Segmentation non-uniforme pour l'approximation polynomiale de fonctions pour processeurs embarqués

Justine Bonnot, Daniel Menard, Erwan Nogues

Compas'2016 : Parallélisme/ Architecture / Système
Lorient, France, du 5 au 8 juillet 2016

Segmentation non-uniforme pour l'approximation polynomiale de fonctions pour processeurs embarqués

Justine Bonnot, Daniel Menard, Erwan Nogues

UEB, INSA Rennes, IETR, UMR 6164
35708 Rennes - France
justine.bonnot@insa-rennes.fr

Résumé
Les applications embarquées intègrent de plus en plus de calculs sophistiqués. Ces calculs sont généralement des composées de fonctions élémentaires, aisément approchées par des polynômes. L'approximation polynomiale permet d'obtenir un compromis entre précision de l'approximation et coût d'évaluation de la fonction. Une implantation logicielle au sein de processeurs en virgule fixe est considérée dans ce papier. Afin d'avoir une erreur d'approximation réduite, l'intervalle I sur lequel la fonction est calculée doit être segmenté. Cet article présente une méthode de calcul de la valeur d'une fonction sur un segment I à l'aide d'une segmentation non-uniforme suivie d'approximation polynomiale. La segmentation non-uniforme permet de réduire le nombre de segments considérés et est modélisée par un arbre. Les caractéristiques de la segmentation fixent l'équilibre entre les contraintes sur la mémoire et sur le temps. En comparaison à la méthode de tabulation directe cette approche réduit de façon importante la taille mémoire.

1. Introduction

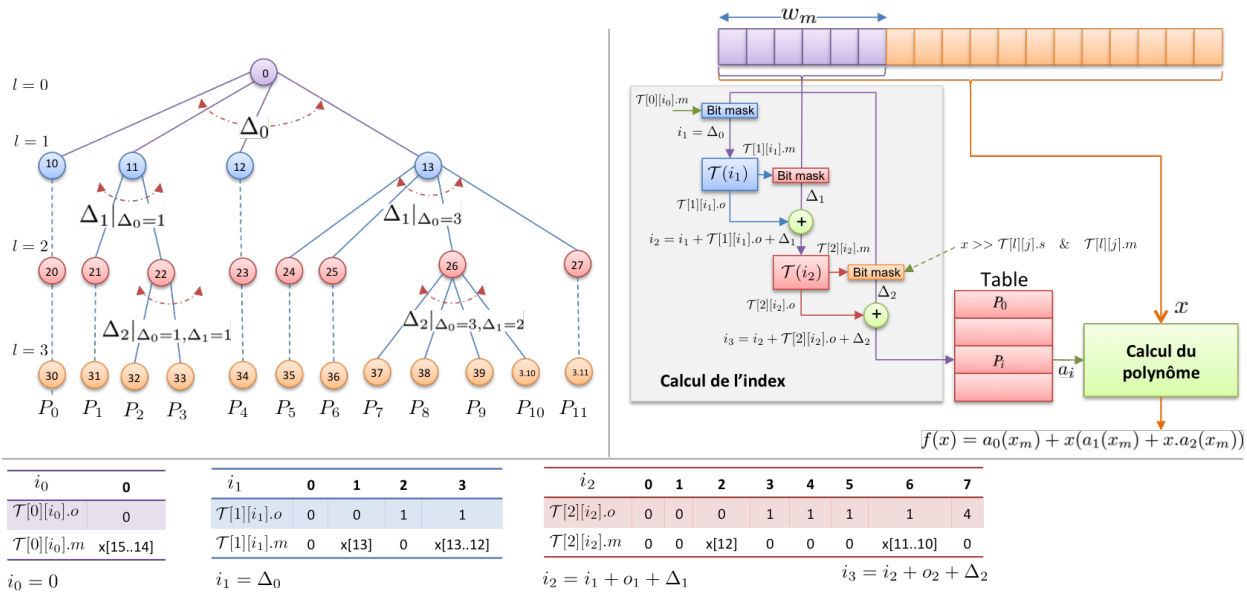
Les progrès en micro-électronique permettent l'intégration sur systèmes embarqués de traitements complexes. Ces traitements intègrent des calculs de fonctions élémentaires. Ces calculs sont utilisés dans des domaines tels que la communication, le traitement du signal ou bien la robotique et peuvent être effectués de façon exacte ou approchée. Le challenge actuel est d'implémenter ces fonctions de façon précise sans toutefois sacrifier les performances de l'application, c'est-à-dire, l'empreinte mémoire, le temps d'exécution ou la consommation énergétique. Pour cela, des algorithmes spécifiques peuvent être adaptés à l'approximation de certaines fonctions [5]. Pour l'approximation de fonctions trigonométriques, hyperboliques ou logarithmiques, l'algorithme de CORDIC (COordinate Rotation DIgital Computer) [6] peut être utilisé mais nécessite le stockage d'une table. Le calcul peut également s'effectuer à l'aide de tabulation directe ou de tables bi/multi-partites. Néanmoins, ces méthodes nécessitent le stockage de tables de tailles importantes et ne peuvent donc pas être portées sur systèmes embarqués. Finalement, la majorité des méthodes proposées est développée pour une implantation matérielle. Ainsi la tabulation directe consiste en la segmentation uniforme de l'intervalle I pour approcher sur chaque segment la fonction par des polynômes de degré 0. La méthode multipartite [2] segmente I et approche f par des fonctions linéaires. Ces méthodes permettent des calculs rapides et des tables de tailles réduites, mais restent limitées à une faible précision et à une implantation matérielle.

Dans cet article, l'implantation logicielle sur système embarqué, i.e. processeurs à faible coût et faible consommation d'énergie, est considérée. Afin de répondre à ces contraintes de coût et de consommation, aucune unité virgule flottante n'est disponible. Les calculs sont effectués en arithmétique virgule fixe. La solution actuelle pour le calcul logiciel de telles fonctions est l'utilisation de libraires telle que *libm* offrant un résultat très précis mais un calcul très lent.
L'*approximation polynomiale* offre un bon compromis pour le calcul de telles fonctions. En implantant l'algorithme de Remez, les outils tels que Sollya [1] fournissent les coefficients du polynôme approchant la fonction f sur l'intervalle I pour un degré fixé. Pour des processeurs virgule fixe embarqués, l'approximation polynomiale fournit un résultat très précis en peu de cycles si l'intervalle I est segmenté assez finement. La segmentation est non-uniforme pour limiter le nombre de polynômes à stocker en mémoire. Le challenge est donc de trouver la segmentation optimale de l'intervalle I. Différentes segmentations non-uniformes [4] ont été proposées mais n'ont été implantées qu'en matériel et ne permettent pas une grande flexibilité en matière de finesse de la segmentation.
La méthode présentée dans ce papier propose une nouvelle technique de segmentation non-uniforme pour l'évaluation logicielle de fonctions, basée sur de l'approximation polynomiale. La segmentation est stockée dans une structure d'arbre. La méthode proposée fournit des courbes de Pareto représentant l'évolution de la taille mémoire requise en fonction du temps d'évaluation de la fonction pour une erreur maximale fixée ϵ_{max}. Le développeur peut ensuite choisir le degré et la finesse de segmentation les plus adaptés à ses contraintes. Finalement, en comparaison à la tabulation directe ou à l'algorithme de CORDIC, l'approche proposée réduit de façon importante respectivement la taille mémoire requise et le temps d'évaluation de la fonction.
La suite de l'article se présente ainsi : la méthode permettant d'obtenir le polynôme correspondant à une valeur d'entrée x est présentée Section 2. La segmentation non-uniforme est détaillée Section 3 et les expériences et comparaisons sont fournies en Section 4.

2. Calcul de l'approximation de $f(x)$ au sein du processeur embarqué

2.1. Arbre de segmentation

Dans les techniques utilisant l'approximation polynomiale, le but est d'approcher une fonction f par un polynôme P sur l'intervalle I. Ces techniques génèrent 2 types d'erreur : l'erreur due à l'approximation $\epsilon_{app} = \|f - P\|_\infty$ et l'erreur ϵ_{fxp} liée aux calculs en précision finie du polynôme P. L'arithmétique virgule-fixe nécessite de conserver un degré de polynôme faible afin d'obtenir une erreur raisonnable ϵ_{fxp}. Un faible degré implique la réduction de la taille des segments. La segmentation non-uniforme permet donc de limiter le nombre de polynômes à stocker en mémoire.
La segmentation non-uniforme est stockée dans une structure d'arbre T comme indiqué figure 1.a. La racine de T stocke les bornes de I et les feuilles contiennent les bornes des segments sur lesquels le critère de l'erreur est vérifié, ainsi un polynôme P_j est associé à chaque feuille n_{lj}. Chaque noeud de l'arbre contient les bornes du segment obtenu par la segmentation du niveau précédent de T. Une fois l'arbre T déterminé, le numéro de la feuille (appelé index) correspondant à une valeur d'entrée x doit être déterminé. La profondeur L de l'arbre est un compromis entre le nombre de polynômes à stocker et le temps de calcul de l'index.
Le schéma de calcul pour approcher f est composé de deux parties, le calcul de l'index et le calcul du polynôme présentés figure 1.c. L'étape du calcul de l'index détermine à partir des w_m bits de poids fort l'index i utilisé pour adresser la table des polynômes $\mathcal{P}$. L'étape du calcul du polynôme évalue le polynôme P_i en x.



i_0	0
$\mathcal{T}[0][i_0].o$	0
$\mathcal{T}[0][i_0].m$	x[15..14]

$i_0 = 0$

i_1	0	1	2	3
$\mathcal{T}[1][i_1].o$	0	0	1	1
$\mathcal{T}[1][i_1].m$	0	x[13]	0	x[13..12]

$i_1 = \Delta_0$

i_2	0	1	2	3	4	5	6	7
$\mathcal{T}[2][i_2].o$	0	0	0	1	1	1	1	4
$\mathcal{T}[2][i_2].m$	0	0	x[12]	0	0	0	x[11..10]	0

$i_2 = i_1 + o_1 + \Delta_1$ $\quad i_3 = i_2 + o_2 + \Delta_2$

FIGURE 1 – a) Exemple d'arbre de segmentation obtenu avec une segmentation non-uniforme (gauche). b) Tables d'indexation associées. c) Schéma de calcul intégrant le calcul de l'index et du polynôme (droite).

2.2. Calcul de l'index

Le but de cette étape est de déterminer l'index associé à la valeur d'entrée x en un temps minimum. L'approche est basée sur l'analyse de bits spécifiques de x codé en virgule fixe. Les bornes des segments sont des sommes de puissances de 2, ces bits peuvent donc être sélectionnés avec un masque et interprétés après avoir été décalés pour être alignés sur le LSB.
L'arbre stockant la segmentation n'est pas équilibré : pour chaque niveau, le nombre de bits à analyser dépend du noeud. A chaque noeud correspond une opération de masquage et de décalage. A chaque noeud intermédiaire n_{lj} (où l est le niveau dans l'arbre et j le noeud) un masque et un offset entre l'index du premier fils du noeud considéré et l'index associé à ce noeud sont définis. Le masque est utilisé pour sélectionner les bits adéquats de x pour passer du noeud n_{lj} au noeud $n_{l+1j'}$ situé au niveau $l + 1$.
La méthode d'indexation utilise une table $\mathcal{T}$, représentée figure 1.b pour l'exemple considéré et qui stocke les tables pour chaque niveau, contenant une structure composée du masque, du décalage et de l'offset pour chaque noeud n_{lj} pour passer du niveau l au $l + 1$.

3. Technique de segmentation non-uniforme pour déterminer les coefficients des polynômes

La méthode proposée requiert les paramètres d'entrée suivants. 1) f, la fonction à approcher, 2) I, l'intervalle sur lequel f est approchée (f doit être continue sur I), 3)ϵ_{max}, l'erreur maximale sur le résultat en sortie, 4) N_d, le degré des polynômes approximant, 5) N_l, la profondeur de l'arbre de segmentation.
Les différentes étapes de l'approche proposée sont représentées figure 2 afin d'obtenir l'arbre de segmentation optimale du segment I. L'arbre final est déterminé après deux étapes : l'arbre binaire T_{bin} est déterminé et ensuite, si l'utilisateur souhaite réduire le nombre de niveaux dans

l'arbre obtenu, un arbre avec un nombre réduit de niveaux $T_{reduced}$ est déterminé. L'arbre final est celui dont la profondeur est égale à la profondeur N_l souhaité.
Une fois $T_{reduced}$ obtenu, le code source associé à l'approximation de la fonction f est généré. La table pour le calcul de l'index $\mathcal{T}$, les coefficients des polynômes formatés en virgule-fixe stockés dans $\mathcal{P}$ ainsi que la table stockant les décalages $\mathcal{D}$ issus du calcul en virgule fixe sont écrits dans un fichier d'en-tête.
Afin d'obtenir les courbes de Pareto décrivant le compromis temps d'exécution-taille mémoire pour l'approximation de f sur I avec une erreur maximale ϵ_{max}, la méthode proposée est appliquée avec différentes profondeurs d'arbre N_l variant de 1 à N_{l-max} et différents degrés de polynômes N_d variant de 1 à N_{d-max}. Le terme N_{d-max} est le degré maximal testé et fixé par le développeur.

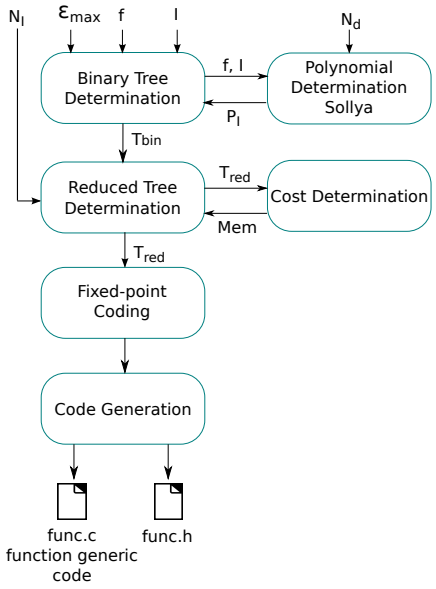


FIGURE 2 – Les différentes étapes de l'algorithme d'approximation de fonctions par des polynômes

3.1. Détermination de l'arbre binaire

3.1.1. Segmentation dichotomique

Cette étape détermine la segmentation non-uniforme de I menant au nombre minimal de segments i.e. au nombre minimal de polynômes à stocker en mémoire. Les bornes des segments sont des sommes de puissances de 2 pour faciliter l'adressage. L'outil Sollya appelle l'algorithme de Remez sur chaque segment tant que l'erreur d'approximation est supérieure à ϵ_{app}. Cet algorithme cherche le minimax, le meilleur polynôme approximant f sur I au sens de la norme infinie. L'erreur maximale ϵ_{max} est distribuée entre ϵ_{app} et ϵ_{fxp}. Un paramètre $\alpha \in [0; 1]$ contrôle cette distribution. La segmentation dichotomique de I est présentée dans l'algorithme récursif 1. Si le critère de l'erreur $\epsilon_{app} < \alpha.\epsilon_{max}$ n'est pas respecté sur un segment, celui-ci est divisé en deux parties égales et l'algorithme est appliqué sur chaque segment obtenu I_a et I_b.
La segmentation de I est modélisée par l'arbre T_{bin}. A chaque appel de l'algorithme récursif 1,

Algorithm 1 The Polynomial Approximation

```
procedure COMPUTE_P(I)
    N_d ; ε_max ; I = [a; b] ; α ;
    P = REMEZ(f, S, N_d)
    ε_app = ||f − P||_∞
    if ε_app ≥ α.ε_max then
        I_a = [a; (b−a)/2 + a]
        I_b = [(b−a)/2 + a; b]
        return COMPUTE_P(I_a)
        return COMPUTE_P(I_b)
    else
        return P
    end if
end procedure
```

les bornes des segments obtenus sont stockées dans un noeud de T_{bin}. La profondeur, N_{l-max}, de l'arbre binaire T_{bin} correspondant au nombre de bits nécessaires pour indexer la table de polynômes $\mathcal{P}$.

3.1.2. Exemple

La segmentation dichotomique est détaillée sur la figure 3. La fonction $\sqrt{-\log x}$ est approchée sur $I = [2^{-5}; 1]$ avec $\epsilon_{max} = 0.002$, $\alpha = 0.5$ et $N_d = 2$. D'après l'arbre obtenu, la fonction possède une forte non-linéarité au voisinage de 1. La correspondance entre les polynômes P_i et les segments obtenus est donnée sous l'arbre.

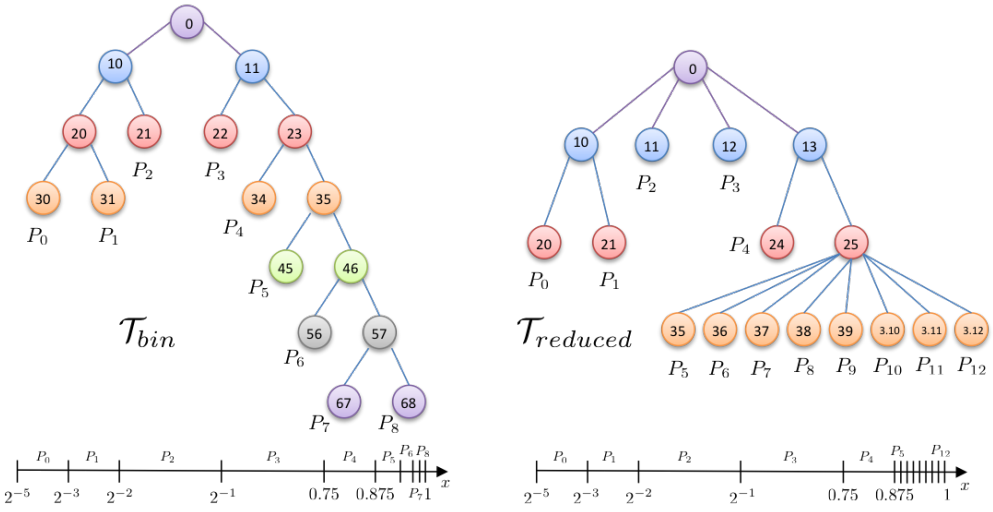


FIGURE 3 – T_{bin} et $T_{reduced}$ avec $f(x) = \sqrt{-\log x}$, $\epsilon_{max} = 10^{-3}$, $\alpha = 0.5$ $N_d = 2$, $I = [0.03125; 1]$.

3.2. Détermination de l'arbre de profondeur réduite $T_{reduced}$

Afin de minimiser le temps de calcul de l'index, l'étape suivante vise à réduire le nombre de niveaux dans l'arbre modélisant la segmentation, réduisant le nombre de tables et donc le nombre de lectures mémoires et de calculs pour l'index. Connaissant le nombre de bits $w_m = N_{l-max}$ à tester pour calculer i, l'arbre stockant la segmentation est modifié et plus d'un bit par niveau est testé. La somme des bits alloués à chaque niveau doit être égale à N_{l-max}. Pour connaître l'allocation optimale de ces bits, un arbre de solutions est calculé. Celui-ci permet de minimiser à la fois le temps d'évaluation de la fonction et les ressources matérielles nécessaires pour stocker les tables $\mathcal{P}$, $\mathcal{D}$ et $\mathcal{T}$.

L'arbre de solutions T_{sol} étudie de façon exhaustive les configurations d'allocation possibles et calcule l'espace mémoire théorique requis par chaque allocation afin de choisir celle nécessitant le minimum d'espace mémoire. La profondeur de l'arbre de solutions est égale au nombre de niveaux requis et chaque branche de l'arbre correspond à un type d'allocation. Chaque noeud de T_{sol} contient le nombre de bits alloués au niveau correspondant. Finalement, le chemin menant à l'espace requis minimum est sélectionné.

3.2.1. Exemple

La détermination de l'arbre de profondeur réduite est détaillée sur l'exemple utilisé Sous-section 3.1.2. La profondeur de T_{bin} est 6. L'arbre de solutions obtenu pour une profondeur réduite de 3 est présenté à la figure 4. Le nombre de bits alloués à chaque niveau est indiqué dans chaque noeud. La mémoire requise par chaque type d'allocation est précisée sous chaque branche en octets.

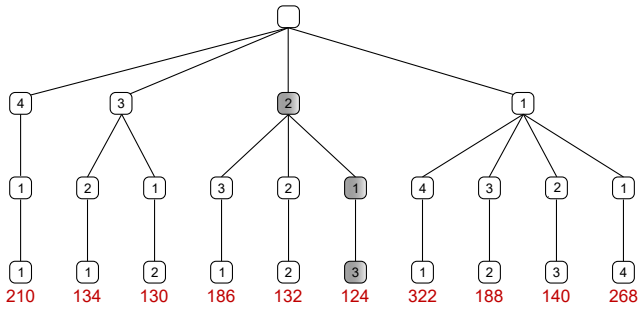


FIGURE 4 – T_{sol} pour $f(x) = \sqrt{-\log x}$ sur $I = [2^{-5}; 1]$ avec $\epsilon_{app} = 10^{-3}$ et $N_d = 2$

L'allocation nécessitant le plus faible espace mémoire est celle de la sixième branche (noeuds en rouge). D'après cette allocation, 2 bits sont alloués au niveau 1 de $T_{reduced}$ (I est segmenté en 4 segments), 1 bit est alloué au second niveau, i.e. si nécessaire, les segments obtenus au niveau 1 sont segmentés en 2 parties. Finalement, 3 bits sont alloués au dernier niveau. Si nécessaire, les segments obtenus au niveau 2 sont segmentés en 8 parties. La figure 3 indique que cette allocation créée 13 polynômes.

4. Expérimentations

La méthode proposée est testée sur deux processeurs à faible coût. Le premier est le DSP C55x de Texas Instruments [3]. Le second processeur utilisé pour tester la méthode proposée est le

Cortex M3 d'ARM.

4.1. Courbes de Pareto fournies par la méthode proposée

L'analyse du compromis entre la taille de la mémoire et le temps d'évaluation de la fonction est réalisée à l'aide des courbes de Pareto. Celles-ci permettent de connaître le degré et la profondeur de l'arbre $T_{reduced}$ optimaux pour le processeur visé et les critères sur la taille mémoire et le temps d'évaluation de la fonction. Celles-ci fournissent les points optimaux pour chaque degré testé.

4.1.1. Expériences avec la fonction $\exp -\sqrt{x}$

La fonction $\exp -\sqrt{x}$ est étudiée sur le segment $I = [2^{-6}; 2^5]$. Les arbres pour des polynômes de degré variant de 1 à 5 ont été calculés avec une profondeur variant de N_{l-max} (arbre binaire) à une profondeur de 2. L'erreur maximale ϵ_{max} est de 10^{-2} et le paramètre $\alpha = 0.5$. Les données et coefficients sont codés sur 16 bits, le codage de l'entrée est $Q_{6,10}$ et les tests ont été effectués sur le DSP C5535 de Texas Instruments. Les courbes de Pareto figure 5 fournissent les points optimaux pour le temps d'évaluation de la fonction ou pour l'évolution de l'espace mémoire. L'approximation par des polynômes de degré 1 à 4 en virgule fixe sur 16 bits est adaptée car l'erreur en virgule fixe obtenue est inférieure à ϵ_{fxp}. Des polynômes de degré 5 ne sont pas adaptés puisque l'erreur est trop élevée. La même fonction est étudiée dans les mêmes conditions d'approximation sur la cible de ARM, le micro-contrôleur Cortex M3. Les courbes de Pareto représentant l'évolution de l'espace mémoire requis en fonction du temps d'évaluation de la fonction sont sur la figure 5 (droite).

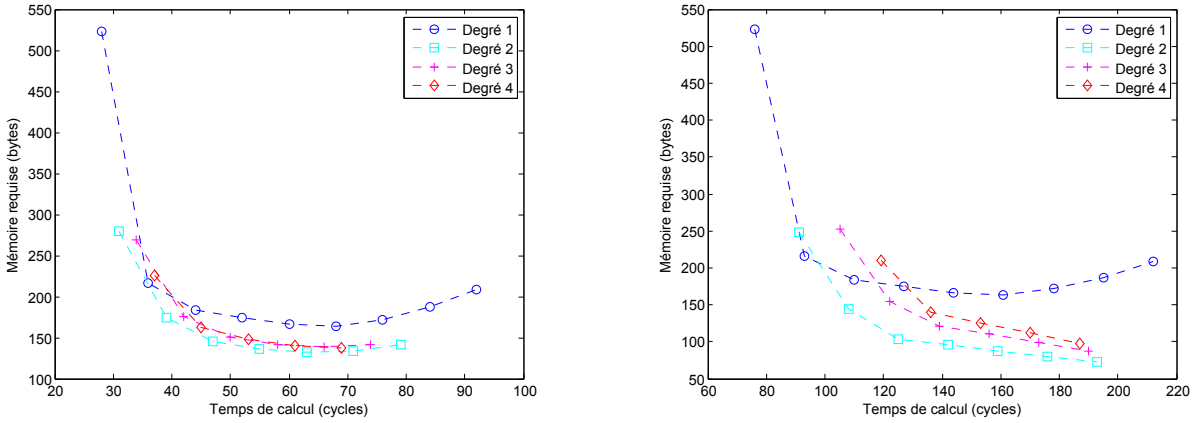


FIGURE 5 – Courbes de Pareto obtenues lors de l'approximation de la fonction $\exp -\sqrt{x}$ sur $[2^{-6}; 2^5]$) pour le DSP C55x (gauche) et Cortex M3 (droite).

D'après les courbes de Pareto présentées à la figure 5, lorsque $T_{reduced}$ a un nombre faible de niveaux, l'approximation nécessite un espace mémoire important car le nombre de polynômes à stocker est élevé mais le temps d'évaluation de la fonction est minimum. Ensuite, le temps d'évaluation de la fonction augmente avec le nombre de niveaux dans l'arbre dû au calcul d'index qui se complexifie tandis que l'espace mémoire nécessaire diminue jusqu'à atteindre un minimum. Une fois ce minimum atteint, l'espace mémoire requis tend à augmenter car la taille de la table $\mathcal{T}$ augmente ainsi que le temps d'évaluation de la fonction en fonction du

nombre de niveaux.

4.2. Comparaison de l'espace mémoire requis par la méthode proposée et par tabulation directe

La méthode proposée est plus lente que la méthode de calcul à l'aide de tabulation directe. Néanmoins, la mémoire requise pour effectuer ces calculs est répertoriée dans la table 1 pour la méthode proposée et pour la tabulation directe et le gain en espace mémoire requis offert par la méthode proposée est significatif.

Fonction	ϵ_{tot}	I	Mem_{tab}	Mem_{prop}
$\exp -\sqrt{x}$	10^{-2}	$[2^{-6}; 2^{5}]$	8192	530
$\sin(x)$	10^{-2}	$[0; \frac{\pi}{2}]$	256	84

TABLE 1 – Comparaison de l'empreinte mémoire pour la méthode proposée Mem_{prop} (valeur moyenne) et par la méthode des tables Mem_{tab}

5. Conclusion

La méthode proposée donne de bons résultats en termes de flexibilité. A l'aide des courbes de Pareto, l'utilisateur peut choisir les caractéristiques d'approximation menant à la segmentation optimale pour les contraintes de son application (temps d'évaluation de la fonction) et du processeur visé (taille mémoire requise). Ainsi, le choix du degré des polynômes approximant ainsi que de la profondeur de l'arbre de segmentation vont dépendre du point désiré sur la courbe de Pareto. Finalement, par rapport à la méthode des tables, la méthode proposée réduit de façon importante la taille mémoire requise.

Bibliographie

1. Chevillard (S.), Joldeş (M.) et Lauter (C.). – Sollya : An environment for the development of numerical codes. – In Fukuda (K.), van der Hoeven (J.), Joswig (M.) et Takayama (N.) (édité par), *Mathematical Software - ICMS 2010*, *Lecture Notes in Computer Science*, volume 6327, pp. 28–31, Heidelberg, Germany, September 2010. Springer.
2. de Dinechin (F.) et Tisserand (A.). – Multipartite table methods. *IEEE Transactions on Computers*, vol. 54, n3, mars 2005, pp. 319–330.
3. Instruments (T.). – *C55x v3.x CPU Reference Guide*. – Dallas, June 2009.
4. Lee (D.-U.), Cheung (R.), Luk (W.) et Villasenor (J.). – Hierarchical segmentation for hardware function evaluation. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 17, n1, janvier 2009, pp. 103–116.
5. Press (W. H.), Teukolsky (S. A.), Vetterling (W. T.) et Flannery (B. P.). – *Numerical recipes in C (2nd ed.) : the art of scientific computing*. – Cambridge University Press New York, NY, USA, 1992.
6. Volder (J. E.). – The CORDIC trigonometric computing technique. *IRE Transactions on Electronic Computers*, vol. EC-8, n3, septembre 1959, pp. 330–334.