Génération de schémas d'évaluation avec contraintes pour des expressions arithmétiques

Christophe Mouilleron
équipe Arénaire, LIP, ENS de Lyon

7 juin 2011

Séminaire de l'équipe-projet CARAMEL


Questions soulevées par l'implantation d'expressions arithmétiques

En complexité algébrique :

- minimisation du nombre d'opérations ([Knuth, 1998] pour x^n, [Pan, 1966] et [Borodin, 1971] pour les polynômes),
- préconditionnement [Knuth, 1962], [Eve, 1964], [Paterson et Stockmeyer, 1973].

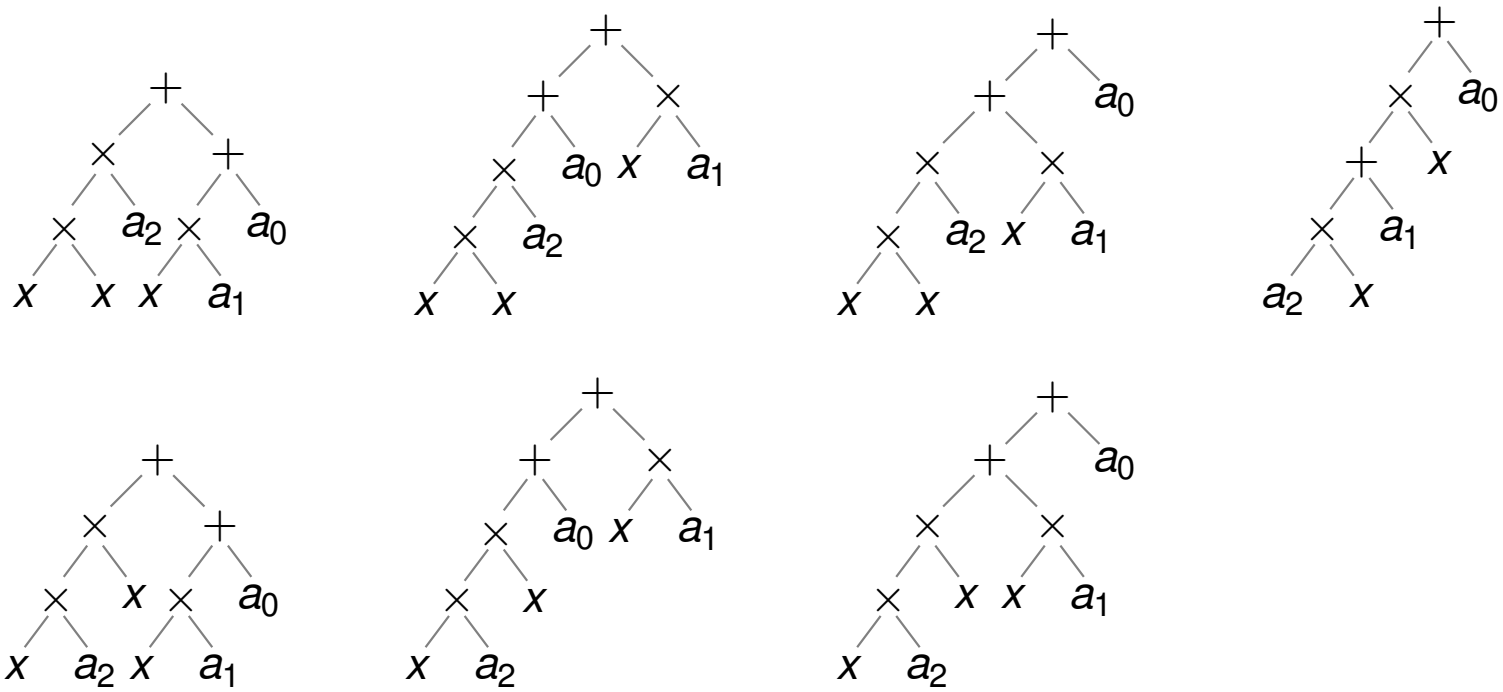
En compilation :

- exploiter au maximum le parallélisme de la machine.

Lors de l'analyse des erreurs arrondis :

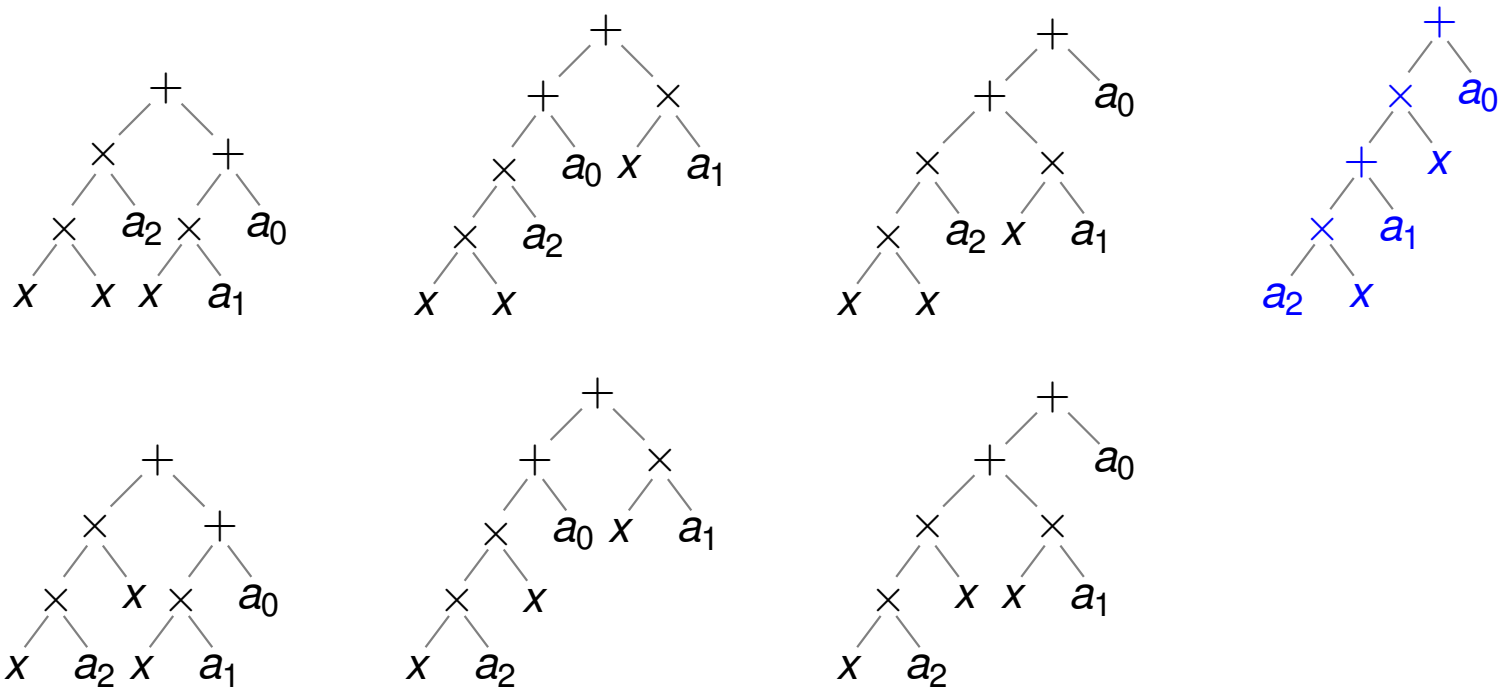
- éviter les pertes de précision,
- garantir une borne d'erreur sur l'évaluation [Higham, 2002].

Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

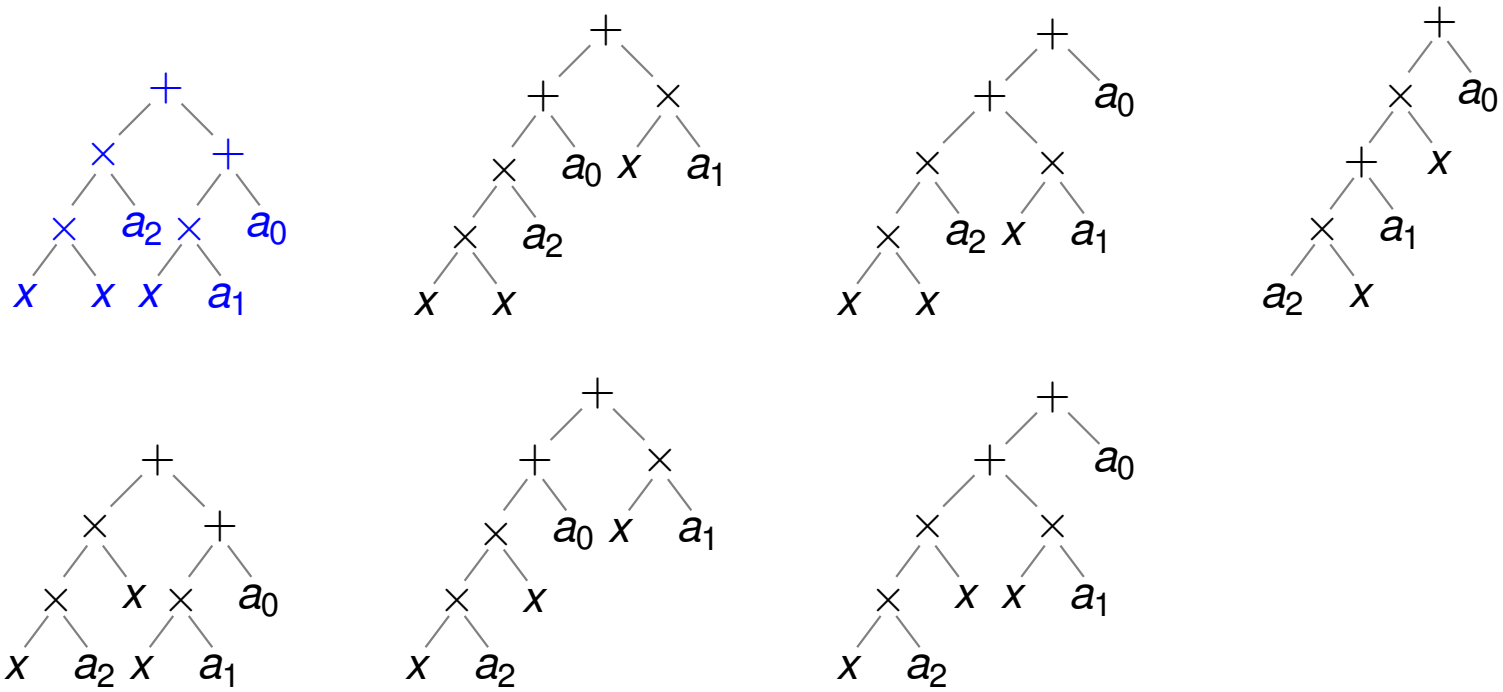
Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

$\rightsquigarrow$ Horner.

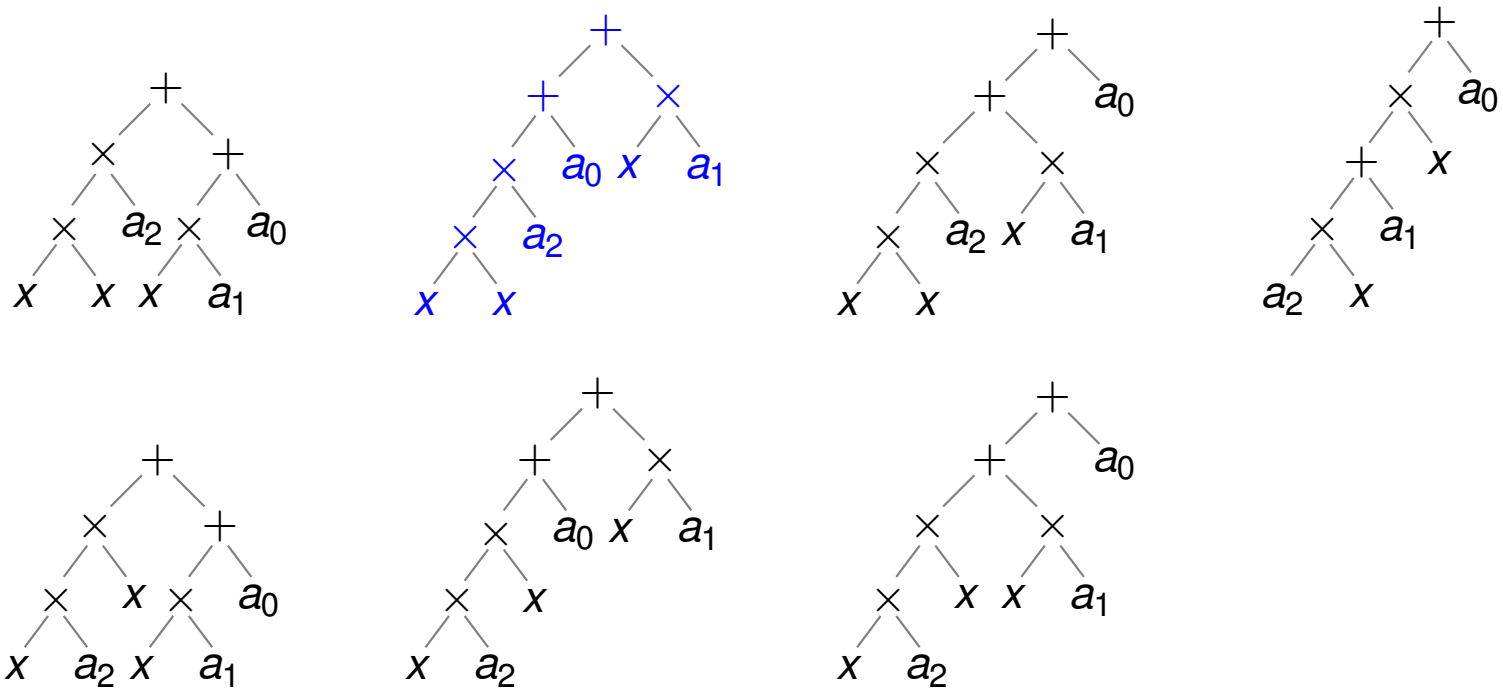
Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

$\rightsquigarrow$ Estrin.

Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

$\rightsquigarrow$ Évaluation pair-impair.

Analyse d'erreur pour l'évaluation d'un polynôme

Théorème [Higham, 2002]

Si $\hat{r}$ est le résultat de l'évaluation de $p(x)$ avec la méthode de Horner en arithmétique virgule flottante double précision, on a : $|\hat{r} - p(x)| \leq \gamma_{2n} \sum_{i=0}^{n} |p_i| . |x|^i$ où $\gamma_n = \frac{nu}{1-nu}$ avec $u = 2^{-53}$.

Théorème [Higham, 2002]

Pour Estrin, on a $\gamma_{n+\lceil \log(n+1) \rceil}$ au lieu de γ_{2n}.

En pratique [Revy, 2006] :

- ces bornes sont pessimistes,
- Horner se comporte souvent mieux que Estrin.

L'outil CGPE (Code Generation for Polynomial Evaluation)

[Revy, 2009], [M. et Revy, 2011]

Problèmes :

- compromis latence/nombre de $\times$ à trouver,
- garantie sur l'erreur d'évaluation.

$\rightsquigarrow$ programme C++ pour générer des schémas et filtrer.

Dans cet exposé :

- formalisation et généralisation de l'approche,
- amélioration des heuristiques de CGPE,
- application à d'autres problèmes.

Plan

Plan

Définition des schémas d'évaluation

Expression arithmétique	$a_0 + a_1 \cdot x + a_2 \cdot x^2$	1
	$\downarrow$	
Parenthésage usuel	$(a_0 + (a_1 \cdot x)) + (a_2 \cdot (x \cdot x))$	1
	$\downarrow$ commutativité associativité distributivité	
Parenthésages possibles	$(a_0 + (a_1 \cdot x)) + (a_2 \cdot (x \cdot x))$	160
	$(a_0 + (a_1 \cdot x)) + ((a_2 \cdot x) \cdot x)$	
	$a_0 + ((a_1 \cdot x) + ((a_2 \cdot x) \cdot x))$	
	$a_0 + ((a_1 + (a_2 \cdot x)) \cdot x)$	
	$((a_1 + (a_2 \cdot x)) \cdot x) + a_0$	...

Définition des schémas d'évaluation

Expression arithmétique	$a_0 + a_1 \cdot x + a_2 \cdot x^2$	1
	$\downarrow$	
Parenthésage usuel	$(a_0 + (a_1 \cdot x)) + (a_2 \cdot (x \cdot x))$	1
	$\downarrow$ commutativité associativité distributivité	
Parenthésages possibles	$(a_0 + (a_1 \cdot x)) + (a_2 \cdot (x \cdot x))$	160
	$(a_0 + (a_1 \cdot x)) + ((a_2 \cdot x) \cdot x)$	
	$a_0 + ((a_1 \cdot x) + ((a_2 \cdot x) \cdot x))$	
	$a_0 + ((a_1 + (a_2 \cdot x)) \cdot x)$	
	$((a_1 + (a_2 \cdot x)) \cdot x) + a_0$ ...	

Définition des schémas d'évaluation

Expression arithmétique $a_0 + a_1 \cdot x + a_2 \cdot x^2$ — 1

↓

Parenthésage usuel $(a_0 + (a_1 \cdot x)) + (a_2 \cdot (x \cdot x))$ — 1

↓ commutativité, associativité, distributivité

Parenthésages possibles [. . .] — 160

↓ $+$ et $\times$ sont commutatifs en arithmétique approchée

Schémas d'évaluation — 7

$(a_0 + (a_1 \cdot x)) + (a_2 \cdot (x \cdot x))$
$(a_0 + (a_2 \cdot (x \cdot x))) + (a_1 \cdot x)$
$a_0 + ((a_1 \cdot x) + (a_2 \cdot (x \cdot x)))$
$a_0 + (a_1 + (a_2 \cdot x)) \cdot x$
$(a_0 + (a_1 \cdot x)) + ((a_2 \cdot x) \cdot x))$
$(a_0 + ((a_2 \cdot x) \cdot x)) + (a_1 \cdot x)$
$a_0 + ((a_1 \cdot x) + ((a_2 \cdot x) \cdot x)))$

Famille $\mathcal{F}$ d'expressions arithmétiques

Hypothèse 1

On dispose d'un ordre total sur les éléments de $\mathcal{F}$.

Hypothèse 2

On sait décomposer une expression arithmétique $f \in \mathcal{F}$, c'est à dire calculer l'ensemble des $(\diamond, \{f_1, f_2\})$ où $\diamond \in \{+, \times\}$ et $f_1, f_2 \in \mathcal{F}$ tels que $f = f_1 \diamond f_2$ et $f_1 < f$, $f_2 < f$.

Exemples :

- pour x^n, on a $x^i \cdot x^j$ avec $i \leq j$ et $i + j = n$,
- pour un polynôme univarié, on a :
  - $\rightarrow$ la séparation du support en 2 sous-supports,
  - $\rightarrow$ les factorisations par x^i pour $i \leq \mathrm{val}(p)$.

Famille $\mathcal{F}$ d'expressions arithmétiques

Hypothèse 1

On dispose d'un ordre total sur les éléments de $\mathcal{F}$.

Hypothèse 2

On sait décomposer une expression arithmétique $f \in \mathcal{F}$, c'est à dire calculer l'ensemble des $(\diamond,\ f_1, f_2\ )$ où $\diamond \in \{+, \times\}$ et $f_1, f_2 \in \mathcal{F}$ tels que $f = f_1 \diamond f_2$ et $f_1 \leq f_2 < f$.

Exemples :

- pour x^n, on a $x^i \cdot x^j$ avec $i \leq j$ et $i + j = n$,
- pour un polynôme univarié, on a :
  - $\rightarrow$ la séparation du support en 2 sous-supports,
  - $\rightarrow$ les factorisations par x^i pour $i \leq \mathrm{val}(p)$.

Famille $\mathcal{F}$ d'expressions arithmétiques

Hypothèse 1

On dispose d'un ordre total sur les éléments de $\mathcal{F}$.

Hypothèse 2

On sait décomposer une expression arithmétique $f \in \mathcal{F}$, c'est à dire calculer l'ensemble des $(\diamond,\ f_1, f_2\ )$ où $\diamond \in \{+, \times\}$ et $f_1, f_2 \in \mathcal{F}$ tels que $f = f_1 \diamond f_2$ et $f_1 \leq f_2 < f$.

Exemples :

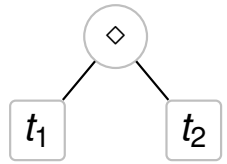
- pour x^n, on a $x^i \cdot x^j$ avec $i \leq j$ et $i + j = n$,
- pour un polynôme univarié, on a :
  - $\rightarrow$ la séparation du support en 2 sous-supports,
  - $\rightarrow$ les factorisations par x^i pour $i \leq \mathrm{val}(p)$.

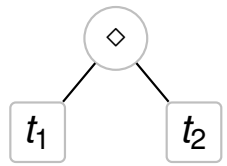
Plan

Algorithme de génération

Algorithme 1 : `génère`

Entrée : $f \in \mathcal{F}$
Sortie : $\mathcal{S} =$ ensemble des schémas d'évaluation pour f

1 $\ell \leftarrow \texttt{decompose}(f)$
2 **if** $\ell = \emptyset$ **then** $\mathcal{S} \leftarrow \{f\}$ **else** $\mathcal{S} \leftarrow \emptyset$
3 **foreach** $(\diamond, f_1, f_2) \in \ell$ **do**
4 $\quad \mathcal{S}_1 \leftarrow \texttt{génère}(f_1)$
5 $\quad$ **if** $f_1 = f_2$ **then**
6 $\quad\quad$ **for** $\{t_1, t_2\} \in \mathcal{P}(\mathcal{S}_1)$ **do** $\mathcal{S} \leftarrow \mathcal{S} \cup \{$  $\}$

7 $\quad$ **else**
8 $\quad\quad \mathcal{S}_2 \leftarrow \texttt{génère}(f_2)$
9 $\quad\quad$ **for** $(t_1, t_2) \in \mathcal{S}_1 \times \mathcal{S}_2$ **do** $\mathcal{S} \leftarrow \mathcal{S} \cup \{$  $\}$

10 **return** $\mathcal{S}$

Passage à l'algorithme de dénombrement

Algorithme 2 : `dénombre`

Entrée : $f \in \mathcal{F}$
Sortie : $r =$ nombre de schémas d'évaluation pour f

```
1  ℓ ← decompose(f)
2  if ℓ = ∅ then r ← 1 else r ← 0
3  foreach (⋄, f1, f2) ∈ ℓ do
4      r1 ← dénombre(f1)
5      if f1 = f2 then
6          r ← r + r1(r1 + 1) / 2
7      else
8          r2 ← dénombre(f2)
9          r ← r + r1 · r2
10 return r
```

Remarques

En pratique, on utilise la technique de mémoïsation pour stocker les résultats des appels récursifs.

Complexité en nombre d'opérations sur des entiers (GMP) :

nombre d'appels récursifs $\times$ coût maximal d'un appel.

Exemples :

- Pour x^n : $O(n) \times O(n) = O(n^2)$
  $\rightsquigarrow$ il existe des algorithmes exploitant les séries génératrices en $O(n \log n)$,
- Pour $p(x)$: $O(2^n) \times O(2^n) = O(4^n)$
  $\rightsquigarrow$ pas de meilleur algorithme à ma connaissance.

Ordres de grandeur pour les nombres de schémas

Deux cas particuliers bien connus :

- le nombre de schémas pour x^n [Wedderburn, Etherington]

$$w_n \sim \frac{\eta\,\xi^n}{n^{3/2}} \text{ où } \begin{cases} \xi \approx 2.48325 \\ \eta \approx 0.31877 \end{cases} \text{ [Otter, 1948],}$$

- le nombre de schémas pour $\sum_{i=0}^{n} a_i$ est $(2n-1)!! \sim \sqrt{2}\left(\frac{2n}{e}\right)^n$.

n	w_n	$(2n-1)!!$	nb de schémas pour $p(x)$	nb de schémas pour $\alpha + y \cdot p(x)$
1	1	1	1	10
2	1	3	7	481
3	1	15	163	88384
4	2	105	11602	57363910
5	3	945	2334244	122657263474
6	6	10395	1304066578	829129658616013
7	11	135135	1972869433837	17125741272619781635
8	23	2027025	8012682343669366	1055157310305502607244946
9	46	34459425	86298937651093314877	190070917121184028045719056344
10	98	654729075	2449381767217281163362301	985436908485543809474905225911916 72

$\rightsquigarrow$ Besoin d'heuristiques fortes à la génération.

Plan

Le standard IEEE 754 d'un coté...

Définition précise de l'arithmétique flottante :

- différents formats (simple précision, double précision, ...),
- valeurs particulières ($\pm 0, \pm\infty$, NaN),
- différents modes d'arrondis (au plus proche pair, ...),
- comportement des fonctions mathématiques usuelles :
  - $\rightarrow$ valeurs particulières (ex : $\sqrt{-0} = -0$),
  - $\rightarrow$ arrondi correct requis/recommandé.

Motivation :

- reproductibilité des calculs,
- résultat indépendant de l'architecture.

Le processeur ST231 de l'autre

C'est un processeur utilisé dans l'embarqué (multimedia) qui ne dispose que de l'arithmétique entière sur 32 bits.

Parallélisme :

- c'est un VLIW avec 4 voies,
- on ne peut lancer que 2 multiplications par cycle,
- quelques contraintes liées à la taille des opérandes.

Coût des opérateurs :

- l'addition coûte $\mathcal{C}_+ = 1$ cycle,
- la multiplication coûte $\mathcal{C}_\times = 3$ cycles.

Implanter un opérateur pour l'arithmétique flottante

$\sqrt{t}$ avec t flottant

Différentes étapes :

- traitement des cas particuliers, $\sqrt{+\infty} = +\infty, \ \ldots$
- réduction d'argument, $\sqrt{t} \rightsquigarrow \sqrt{1+x}$ avec $x \in [0,1)$
- approximation polynomiale, $\|p(x) - \sqrt{1+x}\|_{[0,1]} < \varepsilon_1$
- évaluation du polynôme, $|\widehat{p(x)} - p(x)| < \varepsilon_2$
- routine d'arrondi finale. $\widehat{p(x)} \rightsquigarrow \diamond(\sqrt{t})$

$\rightsquigarrow$ Utilisation de l'arithmétique virgule fixe pour l'évaluation.

Implanter un opérateur pour l'arithmétique flottante

$\sqrt{t}$ avec t flottant

Différentes étapes :

- traitement des cas particuliers, $\sqrt{+\infty} = +\infty, \ \ldots$
- réduction d'argument, $\sqrt{t} \rightsquigarrow \sqrt{1+x}$ avec $x \in [0,1)$
- approximation polynomiale, $\|p(x) - \sqrt{1+x}\|_{[0,1]} < \varepsilon_1$
- évaluation du polynôme, $|\widehat{p(x)} - p(x)| < \varepsilon_2$
- routine d'arrondi finale. $\widehat{p(x)} \rightsquigarrow \diamond(\sqrt{t})$

$\rightsquigarrow$ Utilisation de l'arithmétique virgule fixe pour l'évaluation.

Cas plus général : évaluer $q(x, y) = \alpha + y \cdot p(x)$

[Jeannerod, Knochel, Monat et Revy, 2010]

1. Deuxième variable y due à la réduction d'argument :

$$t = m_t \cdot 2^{e_t} \quad \Rightarrow \quad \sqrt{t} = \sqrt{m_t} \cdot 2^{\frac{e_t}{2}} = \underbrace{\sqrt{1+x}}_{\approx p(x)} \cdot \underbrace{2^{\frac{e_t}{2} - \lfloor \frac{e_t}{2} \rfloor}}_{= y \in \{1, \sqrt{2}\}} \cdot 2^{\lfloor \frac{e_t}{2} \rfloor}$$

2. Routine d'arrondi plus facile lorsqu'on a :

$$\begin{array}{rccccccc} & 0 & < & \widehat{r} & - & y \cdot \sqrt{1+x} & < & 2^{-24} \\ \Leftrightarrow & -2^{-25} & < & \widehat{r} & - & (2^{-25} + y \cdot \sqrt{1+x}) & < & 2^{-25} \end{array}$$

Posons $q(x, y) = 2^{-25} + y \cdot p(x)$. On peut prendre :

$$\left\| q(x,y) - \left(2^{-25} + y\sqrt{1+x}\right) \right\| \leq \sqrt{2} \left\| p(x) - \sqrt{1+x} \right\|_{[0,1]} < 2^{-26}.$$

Donc il suffit alors de calculer $\hat{r}$ tel que $|\hat{r} - q(x, y)| < 2^{-26}$.

Problématique

Étant donnés :

- les coefficients de $p(x)$ ou $q(x, y) = \alpha + y \cdot p(x)$,
- des intervalles de valeurs pour x et y,
- le nombre de cycles entre l'obtention des valeurs de x et y,
- une borne d'erreur ε,

trouver une façon de faire l'évaluation :

1. avec suffisamment de précision
   $\rightarrow$ on doit pouvoir garantir que l'erreur due à l'évaluation sera toujours plus petite que ε,
2. le plus rapidement possible sur notre architecture cible.

Plan

Ajout de contraintes numériques

Pour trouver des schémas précis, on va générer uniquement des schémas :

- évaluables en arithmétique virgule fixe non-signée.
  - → évite un éventuel surcoût à cause de la gestion des signes,
  - → gain en précision de 1 bit.

- dont chaque quantité intermédiaire calculée tient dans un format virgule fixe imposé à l'avance.
  - → force les deux opérandes d'une addition à être du même format pour éviter d'introduire une instruction de décalage.
  - → introduit le besoin d'éliminer les schémas donnant lieu à des résultats non représentables dans le format imposé.

Valeurs associées à chaque schéma d'évaluation

À l'instar de [Martel, 2007], on associe à chaque schéma $\mathcal{G}$:

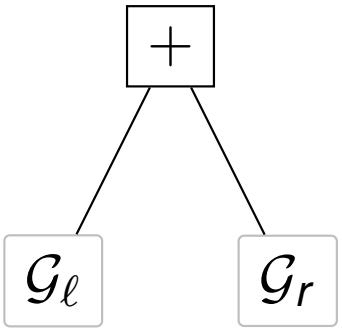
- un intervalle de valeur **value**($\mathcal{G}$),
- un intervalle **error**($\mathcal{G}$) pour encadrer l'erreur d'évaluation,
- le format virgule fixe **format**($\mathcal{G}$) dans lequel sera stockée la valeur absolue du résultat de l'évaluation.

Garantie

Si on a un schéma d'évaluation $\mathcal{G}$ pour l'expression mathématique f, alors le résultat $\hat{r}$ obtenu via $\mathcal{G}$ vérifie :

- $\hat{r} \in$ **value**($\mathcal{G}$),
- on peut garantir le signe de $\hat{r}$,
- $|\hat{r}|$ est au format **format**($\mathcal{G}$),
- $r - \hat{r} \in$ **error**($\mathcal{G}$) où r est le résultat mathématique.

Génération - cas de l'addition



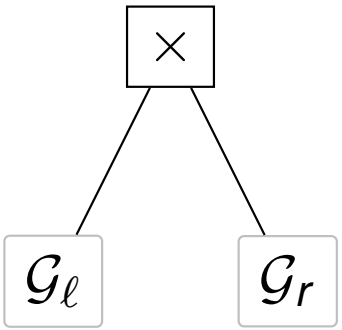
- Si **format**$(\mathcal{G}_\ell) \neq$ **format**$(\mathcal{G}_r)$, le schéma n'est pas valide.
- Si le résultat de l'addition ne tient pas dans le format de **format**$(\mathcal{G})$, le schéma n'est pas valide.
- Sinon, on garde le schéma et on définit :

$$\textbf{value}(\mathcal{G}) = \textbf{value}(\mathcal{G}_\ell) + \textbf{value}(\mathcal{G}_r),$$

$$\textbf{error}(\mathcal{G}) = \textbf{error}(\mathcal{G}_\ell) + \textbf{error}(\mathcal{G}_r).$$

$\rightsquigarrow$ pas d'overflow possible, donc l'addition est sans erreur.

Génération - cas de la multiplication



- Si le résultat de la multiplication ne rentre pas dans le format de **format**($\mathcal{G}$), le schéma n'est pas valide.
- Sinon, on garde le schéma et on définit :

$$\textbf{value}(\mathcal{G}) = \textbf{value}(\mathcal{G}_\ell) \cdot \textbf{value}(\mathcal{G}_r),$$

$$\begin{aligned} \textbf{error}(\mathcal{G}) &= [0, 2^{i-32}] + \textbf{error}(\mathcal{G}_\ell) \cdot \textbf{error}(\mathcal{G}_r) \\ &+ \textbf{error}(\mathcal{G}_\ell) \cdot \textbf{value}(\mathcal{G}_r) + \textbf{error}(\mathcal{G}_r) \cdot \textbf{value}(\mathcal{G}_\ell). \end{aligned}$$

Plan

Approche complète

1. Analyse au niveau des expressions arithmétiques pour déterminer les latences atteignables en parallélisme infini.

2. Génération de schémas tels que :
   - latence suffisamment faible pour conduire à un schéma final de latence optimale,
   - schéma évaluable en arithmétique virgule fixe non signée sans introduire de décalages,
   - pas plus de 50 schémas (tri par nombre de $\times$).

3. Élimination des schémas dont l'ordonnancement en parallélisme borné prend plus de temps que la latence en parallélisme infini.

Timings

Fonction	$x^{1/2}$	$x^{1/3}$	$\log_2(1+x)$	$\frac{1}{\sqrt{1+x^2}}$	$\frac{\exp(1+x)}{1+x}$
Degré	(8,1)	(8,1)	(6,0)	(7,0)	(10,0)
Latence obtenue	13	16	11	11	13
Calcul latence	50ms	56ms	1ms	4ms	124ms
Génération	1s [50]	45s [50]	33ms [6]	41ms [50]	7s [50]
Sélection	32s [11]	93ms [50]	15ms [2]	916ms [2]	12s [18]
Temps total	34s	46s	60ms	1s	19s

$\rightsquigarrow$ gain de $\approx$ 15% par rapport à l'approche dans [Revy, 2009].

Plan

Motivation

Parfois, élever au carré coûte moins cher qu'un vrai produit :

- dans FLIP, 12 cycles contre 21
  [Jeannerod, Joudran-Lu, Monat et Revy, 2011],
- pour les matrices [Bodrato, 2010].

De même, le coût d'une opération peut varier en fonction de la nature des opérandes :

- évaluer $p(x)$ à coefficients scalaires en une matrice
  $\rightsquigarrow$ multiplication "scalaire $\times$ matrice" en $O(n^2)$
  et multiplication "matrice $\times$ matrice" en $O(n^3)$.

Problème : générer des schémas d'évaluation avec le moins d'opérations possible d'un type fixé.

Calcul de l'exponentielle d'une matrice [Higham, 2008]

1. mise à l'échelle pour réduire le conditionnement : $A \to 2^{-s} A =: \hat{A}$
2. approximation de Padé : $\exp(\hat{A}) \approx \frac{p_n(\hat{A})}{q_n(\hat{A})}$
3. élévation au carré s fois.

Pour des raisons de symétrie, $q_n(A) = p_n(-A)$.

Problème : Comment évaluer efficacement les parties paires et impaires de $p_n(x)$ en A ?

Plan

Adaptation de la génération

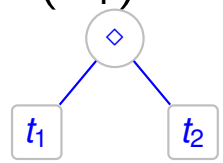
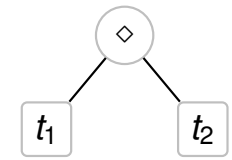
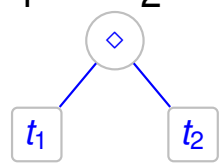
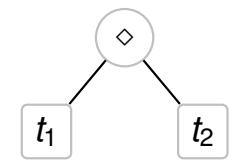
Idée : Restriction aux schémas dont le nombre d'opérations à minimiser est borné par un nouveau paramètre de la fonction de génération récursive.

Principe :

- s'appuyer sur une borne supérieure fine *a priori* pour lancer le nouveau générateur
  $\rightsquigarrow$ `bound`: $\mathcal{F} \rightarrow$`int`,
- compter le nombre d'opérations pour chaque schéma généré et éliminer ceux qui dépassent la borne passée en paramètre
  $\rightsquigarrow$ `hint` : schéma $\rightarrow$ `int`,
- mettre à jour la borne avant un appel récursif
  $\rightsquigarrow$ `update_bound` : decomposition $\times$ `int` $\rightarrow$ `int`.

Algorithme 3 : `génère2`

Entrée : $f \in \mathcal{F}$ et une borne B
Sortie : $\mathcal{S}$ = schémas d'évaluation t pour f tels que $\texttt{hint}(t) \leq B$

1 $\ell \leftarrow \texttt{decompose}(f)$
2 **if** $\ell = \emptyset$ *et* $\texttt{hint}(f) \leq B$ **then** $\mathcal{S} \leftarrow \{f\}$ **else** $\mathcal{S} \leftarrow \emptyset$
3 **foreach** $(\diamond, f_1, f_2) \in \ell$ **do**
4 $\quad B' \leftarrow \texttt{update\_bound}((\diamond, f_1, f_2), B)$
5 $\quad \mathcal{S}_1 \leftarrow \texttt{génère2}\,(f_1, B')$
6 $\quad$ **if** $f_1 = f_2$ **then**
7 $\quad\quad$ **for** $\{t_1, t_2\} \in \mathcal{P}(\mathcal{S}_1)$ **do**
8 $\quad\quad\quad$ **if** $\texttt{hint}\left(\right.$  $\left.\right) \leq B$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \left\{\right.$  $\left.\right\}$
9 $\quad$ **else**
10 $\quad\quad \mathcal{S}_2 \leftarrow \texttt{génère2}\,(f_2, B')$
11 $\quad\quad$ **for** $(t_1, t_2) \in \mathcal{S}_1 \times \mathcal{S}_2$ **do**
12 $\quad\quad\quad$ **if** $\texttt{hint}\left(\right.$  $\left.\right) \leq B$ **then** $\mathcal{S} \leftarrow \mathcal{S} \cup \left\{\right.$  $\left.\right\}$
13 **return** $\mathcal{S}$

Résultats pour x^n

Propriété [Knuth, 1998]

On peut évaluer x^n en $\lceil \log_2 n \rceil + \delta_n$ multiplications, où δ_n est le nombre de 1 dans l'écriture binaire de n.

Trois nouvelles suites ajoutées dans l'encyclopédie de Sloane :

`A186435` nombre de schémas pour x^n avec le moins de $\times$ possible,

`A186437` nombre maximal de carrés dans ces schémas,

`A186520` nombre de ces schémas atteignant le nombre maximal de carrés.

Algorithme classique pour évaluer $p(A)$

[Paterson et Stockmeyer, 1973]

Utilisation de la technique pas de bébé - pas de géant :

- on choisit un entier k entre 1 et n,
- on calcule $A^2, A^3, \ldots, A^k$,
- on voit $p(x)$ comme un polynôme en x^k qu'on évalue par la méthode de Horner.

Exemple pour $n = 7$ et $k = 3$:

$$\Big((p_7 \cdot A + p_6) \cdot A^3 + (p_5 \cdot A^2 + p_4 \cdot A + p_3)\Big) \cdot A^3 + (p_2 \cdot A^2 + p_1 \cdot A + p_0).$$

Choix optimal : $k \approx \sqrt{n}$.
$k = \lceil \sqrt{n} \rceil$ donne $\left\lfloor \frac{n-1}{\lceil \sqrt{n} \rceil} \right\rfloor + \lceil \sqrt{n} \rceil - 1$ multiplications non-scalaires.

Résultats pour $p(A)$ et $\exp(A)$

- on utilise la borne $\left\lfloor \frac{n-1}{\lceil \sqrt{n} \rceil} \right\rfloor + \lceil \sqrt{n} \rceil - 1$,
- la borne diminue de 1 quand la décomposition correspond à un produit "matrice $\times$ matrice".

Constat après expérimentation :

- les schémas obtenus sont des variations de l'algorithme de Paterson et Stockmeyer

Pour le cas de $\exp(A)$:

- on doit maximiser le nombre d'opérations communes aux parties paire et impaire, donc les puissances de A,
- pour $n = 12$, on retrouve le schéma proposé dans [Higham, 2008] parmi 291600 autres.

Plan

En conclusion

En résumé, on a vu :

- comment générer et dénombrer les schémas d'évaluation pour une expression arithmétique,
- comment ajouter des contraintes numériques pour accélérer la recherche de schémas rapides et suffisamment précis dans le cadre de FLIP
- une extension pour s'attaquer à x^n, $p(A)$ et $\exp(A)$.

Perspectives :

- continuer le travail d'analyse pour $p(A)$,
- utilisation du préconditionnement [Knuth, 1962],
- Gestion de transformations plus complexes comme $ab = \frac{1}{4}((a+b)^2 - (a-b)^2)$ [Giorgi, Imbert et Izard, 2009].

Merci pour votre attention !


Bodrato, M. (2010).
A strassen-like matrix multiplication suited for squaring and higher power computation.
In *Proceedings of the 2010 International Symposium on Symbolic and Algebraic Computation*, ISSAC '10, pages 273–280, New York, NY, USA. ACM.


Borodin, A. (1971).
Horner's rule is uniquely optimal.
Theory of machines and computations, pages 45–58.


Eve, J. (1964).
The evaluation of polynomials.
Numerische Mathematik, (6) :17–21.


Giorgi, P., Imbert, L., and Izard, T. (2009).

Optimizing Elliptic Curve Scalar Multiplication for Small Scalars.
In *SPIE - Mathematics for Signal and Information Processing*, volume 7444.


Higham, N. J. (2002).
Accuracy and Stability of Numerical Algorithms.
Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second edition.


Higham, N. J. (2008).
Functions of Matrices : Theory and Computation.
Society for Industrial and Applied Mathematics, Philadelphia, PA, USA.


Jeannerod, C., Knochel, H., Monat, C., and Revy, G. (2010).

Computing floating-point square roots via bivariate polynomial evaluation.
Computers, IEEE Transactions on, PP(99) :1 –1.


Knuth, D. E. (1962).
Evaluation of polynomials by computers.
Communications of the ACM, 5(12) :595–599.


Knuth, D. E. (1998).
Seminumerical Algorithms, volume 2 of *The Art of Computer Programming*.
Addison-Wesley, Third edition.


Martel, M. (2007).
Semantics-based transformation of arithmetic expressions.
In *SAS'07*, volume 4634 of *Lecture Notes in Computer Science*. Springer-Verlag.


Mouilleron, C. and Revy, G. (2011).
Automatic Generation of Fast and Certified Code for Polynomial Evaluation.
accepté pour ARITH 20.


Otter, R. (1948).
The Number of Trees.
The Annals of Mathematics, Second Series, 49(3) :583–599.


Pan, V. Y. (1966).
Methods of Computing Values of Polynomials.
Russian Mathematical Surveys, 21(1) :105–136.


Paterson, M. S. and Stockmeyer, L. J. (1973).
On the number of nonscalar multiplications necessary to evalutate polynomials.

SIAM Journal on Computing, 2(1) :60–66.


Revy, G. (2006).
Analyse et implantation d'algorithmes rapides pour l'évaluation polynomiale sur les nombres flottants.
Master's thesis, École normale supérieure de Lyon, 46 allée d'Italie, F-69364 Lyon cedex 07, France.


Revy, G. (2009).
Implementation of binary floating-point arithmetic on embedded integer processors - Polynomial evaluation-based algorithms and certified code generation.
PhD thesis, Université de Lyon - École Normale Supérieure de Lyon, 46 allée d'Italie, F-69364 Lyon cedex 07, France.