Performances de schémas d'évaluation polynomiale sur architectures vectorielles

Hugues de Lassus Saint-Geniès, Guillaume Revy

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

Performances de schémas d'évaluation polynomiale sur architectures vectorielles

Hugues de Lassus Saint-Geniès et Guillaume Revy

Université de Perpignan Via Domitia, DALI, F-66860, Perpignan, France
Université de Montpellier, LIRMM, UMR 5506, F-34095, Montpellier, France
CNRS, LIRMM, UMR 5506, F-34095, Montpellier, France
hugues.de-lassus@univ-perp.fr, guillaume.revy@univ-perp.fr

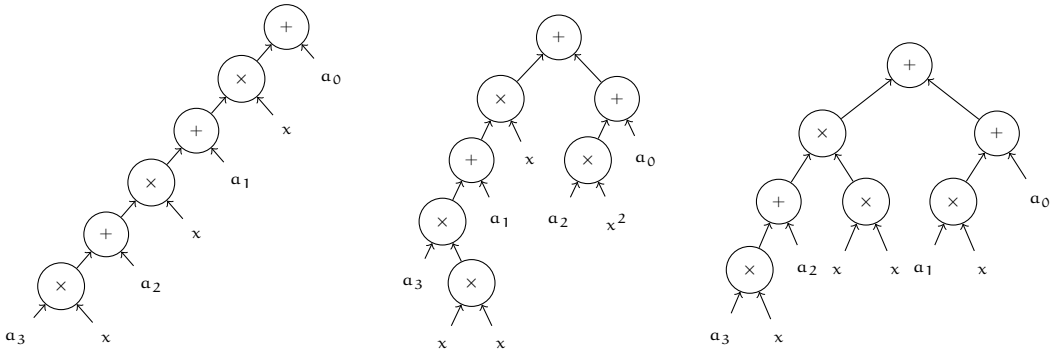
Résumé
Les fonctions élémentaires sont souvent calculées à l'aide d'approximations polynomiales, dont l'efficacité dépend directement de celle du schéma d'évaluation sous-jacent. Cet article montre que le schéma classiquement utilisé (Horner) est rarement le plus performant. En effet, d'autres schémas exploitent mieux les parallélismes des architectures modernes, en réduisant les dépendances de données. Ces résultats ont pour objectif d'être intégrés à un générateur de code performant pour l'évaluation polynomiale dans le cadre de l'approximation de fonctions.
Mots-clés : évaluation polynomiale, performance, vectorisation.

---

1. Introduction

L'implantation de fonctions élémentaires (sin, log, ...) en logiciel passe souvent par une approximation polynomiale de degré raisonnable, sur un intervalle restreint [10, §11.4]. L'évaluation d'un tel polynôme, c'est-à-dire l'ordre dans lequel les opérations sont effectuées, est fixé statiquement par un *schéma d'évaluation* dans le code, pour plusieurs architectures matérielles. Le schéma le plus utilisé est celui de Horner (illustré Figure 1(a)), qui est essentiellement séquentiel. Sa latence [1] est donc relativement élevée [11]. Cependant, sélectionner manuellement un schéma plus rapide pour une architecture donnée peut s'avérer long et complexe.
Notre objectif est de générer automatiquement des schémas qui profitent au mieux des architectures modernes, notamment du parallélisme d'instructions (ILP) et du parallélisme de données (instructions SIMD [2]). Cet article présente une étude de la performance de différents schémas sur une architecture de type Intel Haswell.[3] Nous montrons deux résultats utiles : d'une part, des schémas exposant beaucoup d'ILP peuvent avoir *à la fois* une latence *et* un débit meilleurs que des schémas plus séquentiels, un résultat qui contredit les préconisations usuelles [10, §11.5]. D'autre part, quand la production d'un résultat dénormalisé est très coûteuse, nous notons que l'intervalle d'évaluation prévu pour un polynôme donné est un facteur déterminant pour le choix d'un bon schéma.
L'article est organisé comme suit : La Section 2 rappelle les schémas d'évaluation classiques étudiés dans cet article. La Section 3 analyse les débits de plusieurs schémas sur Intel Haswell. Enfin, la Section 4 conclut sur la génération de code performant pour l'évaluation polynomiale.

1. Sauf pour nos mesures, le terme *latence* est assimilé à la *latence minimale sur parallélisme infini*.
2. *Single Instruction, Multiple Data* : instruction unique appliquée à des données multiples.
3. `https://software.intel.com/haswell`



(a) Horner (b) Horner-2 (c) Estrin

FIGURE 1 – Schémas de Horner, Horner d'ordre 2 et Estrin pour un polynôme de degré 3.

2. Rappels sur l'évaluation polynomiale

Soit P un polynôme de degré n à coefficients flottants non nuls, comme définis dans la norme IEEE 754-2008 [6]. On veut évaluer P en un point x dans le même format. Dans cet article, on ne considère que les schémas d'évaluation sans adaptation de coefficients. (Plus de détails sur l'adaptation de coefficients peuvent être trouvés dans [4, 13].)
Le schéma le plus courant est celui de Horner. Il permet d'évaluer $P = \sum_{i=0}^{n} a_i X^i$ en n additions et n multiplications. Par exemple, pour $n = 7$, on aura :

$$P(x) = a_0 + x \cdot (a_1 + x \cdot (a_2 + x \cdot (a_3 + x \cdot (a_4 + x \cdot (a_5 + x \cdot (a_6 + x \cdot a_7)))))) \,. \tag{1}$$

Son intérêt est triple : il minimise le nombre de multiplications, n'a pas besoin de stocker de résultat intermédiaire [7, p.486] et, pour l'évaluation de fonctions élémentaires, il est stable numériquement [1, ch.9]. Quand le critère de choix est la précision, Horner est donc un bon schéma. Mais, on l'a vu, sa latence est élevée. Or, il existe des schémas plus rapides. Par exemple, le schéma de Horner d'ordre 2, noté Horner-2, a le parenthésage suivant au degré 7 :

$$P(x) = \left(a_0 + x^2 \cdot \left(a_2 + x^2 \cdot \left(a_4 + x^2 \cdot a_6\right)\right)\right) + x \cdot \left(a_1 + x^2 \cdot \left(a_3 + x^2 \cdot \left(a_5 + x^2 \cdot a_7\right)\right)\right) . \tag{2}$$

Ce schéma expose de l'ILP, comme illustré à la Figure 1(b) : les mônomes de degrés pair et impair peuvent être évalués en parallèle. Sa latence est donc environ divisée par deux [7, p.488].
Le schéma d'Estrin, illustré en Figure 1(c), expose encore plus d'ILP en découpant l'arbre d'évaluation selon les puissances de x de la forme 2^i [3]. Par exemple, pour $n = 7$, on aura :

$$P(x) = \left((a_0 + a_1 \cdot x) + x^2 \cdot (a_2 + a_3 \cdot x)\right) + x^4 \left((a_4 + a_5 \cdot x) + x^2 \, (a_6 + a_7 \cdot x)\right) . \tag{3}$$

Dans le cas d'un polynôme de degré $n = 2^k - 1$, l'arbre d'évaluation d'Estrin est bien équilibré, mais il l'est un peu moins dans le cas général. Sa latence est en $O(\log n)$.
Avec une architecture simplifiée infiniment parallèle ayant respectivement des latences de 3 et 5 cycles pour l'addition et la multiplication flottantes,[4] pour $n = 7$ et en utilisant (1), (2) et (3), on peut prévoir des latences de 56, 37 et 24 cycles, pour, respectivement, Horner, Horner-2 et Estrin. Avec un FMA en 5 cycles, on peut s'attendre, respectivement, à des latences de 35, 25 et

4. Latences minimales documentées pour l'architecture Haswell.

15 cycles. Il est donc important de choisir un bon schéma d'évaluation. Cependant, le nombre de schémas possibles au degré n croît rapidement avec n du fait de la combinatoire, ce qui rend impossible une recherche exhaustive dès que $n \geq 6$ [14, ch.6].
D'autres schémas ainsi que la question de l'optimalité asymptotique sur parallélisme fini ou infini ont été étudiés dans [8,11,12]. Dans le cadre de l'évaluation de fonctions élémentaires, nous avons comparé les performances de Horner à deux schémas classiques : Horner-2 et Estrin ; puis, à tous ceux de plus faible latence pour le degré 12 sur notre architecture simplifiée.

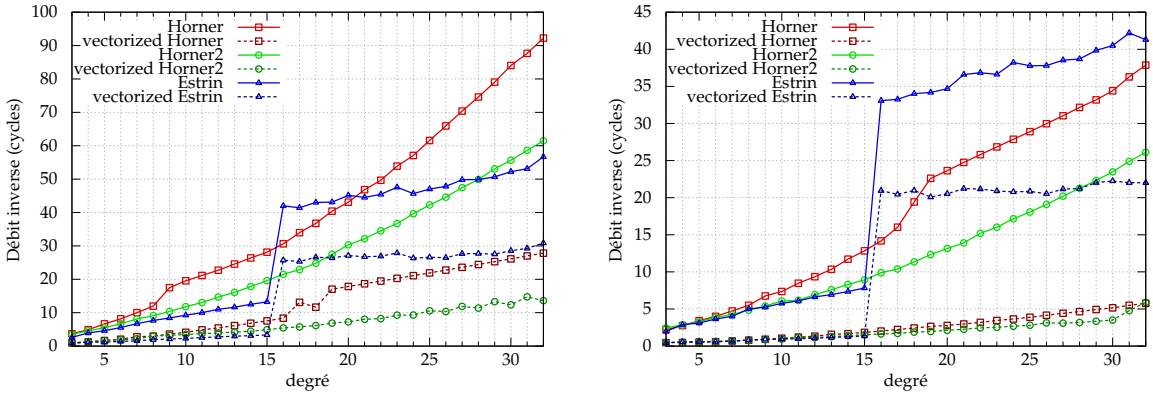
3. Comparaison des performances de schémas d'évaluation polynomiale

Dans cette section, nous analysons les résultats de mesures de performance obtenus pour différents schémas, compilés avec GCC 5.2.0 (option `-O2` activée), sur un processeur Intel Haswell bénéficiant des extensions AVX2 et FMA. Celles-ci opèrent sur 16 registres de 256 bits, en scalaire (sur un mot de poids faible) ou en vectoriel, et disposent d'un additionneur et de deux multiplieurs/FMA [5, §10.14]. GCC 5.2.0 est capable de générer des instructions vectorielles via une passe d'autovectorisation, que nous utilisons pour produire du code vectorisé.
Nous mesurons le débit inverse de plusieurs schémas, c'est-à-dire, le nombre moyen de cycles entre deux résultats. Plus ce nombre est faible, plus le schéma est performant. Dans nos analyses, nous convertissons implicitement la mesure en *débit*. Pour un degré fixé, il dépend *a priori* du schéma et de l'architecture, que nous faisons varier en autorisant ou non l'utilisation du FMA. Sur ce type d'architecture, tant qu'aucun calcul ne dénormalise, nous observons que les schémas exposant le plus d'ILP ont un meilleur débit que les schémas séquentiels.

3.1. Mesures sur des schémas d'évaluation classiques

Dans cette section, afin que l'expérience soit représentative de l'évaluation de fonctions, nous faisons varier le degré du polynôme entre 3 et 32, et nous utilisons les coefficients de la série de Taylor de $\log(1+x)$ en 0, calculés avec Sollya.[5] Nous réalisons plusieurs évaluations sur des vecteurs de 2^{14} nombres flottants uniformément répartis dans l'intervalle $[2^{-12}; 2^{-5}]$.
La Figure 2 présente les débits inverses mesurés en simple précision pour Horner, Horner-2 et Estrin, avec ou sans vectorisation. Sans surprise, le débit diminue avec le degré pour tous les schémas. En revanche, on observe une perte de performance drastique à partir du degré 16, uniquement pour le schéma d'Estrin. Dans la Section 3.2, nous expliquons cette perte de performance par le fait que le résultat intermédiaire x^{16}, calculé par Estrin, dénormalise dans 13% des cas. Sans FMA, Estrin a le meilleur débit jusqu'au degré 15 : pour ce degré, il est plus de deux fois meilleur que Horner et 50% meilleur que Horner-2, pour les deux versions (scalaire et vectorisée). Au degré 16, il est de 27 à 67% inférieur à Horner et de 49 à 79% inférieur à Horner-2, pour les versions scalaire et vectorisée, respectivement. Avec FMA, les trois schémas se valent jusqu'au degré 15, avec un désavantage pour Horner non vectorisé. À partir du degré 16, seul Horner-2 est meilleur que Horner, d'environ 40%. Globalement, jusqu'au degré 32, aucun des trois schémas ne souffre du nombre limité de registres, le séquenceur de GCC parvenant à conserver les résultats intermédiaires dans ces derniers. Cela est encourageant pour l'utilisation de schémas très parallèles dans le cadre de l'évaluation de fonctions.
En faisant les mêmes expériences en double précision, on n'observe pas la perte de performance obtenue pour Estrin en simple précision. Cela conforte l'hypothèse que ceci est causé par un résultat intermédiaire dénormalisé. La diminution du débit avec le degré est alors quasi-linéaire, et le schéma d'Estrin a un débit toujours supérieur à Horner dès le degré 10 dans tous les cas. Nous concluons qu'une latence faible n'entraîne pas toujours une chute du débit, ce qui peut

5. Voir `http://sollya.gforge.inria.fr/` et [2].



(a) sans FMA (b) avec FMA

FIGURE 2 – Moyenne des débits inverses de schémas classiques en simple précision.

rendre des schémas comme Horner-2 ou Estrin plus intéressants si l'on veut privilégier la performance à la précision. Ce résultat est assez contre-intuitif car sur parallélisme fini, on pouvait s'attendre à une saturation des unités arithmétiques pour les schémas très parallèles.

3.2. Influence de l'intervalle d'évaluation

Dans la suite, on note $\circ(x)$ l'arrondi dans une direction quelconque d'un réel x en simple précision. Les polynômes utilisés pour l'évaluation de fonctions élémentaires sont valides sur un petit intervalle. Les autres entrées sont soit traitées séparément, soit ramenées à l'intervalle d'évaluation. Cet intervalle est un voisinage du point d'approximation qui est souvent 0, c'est le cas pour $\ln(1+x)$, $\exp(x)$, ou $\sin(x)$. Les entrées de petites magnitudes élevées à certaines puissances peuvent alors dénormaliser rapidement. Par exemple, pour $x \in [2^{-12}; 2^{-5}]$, x^{16} dénormalise en simple précision dans environ $(2^{-126/16} - 2^{-12})/2^{-5} \approx 13\%$ des cas.

La Figure 3 montre la latence mesurée de deux schémas d'élévation à la puissance 16 de nombres flottants simple précision dans le même intervalle. L'un correspond à une exponentiation naïve, l'autre à une exponentiation binaire. Leurs latences minimales sont de l'ordre de quelques cycles, mais quand x devient suffisamment petit pour que x^{16} dénormalise, on observe un palier pour les deux schémas. Les latences observées passent ainsi de quelques cycles à plus d'une centaine. D'après Agner Fog, cette pénalité provient de traitements micro-codés, pour toutes les opérations dont les opérandes sont normaux et dont le résultat est dénormalisé, ainsi que pour une multiplication dont un opérande est normal et l'autre dénormalisé [5, §10.9]. Les paliers suivants observés pour la méthode naïve commencent quand les mônomes de degré inférieur dénormalisent. On remarque aussi que la version binaire ne retrouve une latence de quelques cycles que bien après que $\circ(x^{16}) = 0$, ce qui vient en partie contredire le manuel de Fog.[6] Cela suggère que le micro-code n'est exécuté que pour un intervalle précis de résultat. Par ailleurs, lorsque $\circ(x^{15}) = 0$, i.e. $|x| \leq 2^{-10}$, on peut voir un palier descendant pour la version naïve. Cela montre que la multiplication par zéro ne souffre pas de l'exécution de micro-code.

Ainsi, sur certaines architectures, le fait qu'un résultat intermédiaire dénormalise diminue drastiquement les performances des calculs, notamment des évaluations polynomiales. Il est alors pertinent de s'assurer que, pour un schéma d'évaluation donné, l'intervalle d'entrée

6. « There is no penalty for overflow, underflow, infinity or not-a-number results » [5, §10.9].

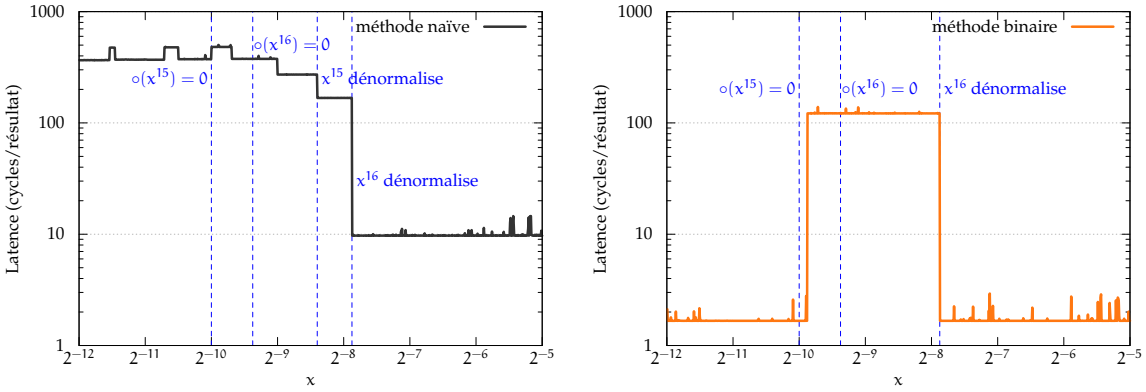


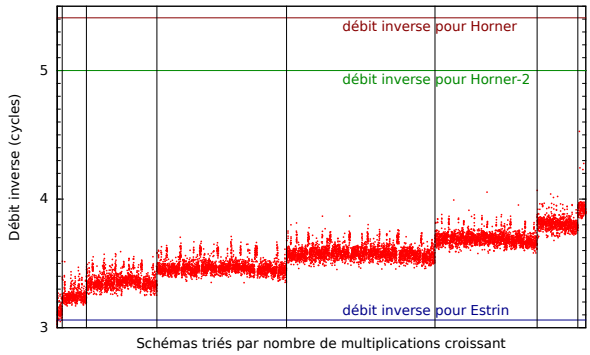
FIGURE 3 – Latences du calcul de x^{16} en simple précision par exponentiations naïve et binaire.

prévu n'engendre pas de dénormalisés. Ce résultat pourrait permettre d'améliorer la sélection automatique d'un schéma performant sur l'ensemble de son intervalle d'utilisation.

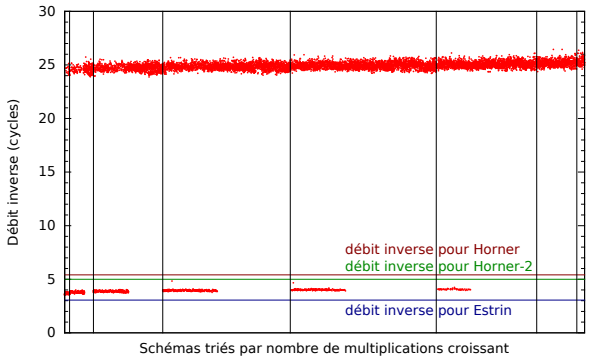
3.3. Des schémas encore plus performants

L'outil CGPE[7] permet notamment de calculer tous les schémas d'évaluation polynomiale de plus faible latence sur parallélisme infini, pour un degré et une architecture donnés. Nous l'avons utilisé pour générer ceux de degré 12, ce qui représente 11 944 schémas de latence 26 avec l'architecture considérée. Les débits de ces schémas compilés sans FMA et avec autovectorisation, ont ensuite été mesurés pour des entrées dont certaines puissances dénormalisent. La Figure 4 représente ces 11 944 mesures par des points. Les traits verticaux distinguent les schémas selon le nombre de multiplications (de 17 à 25) qu'ils impliquent.
Sur la Figure 4(a), on remarque que *tous* les schémas de plus faible latence ont un *meilleur* débit que les schémas de Horner et Horner-2, et seuls une dizaine de schémas sont aussi bons qu'Estrin. Ce sont ceux qui impliquent le moins de multiplications. Assez logiquement, on observe aussi des paliers indiquant des débits plus faibles à mesure que le nombre de multiplications augmente. La Figure 4(b) vient confirmer que le schéma d'Estrin peut être beaucoup plus performant que beaucoup de schémas de latence minimale, car il peut être en mesure de produire moins de résultats intermédiaires dénormalisés. En revanche, la Figure 4(c) montre qu'Estrin est environ 12 fois moins bon que Horner et Horner-2 quand la dénormalisation intervient à partir de x^8, alors qu'une cinquantaine de schémas générés restent plus performants que Horner et Horner-2, notamment parmi ceux qui impliquent peu de multiplications. Cela va dans le sens de nos conclusions tirées en Section 3.2, c'est-à-dire, qu'il faut s'assurer que le schéma que l'on choisit, pour une architecture qui lève des exceptions pour des résultats dénormalisés, ne produise pas lui-même de résultats dénormalisés sur son intervalle d'évaluation.
Une étude plus approfondie des meilleurs schémas générés par CGPE pourrait nous permettre de distinguer des motifs, réutilisables dans le cadre de l'approximation de fonctions par des polynômes univariés. Dans le cas contraire, une mesure automatisée et gloutonne de ces schémas sur une architecture fixée nous permettrait d'en sélectionner un plus performant que ceux couramment utilisés. *A posteriori*, une borne sur la précision du résultat final pourrait être calculée afin de vérifier si un schéma sélectionné est conforme aux exigences demandées.

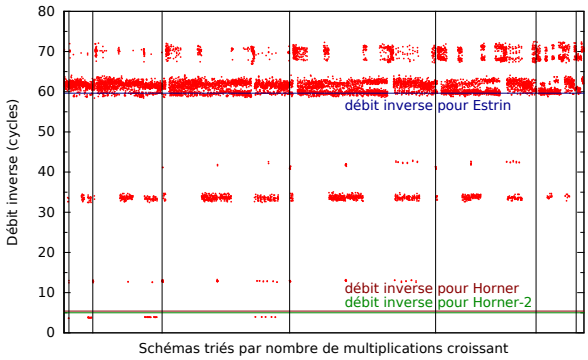
7. Voir `http://cgpe.gforge.inria.fr/` et [9,14].



(a) x^n est normal pour $n \in [1; 12]$



(b) x^n dénormalise quand $n = 12$



(c) x^n dénormalise quand $n = 8$

FIGURE 4 – Débits inverses moyens des schémas de degré 12 et de plus faible latence, triés par nombre de multiplications croissants.

4. Conclusion

Des observations décrites dans cet article, nous tirons plusieurs conclusions quant à la génération de code pour l'évaluation polynomiale. Dans le cadre de l'évaluation polynomiale pour l'approximation de fonctions, nous avons montré que certains types de dénormalisation impactent fortement les performances, ce qui en fait un facteur essentiel pour choisir un schéma performant sur tout son intervalle d'évaluation. Enfin, nous avons vu que des schémas exposant beaucoup d'ILP comme le schéma d'Estrin ou ceux générés par CGPE ont souvent un meilleur débit que les schémas séquentiels comme Horner.

Nous voulons poursuivre nos recherches dans trois directions : Premièrement, nous souhaiterions étudier certains des meilleurs schémas générés par CGPE, notamment ceux impliquant le plus faible nombre de multiplications. Cela pourrait nous permettre de construire directement des schémas performants sans passer par une recherche exhaustive. Ensuite, nous envisageons d'intégrer des critères de choix à CGPE pour éliminer les schémas qui, pour un intervalle d'évaluation et un ensemble de coefficients donnés, pourraient engendrer des résultats intermédiaires dénormalisés. Par exemple, cela peut se déterminer par arithmétique d'intervalles. Enfin, nous aimerions mesurer l'influence qu'a le choix d'un bon schéma sur les performances de l'évaluation de fonctions élémentaires.

À plus long terme, notre but est de développer un générateur de code performant pour l'évaluation polynomiale, dans le cadre de l'approximation de fonctions. Il sera alors intéressant de réfléchir aux compromis entre performance et précision, ainsi qu'aux garanties formelles qu'il sera possible d'apporter.

Bibliographie

1. Boldo (S.). – *Preuves formelles en arithmétiques à virgule flottante*. – Thèse de PhD, École Normale Supérieure de Lyon, 2004.
2. Chevillard (S.), Joldeş (M.) et Lauter (C.). – Sollya : An environment for the development of numerical codes. – In *Mathematical Software - ICMS 2010*, pp. 28–31. Springer, 2010.
3. Estrin (G.). – Organization of computer systems : The fixed plus variable structure computer. – In *Papers Presented at the Western Joint IRE-AIEE-ACM Computer Conference*, pp. 33–40, New York, NY, USA, 1960. ACM.
4. Eve (J.). – The evaluation of polynomials. *Numerische Mathematik*, vol. 6, n1, 1964, pp. 17–21.
5. Fog (A.). – *The microarchitecture of Intel, AMD and VIA CPUs : An optimization guide for assembly programmers and compiler makers*, 2016.
6. *IEEE Standard for Floating-Point Arithmetic*, 2008.
7. Knuth (D. E.). – *The Art of Computer Programming, Volume 2, Seminumerical Algorithms*. – Reading, MA, USA, Addison-Wesley, 1998, third édition.
8. Maruyama (K.). – On the parallel evaluation of polynomials. *Computers, IEEE Transactions on*, vol. C-22, n1, 1973, pp. 2–5.
9. Mouilleron (C.) et Revy (G.). – Automatic Generation of Fast and Certified Code for Polynomial Evaluation. – In *20th IEEE Symposium on Computer Arithmetic (ARITH'20)*, pp. 233–242. IEEE Computer Society, 2011.
10. Muller (J.-M.), Brisebarre (N.), de Dinechin (F.), Jeannerod (C.-P.), Lefèvre (V.), Melquiond (G.), Revol (N.), Stehlé (D.) et Torres (S.). – *Handbook of Floating-Point Arithmetic*. – Birkhäuser Boston, 2010.
11. Munro (I.) et Paterson (M.). – Optimal algorithms for parallel polynomial evaluation. *Journal of Computer and System Sciences*, vol. 7, n2, 1973, pp. 189 – 198.
12. Muraoka (Y.). – *Parallelism Exposure and Exploitation in Programs*. – Thèse de PhD, Dept. of Computer Science, University of Illinois at Urbana-Champaign, 1971.
13. Paterson (M. S.) et Stockmeyer (L. J.). – On the number of nonscalar multiplications necessary to evaluate polynomials. *SIAM Journal on Computing*, vol. 2, n1, 1973, pp. 60–66.
14. Revy (G.). – *Implementation of binary floating-point arithmetic on embedded integer processors - Polynomial evaluation-based algorithms and certified code generation*. – 46 allée d'Italie, F-69364 Lyon cedex 07, France, Thèse de PhD, Université de Lyon - École Normale Supérieure de Lyon, 2009.