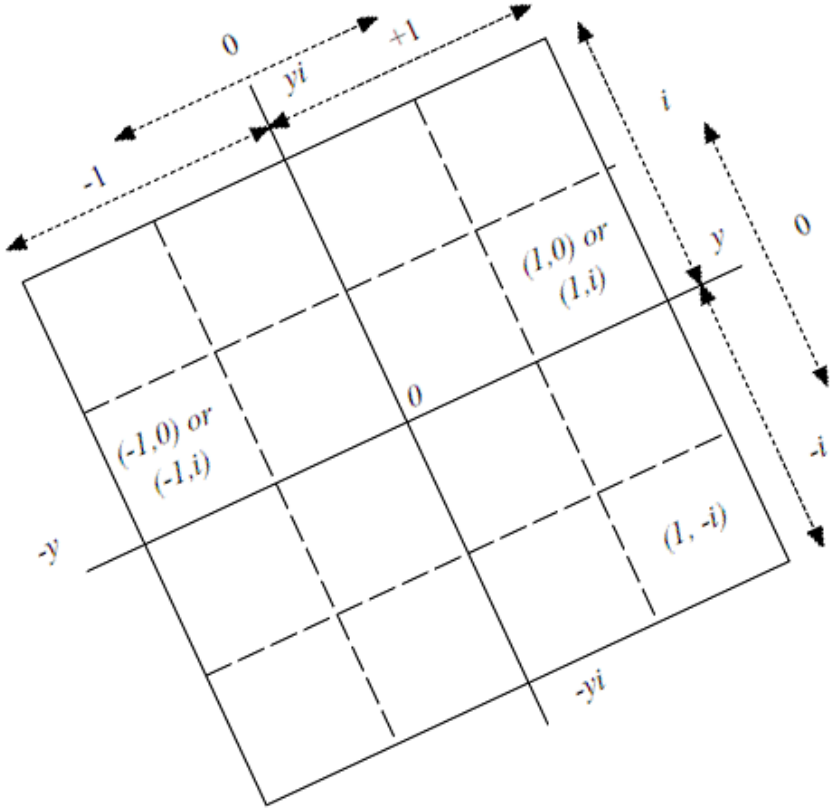
- Itérations (pour $w[0]/y$) : effectuer

$$w[j+1] = rw[j] - q_{j+1}y \tag{4}$$

  r est la base de la division, et $q_{j+1} = q_{j+1}^{\mathcal{R}} + iq_{j+1}^{\mathcal{I}}$, où $q_{j+1}^{\mathcal{R}}$ et $q_{j+1}^{\mathcal{I}} \in$ ensemble $\mathcal{S}$ de chiffres, redondant pour la base r.
- Par récurrence : $\frac{w[j]}{y} = r^j \left[\frac{w[0]}{y} - \left(q_1 r^{-1} + q_2 r^{-2} + \cdots + q_j r^{-j}\right)\right]$.
  $\Rightarrow$ tout choix des q_j pour lequel $||w[j]||_\infty$ reste borné suffit pour garantir

$$0.q_1^{\mathcal{R}} q_2^{\mathcal{R}} q_3^{\mathcal{R}} \cdots q_n^{\mathcal{R}} + i0.q_1^{\mathcal{I}} q_2^{\mathcal{I}} q_3^{\mathcal{I}} \cdots q_n^{\mathcal{I}} \rightarrow z/d;$$

Choix de chiffres en base 2



Solution : pré-normalisation (prescaling)

- But : $q = z/d$, où $z, d \in \mathbb{C}$.
- Pré-normalisation : obtenir (via lecture de table) un facteur d'échelle K tel que

$$||Kd - 1||_\infty < 2^{-p},$$

où p est un paramètre de l'algorithme. Calculer ensuite

$$\begin{cases} w[0] &= Kz \\ y &= Kd \end{cases}$$

Permet de séparer le choix du chiffre du quotient pour la partie réelle et la partie imaginaire.

Sélection des chiffres du quotient

- Le diviseur normalisé y est proche de $1 \Rightarrow q_{j+1}$ peut par exemple être choisi en arrondissant $rw[j]$ à l'entier le plus proche.
- Une estimation de faible précision de $w[j]$ suffit $\Rightarrow w[j]$ en borrow-save $\&$ sélection par arrondi des parties réelle $\&$ imaginaire de $r\hat{w}[j] = rw[j]$ tronquées après le σème chiffre BS.

$$\left\{ \begin{array}{lcl} q_{j+1}^{\mathcal{R}} & = & s\left(\Re(rw[j])\right) \\ q_{j+1}^{\mathcal{I}} & = & s\left(\Im(rw[j])\right) \end{array} \right.$$

où la fonction de sélection s vérifie $|s(x) - x| < 1/2 + 2^{-\sigma}$.

Sélection des chiffres du quotient (suite)

- Les parties réelle et imaginaire des chiffres du quotient sont prises dans $\mathcal{S} = \{-a, -a+1, \ldots, 0, \ldots, +a\}$ où $2a+1 > r$ et, en général, $a \leq r-1$.
- Pour garantir que la fonction de sélection retourne des éléments de $\mathcal{S}$, on doit avoir $||w[j]||_\infty \leq \frac{1}{r}\left(a + \frac{1}{2} - 2^{-\sigma}\right) = \Omega$.
- Notons $\epsilon_y = y - 1$. il vient : $w[j+1] = \epsilon_y q_{j+1} + A + iB$, où $|A|$ et $|B|$ sont $\leq \frac{1}{2} + 2^{-\sigma}$, $||q_{j+1}||_\infty \leq a$ et $||\epsilon_y||_\infty < 2^{-p}$. Ceci donne

$$||w[j+1]||_\infty < 2^{-p+1}a + \frac{1}{2} + 2^{-\sigma}.$$

$\Rightarrow$ l'algorithme est correct si $2^{-p+1}a + \frac{1}{2} + 2^{-\sigma} \leq \Omega$.

r	a	p	σ	Ω
2	1	4	4	$23/32$
2	2	3	3	$19/16$
4	3	5	3	$27/32$
4	4	4	4	$71/74$
8	4	8	6	$287/512$
8	5	6	6	$351/512$
8	6	6	4	$103/128$
8	8	5	5	$271/256$
16	15	7	3	$123/128$
16	16	6	6	$1055/1024$

TAB. 1: Valeurs de p, σ et Ω. p décroît lorsque a augmente, ce qui peut servir car la pré-normalisation demande une table à $2p + 1$ bits d'adresse.

L'itération de division complexe

- Les parties réelle et imaginaire de $w[j+1]$ se calculent en parallèle.

$$\begin{cases} \Re(w[j+1]) &= r\Re(w[j]) - q_{j+1}^{\mathcal{R}}\Re(y) + q_{j+1}^{\mathcal{I}}\Im(y) \\ \Im(w[j+1]) &= r\Im(w[j]) - q_{j+1}^{\mathcal{I}}\Re(y) - q_{j+1}^{\mathcal{R}}\Im(y) \end{cases} \quad (5)$$

- La simplicité de la fonction de sélection (arrondir séparément les parties réelle et imaginaire de $w[j+1]$) rend les itérations simples.
- Temps de notre algorithme : légèrement supérieur à celui d'une division réelle.

Pré-normalisation par lecture directe de table

- Étant donné un diviseur d, trouver K tel que $||Kd-1||_\infty < 2^{-p}$;
- on suppose $\frac{1}{2} \leq ||d||_\infty < 1$ et $d = a + ib$, a et b représentés en virgule fixe.
- Soient $\hat{a}$ et $\hat{b}$ égaux à a et b arrondis au plus proche nombre de q bits. On lit $K = 1/(\hat{a} + i\hat{b})$ dans une table à $2q-1$ bits d'adresse[1]. En notant $\hat{d} = \hat{a} + i\hat{b}$, on obtient $\left|\left|d/\hat{d} - 1\right|\right|_\infty \leq 2^{-q+1}$
- Pour garantir $||y-1||_\infty < 2^{-p}$, il suffit de choisir $q = p+1 \Rightarrow$ la table aura $2q-1 = 2p+1$ bits d'adresse.

[1] On peut supposer $a_1 = 1$: si $a_1 = 0$, alors $b_1 = 1$, donc on peut interchanger $\hat{a}$ et $\hat{b}$.

Lecture directe de table : taille des tables

r	a	p	Taille	Réalisable ?
2	1	4	2^9	oui
	2	3	2^7	oui
4	3	5	2^{11}	oui
	4	4	2^9	oui
8	7	6	2^{13}	oui ?
16	8	10	2^{21}	non

Arrondi

- Si on s'arrête après avoir calculé q_j, l'erreur maximum sur les parties réelle et imaginaire est

$$\sum_{\ell=j+1}^{\infty} a \times r^{-\ell} = \frac{ar^{-j}}{r-1}.$$

- Si $a \leq r-1$ cette erreur est $\leq$ poids du dernier chiffre.

En résumé

- Algorithme dérivé de l'algorithme réel ;
- Prénormalisation pour permettre un choix simple des chiffres du quotient ;
- choix des bases limité par la taille des tables ;
- 4 fois plus rapide que l'algorithme de Smith ;
- facilement implantable en matériel ;
- Arrondi correct réalisable à un coût raisonnable.

Juste un transparent sur le dilemme du fabricant de tables

Posons :

$$x = 1.01100010101010001000011000010011011000101001101101110 \times 2^{678}$$

son logarithme est

$$\log x = \overbrace{111010110.01000111100111101010\cdots110001}^{53 \text{ bits}}\underbrace{0000000000000000\cdots0000000000000000}_{65 \text{ zéros}}1110\ldots$$