Produit précis de nombres flottants

Stef Graillat

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

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

Motivations

- 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 representé 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

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

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

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

Algorithmes de Dekker (1971) et Knuth (1974)

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

function $[x, y] = \texttt{FastTwoSum}(a, b)$

$x = \mathrm{fl}(a + b)$

$y = \mathrm{fl}((a - x) + b)$

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

function $[x, y] = \texttt{TwoSum}(a, b)$

$x = \mathrm{fl}(a + b)$

$z = \mathrm{fl}(x - a)$

$y = \mathrm{fl}((a - (x - z)) + (b - z))$

Transformations exactes pour le produit (1/3)

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

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 $[x, y] = \texttt{Split}(a, b)$

$\mathrm{factor} = \mathrm{fl}(2^s + 1)$ % $\mathbf{u} = 2^{-p}$, $s = \lceil p/2 \rceil$

$c = \mathrm{fl}(\texttt{factor} \cdot a)$

$x = \mathrm{fl}(c - (c - a))$

$y = \mathrm{fl}(a - x)$

Transformations exactes pour le produit (2/3)

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

function $[x, y] = \texttt{TwoProduct}(a, b)$

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

$[a_1, a_2] = \texttt{Split}(a)$

$[b_1, b_2] = \texttt{Split}(b)$

$y = \mathrm{fl}(a_2 \cdot b_2 - (((x - a_1 \cdot b_1) - a_2 \cdot b_1) - a_1 \cdot b_2))$

Cet algorithme nécessite 17 flops

Transformations exactes pour le produit (3/3)

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

- $\texttt{FMA}(a, b, c)$ 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 $[x, y] = \texttt{TwoProductFMA}(a, b)$
  $x = \mathrm{fl}(a \cdot b)$
  $y = \texttt{FMA}(a, b, -x)$

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

Résumé

Théorème 1

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

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

L'algorithme `TwoSum` *nécessite* 6 *flops.*

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

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

L'algorithme `TwoProduct` *nécessite* 17 *flops.*

Algorithme récursif pour le produit

La méthode classique pour l'évaluation du produit de n nombres $a = (a_1, a_2, \ldots, a_n)$

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

est l'algorithme suivant.

Algorithme 6 (Algorithme récursif)

function `res` = `Prod`(a)

$p_1 = a_1$

for $i = 2 : n$

$p_i = \mathrm{fl}(p_{i-1} \cdot a_i)$ % erreur d'arrondi π_i

end

`res` $= p_n$

Cet algorithme nécessite $n - 1$ flops

Analyse d'erreur

Notation

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

Une borne d'erreur directe est

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

Une borne d'erreur validée en flottant est

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

Algorithme compensé pour le calcul du produit (1/2)

Algorithme 7 (Schéma compensé pour le produit)

function $\texttt{res} = \texttt{CompProd}(a)$

$p_1 = a_1$

$e_1 = 0$

for $i = 2 : n$

$[p_i, \pi_i] = \texttt{TwoProduct}(p_{i-1}, a_i)$

$e_i = \mathsf{fl}(e_{i-1} a_i + \pi_i)$

end

$\texttt{res} = \mathsf{fl}(p_n + e_n)$

Cet algorithme nécessite $19n - 18$ flops

Algorithme compensé pour le calcul du produit (2/2)

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

function $\texttt{res} = \texttt{CompProdFMA}(a)$

$p_1 = a_1$

$e_1 = 0$

for $i = 2 : n$

$[p_i, \pi_i] = \texttt{TwoProductFMA}(p_{i-1}, a_i)$

$e_i = \texttt{FMA}(e_{i-1}, a_i, \pi_i)$

end

$\texttt{res} = \mathrm{fl}(p_n + e_n)$

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,*

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

Conditionnement pour le produit :

$$\mathrm{cond}(a) = \lim_{\varepsilon \longrightarrow 0} \sup \left\{ \frac{|(a_1 + \Delta a_1) \cdots (a_n + \Delta a_n) - a_1 \cdots a_n|}{\varepsilon |a_1 a_2 \cdots a_n|} : |\Delta a_i| \leq \varepsilon |a_i| \right\}$$

Un calcul standard montre que

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

Simulations numériques

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
$\longrightarrow$ cela est en partie dû à un meilleur parallèlisme d'instructions [Langlois-Louvet,2007].

Borne d'erreur validée

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*

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

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

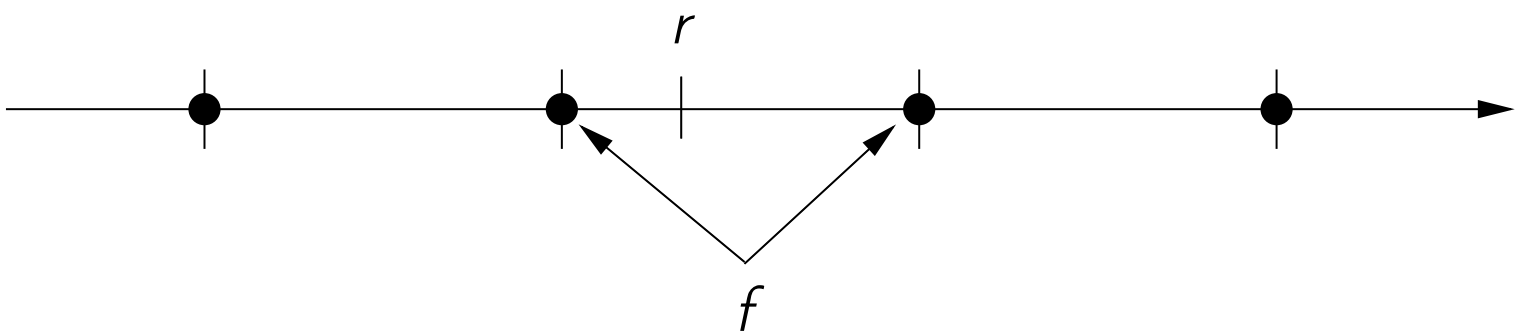
Un nombre flottant $f \in \mathbb{F}$ est un arrondi fidèle du nombre réel $r \in \mathbb{R}$ si

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

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* $\widetilde{r} := \mathsf{fl}(r)$. *Supposons que* $2|\delta| < \mathbf{u}|\widetilde{r}|$. *Alors* $\widetilde{r} \in \square(r + \delta)$, *ce qui signifie que* $\widetilde{r}$ *est un arrondi fidèle de* $r + \delta$.

Arrondi fidèle (3/3)

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

Lemme 3

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

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

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

Borne d'erreur validée et arrondi fidèle

Si

$$\mathrm{fl}\left(2\frac{\gamma_n\gamma_{2n}|a_1a_2\cdots a_n|}{1-(n+3)\mathbf{u}}\right) < \mathrm{fl}(\mathbf{u}|\mathtt{res}|)$$

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

Exponentiation - un algorithme linéaire

Algorithme 9 (Exponentiation par un schéma compensé)

function `res` = `CompLinPower`(x, n)

$p_1 = x$

$e_1 = 0$

for $i = 2 : n$

$[p_i, \pi_i] =$ `TwoProduct`(p_{i-1}, x)

$e_i = \mathsf{fl}(e_{i-1}x + \pi_i)$

end

`res` $= \mathsf{fl}(p_n + e_n)$

Complexité : $\mathcal{O}(n)$

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

Bibliothèque double-double (1/2)

Algorithme 10 (Multiplication de deux double-double)

function $[r_h, r_l] = \texttt{prod\_dd\_dd}(a_h, a_l, b_h, b_l)$

$[t_1, t_2] = \texttt{TwoProduct}(a_h, b_h)$

$t_3 = \mathrm{fl}(((a_h \cdot b_l) + (a_l \cdot b_h)) + t_2)$

$[r_h, r_l] = \texttt{TwoProduct}(t_1, t_3)$

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

function $[r_h, r_l] = \texttt{prod\_dd\_d}(a, b_h, b_l)$

$[t_1, t_2] = \texttt{TwoProduct}(a, b_h)$

$t_3 = \mathrm{fl}((a \cdot b_l) + t_2)$

$[r_h, r_l] = \texttt{TwoProduct}(t_1, t_3)$

Bibliothèque double-double (2/2)

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 $\longrightarrow$ 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 : $\mathcal{O}(\log n)$

Un algorithme logarithmique (2/2)

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 et perspectives

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

Références I


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].

Références II


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.

Références III


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.