Journées FastRelax – LIP6

Renormalisation d'expansions : une vérification formelle

Sylvie Boldo, Miora Joldes, Jean-Michel Muller, Valentina Popescu

30 mai 2017

Plan

Motivations

- Expansions flottantes : plus de précision à bas coût.

Motivations

- Expansions flottantes : plus de précision à bas coût.
- Des algorithmes compliqués.

Motivations

- Expansions flottantes : plus de précision à bas coût.
- Des algorithmes compliqués.
- Des preuves très compliquées avec beaucoup de sous-cas.

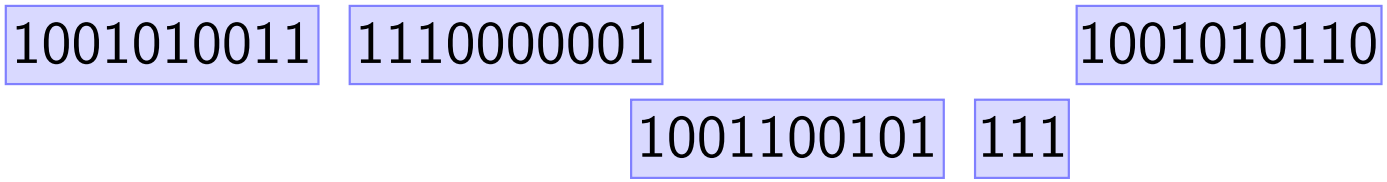
Motivations

- Expansions flottantes : plus de précision à bas coût.
- Des algorithmes compliqués.
- Des preuves très compliquées avec beaucoup de sous-cas.

$\Rightarrow$ une preuve formelle !

Expansions

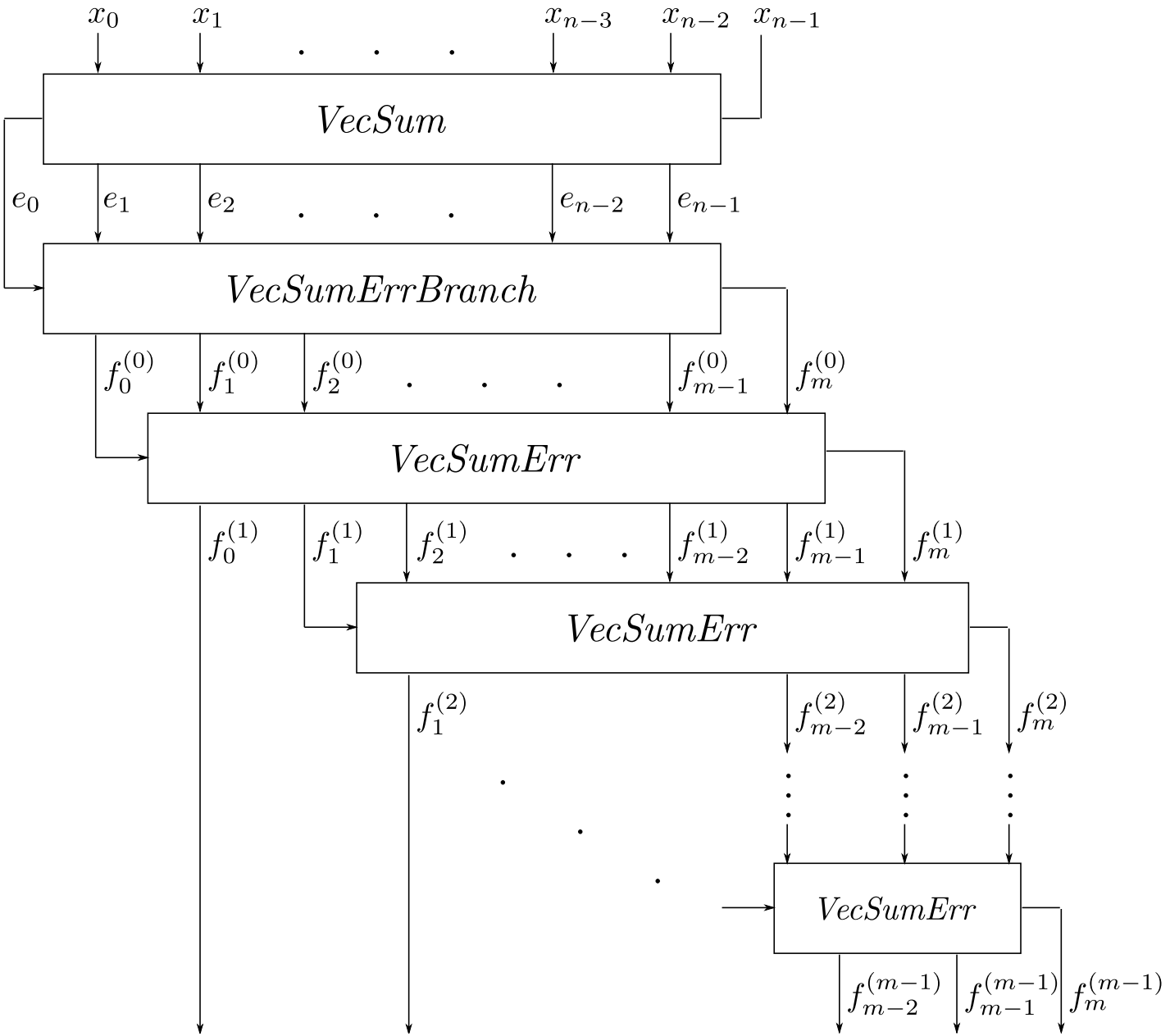
On considère $x = \sum x_i$ avec $x_i \in \mathbb{F}$.



$\hookrightarrow$ opérations ($+, \times, \div, \sqrt{}$, $\ll$ nettoyage $\gg$)
$\hookrightarrow$ preuves (avec possibilité de 0 intermédiaires)

Article **Arithmetic algorithms for extended precision using floating-point expansions** de Mioara Joldes, Olivier Marty, Jean-Michel Muller et Valentina Popescu, IEEE TC, 2016.

Renormalisation



Todo list

- définir les expansions en Flocq
- comprendre et formaliser les propriétés sur les expansions en entrée et sortie des différents niveaux de l'algorithme
- comprendre et formaliser la preuve

Todo list

- définir les expansions en Flocq
- comprendre et formaliser les propriétés sur les expansions en entrée et sortie des différents niveaux de l'algorithme
- comprendre et formaliser la preuve

Problèmes attendus :

- dénormalisés ($\not\in$ preuve papier)

Todo list

- définir les expansions en Flocq
- comprendre et formaliser les propriétés sur les expansions en entrée et sortie des différents niveaux de l'algorithme
- comprendre et formaliser la preuve

Problèmes attendus :

- dénormalisés ($\notin$ preuve papier)
- zéros intermédiaires

Plan

Base

Choix : une expansion est une **liste** de réels.

$+\!+$ inductions, accès au n-ième élément, liste formée des n premiers éléments. . .

Base

Choix : une expansion est une **liste** de réels.

++ inductions, accès au n-ième élément, liste formée des n premiers éléments...

- lemmes manquants sur les listes

Base

Choix : une expansion est une **liste** de réels.

++ inductions, accès au n-ième élément, liste formée des n premiers éléments...

- lemmes manquants sur les listes

– problèmes d'itérateurs : `fold_left / right` ne convenaient pas (*out of scope* de cet exposé)

Formalisation des expansions (1/2)

Pour une propriété P donnée entre deux réels ($P : \mathbb{R} \to \mathbb{R} \to \mathit{Prop}$). On définit une expansion comme une liste de flottants tels que deux éléments successifs en enlevant les zéros vérifient P.

Formalisation des expansions (1/2)

Pour une propriété P donnée entre deux réels ($P : \mathbb{R} \to \mathbb{R} \to Prop$). On définit une expansion comme une liste de flottants tels que deux éléments successifs en enlevant les zéros vérifient P.

```
(* Definition of an expansion with a given property *)

Inductive Exp_P (P:R->R->Prop): list R -> Prop :=
  | Exp_Nil : Exp_P P List.nil
  | Exp_One : forall x : R, format x -> Exp_P  P (x :: nil)
  | Exp_Z1 : forall l: list R, Exp_P P l
                  -> Exp_P  P (0 :: l)
  | Exp_Z2 : forall x: R, forall l: list R, Exp_P P (x::l)
                  ->  Exp_P P (x :: ( 0 :: l))
  | Exp_NZ : forall x y: R, forall l: list R, format x
     -> x <> 0 -> y <> 0 -> Exp_P P (y::l)
     -> P x y -> Exp_P P (x :: (y :: l)).
```

Par exemple `Exp_P` $\leq$ définit une liste croissante, en sautant les zéros.

Formalisation des expansions (2/2)

- des tas de lemmes plus ou moins débiles, et

```
Lemma nth_Exp_P: forall (P:R->R->Prop) l, Forall format l ->
(forall i j, (0 <= i < length l)%nat -> (0 <= j < length l)%nat
  -> (i < j)%nat -> nth i l 0 <> 0 -> nth j l 0 <> 0
  -> P  (nth i l 0) (nth j l 0)) -> Exp_P P l.
```

Formalisation des expansions (2/2)

- des tas de lemmes plus ou moins débiles, et

```
Lemma nth_Exp_P: forall (P:R->R->Prop) l, Forall format l ->
(forall i j, (0 <= i < length l)%nat -> (0 <= j < length l)%nat
  -> (i < j)%nat -> nth i l 0 <> 0 -> nth j l 0 <> 0
  -> P (nth i l 0) (nth j l 0)) -> Exp_P P l.
```

- définition générale pour le VecSum / VecSumErr

Formalisation des expansions (2/2)

- des tas de lemmes plus ou moins débiles, et

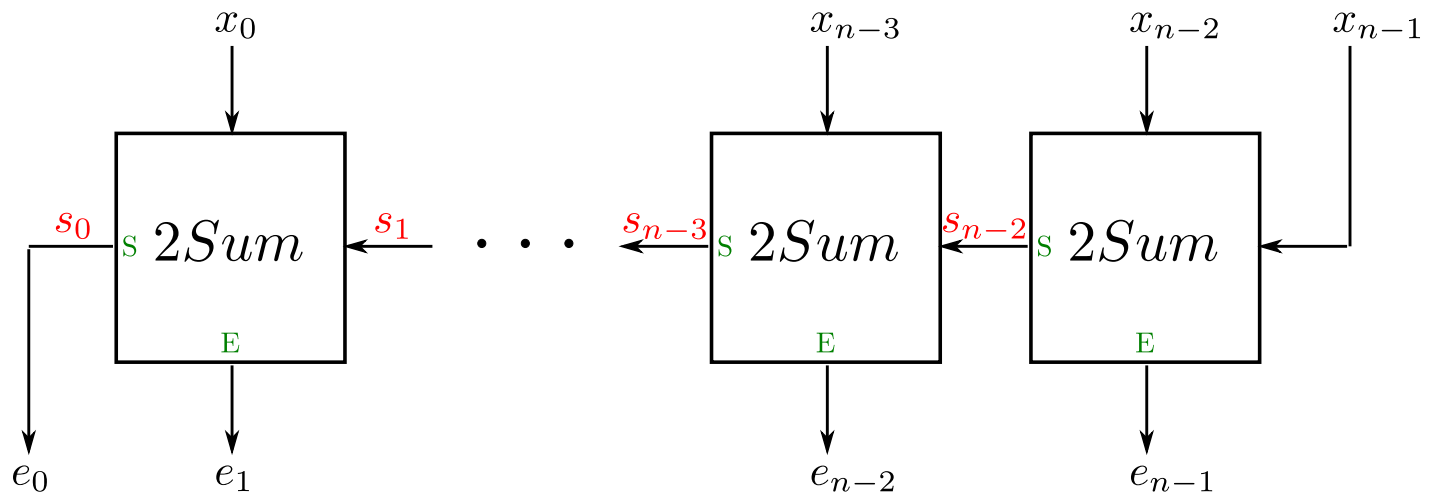
```
Lemma nth_Exp_P: forall (P:R->R->Prop) l, Forall format l ->
(forall i j, (0 <= i < length l)%nat -> (0 <= j < length l)%nat
  -> (i < j)%nat -> nth i l 0 <> 0 -> nth j l 0 <> 0
  -> P (nth i l 0) (nth j l 0)) -> Exp_P P l.
```

- définition générale pour le VecSum / VecSumErr
- et les lemmes qui vont avec.

Plan

VecSum

Première étape de la renormalisation :



Entrée/sortie du VecSum

En entrée, on a une expansion qui « overlap par au plus d bits » (pour $0 < d \leq p-2$), ce qui devient une expansion avec la propriété :

$$P = \text{ fun } x\ y \quad \Rightarrow \quad |x| < 2^d \mathrm{ulp}(y)$$

Entrée/sortie du VecSum

En entrée, on a une expansion qui « overlap par au plus d bits » (pour $0 < d \leq p-2$), ce qui devient une expansion avec la propriété :

$$\begin{aligned} P = \ \text{fun } x\ y \quad &\Rightarrow \quad |x| < 2^d \mathrm{ulp}(y) \\ &\Rightarrow \quad 2^{p-d}\mathrm{ulp}(x) < \mathrm{ulp}(y) \end{aligned}$$

Entrée/sortie du VecSum

En entrée, on a une expansion qui « overlap par au plus d bits » (pour $0 < d \leq p-2$), ce qui devient une expansion avec la propriété :

$$\begin{aligned} P = \text{ fun } x\ y \quad &\Rightarrow \quad |x| < 2^d \mathrm{ulp}(y) \\ &\Rightarrow \quad 2^{p-d}\mathrm{ulp}(x) < \mathrm{ulp}(y) \end{aligned}$$

En sortie, on a une expansion « S-non overlapping (Shewchuck) », ce qui devient une expansion avec la propriété :

$$P = \text{ fun } x\ y \Rightarrow \exists e : Z, \exists n : Z, y = n \times 2^e \wedge |x| < 2^e.$$

Entrée/sortie du VecSum

En entrée, on a une expansion qui « overlap par au plus d bits » (pour $0 < d \leq p-2$), ce qui devient une expansion avec la propriété :

$$\begin{aligned} P = \text{ fun } x\ y \quad &\Rightarrow \quad |x| < 2^d \mathrm{ulp}(y) \\ &\Rightarrow \quad 2^{p-d}\mathrm{ulp}(x) < \mathrm{ulp}(y) \end{aligned}$$

En sortie, on a une expansion « S-non overlapping (Shewchuck) », ce qui devient une expansion avec la propriété :

$$P = \text{ fun } x\ y \Rightarrow \exists e : Z, \exists n : Z, y = n \times 2^e \wedge |x| < 2^e.$$

```
Definition IVS_P:=(fun x y => bpow (p-d) * ulp_flt x <= ulp_flt y).
Definition InputVecSum := Exp_P IVS_P.

Theorem VecSum_correct1: forall l: list R,
   fold_right Rplus 0 l = fold_right Rplus 0 (VecSum l).
```

Preuves & problèmes 1/2

- « D'après Jeannerod et Rump », l'erreur d'une somme est $\leq \ldots$

Théorème (error_sum_n, from JeaRu13)

Soit l une liste de n flottants. Soit $e = \sum_{i=0}^{n} l_i$ et $a = \sum_{i=0}^{n} |l_i|$. Soit $f = \oplus_{i=0}^{n} l_i$ la somme flottante. Alors $|f - e| \leq (n-1)2^{-p}a$.

Preuves & problèmes 1/2

- « D'après Jeannerod et Rump », l'erreur d'une somme est $\leq \ldots$ ✓

Théorème (error_sum_n, from JeaRu13)

Soit l une liste de n flottants. Soit $e = \sum_{i=0}^{n} l_i$ et $a = \sum_{i=0}^{n} |l_i|$. Soit $f = \oplus_{i=0}^{n} l_i$ la somme flottante. Alors $|f - e| \leq (n-1)2^{-p}a$.

Preuves & problèmes 1/2

- « D'après Jeannerod et Rump », l'erreur d'une somme est $\leq \ldots$ ✓

Théorème (error_sum_n, from JeaRu13)

Soit l une liste de n flottants. Soit $e = \sum_{i=0}^{n} l_i$ et $a = \sum_{i=0}^{n} |l_i|$. Soit $f = \oplus_{i=0}^{n} l_i$ la somme flottante. Alors $|f - e| \leq (n-1)2^{-p}a$.

- D'après Joldes et al.,

$$2^d + 2^{2d-p} + 2^{3d-2p} + 2^{4d-3p} + \cdots \leq 2^d \frac{2^p}{2^p - 1}$$

Preuves & problèmes 1/2

- « D'après Jeannerod et Rump », l'erreur d'une somme est $\leq \ldots$ ✓

Théorème (error_sum_n, from JeaRu13)

Soit l une liste de n flottants. Soit $e = \sum_{i=0}^{n} l_i$ et $a = \sum_{i=0}^{n} |l_i|$. Soit $f = \oplus_{i=0}^{n} l_i$ la somme flottante. Alors $|f - e| \leq (n-1)2^{-p}a$.

- D'après Joldes et al., ✗

$$2^d + 2^{2d-p} + 2^{3d-2p} + 2^{4d-3p} + \cdots \leq 2^d \frac{2^p}{2^p - 1}$$

Je propose plutôt $2^d \frac{2^{p-d}}{2^{p-d}-1}$,

Preuves & problèmes 1/2

- « D'après Jeannerod et Rump », l'erreur d'une somme est $\leq \ldots$ ✓

Théorème (error_sum_n, from JeaRu13)

Soit l une liste de n flottants. Soit $e = \sum_{i=0}^{n} l_i$ et $a = \sum_{i=0}^{n} |l_i|$. Soit $f = \oplus_{i=0}^{n} l_i$ la somme flottante. Alors $|f - e| \leq (n-1)2^{-p}a$.

- D'après Joldes et al., ✗

$$2^d + 2^{2d-p} + 2^{3d-2p} + 2^{4d-3p} + \cdots \leq 2^d \frac{2^p}{2^p - 1}$$

Je propose plutôt $2^d \frac{2^{p-d}}{2^{p-d}-1}$, mais c'est pas grave ✓

Preuves & problèmes 2/2

- On peut utiliser des FastTwoSum dans VecSum ✓

Preuves & problèmes 2/2

- On peut utiliser des FastTwoSum dans VecSum ✓

- Gestion des dénormalisés
  $|x| < 2^d \mathrm{ulp}(y)$ ne garantit pas que $|x| \leq |y|$
  Mais $2^{p-d}\mathrm{ulp}(x) < \mathrm{ulp}(y)$ oui (et c'est équivalent sur les normaux).

Preuves & problèmes 2/2

- On peut utiliser des FastTwoSum dans VecSum ✓

- Gestion des dénormalisés ✓
  $|x| < 2^d \mathrm{ulp}(y)$ ne garantit pas que $|x| \leq |y|$
  Mais $2^{p-d}\mathrm{ulp}(x) < \mathrm{ulp}(y)$ oui (et c'est équivalent sur les normaux).

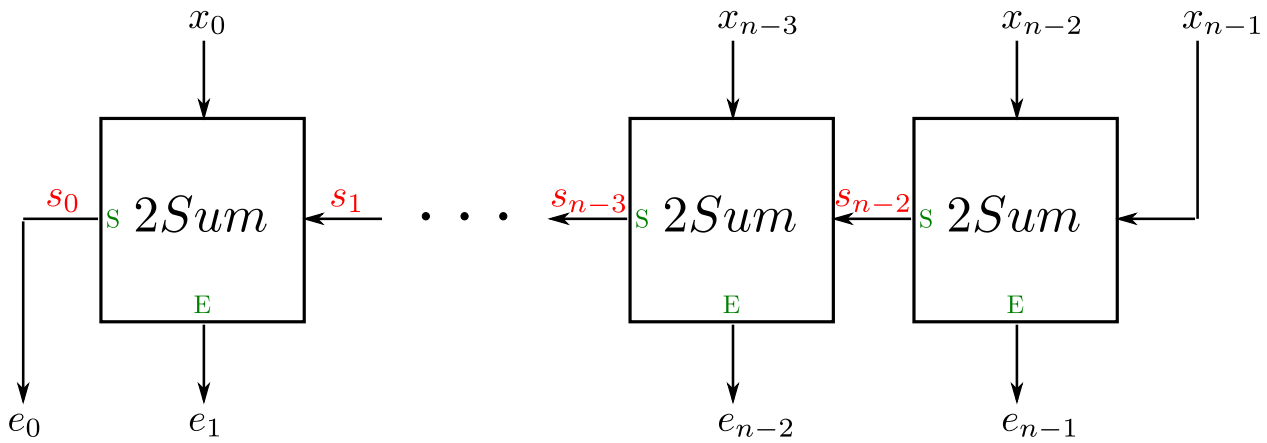
Preuves & problèmes 2/2

- On peut utiliser des FastTwoSum dans VecSum ✓
- Gestion des dénormalisés ✓
  $|x| < 2^d \mathrm{ulp}(y)$ ne garantit pas que $|x| \leq |y|$
  Mais $2^{p-d}\mathrm{ulp}(x) < \mathrm{ulp}(y)$ oui (et c'est équivalent sur les normaux).
- Et les zéros intermédiaires ?

Preuves & problèmes 2/2

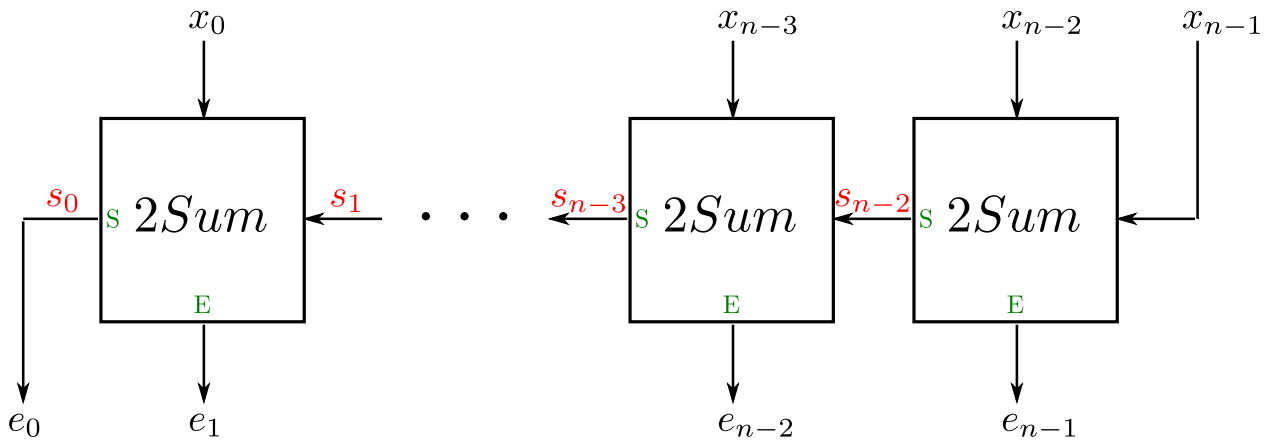
- On peut utiliser des FastTwoSum dans VecSum ✓
- Gestion des dénormalisés ✓
  $|x| < 2^d \mathrm{ulp}(y)$ ne garantit pas que $|x| \leq |y|$
  Mais $2^{p-d}\mathrm{ulp}(x) < \mathrm{ulp}(y)$ oui (et c'est équivalent sur les normaux).
- Et les zéros intermédiaires ? ✓
  AAAAAAAAAAAAAAAAAHHHHHHHHHHHHHHHHHHHHHHHH

Lemme technique 1/2



Sachant que les x_i sont fortement (dé-)croissants, $|x_{i+1}| < 2^d \mathrm{ulp}(x_i)$ si non nuls normaux, je voudrais prouver que la suite $\exp(s_i)$ est également croissante, pour m'en servir comme exposant de non-overlapping.

Lemme technique 1/2



Sachant que les x_i sont fortement (dé-)croissants, $|x_{i+1}| < 2^d \mathrm{ulp}(x_i)$ si non nuls normaux, je voudrais prouver que la suite $\exp(s_i)$ est également croissante, pour m'en servir comme exposant de non-overlapping. ✗

↪ non utile dans la preuve initiale
↪ après re-brainstorming avec JMM, VP, MJ...

Lemme technique 2/2

On note $s_i = \oplus_{j=0}^{i-1} x_i$ et $e_i = \mathrm{errf}(s_i, x_i)$ (et donc $s_{i+1} + e_i = s_i + x_i$).

Lemme technique 2/2

On note $s_i = \oplus_{j=0}^{i-1} x_i$ et $e_i = \mathrm{errf}(s_i, x_i)$ (et donc $s_{i+1} + e_i = s_i + x_i$).

- $\exp(s_i)$ croissante ✗

Lemme technique 2/2

On note $s_i = \oplus_{j=0}^{i-1} x_i$ et $e_i = \mathrm{errf}(s_i, x_i)$ (et donc $s_{i+1} + e_i = s_i + x_i$).

- $\exp(s_i)$ croissante ✗
- sachant que e_{i-1} et e_j sont non nul, $\exp(s_i) \leq \exp(s_j)$ ✗

Lemme technique 2/2

On note $s_i = \oplus_{j=0}^{i-1} x_i$ et $e_i = \mathrm{errf}(s_i, x_i)$ (et donc $s_{i+1} + e_i = s_i + x_i$).

- $\exp(s_i)$ croissante ✗
- sachant que e_{i-1} et e_j sont non nul, $\exp(s_i) \leq \exp(s_j)$ ✗
- quand $s_i \times x_i \geq 0$ ou $2|s_i| < |x_i|$, alors $\exp(s_i) \leq \exp(s_{i+1})$ ✓

Lemme technique 2/2

On note $s_i = \oplus_{j=0}^{i-1} x_i$ et $e_i = \mathrm{errf}(s_i, x_i)$ (et donc $s_{i+1} + e_i = s_i + x_i$).

- $\exp(s_i)$ croissante ✗
- sachant que e_{i-1} et e_j sont non nul, $\exp(s_i) \leq \exp(s_j)$ ✗
- quand $s_i \times x_i \geq 0$ ou $2|s_i| < |x_i|$, alors $\exp(s_i) \leq \exp(s_{i+1})$ ✓
- quand $s_i \times x_i < 0$ et $|x_i| \leq 2|s_i|$ et $s_{i+1} \neq 0$, alors $\exp(s_i) \leq \exp(s_{i+2})$ ✓

Lemme technique 2/2

On note $s_i = \oplus_{j=0}^{i-1} x_i$ et $e_i = \mathrm{errf}(s_i, x_i)$ (et donc $s_{i+1} + e_i = s_i + x_i$).

- $\exp(s_i)$ croissante ✗
- sachant que e_{i-1} et e_j sont non nul, $\exp(s_i) \leq \exp(s_j)$ ✗
- quand $s_i \times x_i \geq 0$ ou $2|s_i| < |x_i|$, alors $\exp(s_i) \leq \exp(s_{i+1})$ ✓
- quand $s_i \times x_i < 0$ et $|x_i| \leq 2|s_i|$ et $s_{i+1} \neq 0$, alors $\exp(s_i) \leq \exp(s_{i+2})$ ✓
- si on supprime les zéros en entrée, on a pou r $i \leq j$

$$\exp(s_i) \leq \max\left(\exp(s_j), \exp(s_{j+1})\right)$$ ✓

VecSum est correct

Théorème

Si l « overlap par au plus d bits » et est de longueur inférieure à $2 + 2^{p-1}$, alors VecSum(l) est « S-non overlapping ».

VecSum est correct

Théorème

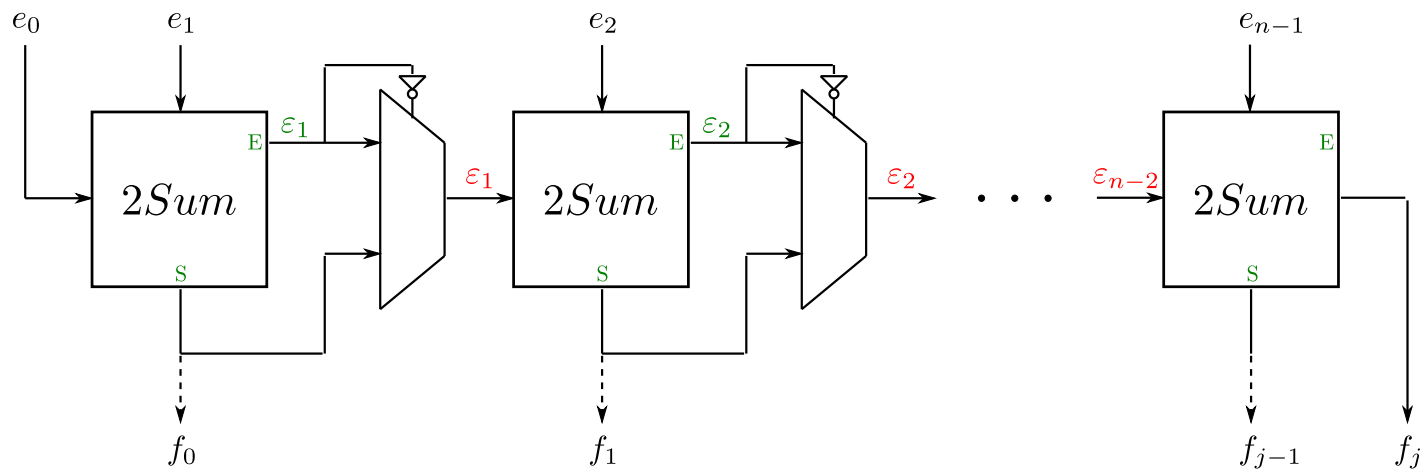
Si l « overlap par au plus d bits » et est de longueur inférieure à $2 + 2^{p-1}$, alors VecSum(l) est « S-non overlapping ».

- en utilisant l'exposant $\max(\exp(s_j), \exp(s_{j+1}))$
- en gérant les cas particuliers...
- OK pour des listes de taille 4.5×10^{15} en *binary64* et plus de 8 millions en *binary32*

Plan

VecSumErrBranch

Deuxième étape de la renormalisation :



Entrée/sortie

En entrée, on a une expansion « S-non overlapping » retournée, une expansion avec la propriété :

$$\text{fun } y\ x \Rightarrow \exists e : Z, \exists n : Z, y = n \times 2^e \wedge |x| < 2^e.$$

En sortie, on a une expansion « ulp-non overlapping » :

$$\text{fun } x\ y \Rightarrow |y| \leq \mathrm{ulp}(x).$$

Propriétés de « S-non overlapping »

Si $(a :: b :: l)$ est « S-non overlapping », alors

- $(a + b :: l)$ est « S-non overlapping »,
- $(\circ(a + b) :: l)$ est « S-non overlapping »,
- $(a + b - \circ(a + b) :: l)$ est « S-non overlapping ».

Propriétés de « S-non overlapping »

Si $(a :: b :: l)$ est « S-non overlapping », alors

- $(a + b :: l)$ est « S-non overlapping »,
- $(\circ(a + b) :: l)$ est « S-non overlapping »,
- $(a + b - \circ(a + b) :: l)$ est « S-non overlapping ».

Théorème

Supposons que $(b :: a :: l)$ *est « S-non overlapping » et que* $b + a - b \oplus a \neq 0$, *alors*

$$\left(\; b \oplus a \; , \; \circ \left((b + a - b \oplus a) + \sum_{i=0}^{n} l_i \right) \right)$$

est « ulp-non overlapping ».

VecSumErrBranch est correct

Théorème

Si l est « S-non overlapping » retourné, alors VecSumErrBranch(l) est « ulp-non overlapping ».

VecSumErrBranch est correct

Théorème

Si l est « S-non overlapping » retourné, alors VecSumErrBranch(l) est « ulp-non overlapping ».

- On peut utiliser des FastTwoSum ✓
- Preuves plus simples (dénormalisés ✓ , zéros intermédiaires ✓)
- subtilité sur les 2 derniers termes

Plan

Théorème final

```
Context { prec_gt_0_ : Prec_gt_0 p }.
Variable d:Z.
Hypothesis d_betw: (0 < d <= p-2)%Z.

Variable l: list R.
Hypothesis Fl: Forall format l.
Hypothesis Hl: InputVecSum d l.

Let res := VecSumErrBranch (rev (VecSum l)).

Lemma Renorm_1: Forall format res.
Lemma Renorm_2:
      fold_right Rplus 0 l = fold_right Rplus 0 res.
Lemma Renorm_3: INR (length l) <= 2 + bpow (p - 1) ->
          OutputVecSumErrBranch res.
```

Conclusion

Je m'attendais à

- des lemmes stupides
- des preuves en plus
- des soucis avec les zéros intermédiaires
- trouver des lemmes marrants
  (comme $s_i \neq 0$, sauf si tous les $(x_k)_{k<i}$ sont nuls)

Conclusion

Je m'attendais à

- des lemmes stupides
- des preuves en plus
- des soucis avec les zéros intermédiaires
- trouver des lemmes marrants
  (comme $s_i \neq 0$, sauf si tous les $(x_k)_{k<i}$ sont nuls)

Je ne m'attendais pas à

- des notations très différentes entre le papier et la preuve formelle
- des inductions, dans des inductions, dans des *case split*
- une erreur dans l'article IEEE-TC
- de GROS soucis avec les zéros intermédiaires

Conclusion

Je m'attendais à

- des lemmes stupides
- des preuves en plus
- des soucis avec les zéros intermédiaires
- trouver des lemmes marrants
  (comme $s_i \neq 0$, sauf si tous les $(x_k)_{k<i}$ sont nuls)

Je ne m'attendais pas à

- des notations très différentes entre le papier et la preuve formelle
- des inductions, dans des inductions, dans des *case split*
- une erreur dans l'article IEEE-TC
- de GROS soucis avec les zéros intermédiaires

Et maintenant ?

- les étapes restants, s'il y en a besoin...