Optimisation d'opérateurs arithmétiques matériels à base d'approximations polynomiales

Romain Michard[*] — **Arnaud Tisserand**[**] — **Nicolas Veyrat-Charvillon**[*]

[*] *Arénaire, LIP, CNRS–ENS Lyon–INRIA–UCB Lyon*
46 allée d'Italie. F-69364 Lyon
{romain.michard,nicolas.veyrat-charvillon}@ens-lyon.fr

[**] *Arith, LIRMM, CNRS–Univ. Montpellier 2*
161 rue Ada. F-34392 Montpellier
arnaud.tisserand@lirmm.fr

RÉSUMÉ. *Cet article présente une méthode pour l'optimisation d'opérateurs arithmétiques matériels dédiés à l'évaluation de fonctions en utilisant des polynômes d'approximation. La méthode, basée sur des outils récents, réduit la taille des coefficients des polynômes et des valeurs intermédiaires tout en bornant l'erreur totale (approximation et évaluation). Elle conduit à des opérateurs petits et rapides tout en garantissant une bonne qualité numérique. La méthode est illustrée sur quelques exemples sur des circuits FPGA.*

ABSTRACT. *This article presents a method for the optimisation of hardware arithmetic operators dedicated to function evaluation using polynomial approximations. Using recent tools, the method reduces the size of the polynomial coefficients and the intermediate values while it keeps the total error bounded (approximation and evaluation). It leads to small and fast operators with a good numerical quality. The method is illustrated on several examples implemented on FPGA circuits.*

MOTS-CLÉS : *arithmétique des ordinateurs, circuit intégré numérique, approximation polynomiale, évaluation de polynôme.*

KEYWORDS: *computer arithmetic, digital integrated circuit, polynomial approximation, polynomial evaluation*

1. Introduction

Les systèmes sur puces embarquent de plus en plus de calculs complexes. En plus des additionneurs et multiplieurs rapides, le support matériel d'opérations évoluées est nécessaire pour certaines applications spécifiques. Des applications en traitement du signal et des images effectuent des divisions, inverses, racines carrées ou racines carrées inverses. D'autres applications utilisent des sinus, cosinus, exponentielles ou logarithmes comme en génération numérique de signaux (Cordesses, 2004).

Différents types de méthodes sont proposés dans la littérature pour l'évaluation de fonctions. On trouve dans (Ercegovac *et al.*, 2003) les bases d'arithmétique des ordinateurs et en particulier les opérations comme la division, l'inverse, la racine carrée et la racine carrée inverse. Dans (Muller, 2006), les principales méthodes d'évaluation des fonctions élémentaires ($\sin$, $\cos$, $\exp$, $\log\ldots$) sont présentées.

Parmi toutes les méthodes possibles, les approximations polynomiales sont souvent utilisées. Elles permettent d'évaluer efficacement la plupart des fonctions que l'on trouve dans les applications embarquées. L'évaluation pratique d'un polynôme s'effectue en utilisant des additions et des multiplications. La surface des multiplieurs est souvent très importante dans le circuit. Ceci limite le degré du polynôme d'approximation ou alors il faut recourir à des opérateurs effectuant plusieurs itérations pour parvenir au résultat final.

En section 2, nous présentons les notations, l'état de l'art du domaine et en particulier un outil récent permettant de borner finement les erreurs: GAPPA (Melquiond, 2005–2007). Dans ce travail, nous montrons comment obtenir des opérateurs spécifiques pour l'évaluation de fonctions par des approximations polynomiales optimisées. La méthode, proposée en section 3, intervient à deux niveaux pour réaliser cette optimisation. À partir du meilleur polynôme d'approximation théorique, nous déterminons des coefficients propices à une implantation matérielle efficace. Ensuite, nous montrons comment réduire la taille des valeurs intermédiaires en utilisant GAPPA. Notre méthode permet de construire des approximations polynomiales à la fois efficaces et garanties numériquement à la conception. Dans la plupart des méthodes précédentes, seule l'erreur d'approximation était prise en compte pour l'optimisation. Ici, l'erreur totale, approximation et évaluation, est optimisée. Enfin, nous illustrons en section 4 l'impact de nos optimisations pour quelques exemples de fonctions sur des circuits FPGA (*field programmable gate array*).

2. Notations et état de l'art

2.1. *Notations*

La fonction à évaluer est f pour l'argument x du domaine $[a, b]$. Toutes les valeurs sont représentées en virgule fixe. L'argument x est dans un format sur n bits tandis que le résultat $f(x)$ est sur m bits, souvent on a $n \approx m$. Le polynôme d'approximation p utilisé pour approcher f est de degré d, ses coefficients sont notés: p_0, p_1,…,

p_d. On a donc $p(x) = \sum_{i=0}^{d} p_i x^i$. La représentation binaire des valeurs est notée $()_2$, par exemple $(11.01)_2$ représente la valeur décimale 3.25. La notation binaire signée *borrow-save* est utilisée pour les coefficients (notation en base 2 avec les chiffres $\{-1, 0, 1\}$, cf. (Ercegovac *et al.*, 2003)) avec le chiffre -1 noté $\overline{1}$.

2.2. *Erreurs*

L'argument x est considéré comme exact. L'*erreur d'approximation* ϵ_{app} est la distance entre la fonction mathématique f et le polynôme d'approximation p utilisé pour approcher f. Pour déterminer cette erreur, nous utilisons la *norme infinie* définie ci-dessous et estimée numériquement par MAPLE (fonction `infnorm`).

$$\epsilon_{\text{app}} = ||f - p||_\infty = \max_{a \leq x \leq b} |f(x) - p(x)|.$$

Du fait de la précision finie des calculs, des erreurs d'arrondi se produisent au cours de l'évaluation du polynôme p en machine. L'*erreur d'évaluation* ϵ_{eval} est due à l'accumulation des erreurs d'arrondi. Même si l'erreur d'évaluation commise lors d'une seule opération est toute petite (une fraction du poids du dernier bit), l'accumulation de ces erreurs peut devenir catastrophique lors de longues séries de calculs si rien n'est prévu pour en limiter les effets.

L'erreur d'évaluation est très difficile à borner finement (Ménard *et al.*, 2002). Le problème est lié au fait que cette erreur dépend de la valeur des opérandes. Elle est donc difficile à connaître *a priori*. Souvent, on considère le pire cas d'arrondi pour chaque opération. C'est à dire la perte de précision correspondant à la moitié du bit de poids le plus faible. C'est à dire $1/2$ LSB (*least significant bit*) en arrondi au plus près et 1 LSB en arrondi dirigé. En virgule, ce qui sera notre cas, on arrive à $1/2$ LSB de précision d'arrondi en utilisant des techniques de biais. Pour l'évaluation d'un polynôme de degré d en utilisant le schéma de Horner (d additions et d multiplications), cette borne pire cas correspond à une perte de d bits de précision. Nous allons voir dans la suite que nous pouvons faire des approximations bien moins pessimistes en utilisant l'outil GAPPA développé par G. Melquiond (cf. section 2.5).

Il est possible de mesurer *a posteriori* l'erreur totale commise pour une valeur particulière de l'entrée x en utilisant $f(x) - \texttt{output}(p(x))$ où $\texttt{output}(p(x))$ est la sortie réelle (physique ou simulée par un simulateur au niveau bit) de l'opérateur. Cette technique est souvent utilisée pour qualifier *a posteriori* la précision d'opérateurs par des simulations exhaustives ou pour les entrées menant aux plus grandes erreurs[1]. D'une manière générale, cette technique de qualification *a posteriori* des opérateurs de calcul est coûteuse en temps de simulation et de conception si il faut modifier certains paramètres des opérateurs.

Dans la suite, nous exprimons les erreurs soit directement soit en terme de précision équivalente. Cette dernière est le nombre de bits n_c justes ou corrects équivalents

1. Ces entrées particulières étant très difficiles à déterminer dans le cas général.

à une certaine erreur ϵ avec $n_c = -\log_2 |\epsilon|$. Par exemple, une erreur de 2.3×10^{-3} est équivalente à une précision de 8.7 bits corrects.

2.3. *Polynôme minimax*

Il existe différents types de polynômes d'approximation (Muller, 2006). Le polynôme *minimax* est, en quelque sorte, le meilleur polynôme d'approximation pour notre problème. Le polynôme minimax de degré d pour approcher f sur $[a, b]$ est le polynôme p^* qui satisfait:

$$||f - p^*||_\infty = \min_{p \in \mathcal{P}_d} ||f - p||_\infty,$$

où $\mathcal{P}_d$ est l'ensemble des polynômes à coefficients réels de degré au plus d. Ce polynôme peut être déterminé numériquement grâce à l'algorithme de Remes (Remes, 1934) (implanté dans la fonction MAPLE `minimax`). Le polynôme minimax est le meilleur car parmi tous les polynômes d'approximation possibles de f sur $[a, b]$, il est celui qui a la plus petite des erreurs maximales ϵ_{app} sur tout l'intervalle. On note ϵ^*_{app} l'erreur d'approximation du polynôme minimax avec $\epsilon^*_{\text{app}} = ||f - p^*||_\infty$.

Mais le polynôme minimax est un polynôme « théorique » car ses coefficients sont dans $\mathbb{R}$ et on suppose son évaluation exacte (c'est à dire faite avec une précision infinie). Pour pouvoir évaluer ce polynôme en pratique, il faut faire deux niveaux d'approximation supplémentaires dont les erreurs peuvent se cumuler avec l'erreur d'approximation théorique ϵ^*_{app}:

– les coefficients de p^* devront être écrits en précision finie dans le format supporté par le circuit;

– les calculs nécessaires à son évaluation seront effectués en précision finie.

Pour le moment, il n'existe pas de méthode théorique permettant d'obtenir le meilleur polynôme d'approximation en pratique (coefficients *et* évaluation dans la précision du format cible). Mais nous allons montrer dans la suite que d'importantes optimisations sont déjà possibles avec une méthode assez simple et des outils facilement accessibles. La figure 1 illustre la problèmatique de l'implantation pratique de polynômes d'approximation en matériel.

2.4. *Vers des polynômes d'approximation avec des coefficients représentables*

L'un des problèmes pour l'implantation pratique d'approximations polynomiales est le fait que les coefficients du polynôme minimax ne sont pas représentables dans le format cible en précision finie. Ceci est vrai aussi pour les autres types d'approximations polynomiales(Chebyshev, Legendre…), cf. (Muller, 2006).

Par exemple, prenons la fonction e^x sur l'intervalle $[1/2, 1]$ avec un polynôme de degré 2 et des coefficients sur 10 bits dont 1 en partie entière, soit $x =$

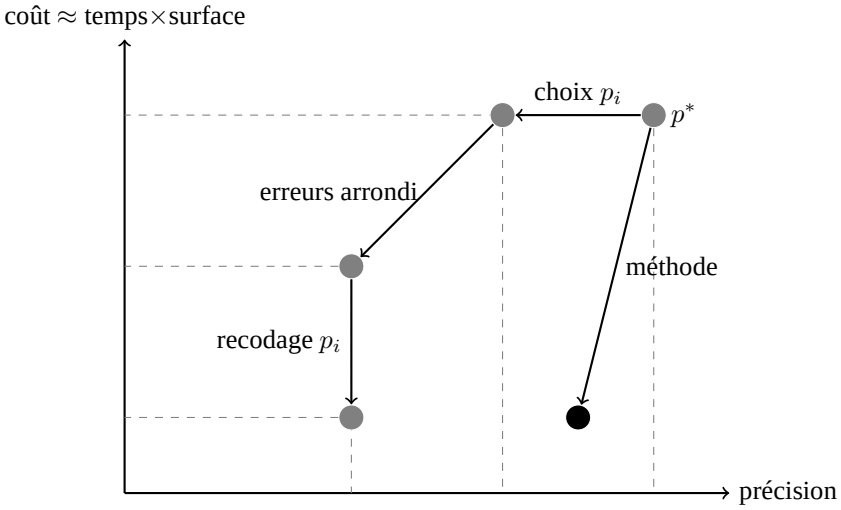

Figure 1. *Résumé du problème d'implantation de polynômes d'approximation en matériel.*

$(x_0.x_1x_2\ldots x_9)_2$. Le polynôme minimax trouvé est[2] $1.116018832+0.535470996x+1.065407154x^2$, il conduit à une précision d'approximation de 9.4 bits environ. C'est la meilleure précision atteignable avec un polynôme de degré 2 sur $[1/2, 1]$. En arrondissant au plus près les 3 coefficients de ce polynôme dans le format cible, on a alors $\frac{571}{512}+\frac{137}{256}x+\frac{545}{512}x^2$ et une précision de l'approximation de 8.1 bits. Après avoir testé tous les modes d'arrondi possibles pour chacun des coefficients dans le format cible, on trouve que le meilleur polynôme est $\frac{571}{512}+\frac{275}{512}x+\frac{545}{512}x^2$ pour une précision de l'approximation de 9.3 bits, soit un gain de 1.2 bit de précision.

Dans les dernières années, différents travaux de recherche ont été menés pour obtenir des « bons » polynômes d'approximation dont les coefficients soient exactement représentables dans le format cible.

On trouve dans (Brisebarre *et al.*, 2006) une méthode basée sur une formulation en un problème de programmation linéaire en nombres entiers. Les coefficients des polynômes sont représentés par des rationnels dont le dénominateur est une puissance de 2 (2^n pour un format fractionnaire sur n bits). Un polytope est construit à partir des contraintes (les tailles) sur tous les coefficients du polynôme. Chaque point du polytope est alors un polynôme dont les coefficients s'écrivent avec les tailles souhaitées. Mais seuls certains de ces polynômes sont des « bons » polynômes d'approximation de f. La liste des polynômes résultats est filtrée en deux phases. L'erreur d'approximation est mesurée en un certain nombre de points d'échantillonage pour chaque point (polynôme) possible du polytope. Le parcours du polytope donne une liste de polynômes

2. en arrondissant les coefficients théoriques sur 10 chiffres décimaux soit envion 32 bits.

pour lesquels la distance entre la fonction et le polynôme testé est plus petite qu'un seuil pour chacun des points d'echantillonage. Cette première liste de polynômes est ensuite traitée pour déterminer numériquement les normes infinies entre chacun des polynômes de la liste et la fonction. Cette méthode, séduisante sur le plan théorique, ne fonctionne pas encore très bien en pratique. Le temps de calcul et le volume de mémoire nécessaires sont trop importants pour le moment.

L'article présenté dans (Brisebarre *et al.*, 2004) utilise (Brisebarre *et al.*, 2006) pour produire des approximations polynomiales avec des coefficients creux (avec beaucoup de 0) pour des petits degrés (3 ou 4). Toutefois, du fait des énormes besoins en temps de calcul et de taille mémoire de (Brisebarre *et al.*, 2006), les approximations trouvées sont limitées à une douzaine de bits de précision. De plus, seule l'erreur d'approximation est prise en compte.

2.5. *Outils pour le calcul de bornes d'erreur globale*

L'étude de méthodes pour borner finement les erreurs de calcul est un domaine de recherche très actif depuis quelques années. Dans le domaine du traitement du signal, des méthodes assimilant les erreurs d'arrondi à du bruit sont utilisées avec succès (Ménard *et al.*, 2002). Différents outils ont été développés pour le calcul scientifique en virgule flottante. Par exemple, FLUCTUAT (Goubault *et al.*, n.d.) est un analyseur statique de code qui permet de détecter certaines pertes de précision sur des programmes flottants. Le logiciel CADNA (Chesneaux *et al.*, n.d.) implante une méthode stochastique pour l'analyse de la précision moyenne des programmes flottants (une version en virgule fixe est actuellement étudiée).

Le logiciel GAPPA (Melquiond, 2005–2007) permet d'évaluer et de prouver des propriétés mathématiques sur des programmes numériques. La caractéristique intéressante de GAPPA dans notre problème est sa capacité à borner finement les erreurs de calcul ou à montrer que des bornes sont en dessous d'un certain seuil.

Voici un exemple simple des possibilités de GAPPA sur la fonction e^x sur $[1/2, 1]$. On suppose tous les calculs effectués en virgule fixe sur 10 bits dont 1 en partie entière. Le polynôme d'approximation utilisé est $p(x) = \frac{571}{512} + \frac{275}{512}x + \frac{545}{512}x^2$, son évaluation se décrit ainsi en GAPPA:

```
1 p0 = 571/512; p1 = 275/512; p2 = 545/512;
2 x = fixed<-9,dn>(Mx);
3 x2 fixed<-9,dn>= x * x;
4 p fixed<-9,dn>= p2 * x2 + p1 * x + p0;
5 Mp = p2 * (Mx*Mx) + p1 * Mx + p0;
6 { Mx in [0.5,1] /\ |Mp-Mf| in [0,0.001385]
7   -> |p-Mf| in ? }
```

La ligne 1 spécifie les coefficients du polynôme (choisis représentables dans le format considéré). Par convention dans la suite, les noms de variables qui commencent par un `M` majuscule représentent les valeurs mathématiques (en précision infinie), et toutes les valeurs en miniscules représentent des variables du programme (des entrées/sorties ou des registres intermédiaires). La ligne 2 indique que `x` est la version circuit de l'argument mathématique `Mx`. La construction `fixed<-9,dn>` indique que l'on travaille en virgule fixe avec le LSB de poids 2^{-9} et l'arrondi vers le bas `dn` (troncature). La ligne 3 indique que la variable `x2` est la version calculée dans le circuit de x^2. La ligne 4 décrit comment le polynôme est évalué en pratique dans le circuit tandis que la ligne 5 décrit son évaluation théorique (en précision infinie car sans l'opérateur d'arrondi `fixed<.,.>`). Enfin, les lignes 6 et 7 indiquent la propriété cherchée (entre accolades). Les hypothèses sont en partie gauche du signe `->`, elles indiquent que la valeur mathématique de x est dans $[1/2, 1]$ et que l'erreur d'approximation entre le polynôme d'appromixation `Mp` (sans erreur d'arrondi) et la fonction mathématique `Mf` est inférieure ou égale à 0.001385 (valeur fournie par MAPLE). La partie droite du signe `->` indique que l'on demande à GAPPA de nous dire dans quel intervalle (`in ?`) est la distance entre la valeur évaluée dans le circuit du polynôme `p` et la fonction mathématique, en incluant erreurs d'approximation et d'évaluation (arrondis).

Le résultat retourné par GAPPA (version supérieure ou égale à `0.6.1`) pour ce calcul est:

```
Results for Mx in [0.5, 1] and |Mp - F| in [0, 0.001385]:
|p - F| in [0, 232010635959353905b-64 {0.0125773, 2^(-6.31303)}]
```

Après avoir répété les hypothèses, GAPPA indique qu'il a trouvé une borne pour $|p - f|$ et qu'il y au moins 6.31 bits corrects.

En modifiant les lignes 3 à 5 par les lignes suivantes, on cherche l'erreur totale commise en utilisant le schéma de Horner, GAPPA indique alors une précision globale de 6.57 bits.

```
3 y1 fixed<-9,dn>= p2 * x + p1;
4 p  fixed<-9,dn>= y1 * x + p0;
5 Mp = (p2 * Mx + p1) * Mx + p0;
```

3. Méthode d'optimisation proposée

Dans un premier temps, nous présentons rapidement la méthode, ensuite nous donnerons des détails sur chacune des différentes étapes et les informations sur les outils nécessaires.

Les données nécessaires en entrée de la méthode sont:

– f la fonction à évaluer;

– $[a, b]$ le domaine de l'argument x;

– le format de l'argument x (nombre de bits n_x);

– μ l'erreur totale maximale cible (donnée en erreur absolue).

Les paramètres déterminés par la méthode sont:

– d le degré du polynôme à utiliser;

– p_0, p_1, p_2,..., p_d les valeurs des coefficients du polynôme représentables dans le circuit;

– n la taille utilisée pour représenter les coefficients;

– n' la taille utilisée pour effectuer les calculs[3].

Notre méthode se résume aux 3 étapes ci-dessous avec des retours possibles à une étape antérieure (rebouclage) dans certains cas:

Étape 1: calcul du *polynôme minimax*.
On cherche p^* de degré d le plus petit possible tel que $\epsilon^*_{\text{app}} < \mu$ avec $\epsilon^*_{\text{app}} = ||f - p^*||_\infty$. Cette première phase donne l'erreur d'approximation ϵ^*_{app} minimale atteignable en supposant tous les calculs faits et coefficients représentés en précision infinie.

Étape 2: détermination des *coefficients* du polynôme à implanter et de leur *taille*.
On cherche ici à la fois les p_i et leur taille minimale n telle que l'erreur d'approximation ϵ_{app} du polynôme p utilisé ($p(x) = \sum_{i=0}^{d} p_i x^i$) soit strictement inférieure à μ.

Étape 3: détermination de la *taille du chemin de données*.
On cherche n' la taille minimale du chemin de données de l'étage de Horner pour effectuer les calculs. Cette dernière phase donne l'erreur d'évaluation ϵ_{eval} qui intègre les erreurs d'approximation et les erreurs d'arrondi. La valeur de n' trouvée garantit que $\epsilon_{\text{eval}} < \mu$.

Dans certains cas, il n'y a pas de solution à une étape. Il faut alors *reboucler* à l'étape précédente pour essayer d'autres solutions.

Différents types de rebouclages sont possibles. Par exemple, en fin de deuxième étape, on peut ne pas trouver des coefficients représentables pour garantir $\epsilon_{\text{app}} < \mu$. Ceci se produit lorsque la « marge » entre ϵ^*_{app} et μ était trop faible à la première étape. Il faut donc retourner à la première étape pour essayer un polynôme de degré plus grand $d \leftarrow d + 1$.

Un autre type classique de rebouclage intervient à la fin de la dernière étape. Si la taille n' trouvée est jugée trop grande devant n, il peut être intéressant de revenir à la deuxième étape pour essayer un n plus grand. Ceci permet d'avoir ϵ_{app} plus petit et donc plus de marge pour les erreurs d'arrondi.

3. Nous verrons que dans certains cas, prendre n et n' légèrement différents peut aider à limiter la taille du circuit.

Dans les descriptions données jusqu'ici, nous utilisons des contraintes du style $\epsilon_{\text{app}} < \mu$ et pas $\epsilon_{\text{app}} \leq \mu$. En pratique, il faut de la marge entre ϵ^*_{app} et μ puis entre ϵ_{app} et μ. Le caractère stricte des contraintes pour les étapes 1 et 2 est donc nécessaire. Pour la dernière étape, peut être pouvons-nous trouver une fonction (probablement triviale et donc peu intéressante) telle que l'on ait $\epsilon_{\text{eval}} = \mu$ à la fin. Cela nous semble très improbable. De plus, l'utilisateur saurait traiter convenablement ce cas.

3.1. *Calcul du polynôme minimax*

Dans cette étape on utilise la fonction `minimax` de MAPLE. On commence avec $d = 1$, et on incrémente d jusqu'à ce que le polynôme minimax p^* trouvé soit tel que $\epsilon^*_{\text{app}} < \mu$.

Voici un exemple pour $f = \log_2(x)$ avec x dans $[1, 2]$:

```
1 > minimax(log[2](x),x=1..2,[1,0],1,'err'); -log[2](err);
2     -.95700010+1.0000000*x
3     4.5371245
4 > minimax(log[2](x),x=1..2,[2,0],1,'err'); -log[2](err);
5     -1.6749034+(2.0246817-.34484766*x)*x
6     7.6597968
7 > minimax(log[2](x),x=1..2,[3,0],1,'err'); -log[2](err);
8     -2.1536207+(3.0478841+(-1.0518750+.15824870*x)*x)*x
9     10.616152
```

Les lignes qui commencent par un signe supérieur (« prompt » MAPLE) sont les commandes entrées par l'utilisateur. Les résultats retournés par MAPLE sont en italique. La ligne 1 signifie que l'on cherche un polynôme de degré 1 et que l'erreur d'approximation trouvée sera placée dans la variable `err`. Les lignes 2 et 3 représentent respectivement le polynôme trouvé et sa précision (en nombre de bits corrects).

Cette étape fournit trois éléments nécessaires pour la suite:

– d le degré du polynôme;

– $p^*(x) = \sum_{i=0}^{d} p_i^* x^i$ le polynôme minimax (théorique);

– ϵ^*_{app} l'erreur *minimale* atteignable en utilisant p^* pour approcher f (en précision infinie).

Les deux autres étapes vont dégrader la qualité de l'approximation (i.e. fournir des erreurs plus grandes que ϵ^*_{app}). Il faut donc laisser un peu de marge entre ϵ^*_{app} et μ. Nous reviendrons sur cette marge.

Nous verrons dans l'exemple 4.2, que certains changements de variables permettent d'obtenir des coefficients d'ordres de grandeur proches entre eux. Ceci limite les recadrages en virgule fixe. Si on évalue $f(x)$ sur $[a, b]$ avec $a \neq 0$, il peut être intéressant de considérer $f(x + a)$ sur $[0, b - a]$.

Toutes les fonctions ne sont pas « approchables facilement » avec des polynômes. Nous renvoyons le lecteur aux ouvrages de références sur l'évaluation de fonctions comme (Muller, 2006, chap. 3) pour les fonctions élémentaires. En pratique, les fonctions usuelles s'approchent bien par des polynômes.

3.2. *Détermination des coefficients du polynôme et de leur taille*

Une fois p^* déterminé, il faut trouver des coefficients représentables en précision finie. On cherche n le nombre de bits du format virgule fixe des p_i tel que n soit le plus petit possible mais avec $\epsilon_{\text{app}} < \mu$.

Nous avons vu en 2.4 que le choix des coefficients est important. En arrondissant simplement les coefficients du polynôme minimax sur n bits, il est peu probable de trouver un bon polynôme d'approximation. L'exemple présenté en 2.4 montre que tester l'ensemble des combinaisons des arrondis des coefficients de p^* permet de trouver un bon polynôme. C'est ce que nous proposons de faire systématiquement dans cette deuxième phase.

Chaque coefficient p_i^* du polynôme peut être arrondi soit vers le haut $p_i = \triangle(p_i^*)$, soit vers le bas $p_i = \triangledown(p_i^*)$. Il y a 2 choix possibles par coefficient et donc 2^{d+1} combinaisons à tester au total comme illustré en figure 2. Pour chaque polynôme p, il faut déterminer $\epsilon_{\text{app}} = ||f - p||_\infty$ (fonction `infnorm` de MAPLE).

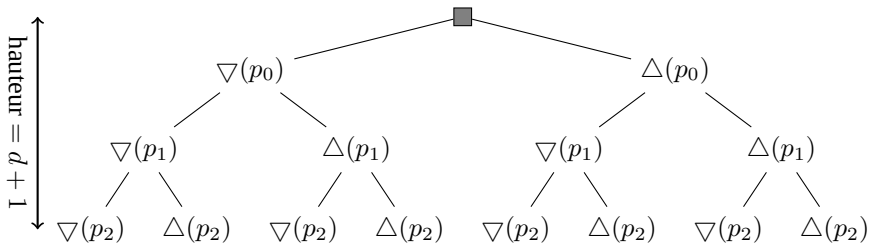

Figure 2. *Différentes combinaisons d'arrondi pour p^* de degré $d = 2$.*

Dans nos applications, d est faible ($d \leq 6$). Il y a donc au mieux quelques centaines de polynômes à tester. En pratique, chaque calcul de ϵ_{app} dure quelques fractions de seconde sur un ordinateur standard.

Nous avons fait un programme MAPLE qui teste les 2^{d+1} combinaisons d'arrondi des p_i^*. Le programme retourne la liste des polynômes candidats pour la troisième étape, c'est à dire ceux pour qui l'erreur d'approximation ϵ_{app} est minimale. On commence par $n = \lceil -\log_2 |\mu| \rceil$ (le nombre de bits correspondant à l'erreur μ), puis on test tous les arrondis des $d+1$ coefficients sur n bits. On recommence en incrémentant n jusqu'à ce que $\epsilon_{\text{app}} < \mu$.

A cette étape, on peut aussi souhaiter modifier plus en « profondeur » certains coefficients. Par exemple, si un coefficient retourné est $0.5002441406 =$

$(0.100000000001)_2$ et que l'on travaille sur un peu plus d'une douzaine de bits fractionnaires, il est peut être intéressant de fixer ce coefficient à 0.5 pour éliminer une opération. La multiplication par une puissance de 2 se réduit à un décalage. Nous verrons un exemple de ce genre de modification dans l'exemple 4.2. Nous ne savons pas encore formaliser simplement ce genre de traitement, mais cela constitue un de nos axes de recherche à travers l'amélioration de la méthode présentée dans (Brisebarre *et al.*, 2006).

Dans ce travail, nous supposons tous les coefficients représentés avec la même taille car on essaie de générer des approximations polynomiales utilisant le schéma de Horner. Dans l'avenir, nous pensons essayer de généraliser notre méthode à des tailles de coefficients différentes. Mais ceci augmente considérablement l'espace de recherche.

3.3. *Détermination de la taille du chemin de données*

La dernière étape permet de spécifier la taille du chemin de données pour l'évaluation suivant le schéma de Horner. Ce schéma est souvent préféré au schéma direct car il nécessite moins d'opérations et donne souvent une erreur d'évaluation ϵ_{eval} plus faible.

$$p(x) = \begin{cases} p_0 + p_1 x + p_2 x^2 + \cdots + p_d x^d & \text{direct } d \text{ add., } d + \lceil \log_2 d \rceil \text{ mul.;} \\ p_0 + x(p_1 + x(p_2 + x(\cdots + x p_d)\cdots)) & \text{Horner } d \text{ add., } d \text{ mul.} \end{cases}$$

Toutefois, dans certains cas particuliers avec des coefficients très creux, le schéma direct peut s'avérer intéressant (cf. exemple en 4.2).

L'étage de Horner permet d'évaluer $u \times v + z$. La taille du chemin de données de cet étage est n'. On commence avec $n' = n$ et on augmente n' tant que l'encadrement de l'erreur d'évaluation ϵ_{eval} par GAPPA avec n' bits pour le chemin de données ne donne pas $\epsilon_{\text{eval}} < \mu$. Pour le moment, le codage en GAPPA se fait à la main. Mais le schéma de Horner ou le schéma direct étant les mêmes aux coefficients près, nous avons les programmes GAPPA types pour lesquels il suffit de préciser les valeurs des coefficients et du degré. La boucle sur n' s'effectue en quelques minutes tout au plus sur un ordinateur standard.

La différence entre n' et n est appelée le nombre de bits de garde. Conserver n plus petit que n' permet, par exemple, de limiter la taille mémoire nécessaire pour stocker les coefficients.

Si la taille du chemin de données est un peu supérieure à n (1 à 3 bits), on pourrait revenir à l'étape 2 pour essayer un n plus grand. Notre expérience montre que le n' final change rarement. Par contre si la valeur de n' est bien plus grande que n, alors il peut être intéressant de reboucler à l'étape 2 avec un n plus grand. Ici encore, nous devons encore travailler pour formaliser ce genre de test.

3.4. *Résumé de la méthode*

La figure 3 résume graphiquement le fonctionnement de la méthode.

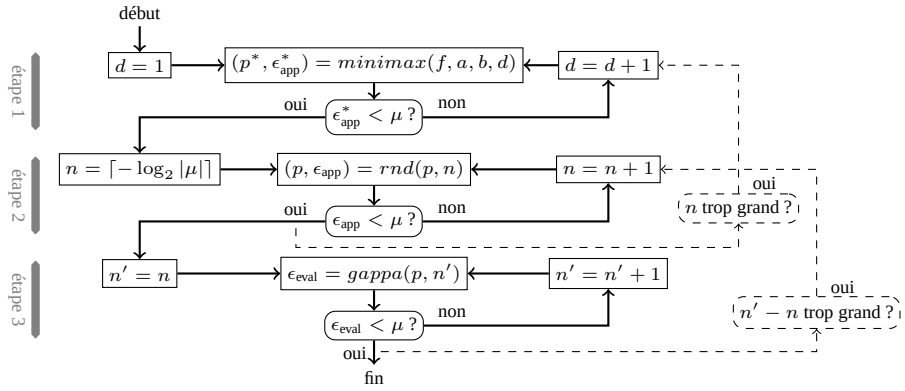

Figure 3. *Résumé de la méthode (les rebouclages en pointillés sont facultatifs).*

4. Exemples d'applications sur FPGA

Les implantations réalisées ci-dessous ont été faites pour des FPGA de la famille Virtex de Xilinx (XCV200-5) avec les outils ISE8.1i de Xilinx. La synthèse et le placement/routage utilisent une optimisation en surface avec effort élevé. Les résultats indiquent toutes les ressources nécessaires pour chaque opérateur (cellules logiques et registres fonctionnels).

4.1. *Fonction* 2^x *sur* $[0, 1]$

On cherche un opérateur pour évaluer la fonction 2^x, avec x dans $[0, 1]$ et une précision globale de 12 bits. La première étape est de trouver un polynôme d'approximation théorique en utilisant MAPLE. On cherche ici à minimiser le degré d du polynôme utilisé. Voici la précision d'approximation ϵ_{app} (en bits corrects) des polynômes minimax de 2^x pour d variant de 1 à 5:

d	1	2	3	4	5
ϵ_{app}	4.53	8.65	13.18	18.04	23.15

Bien évidemment, les polynômes de degré 1 et 2 ne sont pas suffisament précis pour notre but. La solution avec le polynôme de degré 3 conduit à précision estimée dans le pire cas à 10 bits environ ($13.18 - d$, car on borne à d bits la perte de précision pour un schéma de Horner de degré d) ce qui semble trop faible. De plus, les 13.18 bits

corrects correspondent à l'erreur d'approximation avec le polynôme minimax avec des coefficients réels (non représentables dans le format cible).

Sans outil pour aider le concepteur, il semble donc que l'on serait obligé de choisir la solution de degré 4 ou 5. Même la solution avec un polynôme de degré 4 peut conduire à précision trop faible. Certes sa précision théorique est au moins de $18.04 - 4 = 14.04$ bits corrects mais seulement pour des coefficients en précision infinie. Le polynôme minimax de degré 4 est (les coefficients sont affichés avec 10 chiffres décimaux dans le papier, mais MAPLE est capable de les calculer avec plus de précision): $1.0000037045 + 0.6929661227x + 0.2416384458x^2 + 0.0516903583x^3 + 0.0136976645x^4$.

Pour être certain que la solution de degré 4 peut être employée, il faut trouver un format des coefficients pour lequel le polynôme minimax avec ses coefficients arrondis dans ce format ait une précision supérieure ou égale à $12+4 = 16$ bits. Pour un format donné, on teste tous les modes d'arrondi possibles pour les coefficients du polynôme minimax dans le format. On trouve des polynômes acceptables à partir de 14 bits fractionnaires et 1 entier. Pour montrer que le choix de coefficients représentables est important, nous présentons ci-dessous chacune des combinaisons possibles des modes d'arrondi des coefficients et la précision d'approximation du polynôme dont les coefficients s'écrivent exactement dans le format cible. Seuls deux polynômes ont la précision souhaitée (valeurs en gras).

$(\nabla, \nabla, \nabla, \nabla, \nabla)$	12.00	$(\nabla, \nabla, \nabla, \nabla, \triangle)$	13.00
$(\nabla, \nabla, \nabla, \triangle, \nabla)$	13.00	$(\nabla, \nabla, \nabla, \triangle, \triangle)$	14.03
$(\nabla, \nabla, \triangle, \nabla, \nabla)$	13.00	$(\nabla, \nabla, \triangle, \nabla, \triangle)$	14.55
$(\nabla, \nabla, \triangle, \triangle, \nabla)$	14.99	$(\nabla, \nabla, \triangle, \triangle, \triangle)$	13.00
$(\nabla, \triangle, \nabla, \nabla, \nabla)$	13.00	$(\nabla, \triangle, \nabla, \nabla, \triangle)$	**16.13**
$(\nabla, \triangle, \nabla, \triangle, \nabla)$	**17.12**	$(\nabla, \triangle, \nabla, \triangle, \triangle)$	13.00
$(\nabla, \triangle, \triangle, \nabla, \nabla)$	15.71	$(\nabla, \triangle, \triangle, \nabla, \triangle)$	13.00
$(\nabla, \triangle, \triangle, \triangle, \nabla)$	13.00	$(\nabla, \triangle, \triangle, \triangle, \triangle)$	12.00
$(\triangle, \nabla, \nabla, \nabla, \nabla)$	13.00	$(\triangle, \nabla, \nabla, \nabla, \triangle)$	13.00
$(\triangle, \nabla, \nabla, \triangle, \nabla)$	13.00	$(\triangle, \nabla, \nabla, \triangle, \triangle)$	13.00
$(\triangle, \nabla, \triangle, \nabla, \nabla)$	13.00	$(\triangle, \nabla, \triangle, \nabla, \triangle)$	13.00
$(\triangle, \nabla, \triangle, \triangle, \nabla)$	12.99	$(\triangle, \nabla, \triangle, \triangle, \triangle)$	12.00
$(\triangle, \triangle, \nabla, \nabla, \nabla)$	12.99	$(\triangle, \triangle, \nabla, \nabla, \triangle)$	12.98
$(\triangle, \triangle, \nabla, \triangle, \nabla)$	12.91	$(\triangle, \triangle, \nabla, \triangle, \triangle)$	12.00
$(\triangle, \triangle, \triangle, \nabla, \nabla)$	12.79	$(\triangle, \triangle, \triangle, \nabla, \triangle)$	12.00
$(\triangle, \triangle, \triangle, \triangle, \nabla)$	12.00	$(\triangle, \triangle, \triangle, \triangle, \triangle)$	11.41

La solution avec le polynôme de degré 4 est donc utilisable par un bon choix des coefficients du polynôme d'approximation. Mais on va montrer que même celle de degré 3 l'est en pratique. Ce qui constitue une optimisation significative de l'opérateur mais nécessite des outils.

Le polynôme minimax de degré 3 qui approche le mieux théoriquement 2^x sur $[0, 1]$ est:

$$p^*(x) = 0.9998929656 + 0.6964573949x + 0.2243383647x^2 + 0.0792042402x^3.$$

On a alors $\epsilon_{\text{app}} = ||f - p^*||_\infty = 0.0001070344$ soit 13.18 bits de précision. Ceci signifie que, quelque soit la précision des coefficients utilisés pour représenter p^* et celle utilisée pour son évaluation, on ne pourra pas avoir un opérateur avec une précision meilleure que 13.18 bits.

Afin de déterminer la version représentable de p^* que nous allons implanter, il faut trouver la taille des coefficients. Etant donné la fonction et son domaine, le format cherché est constitué d'un bit de partie entière et $n-1$ bits de partie fractionnaire. On cherche n minimal pour une erreur d'approximation ϵ_{app} correspondante la plus proche possible du maximum théorique de 13.18.

$n-1$	12	13	14	15	16
ϵ_{app}	12.38	12.45	13.00	13.00	13.02
nb. candidats	0	0	2	2	7

En effet, pour $n-1 = 14$ bits, tous les modes d'arrondis possibles des coefficients donnent:

$(\nabla, \nabla, \nabla, \nabla)$	11.41	$(\nabla, \nabla, \nabla, \triangle)$	12.00
$(\nabla, \nabla, \triangle, \nabla)$	12.00	$(\nabla, \nabla, \triangle, \triangle)$	12.84
$(\nabla, \triangle, \nabla, \nabla)$	12.00	$(\nabla, \triangle, \nabla, \triangle)$	**13.00**
$(\nabla, \triangle, \triangle, \nabla)$	**13.00**	$(\nabla, \triangle, \triangle, \triangle)$	12.36
$(\triangle, \nabla, \nabla, \nabla)$	12.00	$(\triangle, \nabla, \nabla, \triangle)$	12.25
$(\triangle, \nabla, \triangle, \nabla)$	12.23	$(\triangle, \nabla, \triangle, \triangle)$	12.23
$(\triangle, \triangle, \nabla, \nabla)$	12.13	$(\triangle, \triangle, \nabla, \triangle)$	12.12
$(\triangle, \triangle, \triangle, \nabla)$	12.05	$(\triangle, \triangle, \triangle, \triangle)$	11.64

Les deux polynômes candidats sont donc:

$$\frac{8191}{8192} + \frac{2853}{4096}x + \frac{1837}{8192}x^2 + \frac{649}{8192}x^3 \qquad \text{et} \qquad \frac{8191}{8192} + \frac{2853}{4096}x + \frac{919}{4096}x^2 + \frac{649}{8192}x^3.$$

Tous deux conduisent à une erreur d'approximation de 0.0001220703 (13.00 bits de précision). Il reste maintenant à vérifier que l'évaluation d'au moins un de ces polynômes donne une précision finale d'au moins 12 bits. Voici ci-dessous la précision totale (approximation + évaluation) retournée par GAPPA en utilisant le schéma de Horner et le schéma direct pour évaluer $\frac{8191}{8192} + \frac{2853}{4096}x + \frac{1837}{8192}x^2 + \frac{649}{8192}x^3$ pour différentes tailles du chemin de données n':

n'	14	15	16	17	18	19	20
ϵ_{eval} Horner	11.32	11.93	12.36	12.65	12.81	12.90	12.95
ϵ_{eval} direct	11.24	11.86	12.32	12.62	12.79	12.89	12.94

Les valeurs obtenus pour l'autre polynôme ($p_2 = \frac{919}{4096}$) sont équivalentes. Le schéma de Horner présente un comportement légèrement meilleur que l'évaluation directe (qui en plus est plus coûteuse en nombre d'opérations). Il faut un chemin de

données sur 16 bits au moins pour obtenir un opérateur avec 12 bits de précision au final et ce à partir d'une approximation avec 13.18 bits de précision.

Passer les coefficients sur 16 bits ne modifie pas beaucoup la précision totale car on voit dans la table qui donne ϵ_{app} en fonction de n que l'on passe de 13.00 à 13.02 bits de précision seulement en passant $n-1$ de 14 à 16 pour l'approximation. Avec des coefficients et un chemin de données sur 16 bits, GAPPA indique une précision de 12.38 bits en évaluant le polynôme $p(x) = \frac{32765}{32768} + \frac{22821}{32768}x + \frac{7351}{32768}x^2 + \frac{649}{8192}x^3$.

Deux solutions ont été implantées pour cet opérateur: celle optimisée (degré 3, chemin de données de 16 bits) et celle de base (degré 4, chemin de données de 18 bits). La seconde version correspond à celle que l'on aurait implantée sans l'aide de notre méthode. La table 1 donne les différentes caractéristiques des deux implantations pour un étage de Horner (logique et registres). L'optimisation permet d'obtenir un circuit 17% plus petit mais surtout d'utiliser une approximation de degré 3 plutôt que 4 et donc de gagner 38% en temps de calcul.

version	surface [slices]	période [ns]	nb cycles	durée du calcul [ns]
degré 3, $n' = 16$	193	21.9	3	65.7
degré 4, $n' = 18$	233	26.9	4	107.6

Tableau 1. *Résultats de synthèse pour* 2^x *sur* $[0, 1]$.

4.2. *Racine carrée sur* $[1, 2]$

Pour ce deuxième exemple, nous cherchons à concevoir un opérateur très rapide pour évaluer $\sqrt{x}$ avec x dans $[1, 2]$ et une précision d'au moins 8 bits au final (approximation et évaluation). Le polynôme minimax de degré 1 ne conduit qu'à 6.81 bits de précision, il faut au moins un polynôme de degré 2.

Le polynôme minimax de degré 2 pour $\sqrt{x}$ avec x dans $[1, 2]$ est $0.4456804579 + 0.6262821240x - 0.7119874509x^2$. Il fournit une erreur d'approximation théorique de 0.0007638369 soit 10.35 bits corrects. La précision théorique supérieure à 10 bits permet de supposer que l'on devrait atteindre notre but si on trouve des coefficients représentables sans trop diminuer l'erreur d'approximation.

Toutefois, implanter directement ce polynôme, n'est pas une bonne idée du point de vue du format. Avec x dans $[1, 2]$, il faut travailler un nombre de bits variable pour la partie entière. En effet, l'opération x^2 nécessite deux bits entiers alors que les autres seulement un. Pour éviter ceci, on utilise un changement de variable pour évaluer $\sqrt{1+x}$ avec x dans $[0, 1]$. Le polynôme minimax correspondant est: $1.0007638368 + 0.4838846338x - 0.7119874509x^2$. On obtient la même erreur d'approximation de 10.35 bits corrects. Ceci est tout à fait normal car le changement de variable $x = 1+x$ utilisé ne modifie pas la qualité du polynôme minimax.

A partir de ce polynôme, on pourrait procéder comme pour l'exemple 2^x, mais les coefficients p_0 et p_1 semblent très proches de puissances de 2 et on va essayer de l'utiliser. La première chose à faire est de remplacer p_0 par 1. Le polynôme $1 + 0.4838846338x - 0.7119874509x^2$ offre une précision d'approximation de 9.35 bits, ce qui nous semble satisfaisant.

Le coefficient p_1 semble proche de 0.5. Le polynôme $1 + 0.5x - 0.7119874509x^2$ offre une précision d'approximation de 6.09 bits seulement. p_1 ne peut donc pas être remplacé par 0.5. Toutefois nous allons essayer d'écrire p_1 avec peu de bits à 1 ou -1. Le coefficient p_1 est très proche de $(0.10000\overline{1})_2$. Le polynôme $1 + (0.10000\overline{1})_2x - 0.7119874509x^2$ offre une précision d'approximation de 9.45 bits et en plus le produit p_1x est remplacé par la soustraction $\frac{1}{2}x - \frac{1}{2^6}x$.

Nous procédons à une recherche d'une version avec peu de bits non-nuls de p_2 et nous trouvons $(0.0001001)_2$. Donc le produit p_2x^2 est remplacé par l'addition $\frac{1}{2^4}x^2 + \frac{1}{2^7}x^2$. Le polynôme $1 + (0.10000\overline{1})_2x + (0.0001001)_2x^2$ fournit une précision d'approximation de 9.49 bits. Il ne reste donc plus qu'une seule multiplication pour le calcul de x^2.

On va donc déterminer la précision finale intégrant l'erreur d'évaluation en utilisant GAPPA. En cherchant la taille n' du chemin de données on trouve 10 bits. Le programme à faire prouver par GAPPA est le suivant.

```
1 p0 = 1; p1 = 31/64; p2 = -9/128;
2 x = fixed<-10,dn>(Mx);
3 x2  fixed<-10,dn>= x * x;
4 p   fixed<-10,dn>= p2 * x2 + p1 * x + p0;
5 Mp = p2 * (Mx*Mx) + p1 * Mx + p0;
6 { Mx in [0,1] /\ |Mp-Mf| in [0,0.0013829642]
7   -> |p-Mf| in ? }
```

GAPPA retourne une erreur totale de 8.03 bits. Mais ce programme GAPPA correspond à l'utilisation de mutiplieurs pour effectuer les produits p_1x et p_2x^2. En pratique nous remplaçons ces multiplieurs par des additions/soustractions. Il faut donc donner à GAPPA une description exacte de ce qui est fait par notre architecture. La détermination de la taille minimale du chemin de données est faite en partant de $n = 8$ et en incrémentant n jusqu'à ce que la précision finale soit supérieure ou égale à 8 bits. La recherche donne $n = 13$.

```
p0  = 1;
p11 = 1/2;   p12 = -1/64;
p21 = -1/16; p22 = -1/128;
x = fixed<-8,dn>(Mx);
x2  fixed<-16,dn>= x * x;
p   fixed<-13,dn>= p21 * x2 + p22 * x2 + p11 * x
                   + p12 * x + p0;
Mx2 = Mx * Mx;
Mp  = p21 * Mx2 + p22 * Mx2 + p11 * Mx + p12 * Mx
      + p0;
{ Mx in [0,1] /\ |Mp-Mf| in [0,0.0013829642]
  -> |p-Mf| in ? }
```

Dans ce cas, GAPPA retourne une précision de 8.07 bits avec une seule vraie multiplication pour x^2. L'architecture de l'opérateur est présentée en figure 4. Les cercles gris indiquent un décalage vers la droite du nombre de bits indiqué à l'intérieur du cercle (routage uniquement).

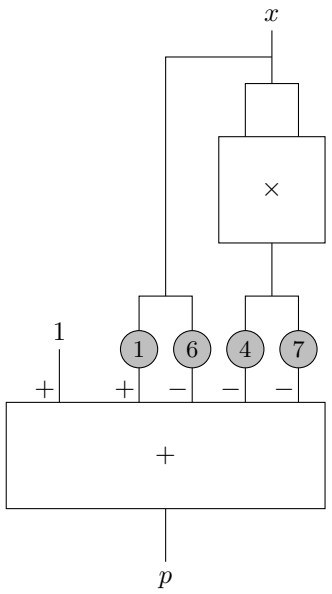

Figure 4. *Architecture de l'opérateur optimisé pour* $\sqrt{1+x}$ *sur* $[0, 1]$.

Nous avons implanté la solution optimisée présentée en figure 4 et celle que l'on aurait implantée sans l'aide de la méthode (degré 2, étage de Horner avec chemin de données sur 11 bits). Les résultats de synthèse correspondants sont présentés dans la table 2. Ici aussi, les gains sont intéressants puisque l'on obtient une amélioration de 40% en surface et de 51% en temps de calcul.

version	surface [slices]	période [ns]	nb cycles	durée du calcul [ns]
degré 2 Horner	103	19.9	2	39.8
degré 2 optimisée	61	19.4	1	19.4

Tableau 2. *Résultats de synthèse pour* $\sqrt{1+x}$ *sur* $[0,1]$.

5. Conclusion et perspectives

Nous avons proposé une méthode pour concevoir et optimiser des opérateurs arithmétiques matériels dédiés à l'évaluation de fonctions par approximation polynomiale. Notre méthode permet, à l'aide d'outils récents, de déterminer une solution avec:

– un degré d petit;

– une taille de coefficients représentables n petite;

– et une taille du chemin de données n' petite.

La méthode ne fournit pas des valeurs de d, n et n' optimales. L'optimum pour ce problème n'est pas connu actuellement même sur le plan théorique. Sur les exemples testés, la méthode permet d'obtenir des circuits plus petits et plus rapides que ceux que l'on pouvait concevoir avant. Toutefois, il reste beaucoup à faire pour les améliorer encore.

De plus, notre méthode permet, grâce à l'utilisation du logiciel GAPPA (Melquiond, 2005–2007), d'obtenir des opérateurs valides numériquement dès la conception. En effet, la méthode permet de déterminer à la fois les erreurs d'approximation et les erreurs d'évaluation. Il n'est plus nécessaire de qualifier *a posteriori* le circuit avec de longues simulations ou tests. La contrainte de précision est vérifiée à la conception.

Dans l'avenir, nous pensons travailler dans plusieurs directions. Pour minimiser l'erreur d'approximation, nous poursuivons nos travaux présentés dans (Brisebarre *et al.*, 2006) et (Brisebarre *et al.*, 2004). Pour minimiser l'erreur d'évaluation tout en évaluant rapidement le polynôme, il reste énormement de travail à faire. En utilisant, par exemple, d'autres schémas d'évaluation de polynômes comme celui proposé par Estrin pour certaines valeurs de d. Enfin, nous travaillons sur l'intégration de cette méthode dans des outils pour générer automatiquement des circuits.

Remerciements

Les auteurs tiennent à remercier chaleureusement Guillaume Melquiond pour son aide sur l'utilisation de son outil GAPPA.

6. Bibliographie

Brisebarre N., Muller J.-M., Tisserand A., « Sparse-Coefficient Polynomial Approximations for Hardware Implementations », *Proc. 38th Asilomar Conference on Signals, Systems and Computers*, Pacific Grove, California, U.S.A., p. 532-535, November, 2004.

Brisebarre N., Muller J.-M., Tisserand A., « Computing Machine-Efficient Polynomial Approximations », *ACM Transactions on Mathematical Software*, vol. 32, n° 2, p. 236-256, June, 2006.

Chesneaux J.-M., Didier L.-S., Jézéquel F., Lamotte J.-L., Rico F., « CADNA: Control of Accuracy and Debugging for Numerical Applications », , http://www-anp.lip6.fr/cadna/, n.d. LIP6–Univ. Pierre et Marie Curie.

Cordesses L., « Direct Digital Synthesis: A Tool for Periodic Wave Generation (Part 1) », *IEEE Signal Processing Magazine*, vol. 21, n° 4, p. 50-54, July, 2004.

Ercegovac M. D., Lang T., *Digital Arithmetic*, Morgan Kaufmann, 2003.

Goubault E., Martel M., Putot S., « FLUCTUAT: Static Analysis for Numerical Precision », , http://www-list.cea.fr/labos/fr/LSL/fluctuat/, n.d. CEA-LIST.

Melquiond G., « GAPPA: génération automatique de preuves de propriétés arithmétiques », , http://lipforge.ens-lyon.fr/www/gappa/, 2005–2007. Arénaire, LIP, CNRS-ENSL-INRIA-UCBL.

Ménard D., Sentieys O., « Automatic Evaluation of the Accuracy of Fixed-Point Algorithms », *in* C. D. Kloos, J. da Franca (eds), *Proc. Design, Automation and Test in Europe (DATE)*, p. 529-537, March, 2002.

Muller J.-M., *Elementary Functions: Algorithms and Implementation*, 2nd edn, Birkhäuser, 2006.

Remes E., « Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation », *C.R. Acad. Sci. Paris*, vol. 198, p. 2063-2065, 1934.