

Calcul haute performance : programmation et algorithmique avancées (HPCA)

Reproductibilité numérique et HPC

Stef Graillat

LIP6/PEQUAN - Sorbonne Université

Cours de Master 2 SFPN / MAIN5



Plan de l'exposé

- 1 Introduction - motivations
- 2 Arithmétique flottante IEEE 754
- 3 Reproductibilité numérique et HPC

Plan de l'exposé

- 1 Introduction - motivations
- 2 Arithmétique flottante IEEE 754
- 3 Reproductibilité numérique et HPC

L'ordinateur calcule mal?

- Bug du Pentium : problème dans le calcul des divisions (erreur dans 1 cas sur 10^{10}). Par exemple $8391667/12582905$ donnait $0.666869\dots$ au lieu de $0.666910\dots$.
Erreur dans l'algorithme du calcul de division
- Excel, dans la version 3.0 à 7.0 : l'entrée de 1.40737488355328 affiche 0.64 !
- Calculatrice de Windows 3.1 : $2.01 - 2.00 = 0.0$!
- dans Maple 7.0 le calcul de

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

donne 1 au lieu de 5001

- dans la version 6.0 de Maple, si on entrait 214748364810 , on obtenait 10

Augmenter la précision ?

Polynôme de Rump :

$$f(x, y) = 333.75y^6 + x^2(11x^2y^2 - y^6 - 121y^4 - 2) + 5.5y^8 + \frac{x}{2y}$$

Sur un IBM 370, on a pour $x = 77617$ et $y = 33096$, on calcule $f(x, y)$:

| précision | valeur calculée |
|-----------|-------------------|
| simple | 1.172603 |
| double | 1.1726039400531 |
| étendue | 1.172603940053178 |

Mais la bonne valeur est $-0.827396059946821368141\cdots$

Plan du cours

- 1 Introduction - motivations
- 2 Arithmétique flottante IEEE 754
- 3 Reproductibilité numérique et HPC

Représentation flottante

Un nombre x est représenté en virgule flottante en base B par :

- son **signe** s_x (0 pour x positif, 1 pour x négatif)
- sa **mantisso** m_x de $n + 1$ chiffres
- son **exposant** e_x , un entier de k chiffres compris entre e_{\min} et e_{\max}

tels que

$$x = (-1)^{s_x} \times m_x \times B^{e_x}$$

avec $m_x = x_0.x_1x_2 \dots x_n$ où $x_i \in \{0, 1, \dots, B - 1\}$.

Pour obtenir la meilleure précision à longueur de mantisse n fixée, la mantisse est **normalisée**, c'est-à-dire que $x_0 \neq 0$, $m_x \in [1, B[$.

Il faut donc un **codage spécial** pour 0.

Révolution en 1985 : la norme IEEE-754

Après de nombreuses années de travail, une **norme** fixe la représentation des données et le comportement des opérations de base en virgule flottante.

Cette norme fixe :

- les **formats** des données
- les **valeurs spéciales**
- les **modes d'arrondi**
- la **précision** des opérations de base
- les règles de **conversion**

En fait, il y a deux normes :

754 *ANSI/IEEE Standard for Binary Floating-Point Arithmetic* en 1985

854 *ANSI/IEEE Standard for Radix-Independent Floating-Point Arithmetic* en 1987 (où $B = 2$ ou 10)

IEEE 754-2008, *ANSI/IEEE Standard for Binary Floating-Point Arithmetic* en 2008

Objectifs de la norme IEEE-754

- permettre de faire des programmes portables
- rendre les programmes déterministes d'une machine à une autre
- conserver des propriétés mathématiques
- permettre/imposer l'arrondi correct
- permettre/imposer des conversions fiables
- faciliter la construction de preuves
- faciliter la gestion des exceptions (fonctions partielles)
- faciliter les comparaisons (unicité de la représentation, sauf pour 0)
- permettre un support pour l'arithmétique d'intervalle

Norme IEEE-754 : Formats de base

$$B = 2$$

| format | nombre de bits | | | |
|------------------|----------------|-------|-------------------------------|----------|
| | total | signe | mantisse | exposant |
| simple précision | 32 | 1 | (1 implicite +) 23 (fraction) | 8 |
| double précision | 64 | 1 | (1 implicite +) 52 (fraction) | 11 |

La **mantisse** (normalisée) m_x du nombre flottant x est représentée par $n + 1$ bits :

$$m_x = 1. \underbrace{x_1 x_2 x_3 \dots x_{n-1}}_{f_x} x_n$$

où les x_i sont des bits.

La partie fractionnaire de m_x est appelée **fraction** (de n bits) : f_x . On a alors : $m_x = 1 + f_x$ et $1 \leq m_x < 2$.

Norme IEEE-754 : exposant

L'exposant e_x est un entier signé de k bits tel que $e_{\min} \leq e_x \leq e_{\max}$

Différentes représentations sont possibles :

- complément à 2,
- signe et magnitude,
- biaisée.

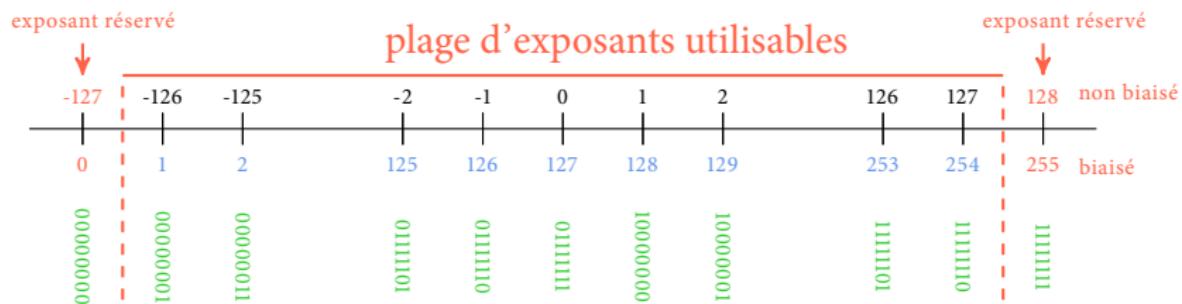
Norme IEEE-754 : choix de la **représentation biaisée** stockée avant la mantisse car cela permet de faire les comparaisons entre flottants dans l'ordre lexicographique (en laissant le signe s_x de côté) et de représenter le nombre 0 avec $e_x = f_x = 0$.

L'exposant stocké physiquement est l'**exposant biaisé eb** tel que $eb_x = e_x + b$ où **b** est le **biais**.

Norme IEEE-754 : exposants réservés

Les exposants non biaisés $e_{\min} - 1$ et $e_{\max} + 1$ (respectivement 0 et $2^k - 1$ en biaisé) sont réservés pour zéro, les dénormalisés et les valeurs spéciales.

| format | taille k | biais b | non-biaisée e_{\min} | e_{\max} | biaisée eb_{\min} | eb_{\max} |
|--------|---------------|-------------------------|---------------------------|------------|------------------------|-------------|
| SP | 8 | $127 (= 2^{8-1} - 1)$ | -126 | 127 | 1 | 254 |
| DP | 11 | $1023 (= 2^{11-1} - 1)$ | -1022 | 1023 | 1 | 2046 |



Norme IEEE-754 : représentation de zéro

$$e_0 = f_0 = 0, s_0 = ?$$

Deux représentations différentes : **-0** et **+0**.

Ce qui est cohérent avec le fait qu'il y ait deux infinis distincts. On a alors : $\frac{1}{+0} = +\infty$ et $\frac{1}{-0} = -\infty$.

La norme impose que le test $-0 = +0$ retourne la valeur vrai.

En simple précision, la représentation machine de $+0$ et -0 est donc :

| x | s_x | eb_x | m_x |
|-----|-------|----------|----------------------------------|
| -0 | 1 | 00000000 | 00000000000000000000000000000000 |
| +0 | 0 | 00000000 | 00000000000000000000000000000000 |

Norme IEEE-754 : valeurs spéciales

- Les **infinis**, $-\infty$ et $+\infty$: $e_x = e_{max} + 1$ et $f_x = 0$.
- Not a Number : NaN (exception, fonction partielle)

Le résultat d'une opération invalide telle que $0/0$, $\sqrt{-1}$ ou $0 \times +\infty$:
 $e_x = e_{max} + 1$ et $f_x \neq 0$.

Les NaN se propagent dans les calculs.

Norme IEEE-754 : valeurs spéciales

- Les **infinis**, $-\infty$ et $+\infty$: $e_x = e_{max} + 1$ et $f_x = 0$.
- Not a Number : NaN (exception, fonction partielle)

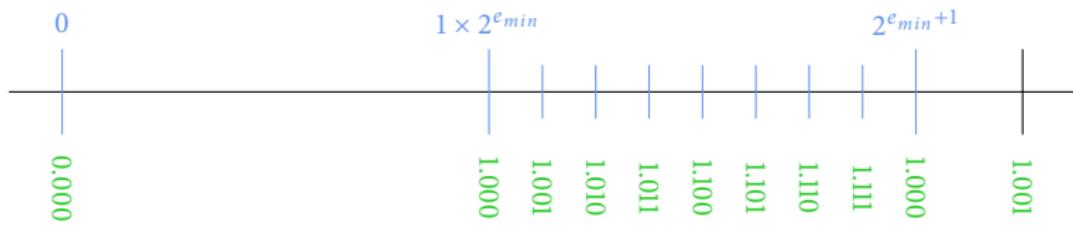
Le résultat d'une opération invalide telle que $0/0$, $\sqrt{-1}$ ou $0 \times +\infty$:
 $e_x = e_{max} + 1$ et $f_x \neq 0$.

Les NaN se **propagent** dans les calculs.

En simple précision, on a donc :

| x | s_x | eb_x | m_x |
|-----------|-------|----------|--------------------------------------|
| $-\infty$ | 1 | 11111111 | 0000000000000000000000000000 |
| $+\infty$ | 0 | 11111111 | 0000000000000000000000000000 |
| NaN | 0 | 11111111 | 0000100100111000000100 (par exemple) |

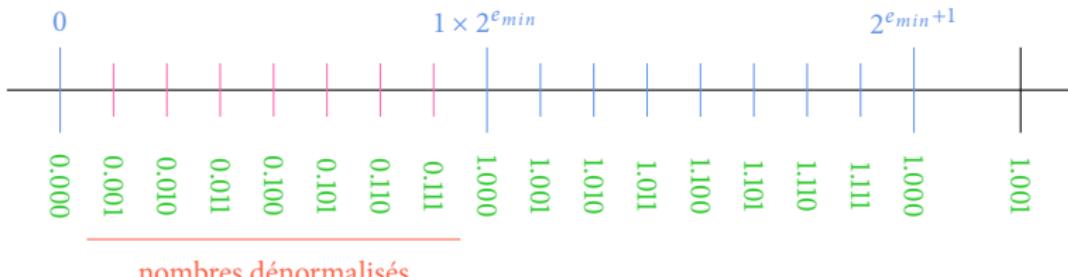
Norme IEEE-754 : autour de zéro



Le plus petit nombre positif normalisé x est tel que $e_x = e_{\min}$, $f_x = 0$, donc $x = 2^{e_{\min}}$.

Le « 1 implicite » dans la mantisse implique qu'il n'y a pas de nombre représentable entre $0 \times 2^{e_{\min}}$ et $1 \times 2^{e_{\min}}$ alors qu'il y en a 2^n entre $1 \times 2^{e_{\min}}$ et $2 \times 2^{e_{\min}}$.

Norme IEEE-754 : autour de zéro, les nombres dénormalisés



Le plus petit nombre positif normalisé x est tel que $e_x = e_{\min}$, $f_x = 0$, donc $x = 2^{e_{\min}}$.

Le « 1 implicite » dans la mantisse implique qu'il n'y a pas de nombre représentable entre $0 \times 2^{e_{\min}}$ et $1 \times 2^{e_{\min}}$ alors qu'il y en a 2^n entre $1 \times 2^{e_{\min}}$ et $2 \times 2^{e_{\min}}$.

L'objectif des **nombres dénormalisés** est d'uniformiser la répartition des nombres représentables autour de 0.

On autorise près de zéro des nombres de la forme $(-1)^{s_x} \times 0.f_x \times 2^{e_{\min}}$: $eb_x = 0$, m_x ne suit pas la convention du « 1 implicite ».

Norme IEEE-754

nombres normalisés et exposants réservés en bref

| Exposant non biaisé | Mantisse | Commentaire |
|-----------------------------------|--------------|---|
| $e_{\min} \leq e_x \leq e_{\max}$ | | nombre normalisé, convention du « 1 implicite » |
| $e_x = e_{\min} - 1$ | $m_x = 0$ | zéro |
| | $m_x \neq 0$ | nombre dénormalisé, proche de zéro, pas de convention du « 1 implicite » |
| $e_x = e_{\max} + 1$ | $m_x = 0$ | infini $(-1)^{s_x} \infty$ |
| | $m_x \neq 0$ | NaN, résultat d'une opération invalide, se propage dans le calcul, $\text{NaN} \neq \text{NaN}$ |

L'utilisation d'exposants réservés conduit à des calculs plus coûteux.

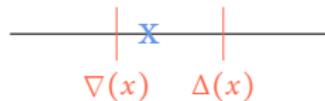
Norme IEEE-754 : nécessité d'arrondir

Si x et y sont deux nombres exactement représentables en machine, alors le résultat d'une opération $res = x \odot y$ n'est, en général, pas représentable en machine.

Par exemple, en base $B = 10$, le nombre $1/3$ n'est pas représentable avec un nombre fini de chiffres.

Il faut **arrondir** le résultat, c'est-à-dire renvoyer un des nombres représentables voisins.

Norme IEEE-754 : modes d'arrondis



La norme propose 4 modes d'arrondi :

- arrondi **vers $+\infty$** (ou par excès), noté $\Delta(x)$: retourne le plus petit nombre machine supérieur ou égal au résultat exact x
- arrondi **vers $-\infty$** (ou par défaut), noté $\nabla(x)$: retourne le plus grand nombre machine inférieur ou égal au résultat exact x
- arrondi **vers 0**, noté $\mathcal{Z}(x)$: retourne $\Delta(x)$ pour les nombres négatifs et $\nabla(x)$ pour les positifs
- arrondi **au plus près**, noté $\circ(x)$: retourne le nombre machine le plus proche du résultat exact x (pour le milieu de deux nombres machine consécutifs on choisit celui dont la mantisse se termine par un 0, on parle d'**arrondi pair**)

Les trois premiers modes d'arrondis sont dits **dirigés**.

Norme IEEE-754 : propriété de l'arrondi correct

Soient x et y deux nombres exactement représentables en machine, \odot une des opérations rationnelles $+$, $-$, \times , $/$ et \diamond le mode d'arrondi choisi parmi les 4 modes IEEE.

La norme IEEE exige que le résultat d'une opération $x \odot y$ soit égal à $\diamond(x \odot_{exact} y)$. Le résultat doit être le même que si on effectuait le calcul en précision infinie puis on arrondissait ce résultat.

Idem pour la racine carrée.

C'est la propriété de l'arrondi correct.

La norme IEEE décrit un algorithme pour l'addition, la soustraction, la multiplication, la division et la racine carrée et exige que ses implémentations produisent le même résultat que cet algorithme.

Un résumé de la norme IEEE-754 en chiffres

| | exposant | fraction | valeur |
|--------------|-------------------------------|------------|-------------------------------|
| normalisés | $e_{min} \leq e \leq e_{max}$ | $f \geq 0$ | $\pm(1.f) \times 2^e$ |
| dénormalisés | $e = e_{min} - 1$ | $f > 0$ | $\pm(0.f) \times 2^{e_{min}}$ |
| zéro (signé) | $e = e_{min} - 1$ | $f = 0$ | ± 0 |
| infinis | $e = e_{max} + 1$ | $f = 0$ | $\pm\infty$ |
| Not a Number | $e = e_{max} + 1$ | $f > 0$ | NaN |

| format | # bits total | k | n | e_{min} | e_{max} | b |
|------------------|--------------|-----|-----|-----------|-----------|------|
| simple précision | 32 | 8 | 23 | -126 | 127 | 127 |
| double précision | 64 | 11 | 52 | -1022 | 1023 | 1023 |

| valeur | simple précision | double précision |
|----------------------------|------------------------------|---------------------------------------|
| plus grand normalisé > 0 | $3.40282347 \times 10^{38}$ | $1.7976931348623157 \times 10^{308}$ |
| plus petit normalisé > 0 | $1.17549435 \times 10^{-38}$ | $2.2250738585072014 \times 10^{-308}$ |
| plus grand dénormalisé > 0 | $1.17549421 \times 10^{-38}$ | $2.2250738585072009 \times 10^{-308}$ |
| plus petit dénormalisé > 0 | $1.40129846 \times 10^{-45}$ | $4.9406564584124654 \times 10^{-324}$ |

Précision des calculs

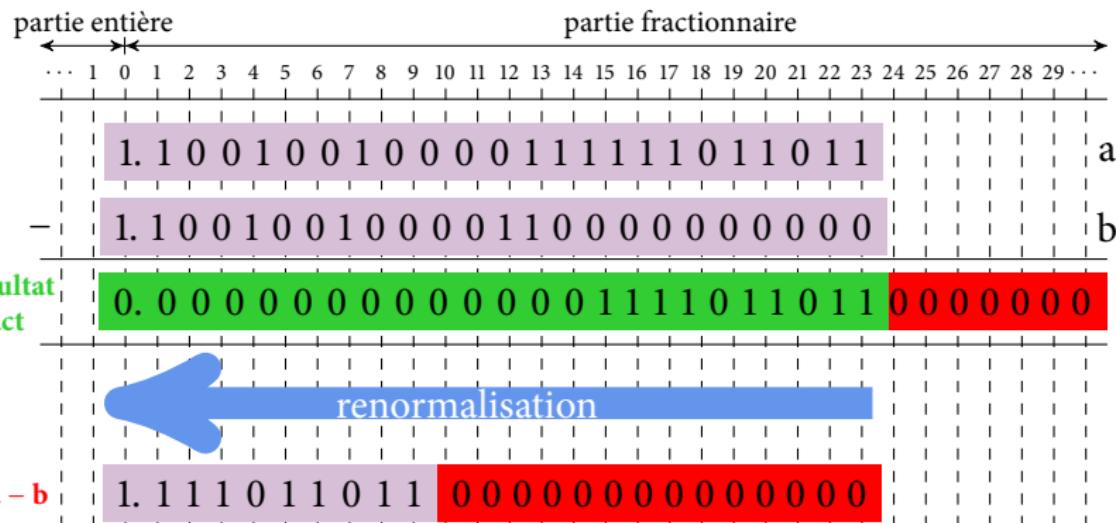
À chaque arrondi, on perd a priori un peu de précision, on parle d'**erreur d'arrondi**.

Même si une opération isolée retourne le meilleur résultat possible (l'arrondi du résultat exact), une suite de calculs peut conduire à d'importantes erreurs du fait du cumul des erreurs d'arrondi.

Les deux sources principales d'erreur d'arrondis au cours des calculs sont la **cancellation** et l'**absorption**. Mais ces erreurs n'interviennent qu'après des mesures physiques approximatives, une modélisation physique hasardeuse et un algorithme de résolution plus ou moins approchée et correctement implémenté.

Phénomène de *cancellation* (ou élimination)

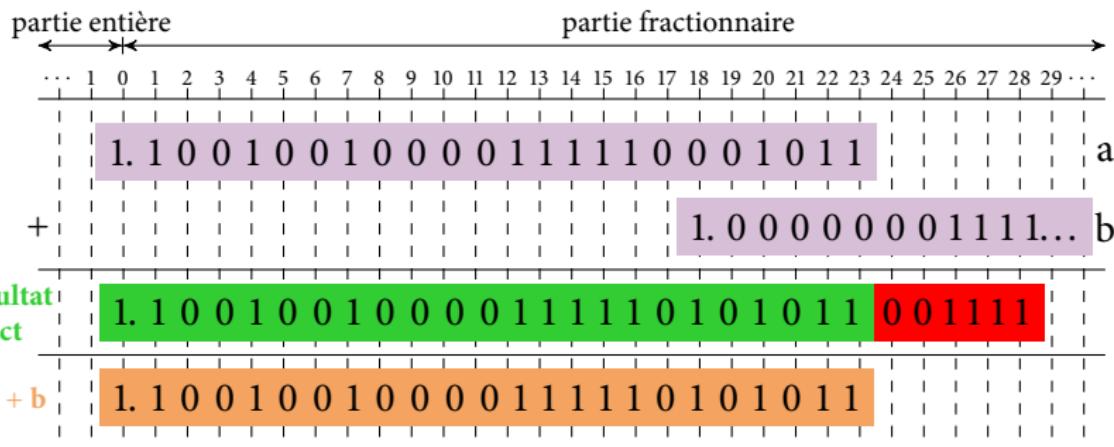
Lors de la soustraction de deux nombres très proches.



Si les opérandes sont eux-mêmes des résultats de calculs avec des erreurs d'arrondi, les 0 ajoutés à droite (partie rouge) sont faux. La cancellation est catastrophique quand il n'y a presque plus de chiffres significatifs.

Phénomène d'absorption

Lors de l'addition de deux nombres ayant des ordres de grandeur très différents où l'on peut « perdre » toute l'information du plus petit des deux nombres.



Combiner avec la *cancellation* ce phénomène peut tourner à la catastrophe. Par exemple en simple précision, si $a = 1$ et $b = 2^{-30}$ alors : $a \oplus b = 1$ et donc $(a \oplus b) \ominus a = 0$.

Exemple du phénomène d'absorption

On calcule numériquement, pour de grandes valeurs de N , la somme :

$$\sum_{i=1}^N \frac{1}{i}$$

Résultats d'un programme C (flottant SP) sur un processeur Pentium4 :

| ordre | N | | | |
|-------------------|--------------|--------------|--------------|--------------|
| | 10^5 | 10^6 | 10^7 | 10^8 |
| exacte | 1.209015e+01 | 1.439273e+01 | 1.669531e+01 | 1.899790e+01 |
| $1 \rightarrow N$ | 1.209085e+01 | 1.435736e+01 | 1.540368e+01 | 1.540368e+01 |
| $N \rightarrow 1$ | 1.209015e+01 | 1.439265e+01 | 1.668603e+01 | 1.880792e+01 |

Multiplication-Addition Fusionnée (FMA)

Le FMA (fused multiply and add) permet de faire le calcul suivant en une seule instruction (au lieu de deux)

$$r = a + b \times c.$$

On a donc **un seul arrondi**

Pratique pour l'évaluation d'un produit scalaire et l'évaluation polynomiale via Horner.

Transformations exactes pour la somme

On travaille en arrondi au plus près

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

Algorithme de Dekker (1971)

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

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

Transformations exactes pour la somme

Algorithme de Knuth (1974)

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 (1/2)

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

Algorithme TwoProduct de Veltkamp et Dekker (1971)

$a = x + y$ et x et y ne se chevauchent pas avec $|y| \leq |x|$.

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

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

Transformations exactes pour le produit (2/2)

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))
```

Transformations exactes pour le produit avec FMA

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 le produit de 2 flottants)

```
function [x, y] = TwoProductFMA(a, b)
    x = fl(a · b)
    y = FMA(a, b, -x)
```

Représentation : les expansions

Un nombre flottant multiprécision est représenté comme une somme non évaluée de nombres flottants

$$\sum_{i=0}^n f_i$$

où les f_i sont des nombres flottants et si possible tels que les mantisses ne se chevauchent pas.

La bibliothèque double-double

Un nombre double-double est une paire (a_h, a_l) de flottants IEEE 754 vérifiant $a = a_h + a_l$ et $|a_l| \leq \mathbf{u}|a_h|$.

Algorithme 6

Addition d'un double b et d'un double-double (a_h, a_l)

```
function [ch, cl] = add_dd_d(ah, al, b)
    [th, tl] = TwoSum(ah, b)
    [ch, cl] = FastTwoSum(th, (tl ⊕ al))
```

La bibliothèque double-double

Algorithme 7

Produit d'un double-double (a_h, a_l) par un double b

function $[c_h, c_l] = \text{prod_dd_d}(a_h, a_l, b)$

$[s_h, s_l] = \text{TwoProduct}(a_h, b)$

$[t_h, t_l] = \text{FastTwoSum}(s_h, (a_l \otimes b))$

$[c_h, c_l] = \text{FastTwoSum}(t_h, (t_l \oplus s_l))$

La bibliothèque double-double

Algorithme 8

Addition d'un double-double (a_h, a_l) avec un double-double (b_h, b_l)

```
function [ch, cl] = add_dd_dd(ah, al, bh, bl)
    [sh, sl] = TwoSum(ah, bh)
    [th, tl] = TwoSum(al, bl)
    [th, sl] = FastTwoSum(sh, (sl ⊕ th))
    [ch, cl] = FastTwoSum(th, (tl ⊕ sl))
```

Si a, b sont des double-double et $\odot \in \{+, \times\}$, alors on peut montrer que

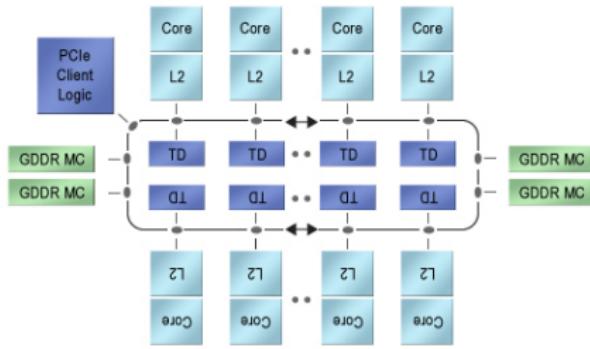
$$\text{fl}(a \odot b) = (1 + \delta)(a \odot b),$$

avec $|\delta| \leq 4 \cdot 2^{-106}$.

Plan de l'exposé

- 1 Introduction - motivations
- 2 Arithmétique flottante IEEE 754
- 3 Reproductibilité numérique et HPC

Du multi-coeurs au many-coeurs

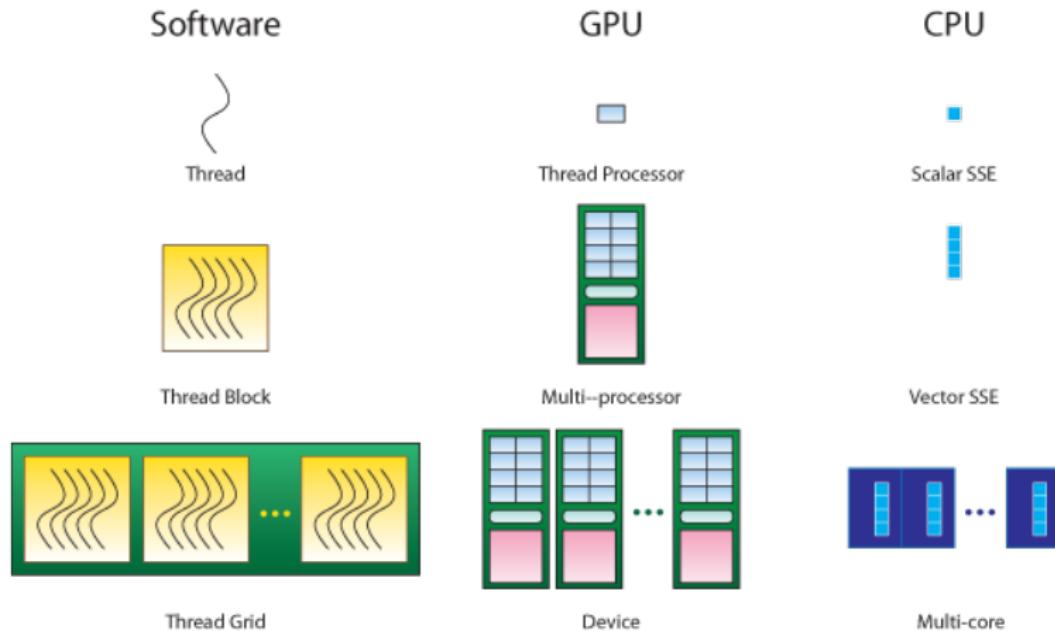


Intel Xeon Phi : 50 coeurs x86



NVIDIA K20 : 2496 coeurs CUDA

Exécution sur many-coeurs



Reproductibilité numérique

Les opérations en arithmétique flottante génèrent des erreurs d'arrondi

Les opérations ($+$, \times) sont commutatives mais pas associatives :

$$(-1 + 1) + 2^{-53} \neq -1 + (1 + 2^{-53}) \quad \text{en double précision.}$$

Conséquence : le résultat d'un calcul flottant dépend de l'ordre des opérations.

Reproductibilité : capacité à obtenir un résultat identique bit à bit sur les mêmes données en entrée pour différentes exécutions sur différents matériels.

Motivations pour la reproductibilité

Demande pour des calculs numériques reproductibles :

- débogage : analyse du code étape par étape et nécessité de relancer l'exécution du code plusieurs fois sur les mêmes données
- comprendre la fiabilité du résultat
- raisons contractuelles (sécurité, etc.)
- ...

Sources de non-reproductibilité

Une bibliothèque scientifique avec des calculs en précision finie est sujette à de la non-reproductibilité pour diverses raisons :

- changement dans la présentation des données :
 - données,
 - partitionnement des données,
 - ordre des données,
- changement dans les ressources matérielles :
 - utilisation du FMA,
 - utilisation de précision intermédiaire (64 bits, 80 bits, 128 bits, etc.),
 - chemin de données (SSE, AVX, GPU warp, etc.),
 - taille des lignes de cache,
 - nombre de processeurs,
 - topologie du réseau,
 - ...

Compiler options

```
#include <stdio.h>
int main(){
    volatile double a,b;
    double x,y;

    a=1.0/3;
    b=0.1/2048;

    x = a + b;
    y = (a - (a + b)) + b;
    if (y==0.0){
        printf("Error\n");
    }
    else {
        printf("Correct\n");
    }

    return 0;
}
```

> gcc -O3 -o repro repro.c
> ./repro
Correct

> gcc -O3 -ffast-math -o repro repro.c
> ./repro
Error

Reproductibilité numérique pour l'Exascale

Exascale : capacité à effectuer 10^{18} opérations flottantes par seconde en utilisant $\mathcal{O}(10^9)$ processeurs

- ordonnancement dynamique
- hétérogénéité du réseau
- temps de communication important

Coût = opérations arithmétiques + communications

Reproductibilité numérique pour l'Exascale

ExaScale Computing Study: Technology Challenges in Achieving Exascale Systems

Peter Kogge, Editor & Study Lead

Keren Bergman

Shekhar Borkar

Dan Campbell

William Carlson

William Dally

Monty Denneau

Paul Franzon

William Harrod

Kerry Hill

Jon Hiller

Sherman Karp

Stephen Keckler

Dean Klein

Robert Lucas

Mark Richards

Al Scarpelli

Steven Scott

Allan Snavely

Thomas Sterling

R. Stanley Williams

Katherine Yelick

September 28, 2008

This work was sponsored by DARPA IPTO in the ExaScale Computing Study with Dr. William Harrod as Program Manager; AFRL contract number **FA8650-07-C-7724**. This report is published in the interest of scientific and technical information exchange and its publication does not constitute the Government's approval or disapproval of its ideas or findings



NOTICE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation, or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

APPROVED FOR PUBLIC RELEASE, DISTRIBUTION UNLIMITED.

Reproductibilité numérique pour l'Exascale

March 2014

Applied Mathematics Research for Exascale Computing

The image features a dense mathematical equation in the center, which appears to be a system of partial differential equations related to fluid dynamics or computational fluid mechanics. The equation is framed by a grid of circuit board tracks. At the bottom left, there is a small logo for the U.S. Department of Energy's Office of Science Advanced Scientific Computing Research Program.

Exascale Mathematics Working Group

| | |
|-----------------|----------------|
| Jack Dongarra | co-chair, ORNL |
| Victor Eijkhout | co-chair, ORNL |
| John Bell | LBNL |
| Luis Caffarelli | UT Austin |
| Robert Byrd | SINUM |
| Michael Heston | SINUM |
| Paul Hovland | ANL |
| Elmer Jessup | LBNL |
| Cayley Williams | LBNL |
| Stefan Wild | ANL |

DOE ASCR Point of Contact:
Karen Papadimitriou

Sponsored by:
U.S. Department of Energy
Office of Science
Advanced Scientific Computing Research Program

 **Top Ten Exascale Research Challenges**

DOE ASCAC Subcommittee Report
February 10, 2014

U.S. DEPARTMENT OF ENERGY
Office of Science

Sponsored by the U.S. Department of Energy, Office of Science,
Office of Advanced Scientific Computing Research

Top 10 Challenges to Exascale

3 Hardware, 4 Software, 3 Algorithms/Math Related

.. Energy efficiency:

- Creating more energy efficient circuit, power, and cooling technologies.

.. Interconnect technology:

- Increasing the performance and energy efficiency of data movement.

.. Memory Technology:

- Integrating advanced memory technologies to improve both capacity and bandwidth.

.. Scalable System Software:

- Developing scalable system software that is power and resilience aware.

.. Programming systems:

- Inventing new programming environments that express massive parallelism, data locality, and resilience

.. Data management:

- Creating data management software that can handle the volume, velocity and diversity of data that is anticipated.

.. Scientific productivity:

- Increasing the productivity of computational scientists with new software engineering tools and environments.

.. Exascale Algorithms:

- Reformulating science problems and refactoring their solution algorithms for exascale systems.

.. Algorithms for discovery, design, and decision:

- Facilitating mathematical optimization and uncertainty quantification for exascale discovery, design, and decision making.

.. Resilience and correctness:

- Ensuring correct scientific computation in face of faults, reproducibility, and algorithm verification challenges.



Reproductibilité numérique

Source de non-reproductibilité numérique : les erreurs d'arrondi entraîne une dépendance du résultat calculé par rapport à l'ordre des calculs.

Pour être reproductible, on peut :

- fixer l'ordre des calculs :
 - en séquentiel : coût trop important sur des machines parallèles
 - arbre de réduction fixe : coût lié aux communications
- éliminer/réduire les erreurs d'arrondi :
 - arithmétique à virgule fixe : plage d'exposant limitée
 - augmenter la précision : reproductibilité avec une plus grande probabilité (mais pas certaine).
 - faire des calculs flottants sans erreur d'arrondi (pré-arrondi)
 - arithmétique exacte et précise (arrondi à la fin)

Reproductibilité numérique pour la sommation

But : on veut calculer $\sum_{i=1}^n x_i$ pour x_i des nombres flottants.

Algorithme 9 (Algorithme de sommation classique)

```
function res = Sum(x)
```

```
    s = 0;
```

```
    for i = 1 : n
```

```
        s = fl(s + xi)
```

```
    res = s
```

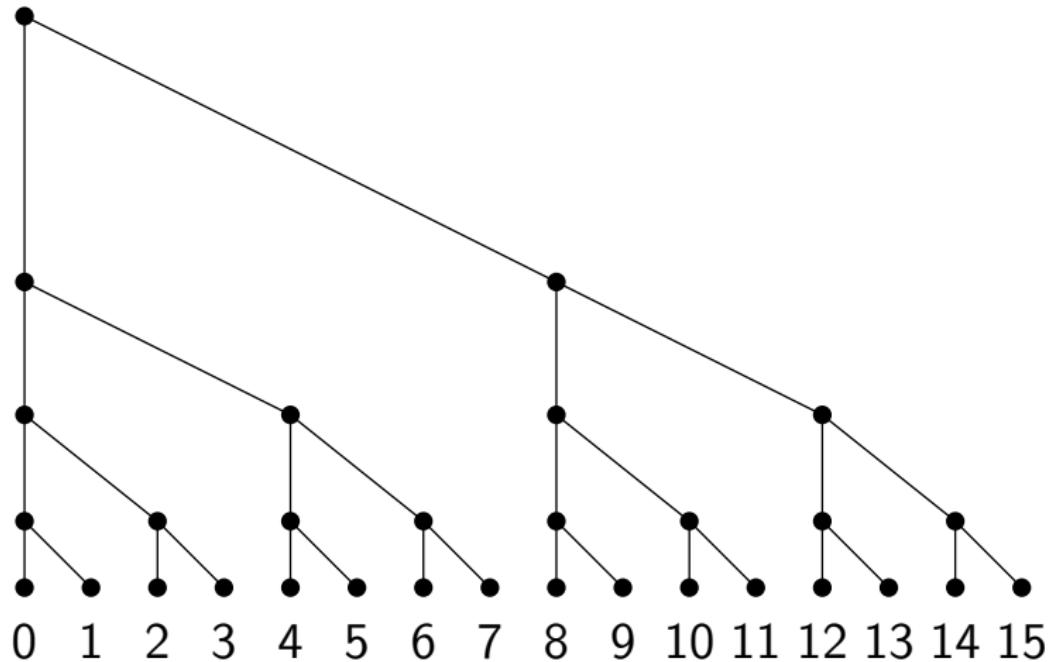
Arbre de réduction reproductible

Idée : fixer l'arbre de réduction dont la forme ne dépend pas des ressources

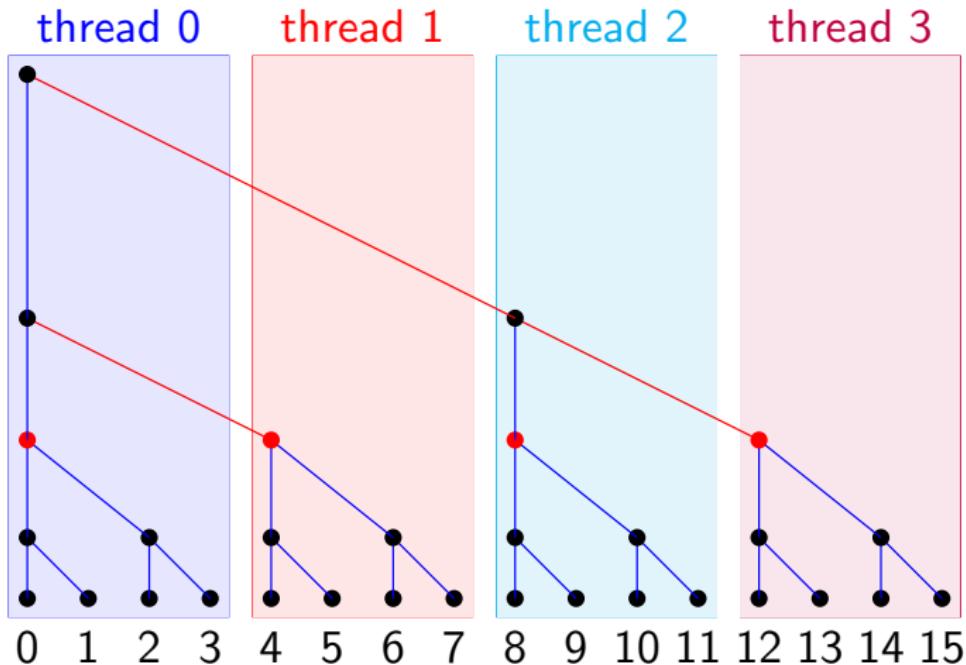
Stratégie :

- décomposer le vecteur en blocs de taille fixe
- imposer un arbre de réduction sur les blocs (et pas sur les threads)

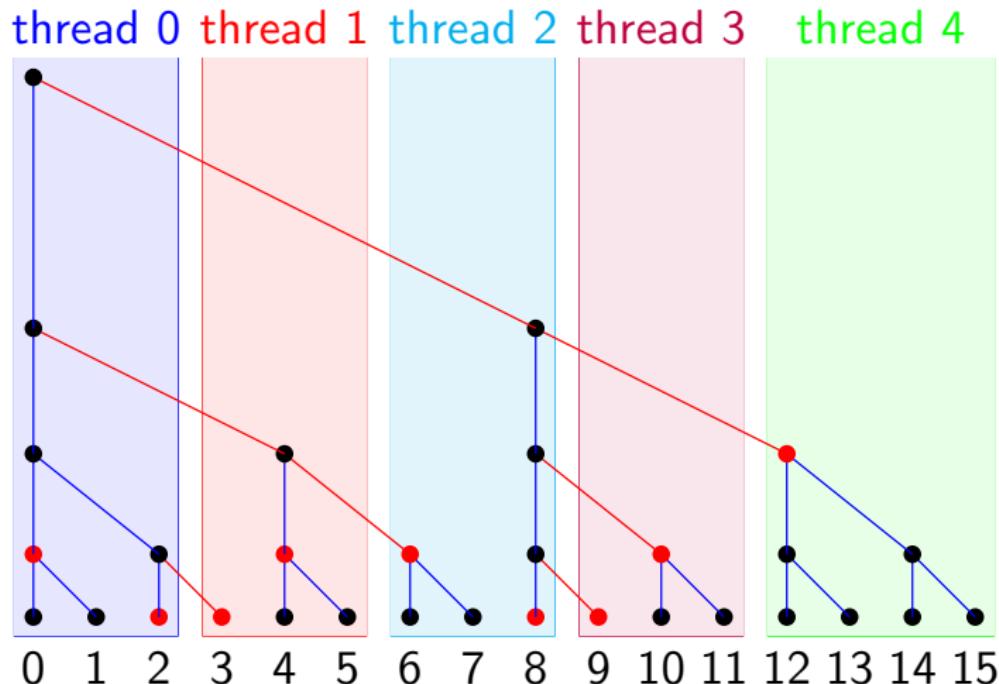
Arbre de réduction reproductible



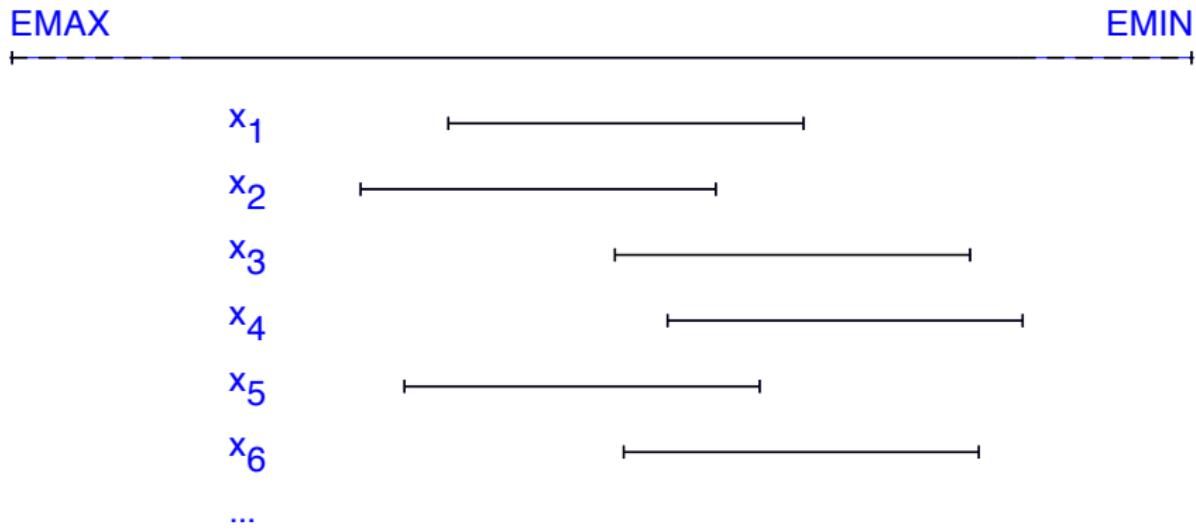
Arbre de réduction reproductible



Arbre de réduction reproductible

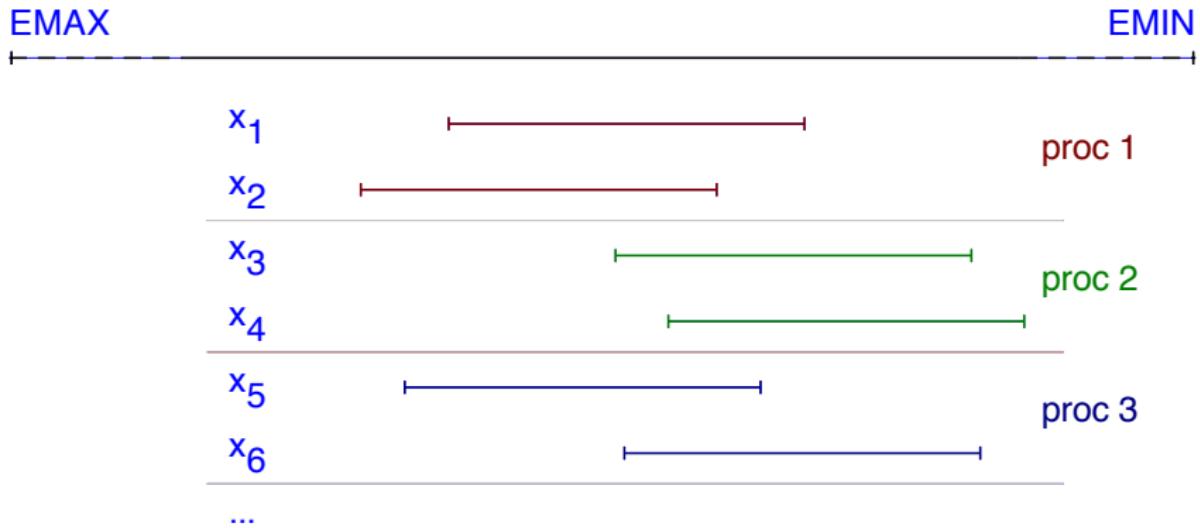


Méthode de pré-arrondi



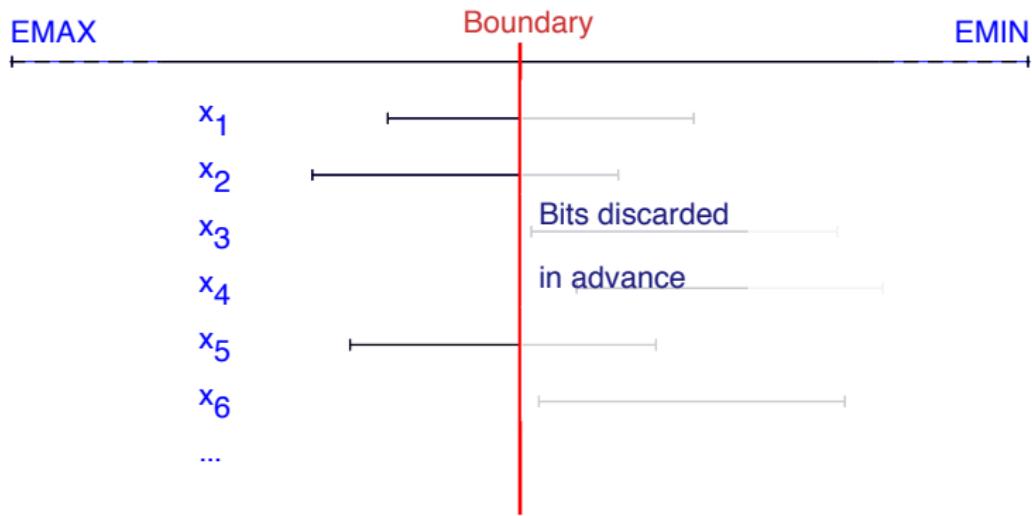
Des erreurs d'arrondi apparaissent à chaque addition. Les erreurs de calcul dépendent des résultats intermédiaires qui dépendent eux-même de l'ordre des calculs.

Méthode de pré-arrondi



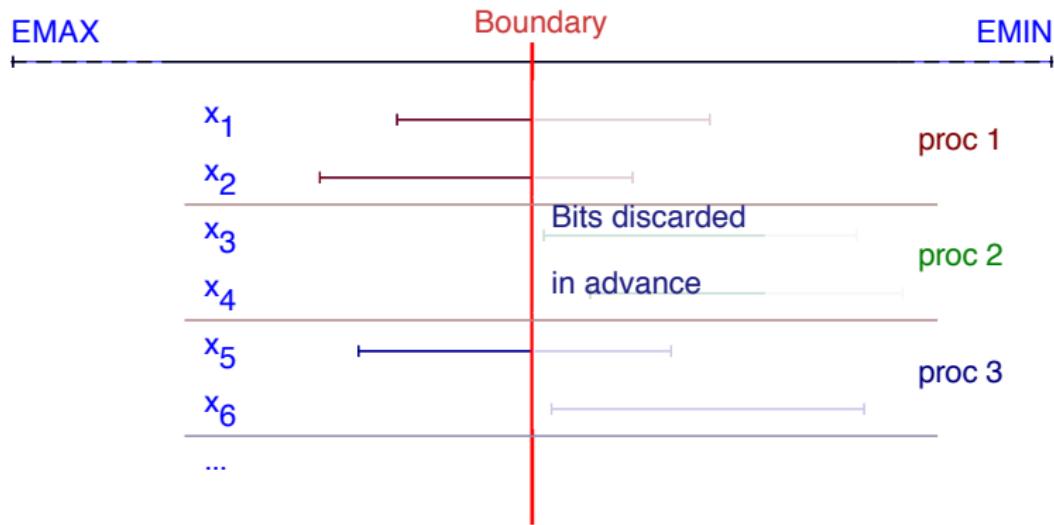
Des erreurs d'arrondi apparaissent à chaque addition. Les erreurs de calcul dépendent des résultats intermédiaires qui dépendent eux-même de l'ordre des calculs.

Méthode de pré-arrondi



Pas d'erreur d'arrondi à chaque addition. Les erreurs de calcul dépendent de la frontière qui dépende de $\max |x_i|$ et pas de l'ordre des calculs.

Méthode de pré-arrondi



Pas d'erreur d'arrondi à chaque addition. Les erreurs de calcul dépendent de la frontière qui dépende de $\max |x_i|$ et pas de l'ordre des calculs.

Solution avec superaccumulateur

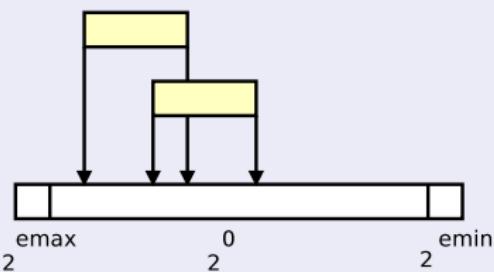
- Utiliser à la fois les expansions frottantes (FPE) et le superaccumulateur :
 - calculs rapides et précis avec des expansions flottantes (FPE)
 - calculs sans erreur avec le superaccumulateur si nécessaire

Algorithm 1 FPE de taille n

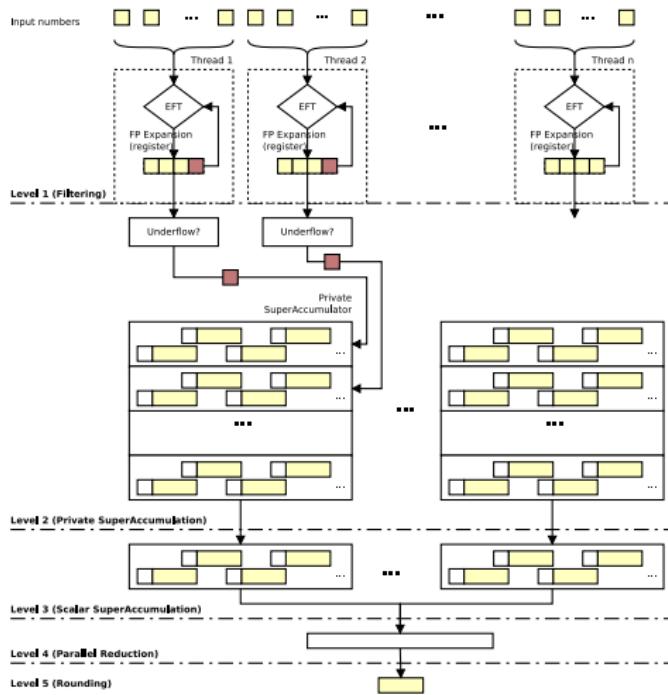
Function = ExpansionAccumulate(x)

```
1: for  $i = 0 : n - 1$  do
2:    $(a_i, x) \leftarrow \text{TwoSum}(a_i, x)$ 
3: end for
4: if  $x \neq 0$  then
5:   Superaccumulate( $x$ )
```

Superaccumulateur de Kulisch

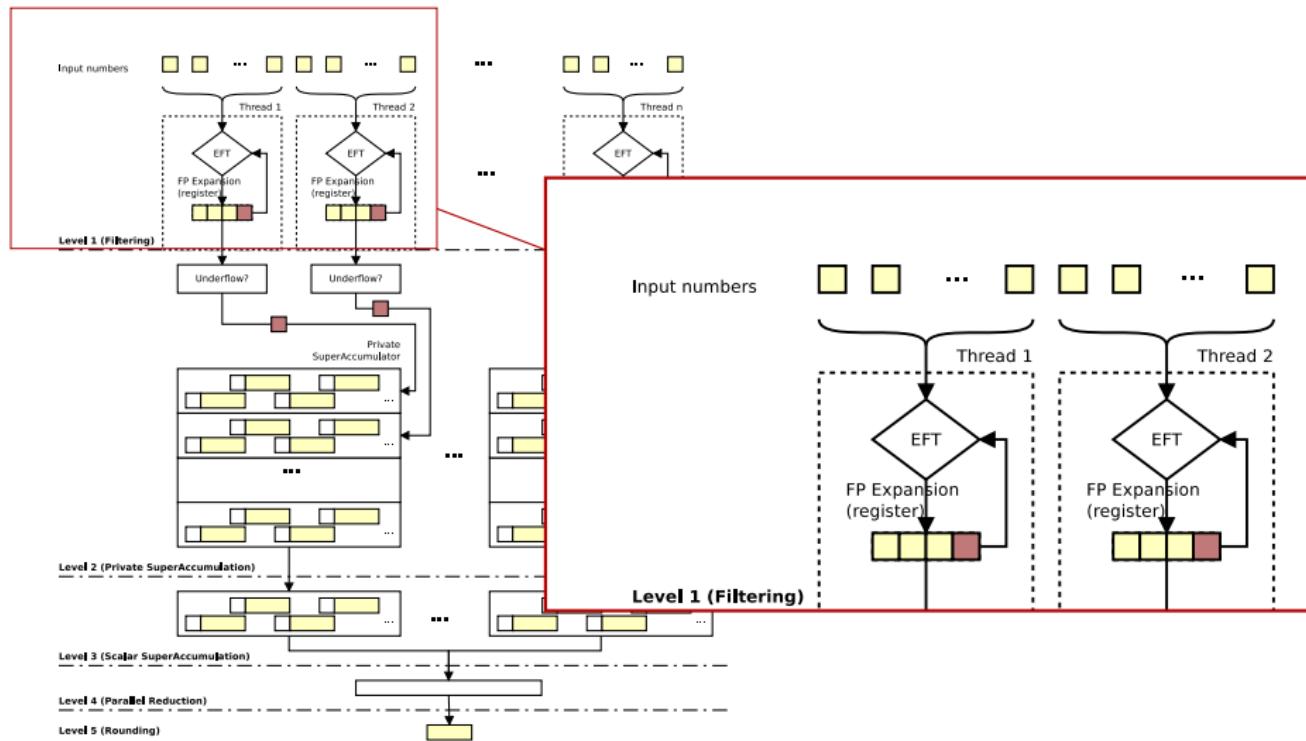


Algorithme de sommation reproductible multi-niveau

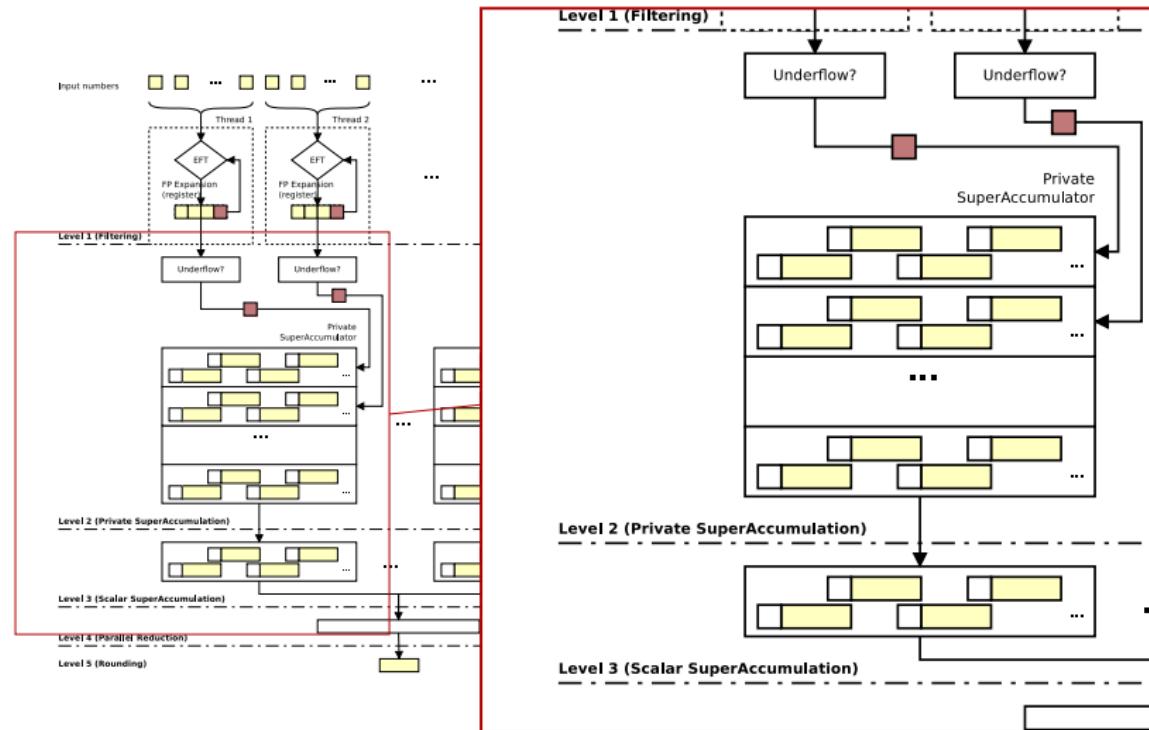


- Algorithmes parallèles à 5 niveaux
- Adaptés aux architectures actuelles
- Basé sur des expansions avec EFT et sur le superaccumulateur
- Garantit l'arrondi correct
- résultat reproductible bit à bit

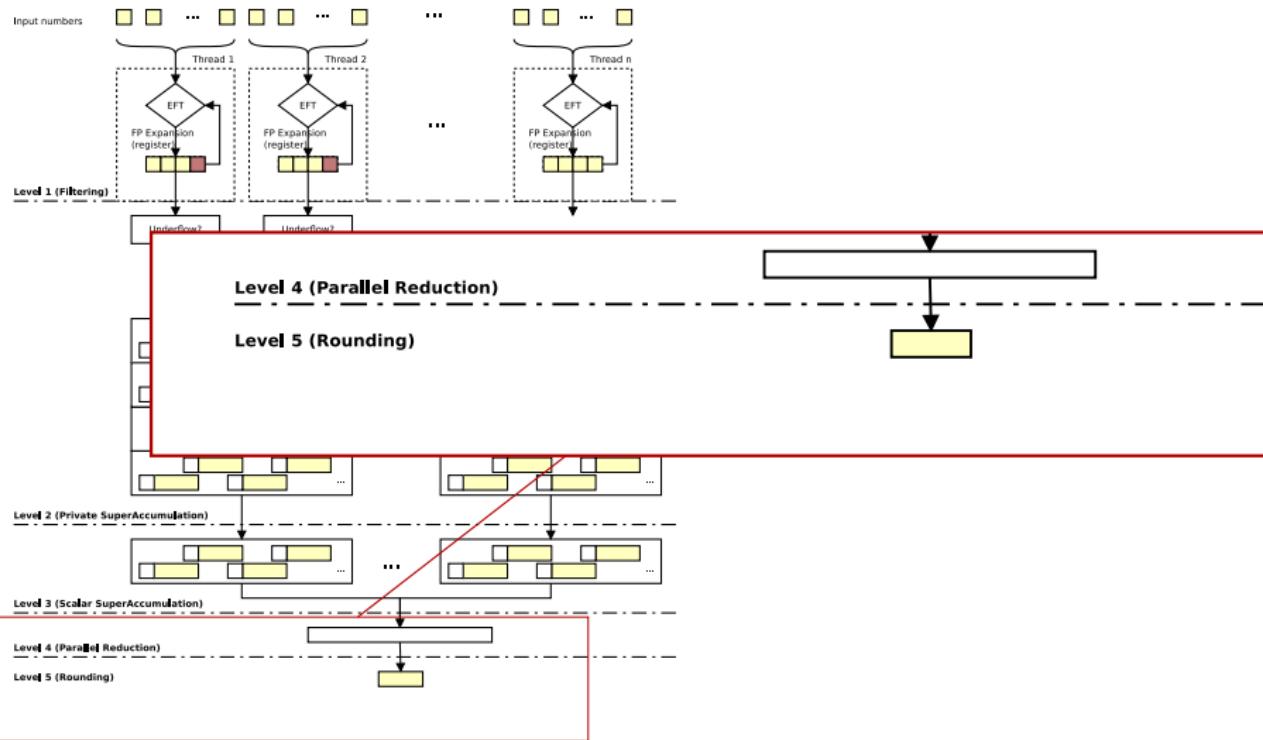
Niveau 1 : filtrage



Niveau 2 et 3 : superaccumulateur

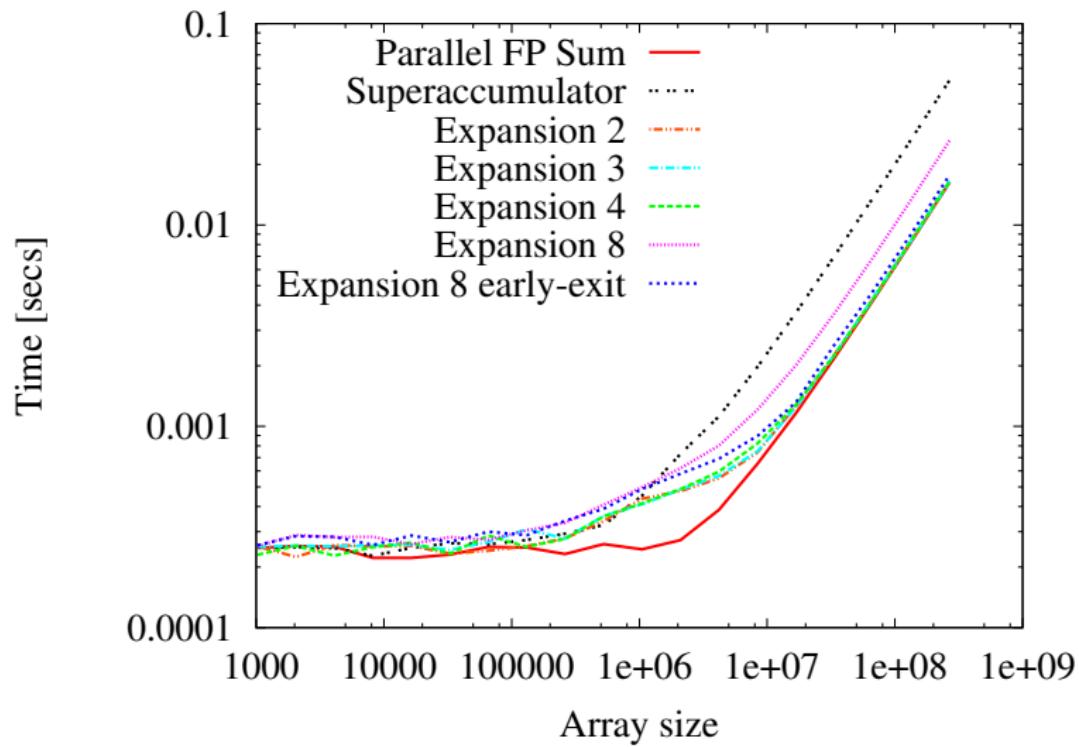


Niveau 4 et 5 : réduction et arrondi



Sommation parallele reproductible

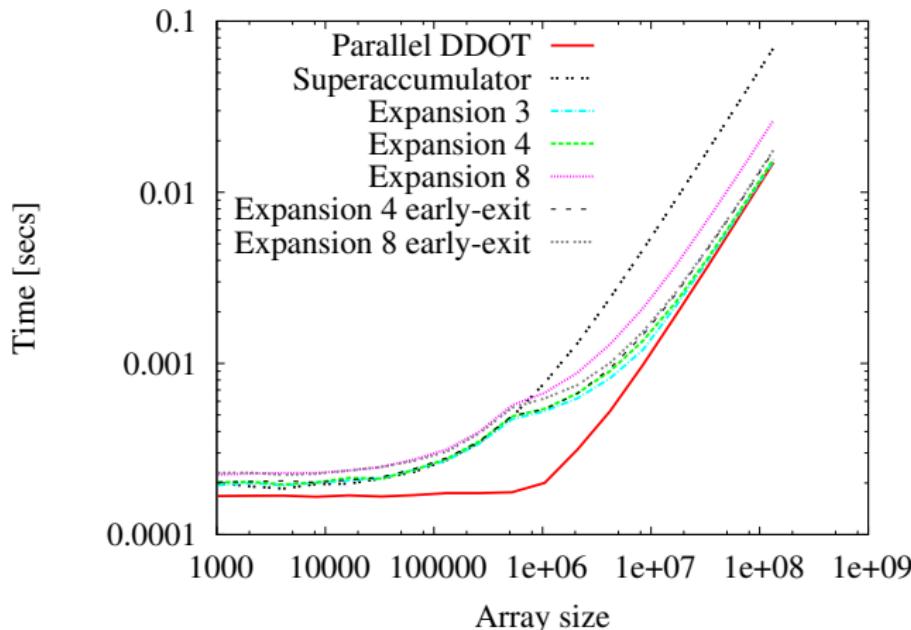
Performance sur NVIDIA Tesla K20c



Produit scalaire parallèle reproductible

Performance sur NVIDIA Tesla K20c

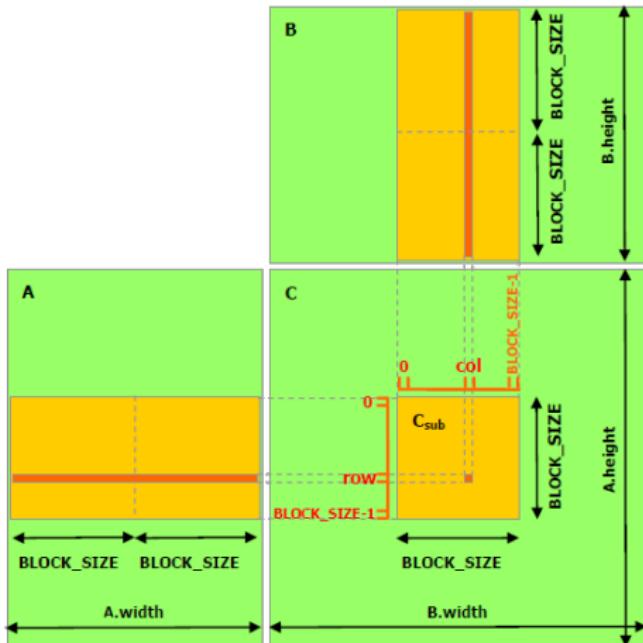
$$\text{DDOT} : \alpha := x^T y = \sum_i^N x_i y_i$$



- Basé sur [TwoProduct](#) et la sommation reproductible
- $\text{TwoProduct}(a, b)$
 - 1: $r \leftarrow a * b$
 - 2: $s \leftarrow FMA(a, b, -r)$

DGEMM reproductible (produit matriciel)

$$\text{DGEMM} : C := \alpha AB + \beta C$$

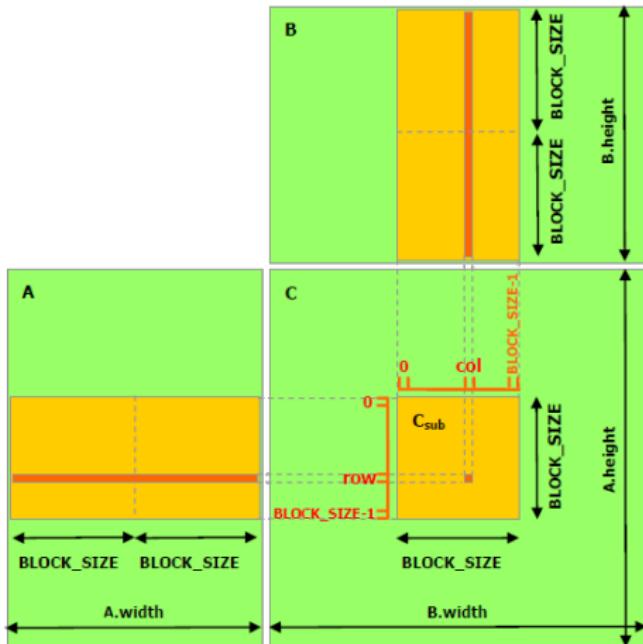


- Une FPE et un superaccumulateur par thread
- Algorithme en 3 étapes :
 - Filtrage
 - Superaccumulateur privé
 - Arrondi correct
- Chaque thread calcule plusieurs éléments de la matrice C pour réduire l'utilisation de la mémoire

Source : CUDA C Programming Guide

DGEMM reproductible (produit matriciel)

$$\text{DGEMM} : C := \alpha AB + \beta C$$



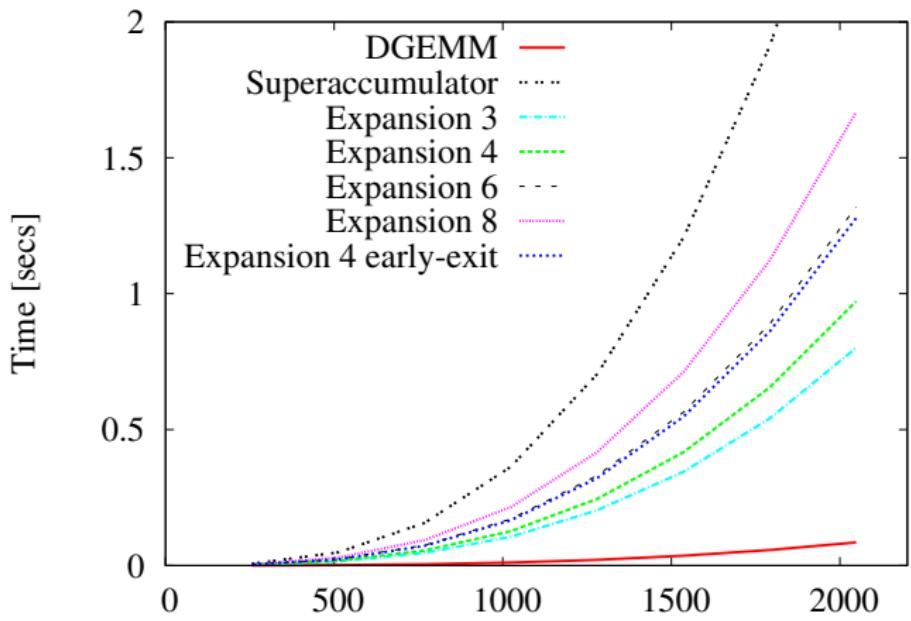
- Une FPE et un superaccumulateur par thread
- Algorithme en 3 étapes :
 - Filtrage
 - Superaccumulateur privé
 - Arrondi correct
- Chaque thread calcule plusieurs éléments de la matrice C pour réduire l'utilisation de la mémoire

Source : CUDA C Programming Guide

Multiplication matricielle parallèle

Performance sur NVIDIA Tesla K20c

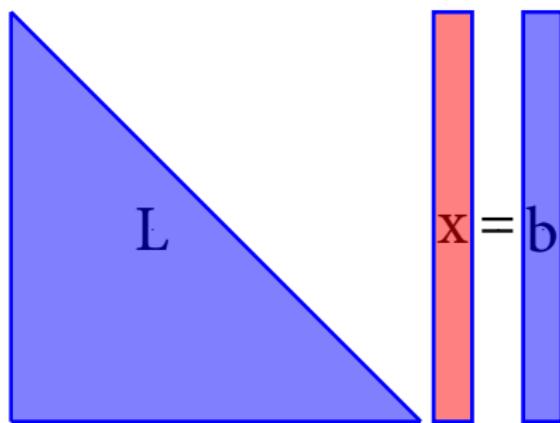
$$\text{DGEMM} : C := \alpha AB + \beta C$$



- $12dn^3$ flops, d taille d'une FPE
- Jusqu'à $76n^3$ d'utilisation mémoire

Système triangulaire

TRSV (Triangular solver) : $Lx = b$

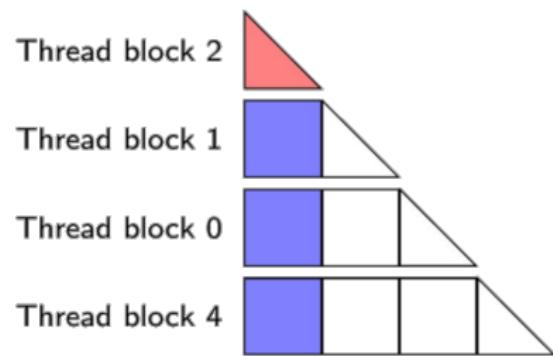
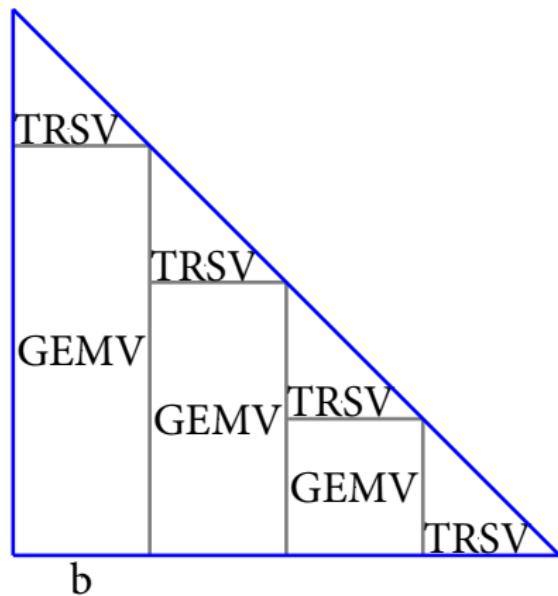


Algorithm 2 Substitution directe

```
1:  $x_1 \leftarrow b_1/l_{11}$ 
2: for  $i = 2 \rightarrow n$  do
3:    $s \leftarrow b_i$ 
4:   for  $j = 1 \rightarrow i - 1$  do
5:      $s \leftarrow s - l_{ij}x_j$ 
6:   end for
7:    $x_i \leftarrow s/l_{ii}$ 
8: end for
```

Système triangulaire

Partitionnement de la matrice

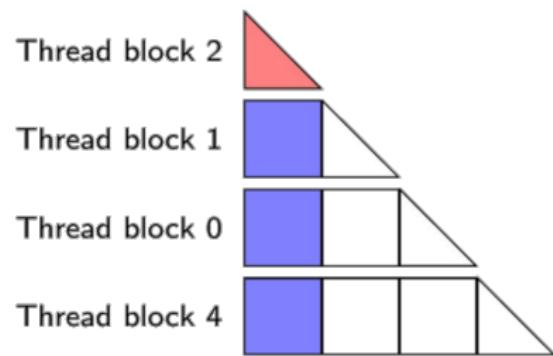
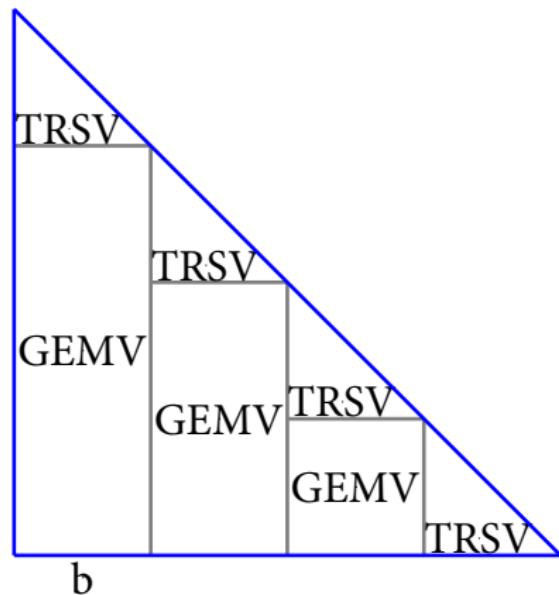


Source: A fast triangular solve on GPUs by Hogg

Figure – Partitionnement de L dans GotoBLAS

Système triangulaire

Partitionnement de la matrice



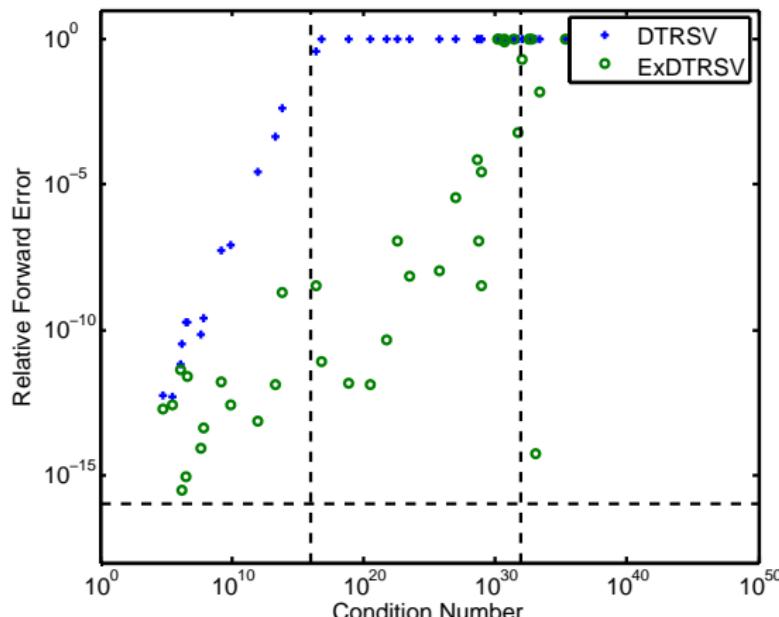
Source: A fast triangular solve on GPUs by Hogg

Figure – Partitionnement de L dans GotoBLAS

Système triangulaire

Précision

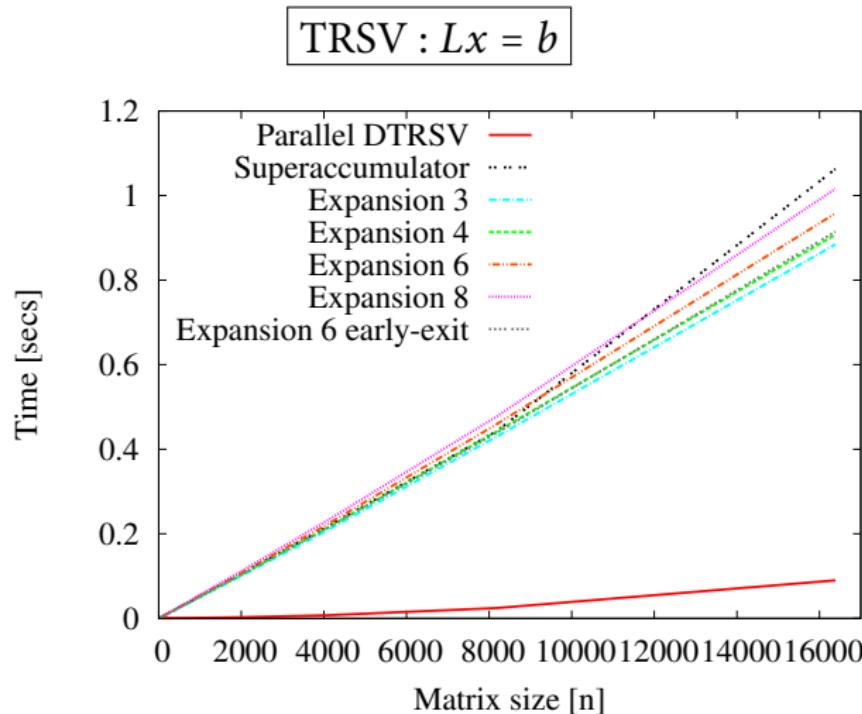
$$\frac{\|x - \widehat{x}\|}{\|x\|} \leq \text{nucond}(T, x) + O(u^2)$$



```
1:  $x_1 \leftarrow fl(b_1/l_{11})$ 
2: for  $i = 2 \rightarrow n$  do
3:    $s \leftarrow b_i$ 
4:   for  $j = 1 \rightarrow i - 1$  do
5:      $s \leftarrow s - l_{ij}x_j$ 
6:   end for
7:    $x_i \leftarrow fl(RNDN(s)/l_{ii})$ 
8: end for
```

TRSV reproductible

Performance sur NVIDIA Quadro K5000



- Utilisation de $n \times b$ threads et superaccumulateur
- Utilisation forte de la mémoire et de superaccumulateur
→ peu de performance
- Mais les résultats sont reproductibles

Décomposition LU reproductible ?

Travail en cours en se basant sur des algorithmes par bloc et sur DGEMM, TRSV reproductibles

→ Résoudre des systèmes linéaires de façon reproductible

Bibliothèques d'algèbre linéaire reproductible

- **ReproBLAS** : <http://bebop.cs.berkeley.edu/reproblas/>
développée à University of California, Berkeley par Jim Demmel et Hong Diep Nguyen
- **ExBLAS** : <https://exblas.lip6.fr/>
développée au LIP6, Sorbonne Université par Sylvain Collange, David Defour, Stef Graillat et Roman Iakymchuk