Calcul scientifique précis et efficace sur le processeur CELL

NGUYEN Hong Diep
Encadrants : Jean-Luc Lamotte, Stef Graillat

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

12 septembre 2007

Plan de l'exposé

Plan de l'exposé

Plan du stage

Le stage se compose de trois parties :

1. partie bibliographique (2 mois)
   - l'arithmétique flottante et la méthodologie d'augmentation de précision,
   - le processeur CELL et sa programmation parallèle.
2. partie théorique (1 mois) : nouveaux algorithmes sur la précision étendue en mode d'arrondi vers zéro, avec 11 théorèmes et 9 démonstrations (voir rapport).
3. partie expérimentale (3 mois) : la bibliothèque double-simple et quad-simple

Plan de l'exposé

Nombres à virgule flottante

Un nombre flottant normalisé $x \in \mathbb{F}$ est un nombre qui s'écrit sous la forme

$$x = \pm \underbrace{x_0.x_1 \ldots x_{p-1}}_{mantisse} \times b^e, \quad 0 \leq x_i \leq b-1, \quad x_0 \neq 0$$

$$0.0 \ldots 01 \times b^e \rightarrow ulp(x)$$

b : la base, p : précision, e : exposant vérifiant $e_{\min} \leq e \leq e_{\max}$

Un nombre réel : la mantisse est une chaîne infinie.
Précision machine (erreur relative) $\epsilon = b^{1-p}$

Pour représenter un nombre réel par un flottant : utiliser une approximation de $\mathbb{R}$ par $\mathbb{F}$ (arrondi fl : $\mathbb{R} \rightarrow \mathbb{F}$) $\Rightarrow$ erreur d'arrondi.

Augmenter la précision

Matériel : augmenter le nombre de bits représentants. Précision de la machine :

- simple précision : 32 bits (1 + 8 + 23)
- double précision : 64 bits (1 + 11 + 52)

Logiciel : découper un nombre réel en plusieurs parties pour représenter. Soit N un nombre reel :

$$N = \pm \underbrace{\underbrace{x_0.x_1 \ldots x_{p-1}}_{m} \underbrace{x_p \ldots x_{2p-1}}_{m_1} \underbrace{x_{2p} \ldots x_{3p-1}}_{m_2} \ldots}_{mantisse} \times 2^e$$

avec $x_i \in \{0, 1\}$

$\Rightarrow$ Précision étendue

Précision étendue

Définition 1 (La précision étendue)

Un nombre de précision étendue de n est une somme non-évaluée de n flottants de la machine.

$$x = x_1 + x_2 + \ldots + x_n$$

Normalisation :

1. au plus près : $|x_{k+1}| \leq \frac{1}{2} ulp(x_k)$.
2. vers zéro : $|x_{k+1}| < ulp(x_k)$ et ils ont le même signe.

précision de la machine : simple précision

- n=2 : double-simple
- n=4 : quad-simple

Transformations exactes

Soient $a, b \in \mathbb{F}$, et $\circ$ une opération flottante quelconque $\circ \in \{+, -, \cdot, /\}$

$$\begin{array}{rcl} (a \circ b) & \in & \mathbb{R} \\ & \notin & \mathbb{F} \\ \rightarrow fl(a \circ b) & \neq & (a \circ b) \end{array}$$

$(a \circ b) - fl(a \circ b) = err$: erreur d'arrondi
transformations exactes (EFT) : trouver un couple de deux flottants (x, y) tel que :

- $x \approx a \circ b$
- $a \circ b = x + y$

Transformations exactes

Algorithme de Priest (arrondi au plus près)

Algorithme 1 (Transformation exacte sur la sommation)

function $[x, y] = \texttt{TwoSum}(a, b)$
  if $(|b| > |a|)$
    $\text{swap}(a, b)$
  $x = fl(a + b)$
  $d = fl(x - a)$
  $y = fl(b - d)$

Algorithme classique

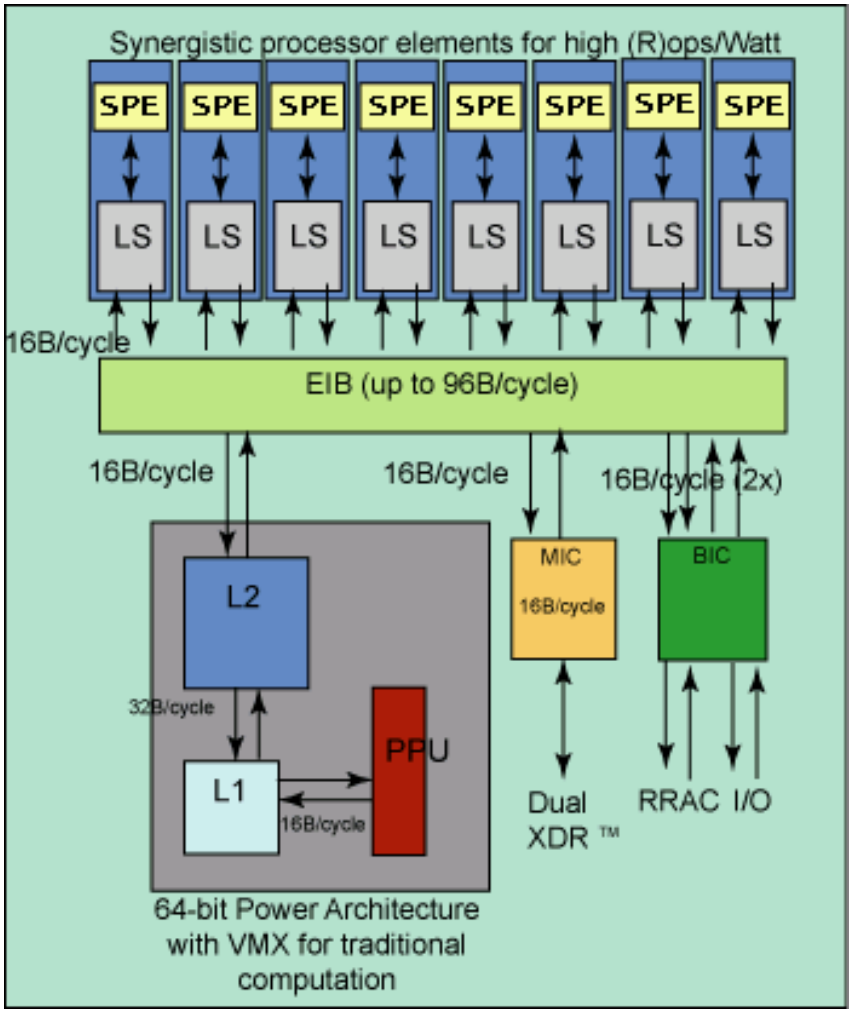
Algorithme 2 (Transformation exacte sur le produit)

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

Plan de l'exposé

Le processeur CELL

CELL : Processeur conçu conjointement par IBM, Sony et Toshiba équipe notamment la console de jeu vidéo Playstation 3 de Sony.
Performance de crête $>$ 200 GFLOPs en simple précision.

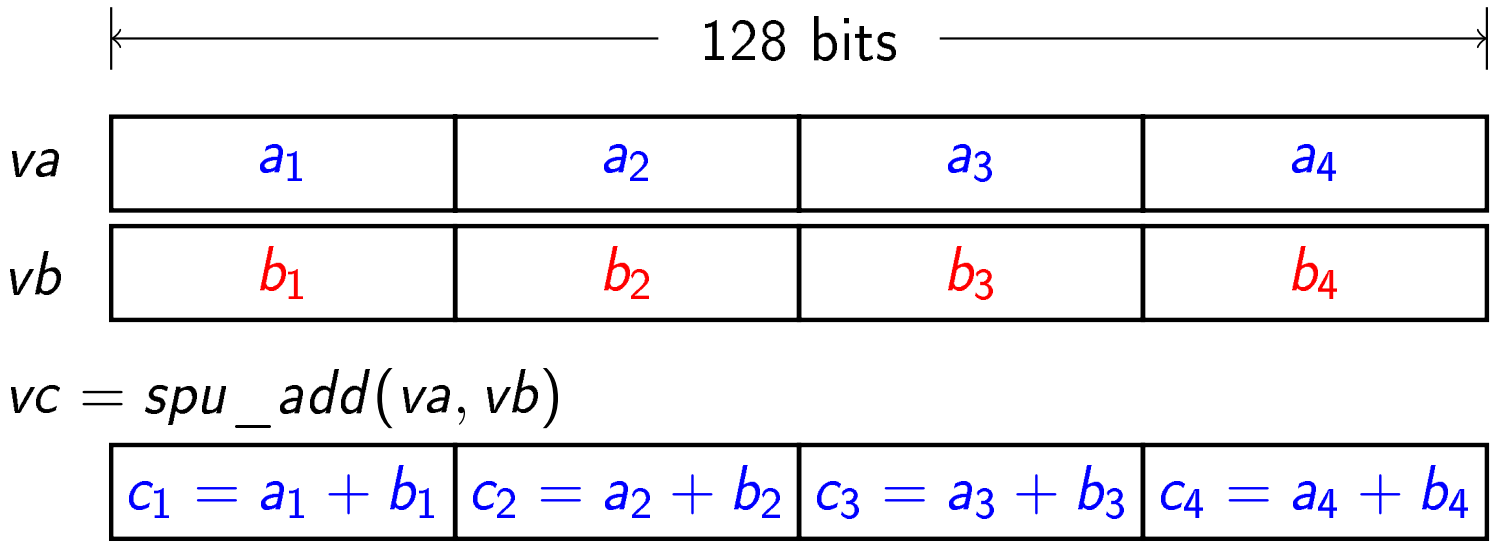


Synergistic Processing Element (SPE)

Le SPE fournit une importante puissance de calcul sur des vecteurs :

- 4 nombres flottants de simple précision.
- 2 nombres flottants de double précision.

se base sur la fonction FMA (Fused Multiply-Add).



Synergistic Processing Element (SPE)

Mode d'arrondi :

- Simple précision : arrondi vers zero,
- Double précision : tous les quatre modes.

Pipelines :

- Simple précision : complètement pipelinée (1 instruction / 1 cycle),
- Double précision : pas complètement pipelinée (1 instruction / 7 cycles).

Performance

- Simple précision : $4 \times 2 \times 3.2 = 25.6 GFLOPs$.
- Double précision : $2 \times 2 \times 3.2/7 = 1.8 GFLOPs$

Plan de l'exposé

Transformations exactes

Algorithme 3 (Transformation exacte sur la sommation vers zéro)

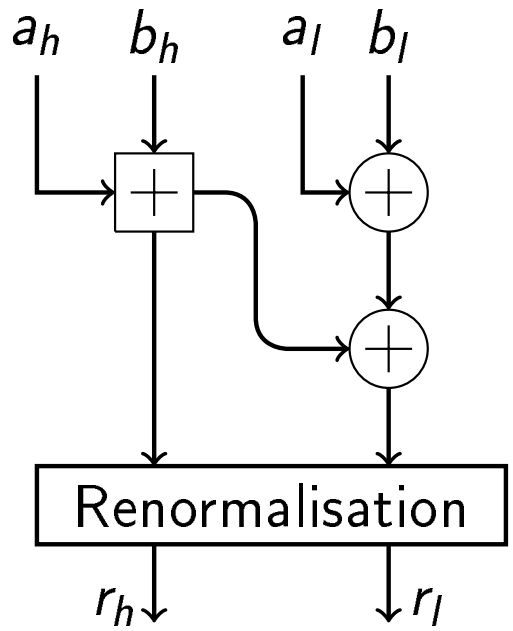
function $[x, y] = \texttt{TwoSum} - \texttt{toward} - \texttt{zero}(a, b)$
  if $(|b| > |a|)$
    $\text{swap}(a, b)$
  $x = fl(a + b)$
  $d = fl(x - a)$
  $y = fl(b - d)$
  if$(|2 * b| < |d|)$
    $x = a, y = b$

Théorème 1

L'algorithme `TwoSum-toward-zero` *transforme deux flottants* a *et* b *dans un couple de deux flottants* (s, e) *qui satisfait :*

$$s + e = a + b \text{ et } |e| < ulp(s)$$

La sommation de deux double-simples



Théorème 2

Soit $a = a_h + a_l$ *et* $b = b_h + b_l$ *deux double-simples d'entrée,* $r = r_h + r_l$ *le résultat et* δ *l'erreur de l'algorithme* `add_ds_ds`*. L'erreur de cet algorithme satisfait*

$$r = a + b + \delta$$
$$|\delta| < max(2^{-23} * |a_l + b_l|, 2^{-43} * |a_h + a_l + b_h + b_l|) + 2^{-45} * |a + b|.$$

La transformation exacte : code

```
1  | Two-Sum-toward-zero (a,b)
2  |    comp = spu_cmpabsgt(b,a)
3  |    hi = spu_sel(a, b, comp)
4  |    lo = spu_sel(b, a, comp)
5  |    s = spu_add(a , b)
6  |    d = spu_sub(s , hi)
7  |    e = spu_sub(lo , d)
8  |    tmp = spu_mul(2 , lo)
9  |    comp = spu_cmpabsgt(d, tmp)
10 |    s = spu_sel(s, hi, comp)
11 |    e = spu_sel(e, lo, comp)
12 | return (s,e)
```

```
cycles
 12
  -34
    45
012345
      -678901
          ----234567
        789012
              34
               -56
                 --89
```

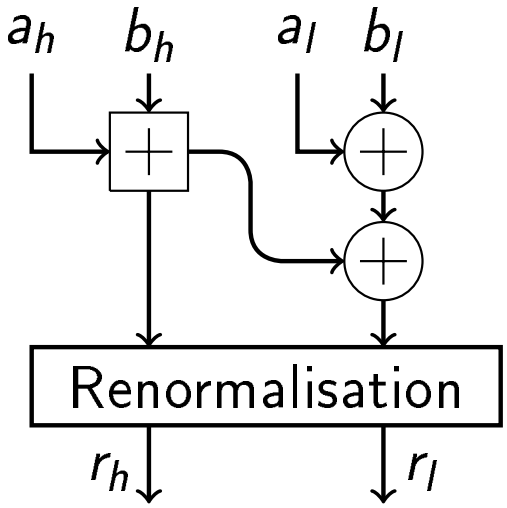
Coût : 20 cycles

Renormalisation

```
1 | Renormalise2-toward-zero (a,b)
2 |    s = spu_add(a , b)
3 |    comp = spu_cmpabsgt(b,a)
4 |    hi = spu_sel(a, b, comp)
5 |    lo = spu_sel(b, a, comp)
6 |    d = spu_sub(s , hi)
7 |    e = spu_sub(lo , d)
8 | return (s,e)
```

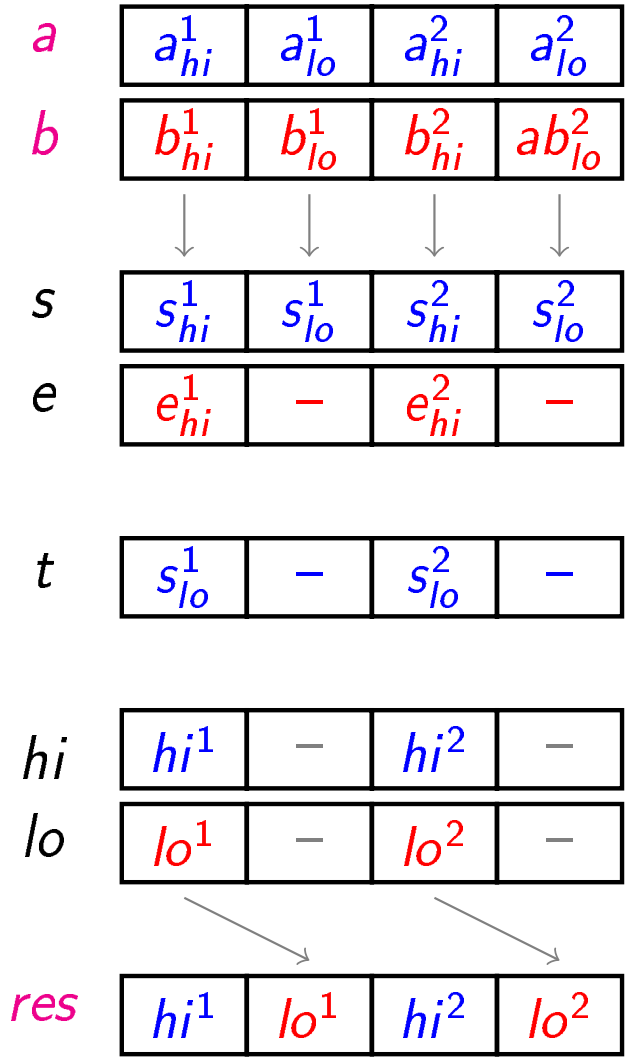
Coût : 18 cycles

La sommation de double-simple : version naturelle




```
1 | add_ds_ds_vect(a , b)
2 |   (s,e) = Two-Sum-toward-zero(a , b)
3 |   t = spu_shuffle(s,s,switch-vect)
4 |   t1 = spu_add(t , e)
5 |   (hi,lo) = Renormalise2-toward-zero(s,t1)
6 |   res = spu_shuffle(hi,lo,merge-vect)
7 | return res
```

Coût : **50** cycles / **2** opérations



La sommation de double-simple : version 2

```
1  | add_ds_ds_2vect (vect_a1, vect_a2, vect_b1, vect_b2)
2  |    a_hi = spu_shuffle(vect_a1, vect_a2, _merge1_vect_)
3  |    a_lo = spu_shuffle(vect_a1, vect_a2, _merge2_vect_)
4  |    b_hi = spu_shuffle(vect_b1, vect_b2, _merge1_vect_)
5  |    b_lo = spu_shuffle(vect_b1, vect_b2, _merge2_vect_)
6  |    (s, e) = Two-Sum-toward-zero (a_hi, b_hi)
7  |    t1 = spu_add(a_lo , b_lo)
8  |    tmp = spu_add(t1 , e)
9  |    (hi, lo) = Renormalise2-toward-zero (s , tmp)
10 |    vect_c1 = spu_shuffle(hi, lo, _merge1_vect_)
11 |    vect_c2 = spu_shuffle(hi, lo, _merge2_vect_)
12 | return (vect_c1, vect_c2)
```

Coût : **64** cycles / **4** opérations

La sommation de double-simple : version optimisée

La version 2 accélère beaucoup la performance de la sommation.
On gaspille encore des cycles.
$\Rightarrow$ Effectuer deux fois la version 2 en même temps dans une seule fonction

Coût : **72** cycles / **8** opérations

Plan de l'exposé

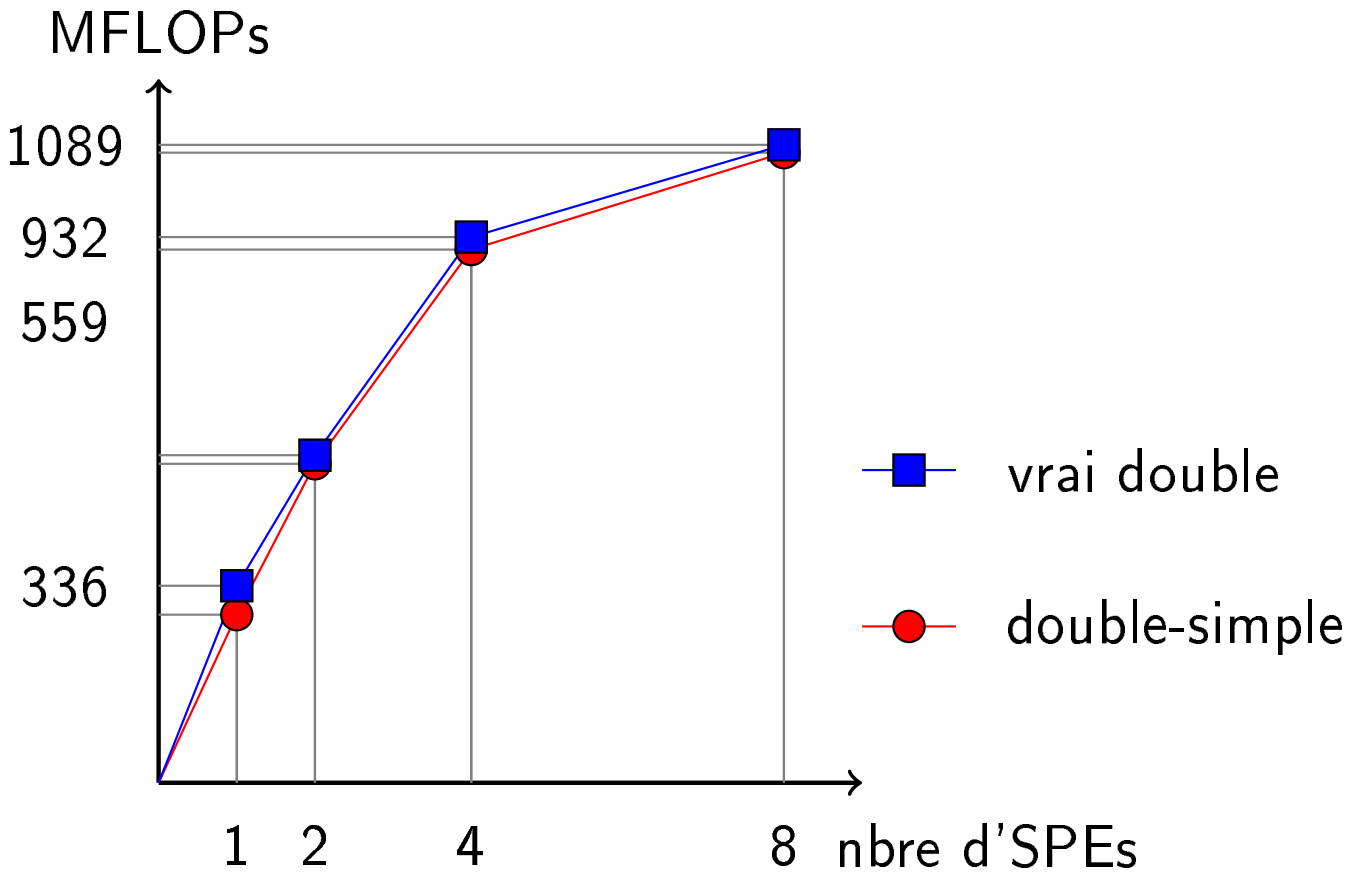
Résultats théoriques

Si le SPE a une fréquence de 3.2GHz.
La performance de crête en double précision : $2 \times 2 \times 3.2/7 = 1.8 GFLOPs$.

Fonction	**Temps de calcul**	**Performance**
Add_ds_ds_vect	50 cycles / 2	128 MFLOPs
Add_ds_ds_2vect	64 cycles / 4	200 MFLOPs
Add_ds_ds_4vect	72 cycles / 8	355 MFLOPs
Mul_ds_ds_vect	49 cycles / 2	130 MFLOPs
Mul_ds_ds_2vect	60 cycles / 4	213 MFLOPs
Mul_ds_ds_4vect	63 cycles / 8	406 MFLOPs
Div_ds_ds_2vect	111 cycles / 4	115 MFLOPs
Div_ds_ds_4vect	125 cycles / 8	204 MFLOPs
Add_qs_qs_4vect	449 cycles / 4	28.5 MFLOPs
Mul_qs_qs_4vect	583 cycles / 4	21.9 MFLOPs
Div_qs_qs_4vect	2667 cycles / 4	4.79 MFLOPs

Résultats expérimentaux : performance de double-simple

Sommation de deux vecteurs de 8,388,608 éléments. 100 passages



Résultats expérimentaux : précision

Effectuer un grand nombre de fois (2^{24}) des opérations sur des entrées au hasard.

Opération	Différence Maximale	bits de précision
add_ds_ds_4vect	0.0e+00	46
mul_ds_ds_4vect	2.964e-14	44.9
div_ds_ds_4vect	2.373e-14	45.2
add_qs_qs_4vect	7.675e-30	96.7
mul_qs_qs_4vect	3.259e-29	94.6
div_qs_qs_4vect	4.742e-29	94

Plan de l'exposé

Conclusions

Travail effectué :

1. théorique :
   - proposer de nouvelles transformations exactes en arrondi vers zéro,
   - proposer des algorithmes formant la méthodologie pour les précisions étendues avec 11 théorèmes et 9 démonstrations.
2. expérimentale :
   - implémenter une bibliothèque double-simple, une bibliothèque double-double, une bibliothèque quad-simple avec plus de 2000 lignes de code SIMD sur le processeur CELL.
   - tester les deux bibliothèque double-simple et quad-simple pour récupérer ses performances réelles et aussi ses précisions réelles.

Perspectives

Dans le futur :

- Traiter des exceptions numériques.
- Compléter la bibliothèque double-simple et quad-simple
  - Opérations binaires,
  - Opérations algébriques (puissance, racine, ...)
  - Opérations transcendantes (exposant, logarithme, ...)
- Les méthodes compensées.

Fin

Merci de votre attention !

Références I


Yozo Hida, Xiaoye S.Li, and David H.Baily.
Quad-double arithmetic : Algorithms, implementations, and application.
2000.


Donald Knuth.
The art of computer programming : Seminumerical algorithms, vol.2, Reading, Massachusetts : Addison-Wesley,2000.


Takeshi Ogita, Siegfried M. Rump, and Shin'ichi Oishi.
Accurate sum and dot product.
SIAM J. Sci. Comput., 26(6) :1955–1988, 2005.


Christoph Quirin Lauter.
Basic building blocks for a triple-double intermediate format
Tech. report, INRIA, 2005

Références II

Douglas M. Priest.
On Properties of Floating Point Arithmetics : Numerical Stability and the Cost of Accurate Computations.
PhD thesis, Mathematics Department, University of California, Berkeley, CA, USA, November 1992.

T.J Dekker.
A floating-point technic for extending the available precision.
Numer . Math., 18 : 224–242, 1971.