

Produit précis de nombres flottants

Stef Graillat

LIP6/PEQUAN - Université Pierre et Marie Curie (Paris 6)

RAIM'08, 2^e Rencontres Arithmétique de l'Informatique Mathématique
Lille, France, 3-5 juin 2008



- Déterminant d'une matrice triangulaire

$$T = \begin{bmatrix} t_{11} & t_{12} & \cdots & t_{1n} \\ & t_{22} & & t_{2n} \\ & & \ddots & \vdots \\ & & & t_{nn} \end{bmatrix}.$$

$$\det(T) = \prod_{i=1}^n t_{ii}.$$

- Évaluation polynomiale pour un polynôme représenté par ses racines
 $p(x) = a_n \prod_{i=1}^n (x - x_i)$
- Exemple de Sterbenz dans son livre *Floating-point computation*

Que sont les transformations exactes (EFT) ?

Arithmétique à virgule flottante conforme à la norme IEEE 754 avec **arrondi au plus près** avec unité d'arrondi u (ni underflow ni overflow)

Les **transformations exactes** sont des algorithmes calculant les erreurs d'arrondi générées par les opérations élémentaires

$$a, b \text{ en entrée } \in \mathbb{F}, \quad a \circ b = \text{fl}(a \circ b) + e, \text{ avec } e \in \mathbb{F}$$

Algorithmes clé pour des **calculs précis**

- expansions de longueur fixe : double-double (Briggs, Bailey, Hida, Li), quad-double (Bailey, Hida, Li)
- expansions de longueur arbitraire : Priest, Shewchuk
- **algorithmes compensés** : Kahan, Priest, Ogita-Rump-Oishi, Graillat-Langlois-Louvet

Transformations exactes pour la somme

$$x = \text{fl}(a \pm b) \Rightarrow a \pm b = x + y \quad \text{with } y \in \mathbb{F},$$

Algorithmes de Dekker (1971) et Knuth (1974)

Algorithme 1 (Transformation exacte pour la somme de 2 flottants avec $|a| \geq |b|$)

```
function [x, y] = FastTwoSum(a, b)
    x = fl(a + b)
    y = fl((a - x) + b)
```

Algorithme 2 (Transformation exacte pour la somme de 2 flottants)

```
function [x, y] = TwoSum(a, b)
    x = fl(a + b)
    z = fl(x - a)
    y = fl((a - (x - z)) + (b - z))
```

Transformations exactes pour le produit (1/3)

$$x = \text{fl}(a \cdot b) \Rightarrow a \cdot b = x + y \quad \text{avec } y \in \mathbb{F},$$

Algorithme TwoProduct de Veltkamp et Dekker (1971)

$$a = x + y \quad \text{and} \quad x \text{ et } y \text{ ne se chevauchent pas avec } |y| \leq |x|.$$

Algorithme 3 (Séparation exacte d'un flottant en deux parties)

```
function [x, y] = Split(a, b)
    factor = fl(2s + 1)           % u = 2-p, s = [p/2]
    c = fl(factor · a)
    x = fl(c - (c - a))
    y = fl(a - x)
```

Algorithme 4 (Transformation exacte pour le produit de 2 flottants)

```
function [x, y] = TwoProduct(a, b)
    x = fl(a · b)
    [a1, a2] = Split(a)
    [b1, b2] = Split(b)
    y = fl(a2 · b2 - (((x - a1 · b1) - a2 · b1) - a1 · b2))
```

Cet algorithme nécessite 17 flops

Étant donnés $a, b, c \in \mathbb{F}$,

- $\text{FMA}(a, b, c)$ est l'arrondi au plus près de $a \cdot b + c \in \mathbb{F}$

Algorithme 5 (Transformation exacte pour la somme de 2 flottants)

```
function  $[x, y] = \text{TwoProductFMA}(a, b)$ 
```

```
   $x = \text{fl}(a \cdot b)$ 
```

```
   $y = \text{FMA}(a, b, -x)$ 
```

Un FMA est présent sur les PowerPC, Itanium, processeurs Cell.

Théorème 1

Soit $a, b \in \mathbb{F}$ et $x, y \in \mathbb{F}$ tels que $[x, y] = \text{TwoSum}(a, b)$. Alors,

$$a + b = x + y, \quad x = \text{fl}(a + b), \quad |y| \leq \mathbf{u}|x|, \quad |y| \leq \mathbf{u}|a + b|.$$

L'algorithme `TwoSum` nécessite 6 flops.

Soit $a, b \in \mathbb{F}$ et $x, y \in \mathbb{F}$ tels que $[x, y] = \text{TwoProduct}(a, b)$. Alors,

$$a \cdot b = x + y, \quad x = \text{fl}(a \cdot b), \quad |y| \leq \mathbf{u}|x|, \quad |y| \leq \mathbf{u}|a \cdot b|,$$

L'algorithme `TwoProduct` nécessite 17 flops.

Algorithme récursif pour le produit

La méthode classique pour l'évaluation du produit de n nombres

$$a = (a_1, a_2, \dots, a_n)$$

$$p = \prod_{i=1}^n a_i$$

est l'algorithme suivant.

Algorithme 6 (Algorithme récursif)

```
function res = Prod(a)
    p1 = a1
    for i = 2 : n
        pi = fl(pi-1 · ai)      % erreur d'arrondi πi
    end
    res = pn
```

Cet algorithme nécessite $n - 1$ flops

Notation

$$\gamma_n := \frac{n\mathbf{u}}{1 - n\mathbf{u}} \quad \text{for } n \in \mathbb{N}.$$

Une borne d'erreur directe est

$$|a_1 a_2 \cdots a_n - \text{res}| \leq \gamma_{n-1} |a_1 a_2 \cdots a_n|$$

Une borne d'erreur validée en flottant est

$$|a_1 a_2 \cdots a_n - \text{res}| \leq \text{fl} \left(\frac{\gamma_{n-1} |\text{res}|}{1 - 2\mathbf{u}} \right)$$

Algorithme 7 (Schéma compensé pour le produit)

```
function res = CompProd(a)
    p1 = a1
    e1 = 0
    for i = 2 : n
        [pi, πi] = TwoProduct(pi-1, ai)
        ei = fl(ei-1ai + πi)
    end
    res = fl(pn + en)
```

Cet algorithme nécessite $19n - 18$ flops

Algorithme 8 (Schéma compensé pour le produit avec TwoProductFMA et FMA)

```
function res = CompProdFMA(a)
    p1 = a1
    e1 = 0
    for i = 2 : n
        [pi, πi] = TwoProductFMA(pi-1, ai)
        ei = FMA(ei-1, ai, πi)
    end
    res = fl(pn + en)
```

Cet algorithme nécessite $3n - 2$ flops

Théorème 2

Soit res le résultat de l'algorithme `CompProd` appliqué à $a_i \in \mathbb{F}$, $1 \leq i \leq n$, et $p = \prod_{i=1}^n a_i$. Alors,

$$|\text{res} - p| \leq \mathbf{u}|p| + \gamma_n \gamma_{2n}|p|$$

Conditionnement pour le produit :

$$\text{cond}(a) = \limsup_{\varepsilon \rightarrow 0} \left\{ \frac{|(a_1 + \Delta a_1) \cdots (a_n + \Delta a_n) - a_1 \cdots a_n|}{\varepsilon |a_1 a_2 \cdots a_n|} : |\Delta a_i| \leq \varepsilon |a_i| \right\}$$

Un calcul standard montre que

$$\text{cond}(a) = n$$

Ordinateur portable avec un Pentium M à 1.73GHz et gcc version 4.0.2.

Tab.: Temps de calcul mesuré avec Prod normalisé to 1.0

n	Prod	CompProd
100	1.0	3.5
500	1.0	4.4
1000	1.0	5.0
10000	1.0	4.9
100000	1.0	5.5

Le ratio théorique pour CompProd est 19.

Les **algorithmes compensés** sont généralement plus rapides que les performances théoriques

→ cela est en partie dû à un meilleur parallélisme d'instructions

[Langlois-Louvet,2007].

Lemme 1

Soit res le résultat de l'algorithme CompProd appliqué à $a_i \in \mathbb{F}$, $1 \leq i \leq n$ et soit $p = \prod_{i=1}^n a_i$. Alors l'erreur directe absolue affectant le calcul est bornée par

$$|\text{res} - p| \leq \text{fl} \left(\left(\mathbf{u}|\text{res}| + \frac{\gamma_n \gamma_{2n} |a_1 a_2 \cdots a_n|}{1 - (n+3)\mathbf{u}} \right) / (1 - 2\mathbf{u}) \right).$$

Arrondi fidèle (1/3)

Prédécesseur et successeur flottant d'un nombre réel r satisfaisant $\min\{f : f \in \mathbb{R}\} < r < \max\{f : f \in \mathbb{F}\}$:

$$\text{pred}(r) := \max\{f \in \mathbb{F} : f < r\} \quad \text{et} \quad \text{succ}(r) := \min\{f \in \mathbb{F} : r < f\}.$$

Définition 1

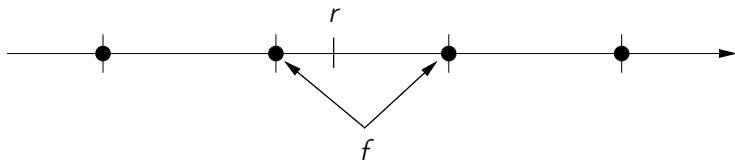
Un nombre flottant $f \in \mathbb{F}$ est un arrondi fidèle du nombre réel $r \in \mathbb{R}$ si

$$\text{pred}(f) < r < \text{succ}(f).$$

On note alors $f \in \square(r)$. Si $r \in \mathbb{F}$, alors nécessairement $f = r$.

Arrondi fidèle signifie que le résultat calculé est égal au résultat exact si celui-ci est un nombre flottant et autrement est l'un des deux flottants adjacents au résultat exact.

Arrondi fidèle (2/3)



Lemme 2 (Rump, Ogita et Oishi, 2005)

Soit $r, \delta \in \mathbb{R}$ et $\tilde{r} := \text{fl}(r)$. Supposons que $2|\delta| < \mathbf{u}|\tilde{r}|$. Alors $\tilde{r} \in \square(r + \delta)$, ce qui signifie que \tilde{r} est un arrondi fidèle de $r + \delta$.

Soit $\text{res} = \text{CompProd}(p)$

Lemme 3

Si $n < \frac{\sqrt{1-u}}{\sqrt{2}\sqrt{2+u}+2\sqrt{(1-u)u}} u^{-1/2}$ alors res est un arrondi fidèle de p .

Si $n < \alpha u^{-1/2}$ où $\alpha \approx 1/2$ alors le résultat est fidèlement arrondi

En double précision où $u = 2^{-53}$, si $n < 2^{25} \approx 5 \cdot 10^7$, on obtient un arrondi fidèle

Si

$$\text{fl} \left(2 \frac{\gamma_n \gamma_{2n} |a_1 a_2 \cdots a_n|}{1 - (n+3)\mathbf{u}} \right) < \text{fl}(\mathbf{u}|\text{res}|)$$

alors on a un arrondi fidèle du résultat. Cela permet *a posteriori* de vérifier la précision du résultat calculé.

Algorithme 9 (Exponentiation par un schéma compensé)

```
function res = CompLinPower(x, n)
    p1 = x
    e1 = 0
    for i = 2 : n
        [pi, πi] = TwoProduct(pi-1, x)
        ei = fl(ei-1x + πi)
    end
    res = fl(pn + en)
```

Complexité : $\mathcal{O}(n)$

En double précision où $u = 2^{-53}$, si $n < 2^{25} \approx 5 \cdot 10^7$, alors on a un arrondi fidèle

Algorithme 10 (Multiplication de deux double-double)

```
function  $[r_h, r_l] = \text{prod\_dd\_dd}(a_h, a_l, b_h, b_l)$   
     $[t_1, t_2] = \text{TwoProduct}(a_h, b_h)$   
     $t_3 = \text{fl}(((a_h \cdot b_l) + (a_l \cdot b_h)) + t_2)$   
     $[r_h, r_l] = \text{TwoProduct}(t_1, t_3)$ 
```

Algorithme 11 (Multiplication d'un double-double par un double)

```
function  $[r_h, r_l] = \text{prod\_dd\_d}(a, b_h, b_l)$   
     $[t_1, t_2] = \text{TwoProduct}(a, b_h)$   
     $t_3 = \text{fl}((a \cdot b_l) + t_2)$   
     $[r_h, r_l] = \text{TwoProduct}(t_1, t_3)$ 
```

Théorème 3 (Lauter 2005)

Soit $a_h + a_l$ et $b_h + b_l$ les entrées double-double de l'algorithme `prod_dd_dd`. Alors les valeurs retournées r_h et r_l satisfont

$$r_h + r_l = ((a_h + a_l) \cdot (b_h + b_l))(1 + \varepsilon)$$

avec ε bornée par : $|\varepsilon| \leq 16\mathbf{u}^2$. De plus, on a $|r_l| \leq \mathbf{u}|r_h|$.

Un algorithme logarithmique (1/2)

Algorithme **droite-gauche** [Kornerup-Lefevre-Muller,2007]

Ici, algorithme **gauche-droite** \rightarrow produit double-double \times double-double
remplacé par double \times double-double

Algorithme 12 (Exponentiation avec un schéma compensé)

```
function res = CompLogPower(x, n)           % n = (n_t n_{t-1} \cdots n_1 n_0)_2
[h, l] = [1, 0]
for i = t : -1 : 0
    [h, l] = prod_dd_dd(h, l, h, l)
    if n_i = 1
        [h, l] = prod_dd_d(x, h, l)
    end
end
res = fl(h + l)
```

Complexité : $\mathcal{O}(\log n)$

Théorème 4

Les valeurs h et l renvoyées par l'algorithme `CompLogPower` vérifient

$$h + l = x^n(1 + \varepsilon)$$

avec

$$(1 - 16\mathbf{u}^2)^{n-1} \leq 1 + \varepsilon \leq (1 + 16\mathbf{u}^2)^{n-1}.$$

Par exemple, en double précision avec $\mathbf{u} = 2^{-53}$, si $n < 2^{49} \approx 5 \cdot 10^{14}$, alors on obtient à arrondi fidèle

ratio gauche-droite/droite-gauche ≈ 1.07

Conclusion

- Algorithme compensé pour le produit
- Arrondi fidèle
- Borne d'erreur dynamique validée

Perspectives

- Arrondi au plus près pour le produit de nombres flottants

Merci de votre attention



Stef Graillat.

Accurate floating point product and exponentiation.

Technical report, 2007.

[hal-00164607](#).



Peter Kornerup, Christoph Lauter, Vincent Lefèvre, Nicolas Louvet,
and Jean-Michel Muller.

Computing correctly rounded integer powers in floating-point
arithmetic, 2008.

[ensl-00278430](#), version 1.



Peter Kornerup, Vincent Lefevre, and Jean-Michel Muller.

Computing integer powers in floating-point arithmetic, 2007.

[arXiv :0705.4369v1 \[cs.NA\]](#).



Philippe Langlois and Nicolas Louvet.

How to ensure a faithful polynomial evaluation with the compensated horner algorithm.

In Proceedings of the 18th IEEE Symposium on Computer Arithmetic (ARITH '07), Montpellier, France, pages 141–149. IEEE Computer Society, Los Alamitos, CA, USA, 2007.



Philippe Langlois and Nicolas Louvet.

More instruction level parallelism explains the actual efficiency of compensated algorithms, 2007.

hal-00165020, version 1.



Christoph Quirin Lauter.

Basic building blocks for a triple-double intermediate format.

Research Report RR-5702, INRIA, September 2005.



Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi.
Accurate sum and dot product.

SIAM J. Sci. Comput., 26(6) :1955–1988, 2005.



Siegfried M. Rump, Takeshi Ogita, and Shin'ichi Oishi.
Accurate floating-point summation.

Technical Report 05.12, Faculty for Information and Communication
Sciences, Hamburg University of Technology, nov 2005.