Génération automatique de code rapide et certifié pour évaluer un polynôme

Christophe Mouilleron
École Normale Supérieure de Lyon, Université de Lyon

Travail en collaboration avec Guillaume Revy (*Université de Perpignan*)

Séminaire du projet ALGORITHMS – 29 novembre 2010


Le standard IEEE 754

Définition précise de l'arithmétique flottante :

- différents formats (simple précision, double précision, ...),
- valeurs particulières (± 0, $\pm\infty$, NaN),
- différents modes d'arrondis (au plus proche pair, ...),
- comportement des fonctions mathématiques usuelles :
  - $\rightarrow$ valeurs particulières ($1/+\infty, \ldots$),
  - $\rightarrow$ arrondi correct demandé.

Motivation :

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

Notre cible : le ST231

On dispose de l'arithmétique entière sur 32 bits (multiplication tronquée) $\approx$ arithmétique en virgule fixe.

Coût des opérateurs :

- l'addition coûte $\mathcal{C}_A = 1$ cycle,
- la mutliplication coûte $\mathcal{C}_M = 3$ cycles,
- le décalage coûte 1 cycle.

Parallélisme :

- c'est un VLIW avec 4 voies,
- on est limité à 2 multiplications par cycle,
- quelques contraintes liés à la taille des opérandes.

Implanter un opérateur pour l'arithmétique flottante

Outils techniques :

- réduction d'argument,
- approximation polynomiale (Sollya),
- routine d'arrondi finale.

Remarque : Cette routine d'arrondi est plus simple si on a une *approximation dirigée*[1].

Points clés :

- traitement des cas particuliers,
- évaluation du polynôme d'approximation,
- contrôle des différentes erreurs de façon à garantir qu'on est capable de retourner l'arrondi correct au final.

1. voir [Ercegovac and Lang, 2004, p. 459].

Exemple [2] : l'opérateur $\sqrt{\cdot}$ pour la simple précision

Entrée : $t = m_t \cdot 2^{e_t}$ avec $m_t \in [1, 2)$.

On a :

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

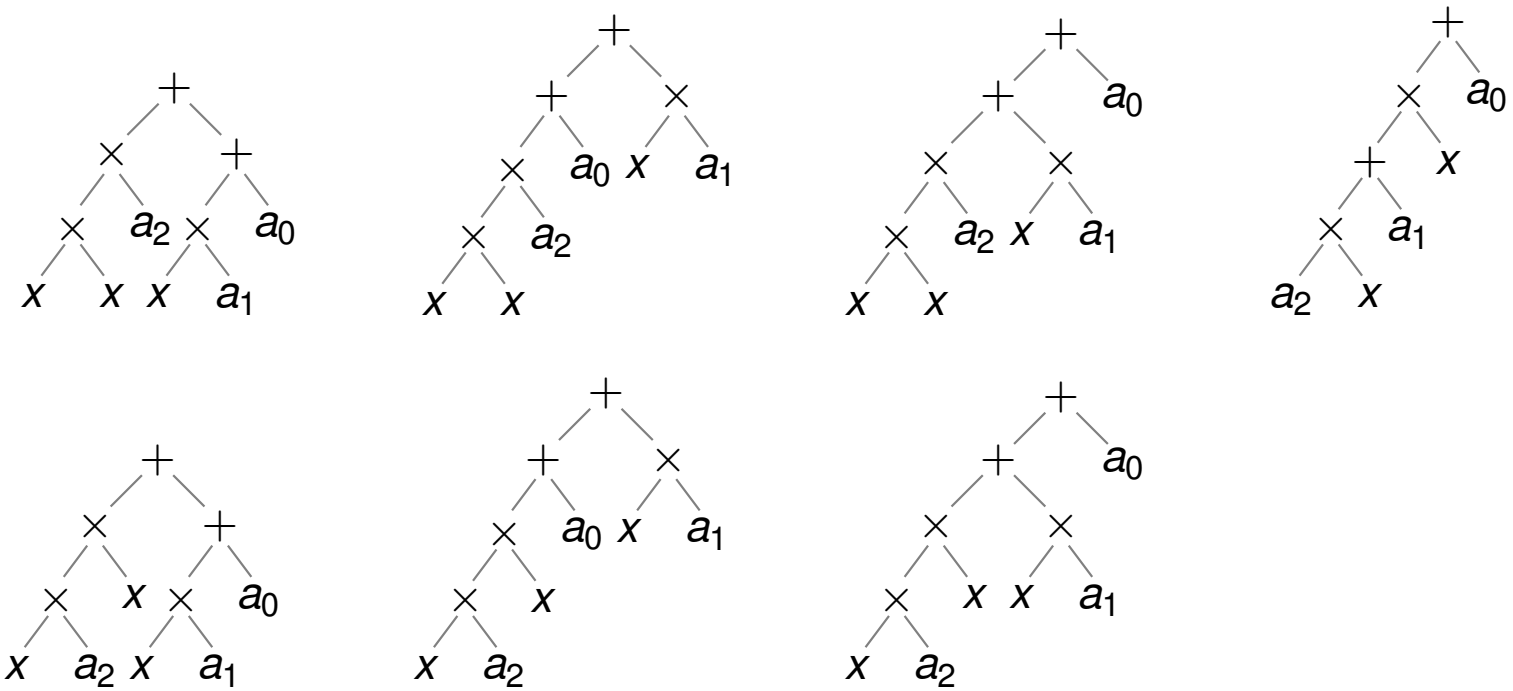
où $x \in [0, 1)$, $y \in \{1, \sqrt{2}\}$ et $p(x)$ est un polynôme d'approximation pour $x \mapsto \sqrt{1+x}$ sur $[0, 1)$.

Pour avoir une approximation dirigée, on ajoute $q_{0,0} = 2^{-25}$.

$\rightsquigarrow$ **cas général** : évaluation de $q(x, y) = q_{0,0} + y \cdot p(x)$.

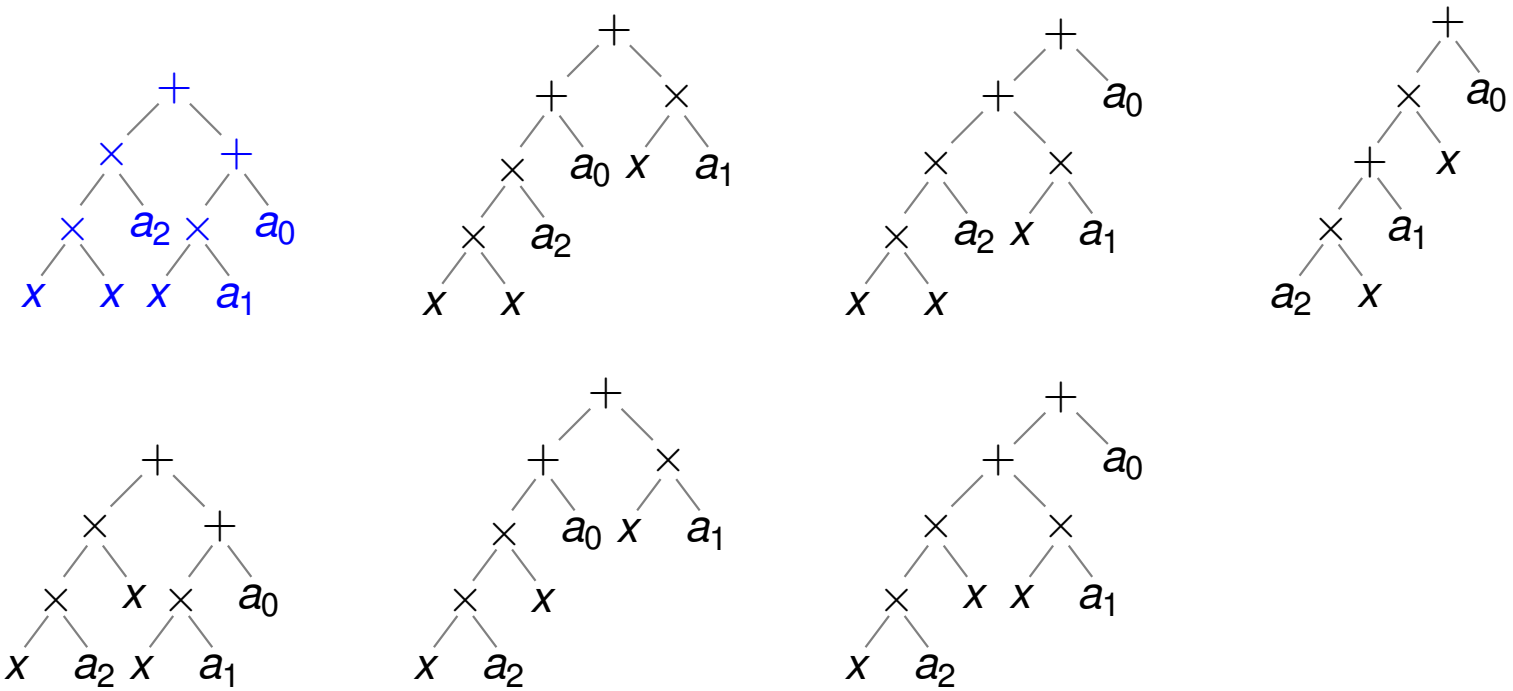
2. voir [Jeannerod et al., 2010].

Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

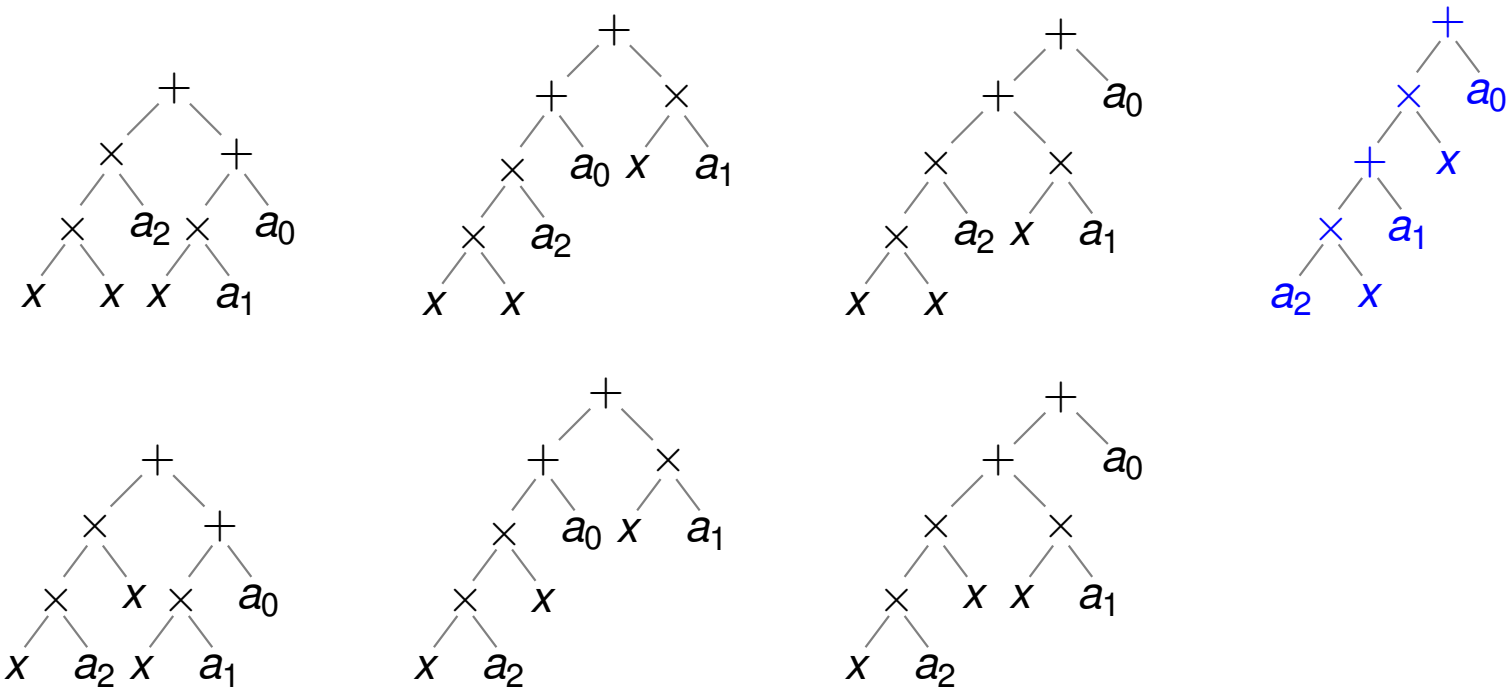
Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

$\rightsquigarrow$ Évaluation naïve.

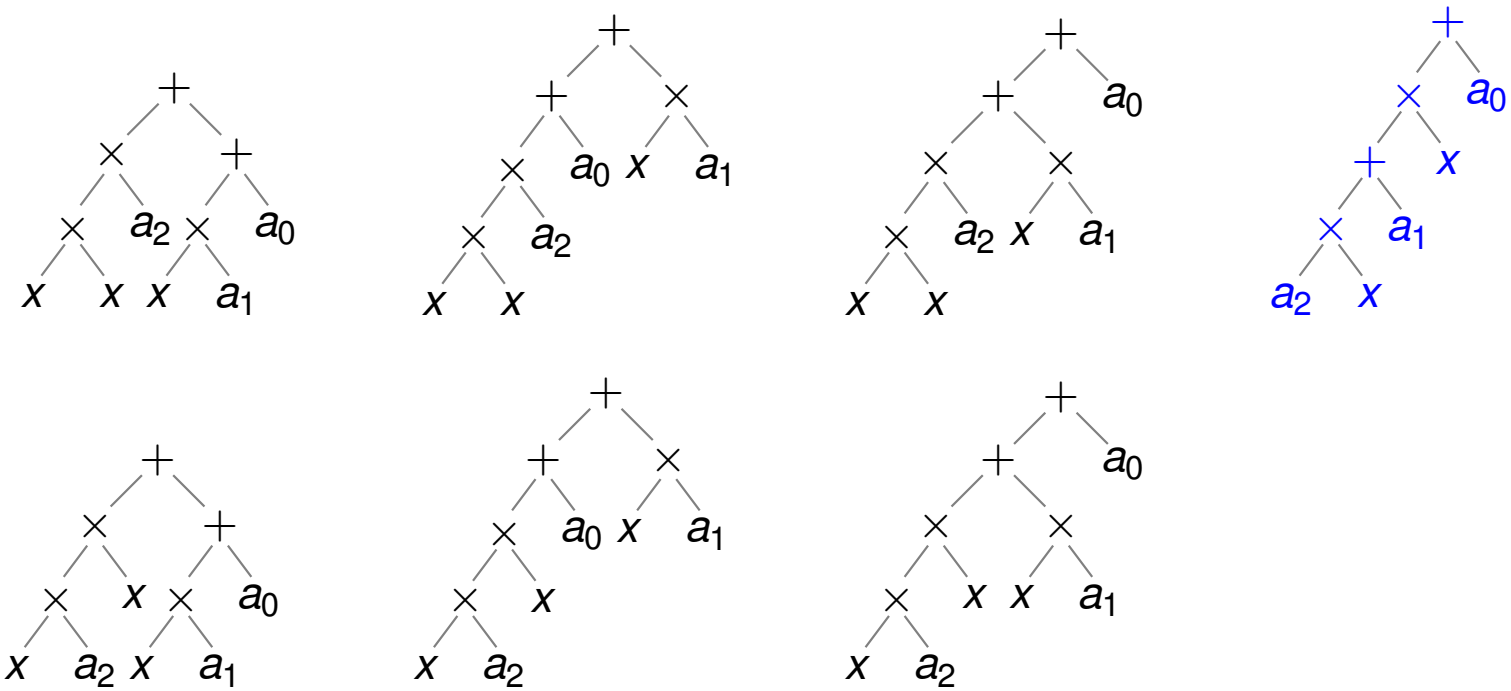
Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

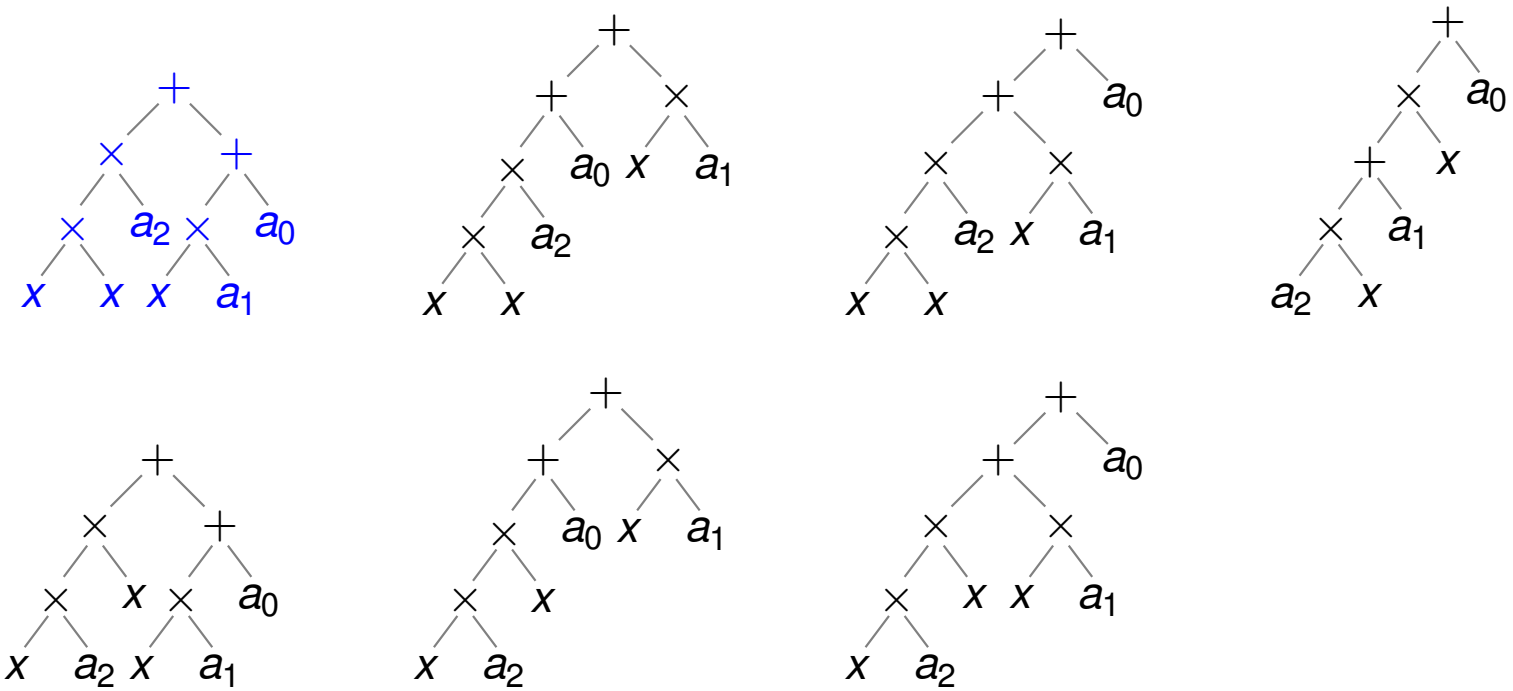
$\rightsquigarrow$ Horner.

Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.
$\rightsquigarrow$ Estrin1.

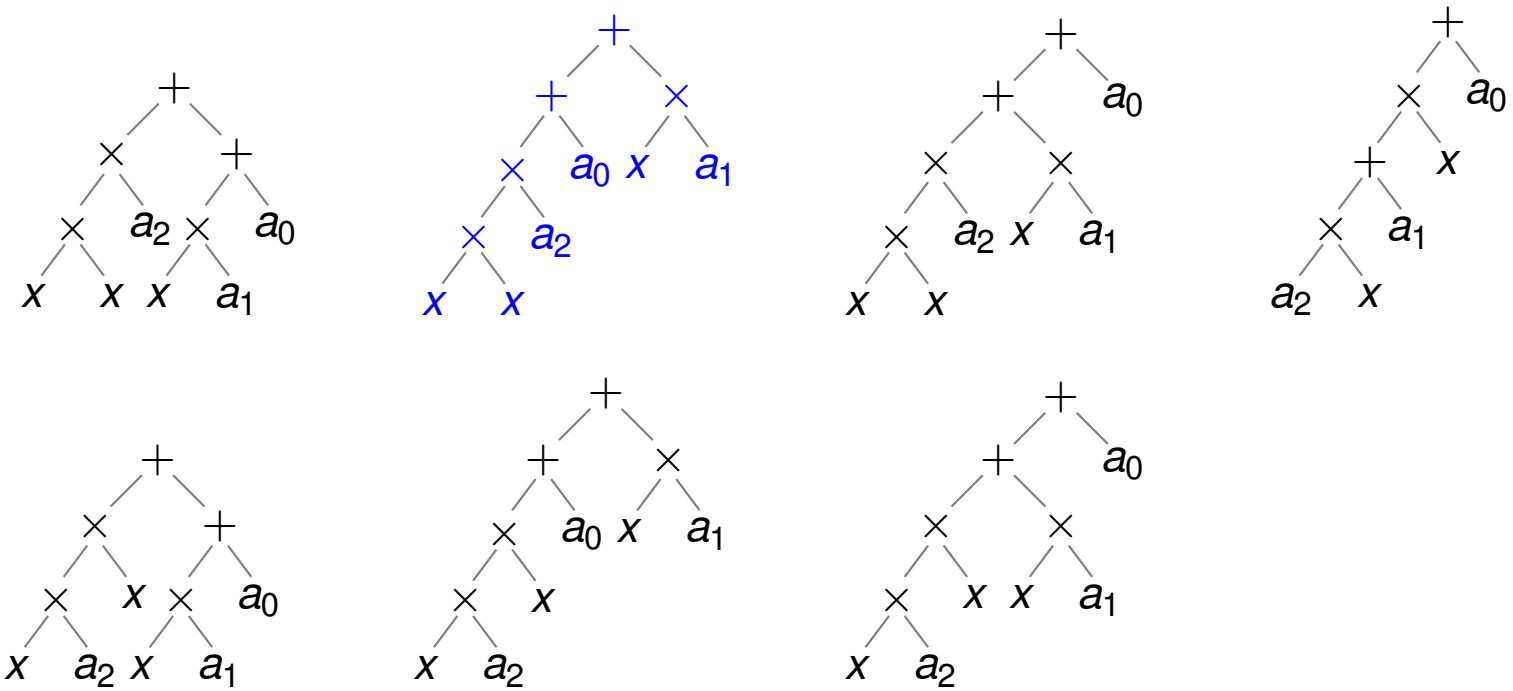
Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

$\rightsquigarrow$ Estrin2.

Exemple : évaluer un polynôme univarié de degré 2



7 schémas d'évaluation dont quelques uns classiques.

$\rightsquigarrow$ Évaluation pair-impair.

Remarques sur l'évaluation de polynômes

Concernant la qualité numérique :

- différentes bornes d'erreurs théoriques [Higham, 2002],
- différence de qualité en pratique, pas forcément en accord avec la borne théorique [Revy, 2006].

Concernant la latence :

- nombre d'additions constant,
- nombre de multiplications variable
  $\rightsquigarrow$ optimalité du schéma de Horner [Pan, 1966],
- certains schémas ont plus de parallélisme que d'autres
  $\rightsquigarrow$ latence variable.

Remarques sur l'évaluation de polynômes

Concernant la qualité numérique :

- différentes bornes d'erreurs théoriques [Higham, 2002],
- différence de qualité en pratique, pas forcément en accord avec la borne théorique [Revy, 2006].

Concernant la latence :

- nombre d'additions constant,
- nombre de multiplications variable
  $\rightsquigarrow$ optimalité du schéma de Horner [Pan, 1966],
- certains schémas ont plus de parallélisme que d'autres
  $\rightsquigarrow$ latence variable.

L'outil CGPE [3]

But = trouver un schéma d'évalution :

- dont on peut borner l'erreur par ε donné en entrée,
- avec la plus petite latence possible sur l'architecture cible.

L'outil est composé de deux étapes :

1. génération de schémas d'évaluation,
2. sélection parmi les schémas générés :
   - $\rightarrow$ élimination des schémas non évaluables en arithmétique non-signée,
   - $\rightarrow$ vérification du scheduling,
   - $\rightarrow$ vérification de la qualité numérique (Gappa).

3. voir [Mouilleron and Revy, 2010].

Problèmes soulevés

Problèmes soulevés :

- nombre de schémas pour un polynôme donné,
- génération des schémas d'évaluation,
- recherche de la latence minimale,
- recherche de l'arbre le plus précis,
- recherche d'un compromis (voir [Thévenoux et al., 2010]).

Plan

Plan

Expressions mathématiques (1/2)

On va considérer une famille $\mathcal{F}$ d'expressions mathématiques (= objets arithmétiques).

Quelques exemples concrets :

- $\mathcal{F} = \{x^n / \ n \in \mathbb{N}\}$ puissances de x,
- $\mathcal{F} = \{\sum_{i \in I} a_i / \ I \subset \mathbb{N}\}$ sommes de variables,
- $\mathcal{F} = \{\sum_{i \in I} a_{k+i} \cdot x^i / I \subset \mathbb{N}, \ k \in \mathbb{N}\} \cup \{x^i / i \in \mathbb{N}\}$ polynômes.

On veut se ramener à des opérations binaires, donc on se donne des règles de parenthésage implicite.

Expressions mathématiques (2/2)

Hypothèse 1

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

Hypothèse 2

On sait décomposer une expression mathématique $f \in \mathcal{F}$, 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 $f = f_1 \diamond f_2$.

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 $i \leq \mathrm{val}(p)$.

Ensemble des expressions parenthésées

On se donne une formule mathématique $f \in \mathcal{F}$.

On part du parenthésage implicite et on déduit l'ensemble $\mathcal{P}_f$ des formules parenthésées pour f par :

- associativité de $+$ et $\times$,
- commutativité de $+$ et $\times$,
- distributivité de $\times$ par rapport à $+$ et factorisation.

Exemple : pour $a_1 + a_2 + a_3$ on a 12 expressions parenthésées.

$$
\begin{array}{lll}
a_1 + (a_2 + a_3) & a_2 + (a_1 + a_3) & a_3 + (a_1 + a_2) \\
a_1 + (a_3 + a_2) & a_2 + (a_3 + a_1) & a_3 + (a_2 + a_1) \\
(a_2 + a_3) + a_1 & (a_1 + a_3) + a_2 & (a_1 + a_2) + a_3 \\
(a_3 + a_2) + a_1 & (a_3 + a_1) + a_2 & (a_2 + a_1) + a_3
\end{array}
$$

Schémas d'évaluation

En arithmétique réelle approchée, $+$ et $\times$ sont commutatives, mais pas associatives.

L'ensemble $\mathcal{S}_f$ des schémas d'évaluation pour $f \in \mathcal{F}$ est le quotient $\mathcal{P}_f/_{\equiv}$ où $\varphi_1 \equiv \varphi_2$ si on peut passer de φ_1 à φ_2 uniquement en utilisant la commutativité de $+$ et $\times$.

Si on raisonne en termes d'arbres :

- échanger les fils d'un nœud ne change pas le schéma.

Plan

Cas de x^n, suite de Wedderburn-Etherington

- La racine de l'arbre est forcément un $\times$.
- Il faut éviter de compter les arbres deux fois à cause de la commutativité. **Solution** : on met le plus grand à droite.

Cela donne la relation de récurrence :

$$\Omega(0) = 0,\ \Omega(1) = 1,$$

$$\Omega(n) = \sum_{i=1}^{(n-1)/2} \Omega(i)\Omega(n-i) \qquad \text{si } n > 1, n \text{ impair}$$

$$\Omega(n) = \frac{\Omega(n/2)(\Omega(n/2)+1)}{2} + \sum_{i=1}^{n/2-1} \Omega(i)\Omega(n-i) \qquad \text{si } n > 1, n \text{ pair}$$

$\rightarrow$ http://www.research.att.com/~njas/sequences/A001190

Idée de l'algorithme dans le cas général

Idées :

- la racine de l'arbre est soit un $+$, soit un $\times$,
- il faut continuer d'appliquer le principe du plus grand à droite et traiter correctement le cas d'égalité.

Pour l'implantation en pratique :

- mémoization pour éviter les appels récursifs identiques,
- code C++ avec une classe `Expression` passée en template,
- variante en C pour les polynômes univariés, optimisée pour éviter les tests dans la boucle interne.

Algorithme de dénombrement

Algorithme 1: `Count`

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

1 **si** f *est atomique* **alors** Retourner 1
2 $n \leftarrow 0$
3 **pour chaque** $(\diamond, \{f_1, f_2\})$ *tel que* $f = f_1 \diamond f_2$ **faire**
4 **si** $f_1 = f_2$ **alors** $n \leftarrow \dfrac{\texttt{Count}(f_1) \cdot (\texttt{Count}(f_1) + 1)}{2}$
5 **sinon** $n \leftarrow \texttt{Count}(f_1) \cdot \texttt{Count}(f_2)$
6 Retourner n

coût = nombre d'appels $\times$ coût maximal de la boucle ligne 3
. . . en opérations élémentaires sur les grands entiers.

Algorithme de dénombrement

Algorithme 1: `Count`

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

1 **si** f *est atomique* **alors** Retourner 1
2 $n \leftarrow 0$
3 **pour chaque** $(\diamond, \{f_1, f_2\})$ *tel que* $f = f_1 \diamond f_2$ **faire**
4 **si** $f_1 = f_2$ **alors** $n \leftarrow \dfrac{\texttt{Count}(f_1) \cdot (\texttt{Count}(f_1) + 1)}{2}$
5 **sinon** $n \leftarrow \texttt{Count}(f_1) \cdot \texttt{Count}(f_2)$
6 Retourner n

coût = nombre d'appels $\times$ coût maximal de la boucle ligne 3
. . . en opérations élémentaires sur les grands entiers.

Algorithme de dénombrement : cas univarié

Algorithme 2: `CountPx`

Entrée : S ensemble représentant un support
Sortie : $n =$ nb. de schémas d'évaluation pour le support S

1 **si** $S = \emptyset$ *ou* $S = \{0\}$ **alors** Retourner 1
2 $n \leftarrow 0$
3 **pour chaque** $s \subset S$ *tel que* $s \neq \emptyset, \max s = \max S$ **faire**
4 $\quad n \leftarrow n + \texttt{CountPx}(S \setminus s) \cdot \texttt{CountPx}(s)$
5 **pour** i *allant de* 1 *à* $\min S$ **faire**
6 $\quad s \leftarrow \{x - i,\ x \in S\}$
7 $\quad n \leftarrow n + \texttt{CountXn}(i) \cdot \texttt{CountPx}(s)$
8 Retourner n

Plan

La suite de Wedderburn-Etherington (A001190)

Pour x^n, on retrouve les premiers termes de la suite de Wedderburn-Etherington.

L'encyclopédie de Sloane nous donne les résultats suivants :

- la série génératrice $W(x) = \sum_{i \geq 1} \Omega(i) x^i$ vérifie

$$W(x) = x + \frac{1}{2}(W(x)^2 + W(x^2)),$$

- l'équivalent asymptotique donné par [Otter, 1948] est

$$\Omega(n) \sim_{n \to \infty} \frac{\eta \, \xi^n}{n^{3/2}}, \text{ où } \eta \approx 0.31877, \ \xi \approx 2.48325.$$

La suite A085748 pour $\alpha \cdot x^n$

Pour le nombre de schémas d'évaluation de $\alpha \cdot x^n$:

- on retrouve la suite A085748,
- la série génératrice $A(x)$ est reliée à celle de la suite de Wedderburn-Etherington par $A(x) = \frac{1}{1-W(x)}$,
- pas d'équivalent asymptotique connu (?).

Pour montrer le deuxième point :

- on note $\mathcal{W}$ l'ensemble des schémas d'évaluation des $x^n, n \in \mathbb{N}$ et $\mathcal{A}$ celui des schémas pour les $\alpha \cdot x^n, n \in \mathbb{N}$,
- on remarque que $\mathcal{A} = \mathrm{SEQ}(\mathcal{W})$ car chaque schéma pour $\alpha \cdot x^n$ peut se voir comme un produit $(((\alpha \cdot x^{i_1}) \cdot x^{i_2}) \dots) \cdot x^{i_k}$ avec $i_1 + \cdots + i_k = n$ et réciproquement.

La suite A001147 pour $x_1 + \cdots + x_n$

On a une formule close :

$$\sigma_n = (2n-3)!! = \prod_{i=1}^{n-2} (2i+1) = \frac{n!\,\mathrm{Catalan}(n-1)}{2^{n-1}} \sim_{n\to+\infty} \frac{1}{n\sqrt{2}} \left(\frac{2n}{e}\right)^n.$$

Pour évaluer un polynôme de degré n, on peut :

1. calculer chaque x^i pour $2 \leq i \leq n$,
2. faire les produits $a_i \cdot x^i$,
3. sommer les $n+1$ termes ainsi obtenus.

$\rightsquigarrow$ On s'attend à un grand nombre de schémas pour les polynômes car on a un choix (au moins) exponentiel en 1 et 3.

La suite A169608 pour $p(x)$

n	nb de schémas
0	1
1	1
2	7
3	163
4	11602
5	2334244
6	1304066578
7	1972869433837
8	801268234366936 6
9	8629893765109331 4877
10	2449381767217281163362301
11	1819460422818643352966991042 07
12	3521464283035276847373650489107 9096

temps de calcul :
$\approx$ 4j pour $n = 25$,

Suite ajoutée dans l'encyclopédie de Sloane :

$\longrightarrow$ http://www.research.att.com/~njas/sequences/A169608

La suite A169608 pour $p(x)$

n	nb de schémas
0	1
1	1
2	7
3	163
4	11602
5	2334244
6	1304066578
7	1972869433837
8	8012682343669366
9	862989376510933 14877
10	244938176721728116336 2301
11	1819460422818643352966991 04207
12	3521464283035276847373650489107 9096

temps de calcul :
$\approx$ 4j pour $n = 25$,

croissance
sur-exponentielle,

Suite ajoutée dans l'encyclopédie de Sloane :

$\longrightarrow$ http://www.research.att.com/~njas/sequences/A169608

La suite A169608 pour $p(x)$

n	nb de schémas
0	1
1	1
2	7
3	163
4	11602
5	2334244
6	1304066578
7	1972869433837
8	801268234366936 6
9	8629893765109331 4877
10	2449381767217281163362301
11	1819460422818643352966991042 07
12	3521464283035276847373650489107 9096

temps de calcul :
$\approx$ 4j pour $n = 25$,

croissance
sur-exponentielle,

analyse exhaustive
prohibitive si $n > 5$.

Suite ajoutée dans l'encyclopédie de Sloane :

$\longrightarrow$ http://www.research.att.com/~njas/sequences/A169608

La suite A173157 pour $q(x, y) = q_{0,0} + y \cdot p(x)$

n	nb de schémas
0	1
1	10
2	481
3	88384
4	57363910
5	122657263474
6	829129658616013
7	17125741272619781635
8	1055157310305502607244946
9	190070917121184028045719056344
10	98543690848554380947490522591191672

Suite ajoutée dans l'encyclopédie de Sloane :

→ http://www.research.att.com/~njas/sequences/A173157

La suite A173157 pour $q(x, y) = q_{0,0} + y \cdot p(x)$

n	nb de schémas
0	1
1	10
2	481
3	88384
4	57363910
5	1226572633474
6	829129658616013
7	171257412726197 81635
8	10551573103055026072 44946
9	1900709171211840280457190 56344
10	985436908485543809474905225911 91672

croissance sur-exponentielle,

Suite ajoutée dans l'encyclopédie de Sloane :

→ http://www.research.att.com/~njas/sequences/A173157

La suite A173157 pour $q(x, y) = q_{0,0} + y \cdot p(x)$

n	nb de schémas
0	1
1	10
2	481
3	88384
4	57363910
5	122657263474
6	829129658616013
7	17125741272619781635
8	1055157310305502607244946
9	190070917121184028045719056344
10	98543690848554380947490522591191672

croissance sur-exponentielle,

analyse exhaustive prohibitive si $n > 4$.

Suite ajoutée dans l'encyclopédie de Sloane :

→ http://www.research.att.com/~njas/sequences/A173157

Bilan

Bilan :

- nouvelle interprétation pour une suite présente dans l'encyclopédie de Sloane,
- deux suites ajoutées dans l'encyclopédie :

$\longrightarrow$ http://www.research.att.com/~njas/sequences/A169608

$\longrightarrow$ http://www.research.att.com/~njas/sequences/A173157,

- la question de l'étude asymtpotique des deux nouvelles suites reste ouverte.

Pour implanter des opérateurs en arithmétique simple précision, on doit évaluer $p(x)$ ou $q(x, y) = q_{0,0} + y \cdot p(x)$ avec $\deg p(x)$ entre 6 et 12.

$\rightsquigarrow$ **Besoin d'heuristiques fortes**.

Plan

Heuristiques dans la version stable de CGPE

Pour limiter le nombre de schémas générés :

- suppression à la volée des schémas trop lents,
- limitation à un nombre fixé de schémas pour chaque expression mathématique,
- restriction sur les décompositions considérées.

Pour le premier point, on utilise une estimation τ :

- si τ trop grand, on n'élimine pas assez de schémas,
- si τ trop petit, on n'obtient aucun schéma et il faut incrémenter τ et relancer le calcul.

Plan

Du dénombrement à la latence minimale

On va chercher à trouver la latence minimale pour évaluer une expression mathématique **en parallélisme infini**.

Motivations :

- obtenir une borne inférieure assez fine pour la latence sur parallélisme borné,
- s'en servir pour supprimer à la volée les schémas trop lents (heuristique).

Idée :

- adapter l'algorithme de dénombrement pour calculer les latences minimales à la place du nombre de schémas d'évaluation = $\times \to \max$, $+ \to \min$.

Algorithme de calcul de latence minimale

Algorithme 3: `MinLat`

Entrée : une expression mathématique f
Données : les coûts $\mathcal{C}_+$ et $\mathcal{C}_\times$ pour $+$ et $\times$
Sortie : latence minimale pour évaluer f

1 **si** f *atomique* **alors** Retourner 0
2 $r \leftarrow \infty$
3 **pour chaque** $(\diamond, \{f_1, f_2\})$ *tel que* $f = f_1 \diamond f_2$ **faire**
4 $\quad r_1 \leftarrow$ `MinLat` (f_1)
5 $\quad r_2 \leftarrow$ `MinLat` (f_2)
6 $\quad r \leftarrow \min\{r,\ \mathcal{C}_\diamond + \max\{r_1, r_2\}\}$
7 Retourner r

Remarques

Pour le calcul de la latence minimale :

- on utilise toujours la mémoization,
- pas besoin de traiter à part le cas où $f_1 = f_2$,
- gestion des délais possible.

En pratique, on a du code plus général :

- on peut conserver les décompositions qui atteignent le minimum pour en déduire des schémas optimaux,
- calcul de la répartition des schémas par latence,
- optimisation d'autres critères :
  - $\rightarrow$ borne sur l'erreur due à l'évaluation,
  - $\rightarrow$ nombre de multiplications (heuristique).

Latence pour l'évaluation d'un polynôme univarié

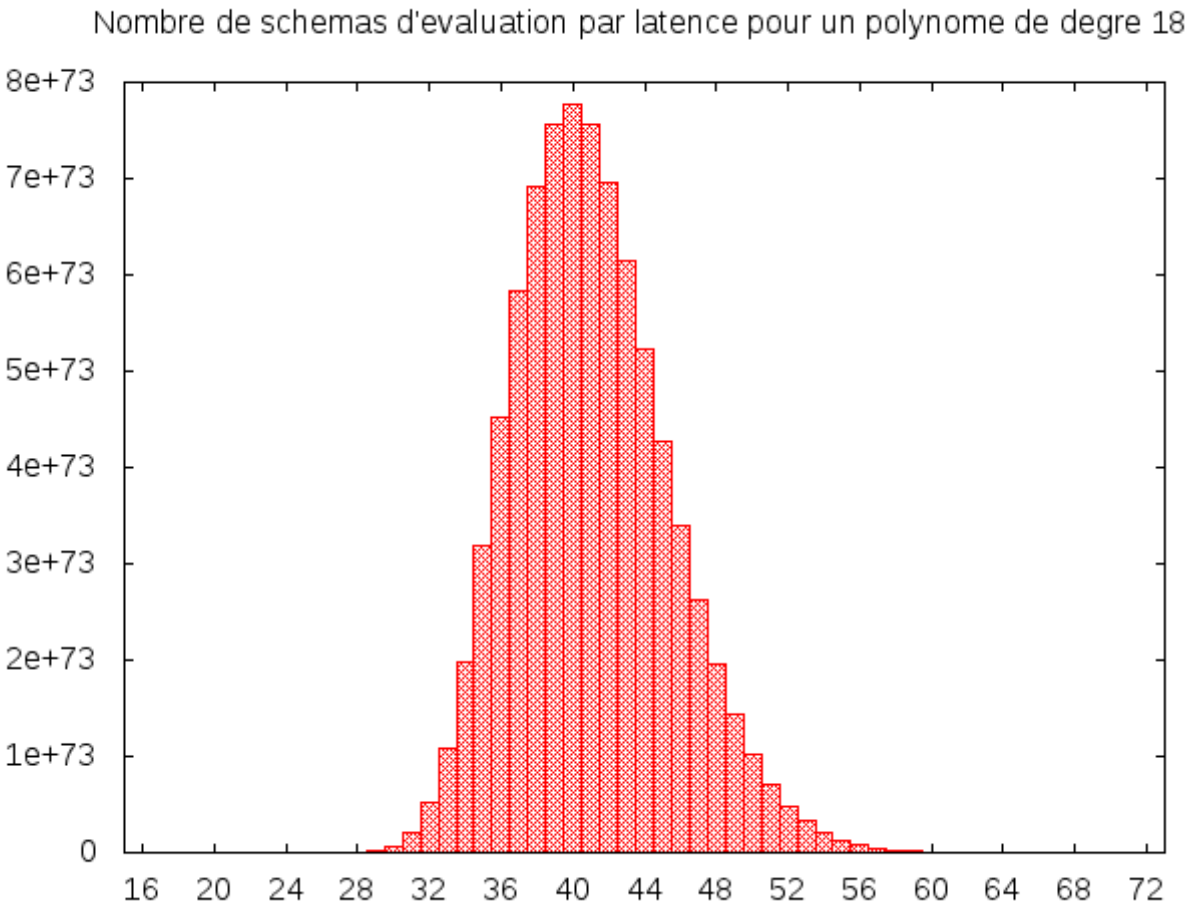
Temps de calcul : $\approx$ 1s pour $d = 12$, $\approx$ 1j pour $d = 25$.

n	0	1	2	3	4	5	6	7	8	9	10	11	12
borne inf.	0	4	7	7	10	10	10	10	13	13	13	13	13
latence min.	0	4	7	8	10	10	11	11	13	13	13	13	14
Estrin	0	4	7	8	10	10	11	12	13	13	13	13	14

n	13	14	15	16	17	18	19	20	21	22	23	24	25
borne inf.	13	13	13	16	16	16	16	16	16	16	16	16	16
latence min.	14	14	15	16	16	16	16	16	16	16	16	17	17
Estrin	14	15	16	17	17	17	17	17	17	17	17	17	17

- borne inférieure $=$ temps pour évaluer $a_0 + a_n \cdot x^n$
  $= \mathcal{C}_+ + \mathcal{C}_\times \cdot \lceil \log_2(n+1) \rceil$,
- cette borne est utilisée dans CGPE, mais elle est pessimiste principalement juste avant une puissance de 2.

Répartition des schémas par latence



temps de calcul :
$\approx$ 1j pour $n = 18$.

répartition semblable
à une gaussienne.

déjà $\approx 8.10^{24}$ schémas
de latence 16.

$\rightsquigarrow$ suffisamment de schémas de latence minimale pour espérer en trouver un valide numériquement.

Utilisation d'une pré-analyse

Idée :

1. lancer le calcul de latence minimale pour identifier les expressions qui apparaissent dans des schémas optimaux,
2. ne générer des schémas que pour ces expressions.

En pratique :

- on a des schémas optimaux du point de vue de la latence,
- on a une vraie latence cible pour la phase de génération,
- les schémas se schedulent mieux (gain de temps) ...
- ... mais on a plus de schémas à analyser numériquement.

$\rightsquigarrow$ travail en cours pour éviter d'être trop pénalisé par l'analyse de la qualité numérique.

Plan

Restriction sur les décompositions

On a deux points "critiques" dans les algorithmes :

- la boucle principale sur les décompositions d'une expression mathématique donnée,
- le nombre d'expressions considérées récursivement.

Idée : se restreindre à certains types de décompositions pour réduire le nombre d'itérations dans la boucle.
Effet de bord : moins d'expressions rencontrées au final.

On peut appliquer cette restriction lors de la génération des schémas et/ou lors d'une pré-analyse.

Exemple : restriction aux découpages haut/bas

Si $f = \sum_{i=i_0}^{i_1} a_{k+i} \cdot x^i$, alors on ne va considérer que les décompositions suivantes :

- les factorisations par x^j pour $0 < j \leq i_0$,
- les sommes de la forme

$$\left(\sum_{i=i_0}^{j} a_{k+i} \cdot x^i \right) + \left(\sum_{i=j+1}^{i_1} a_{k+i} \cdot x^i \right)$$

pour $i_0 < j < i_1$.

Récursivement, en partant de $p(x) = \sum_{i=0}^{n} a_i \cdot x^i$, on obtient les $\binom{n+1}{2}$ expressions correspondant à une suite de coefficients contigus dans $p(x)$ et leurs $O(n)$ formes factorisées associées.

$\rightsquigarrow$ $O(n^3)$ expressions rencontrées au lieu de $O(2^n)$.

Impact de différentes restrictions

Restriction	nb. expressions considérées	coût par expression	coût total
aucune	$O(2^n)$	$O(2^n)$	$O(2^{2n})$
un support de taille $< k$	$O(2^n)$	$O(n^k)$	$O(2^n n^k)$
haut/bas	$O(n^3)$	$O(n)$	$O(n^4)$
haut/bas + taille haut $< k$	$O(kn^2)$	$O(k)$	$O(k^2 n^2)$

En pratique :

- pour la limitation sur la taille d'un support, on prend $k \leq 5$,
- CGPE utilise le découpage haut/bas qui donne les mêmes résultats pour la latence qu'en l'absence de restriction.

$\rightsquigarrow$ temps pour l'étape de génération de l'ordre de la seconde.

Plan

Conclusions

Étude du dénombrement :

- deux suites ajoutées dans l'encyclopédie de Sloane,
- illustration des limites de la recherche exhaustive et du besoin d'heuristiques efficaces,
- la question des équivalents asymptotiques reste ouverte.

Au niveau des heuristiques :

- on peut se restreindre à des schémas optimaux pour un certain critère (latence en parallélisme infini),
- certaines restrictions sur les décompositions considérées permettent d'obtenir des algorithmes polynomiaux.

Travail en cours et perspectives

Travail en cours :

- intégration d'une partie des filtres numériques au niveau de la génération des schémas d'évaluation,
- traitement d'expressions arithmétiques plus générales (fractions rationnelles, polynômes de matrices),

Perspectives :

- extension aux opérateurs à 3 opérandes (FMA),
- interaction avec le projet LEMA (framework général pour générer du code numérique) :
  - $\rightarrow$ ajout d'un support pour CGPE dans LEMA,
  - $\rightarrow$ utilisation de LEMA pour réécrire/améliorer les scripts en amont de CGPE pour la génération de la libraire FLIP.


Ercegovac, M. and Lang, T. (2004).
Digital arithmetic.
Morgan Kaufmann.


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


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.


Mouilleron, C. and Revy, G. (2010).

Automatic Generation of Fast and Certified Code for Polynomial Evaluation.
http://prunel.ccsd.cnrs.fr/ensl-00531721/PDF/MouRev11.pdf.


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.


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.

Thévenoux, L., Langlois, P., and Martel, M. (2010). Accuracy versus time : a case study with summation algorithms.
In *Proc. of the 4th International Workshop on Parallel and Symbolic Computation (PASCO '10)*, pages 121–130, New York, NY, USA. ACM.