Arithmétique redondante : impacts sur l'architecture des machines

Jean-Michel MULLER
Laboratoire LIP,
Ecole Normale Supérieure de Lyon
46 Allée d'Italie,
69364 Lyon cedex 07

1 - Introduction

Les systèmes redondants de représentation des nombres, introduits par Avizienis [1] en 1961, sont souvent basés sur l'utilisation de chiffres qui peuvent être négatifs. Par exemple, en base 10, on peut représenter tous les nombres avec des chiffres allant de -5 à +5. Ces systèmes permettent d'effectuer des additions sans propagation de retenue, et sont donc très attractifs pour faire de l'arithmétique rapide.

Les opérateurs arithmétiques "série" reçoivent les chiffres des opérandes et fournissent les chiffres du résultat un par un. Un opérateur série efficace commence à fournir les premiers chiffres du résultat avant d'avoir reçu tous les chiffres des opérandes : ceci permet d'évaluer rapidement des expressions, en effectuant un *pipe-line* au niveau des chiffres. On dira qu'un opérateur série est de délai δ s'il fournit le $i$ème chiffre du résultat après avoir reçu les $i + \delta$ premiers chiffres des opérandes.

Considérons l'exemple suivant : on évalue $y + \sin(x^2)$, et on dispose d'un additionneur de délai 2, d'un quadrateur de délai 3, et d'un opérateur de calcul du sinus, de délai 4. L'ensemble du calcul se fait avec un délai égal à 9 : le premier chiffre du résultat commence à sortir après l'entrée du 10ème chiffre de = x, c'est-à-dire bien avant que la première opération ne soit terminée, et ceci quelle que soit la taille des opérandes. On ne peut profiter de cette possibilité de pipe-line au niveau du chiffre que si dans tous les opérateurs les chiffres circulent dans le même sens, ce qui semble *a priori* impossible : Les algorithmes usuels évaluent une somme ou un produit de droite à gauche, mais par contre, on n'évalue les chiffres du quotient ou du maximum de deux nombres que de gauche à droite. Ce problème est résolu en utilisant des systèmes *redondants* d'écriture des nombres, qui permettent d'effectuer des additions sans propagations de retenues, et grâce auxquels tous les calculs peuvent être faits en série, de la gauche vers la droite. Les opérateurs série dans lesquels les chiffres circulent en commençant par la gauche sont appelés opérateurs *en ligne*, ils ont été introduits en 1977 par Ercegovac et Trivedi [4].

Dans la suite de cet exposé, je vais donner deux exemples d'application des systèmes redondants d'écriture des nombres : l'évaluation rapide de séries entières grâce à l'arithmétique "en ligne", et la réduction d'argument rapide.

2 - Evaluation de polynômes grâce à l'arithmétique en ligne

Dans sa thèse de doctorat [5] Jean-Claude Bajard propose d'évaluer en ligne (c'est-à-dire en série, poids forts en tête) des séries entières. Bien entendu, en pratique, les séries sont tronquées à un rang dépendant de la précision désirée, par conséquent, ce sont en fait des polynômes que nous évaluons. Si je parle "d'évaluation de séries" plutôt que "d'évaluation de polynômes", c'est parce que nous obtenons des conditions sur les coefficients qui sont relativement contraignantes (décroissance rapide), mais qui sont

satisfaites, tout au moins à partir d'un certain rang, par les développements en série entière des fonctions mathématiques usuelles. Donnons nous une cellule élémentaire qui, en fonction de 3 entrées c, d et x comprises entre -1/2 et 1/2, calcule cx+d en ligne, avec un délai égal à 2 :

L'idée naïve, pour évaluer une série entière $S(x) = a_0 + a_1x = + a_2x^2 + a_3x^3 + ...$ est d'utiliser le *Schéma de Horner* (c'est-à-dire d'écrire $S(x) = a_0 + x(a_1 + = x(a_2 + x$ (...)))...), et de disposer nos cellules élémentaires comme suit :

Toutefois, cette architecture naïve comporte un inconvénient : le délai de ce calcul est énorme, puisqu'il est égal à la somme des délais des cellules élémentaires utilisées. On peut remédier à ce problème comme suit : si dans la cellule élémentaire présentée plus haut on entre 4c et 4d, alors on sort 4(cx + d) avec délai 2, c'est à dire cx + d avec délai 0. Il est possible, en décalant en conséquence les coefficients de la série entière S (c'est-à-dire en multipliant le coefficient a_i par 4^i) de faire tous les calculs élémentaires avec délai zéro (à la condition que 4^ia_i soit une entrée admissible d'une cellule élémentaire, c'est-à-dire qu'il soit inférieur à 1/2 : c'est de là que vient la condition sur les coefficients), et donc de faire l'ensemble du calcul avec délai zéro. Il reste ensuite, afin de pouvoir cadencer le calcul plus rapidement, à placer des barrières temporelles toutes les p cellules - et donc perdre un peu en délai - on fera varier p pour chercher un compromis entre le délai et la vitesse à laquelle on peut cadencer le circuit.

3 - La réduction d'argument

Les algorithmes généralement utilisés pour calculer les fonctions élémentaires en machine (c'est-à-dire les fonctions sinus, cosinus, exponentielle...) ne donnent un résultat correct que dans un intervalle borné. Pour calculer une fonction élémentaire f(x) pour tout x, il est nécessaire de trouver une "transformation" qui permette de déduire f(x) d'une fonction $g(x^*)$, où :

- x^* est déduit de x, x^* est appelé l'*argument réduit*
- x^* appartient au domaine de convergence d'un algorithme 0 permettant de calculer g (en général g = f).

En pratique, deux types de réduction apparaissent :

1. **Réduction additive.** $x^* = x - kC$, où k est un entier et C une constante.

2. **Réduction multiplicative.** $x^* = x/C^k$, où k est un entier et C une constante.

3.1 - Exemples

3.1.1. Calcul de la fonction cosinus

Supposons que nous désirions évaluer cos(x), et que l'algorithme utilisé pour calculer le sinus ou le cosinus de l'argument réduit converge dans $[-\pi/4 , +\pi/4]$. On peut choisir $C = \pi/2$, et le calcul de cos(x) se fait comme suit :

- Calculer x^* et k tels que $x^* \in [-\pi/4, +\pi/4]$ et $x^* = x - k\pi/2$
- Calculer $g(x^*,k) =$

$$(1) \qquad \begin{cases} \cos(x^*) & \text{si} \quad k \bmod 4 = 0 \\ -\sin(x^*) & \text{si} \quad k \bmod 4 = 1 \\ -\cos(x^*) & \text{si} \quad k \bmod 4 = 2 \\ \sin(x^*) & \text{si} \quad k \bmod 4 = 3 \end{cases}$$

- On obtient alors: $\cos(x) = g(x^*,k)$.

Cette réduction d'argument est *additive*. Examinons un autre exemple de réduction additive.

3.1.2. Calcul de la fonction exponentielle

Supposons que l'on veuille calculer e^x sur une machine fonctionnant en base 2, et que l'algorithme utilisé pour calculer le sinus ou le cosinus de l'argument réduit converge dans $[0,\ln(2)]$. On peut choisir $C = \ln(2)$, et utiliser l'algorithme suivant :

- Calculer $x^* \in [0,\ln(2)]$ et k tels que $x^* = x - k\ln(2)$.
- Calculer $g(x^*,k) = e^{x^*}$
- Calculer $e^x = 2^k g(x^*,k)$

En base 2, la multiplication finale par 2^k est élémentaire.

3.1.3. Calcul du logarithme

Supposons que l'on veuille calculer $\ln(x)$ $(x > 0)$ sur une machine fonctionnant en base 2, et que l'algorithme utilisé pour calculer le logarithme de l'argument réduit converge dans $[1/2,1]$. On peut choisir $C = 2$, et utiliser l'algorithme suivant:

- Calculer $x^* \in [1/2,1]$ et k tels que $x^* = x/2^k$.
- Calculer $g(x^*,k) = \ln(x^*)$
- Calculer $\ln(x) = g(x^*,k) + k\ln(2)$.

Cette dernière réduction est *multiplicative*. En pratique, la réduction multiplicative n'apparaît que pour les logarithmes et les racines nèmes. Elle s'effectue donc aisément, car pour ces fonctions on peut toujours se ramener au cas où C est une puissance de la base, ce qui rend le calcul de x/C^k élémentaire. Par la suite, nous nous concentrerons donc seulement sur le problème de la réduction additive

3.2 - Réduction d'argument additive

Dans tout ce qui suit, nous supposons que nous disposons d'un algorithme de calcul de g dans un intervalle I de la forme $[-C/2-\varepsilon,+C/2+\varepsilon]$ (on parlera alors de "*réduction symétrique*") ou $[-\varepsilon,C+\varepsilon]$ (on parlera alors de "*réduction positive*"), où $\varepsilon > 0$. On cherche à évaluer $x^* \in I$, et un entier k tels que:

$$(2) \qquad x^* = x - kC$$

Puisque $\varepsilon > 0$, x^* et k ne sont pas définis de manière unique par (2). Cette latitude de choix (encore une redondance !) va permettre de concevoir des algorithmes efficaces.

Notons $\lfloor x \rceil$ l'entier le plus proche de x (ou *un* des entiers en cas d'ambiguité). L'algorithme usuel de réduction d'argument est:

- Calculer $k = \lfloor x/C \rfloor$ (dans le cas d'une réduction positive), ou $k = \lfloor x/C \rceil$ (dans le cas d'une réduction symétrique) par le biais d'une multiplication ou d'une division.
- Calculer $x^* = x - kC$ au moyen d'une multiplication par k suivie d'une soustraction.

Pour de large valeurs de x, ce procédé risque d'être imprécis (lors de la deuxième étape, on soustrait des quantités qui peuvent être très proches). Nous allons maintenant proposer un algorithme appelé *réduction modulaire*, qui permet d'effectuer la réduction d'argument rapidement et avec précision.

4 - L'algorithme de réduction modulaire

4.1 - Réduction en virgule fixe

Supposons que les nombres manipulés sont écrits en virgule fixe et en base 2, et qu'ils sont inférieurs à 2^N. Ces nombres ont une partie entière qui s'écrit sur N bits et une partie fractionnaire qui s'écrit sur p bits. La chaîne de chiffres :

$$x_{N-1}\, x_{N-2}\, x_{N-3} \cdots x_0 . x_{-1}\, x_{-2} \cdots x_{-p} \qquad (x_{-i} = 0,1)$$

représente le nombre $\sum_{i=-p}^{N-1} x_i 2^i$. Notons ν l'entier tel que $2^\nu < C \leq 2^{\nu+1}$.

Pour tout $i \geq \nu$, soit $m_i \in [-C/2, C/2)$ tel que $2^i - m_i$ /C est un entier (on écrira "$m_{-i} \equiv 2^i \bmod C$"). L'algorithme de réduction d'argument modulaire consiste à effectuer les étapes suivantes:

4.1.1. Première réduction

On calcule le nombre:

$$(3) \qquad \begin{cases} r = (x_{N-1}\, m_{N-1}) + (x_{N-2}\, m_{N-2}) + (x_{N-3}\, m_{N-3}) + \ldots + (x_\nu\, m_\nu) \\ \qquad\qquad + x_{\nu-1}\, x_{\nu-2}\, x_{\nu-3} \cdots x_0 . x_{-1}\, x_{-2} \cdots x_{-p} \end{cases}$$

Comme les x_i valent 0 ou 1, ce calcul se fait en effectuant la somme de $N-\nu+1$ termes. Le résultat r de cette première réduction est compris entre $-(N-\nu+2)C/2$ et $+(N-\nu+2)C/2$.

4.1.2. Seconde réduction

Notons r_i le ième chiffre binaire du résultat de la première réduction:

$r = r_l\, r_{l-1}\, r_{l-2} \cdots r_0 . r_{-1}\, r_{-2} \cdots$

où $l = \lfloor \log_2(N-\nu+2) \rfloor$.

Soit $\hat{r}$ le nombre obtenu en tronquant la représentation de r après le $\lfloor -\log_2(\varepsilon) \rfloor$ème bit, c'est-à-dire:

$\hat{r} = r_l\, r_{l-1}\, r_{l-2} \cdots r_0 . r_{-1}\, r_{-2} \cdots r_{\lfloor \log_2(\varepsilon) \rfloor}$

$\hat{r}$ s'écrit sur m bits, où $m = \lfloor -\log_2(N-v+2) \rfloor + \lfloor -\log_2(\varepsilon) \rfloor$ est très petit dans tous les cas pratiques. Si on choisit $k = \lfloor \hat{r} / C \rceil$ (resp. $\lfloor \hat{r} / C \rfloor$) alors r - kC appartient à [-C/2-ε,+C/2+ε] (resp. [-ε, C+ε]), et constitue donc le résultat correct de la réduction symétrique (resp. positive). Comme k peut être déduit de $\hat{r}$, cette seconde réduction s'effectue en cherchant kC dans une table à 2^m bits d'entrée à l'adresse constituée par les bits de $\hat{r}$.

Durant cette réduction, on additionne N-v+1 termes. Si ces termes (à savoir les m_i et la valeur kC de la deuxième réduction) sont représentés en virgule fixe avec q bits fractionnaires, alors la différence entre le résultat du calcul et l'argument réduit exact est majorée par $2^{-q-1}(N-v+1)$.

4.2 - Exemple

Si on désire calculer des sinus et des cosinus de nombres compris entre -2^{20} et 2^{20}, et si l'algorithme de calcul sur les arguments réduits est Cordic [23], [25], l'intervalle de convergence I est [-1.743 ..., +1.743...]. Etant donné que $1.743 > \pi/2$, nous effectuerons une *réduction symétrique*, avec $C = \pi$ et $\varepsilon = +1.743 ... - \pi/2 = 0.172 ... > 2^{-3}$. On obtient alors

- N = 20 et v = 2
- La première réduction se fait en additionnant 19 termes
- $r \in [-10\pi,+10\pi]$, donc, puisque $10\pi < 2^5$, la seconde réduction nécessite l'emploi d'une table à 5+3 = 8 bits d'adresse.
- Pour obtenir l'argument réduit avec p bits fractionnaires significatifs, il faut mémoriser les m_i et les valeurs kC avec p+5 bits fractionnaires.

4.3 - Réduction en virgule flottante

Si la valeur d'entrée x est un nombre virgule flottante :

$x = 0.x_1 x_2 x_3 ... x_n \, 2^{\text{exposant}}$

La réduction d'argument s'effectue comme en virgule fixe. Durant la première réduction, on remplace la somme des termes m_i par celle des termes $m_{\text{exponent-i}}$. La différence essentielle entre cette méthode de réduction et les autres est que durant la réduction, on additionne des nombres (les m_i) qui sont représentés en virgule fixe, et dont l'ordre de grandeur est le même. Ceci rend la réduction d'argument très précise.

5 - Architectures pour la réduction d'argument modulaire

La première réduction consiste en l'addition de N-v+1 nombres. Ce problème est trivialement très semblable à celui de la multiplication de deux nombres (multiplier $x = \sum_{i=0}^{q} x_i 2^i$ par $y = \sum_{j=0}^{q} y_i 2^j$ se réduit à calculer la somme des q+1 termes $y_i 2^j x$). Par conséquent, la plupart des algorithmes et architectures classiques de multiplication (voir par exemple [8], [6], [9], [11], [13], [19], [22], [24], peuvent s'adapter au calcul de la réduction d'argument modulaire. L'architecture présentée =46ig. 1 est dérivée du multiplieur de Braun [8], tandis que celle présentée =46ig. 2 est un arbre de Wallace [24].

Cette similitude entre la réduction modulaire et la multiplication permet d'effectuer les deux opérations avec un même opérateur.

6 - Conclusion

Nous avons montré par quelques exemples que les systèmes redondants de représentation des nombres peuvent avoir des impacts non négligeables sur l'architecture des machines (opérateurs "en ligne", réduction d'argument très rapide à l'aide d'opérateurs très peu différents d'un multiplieur).

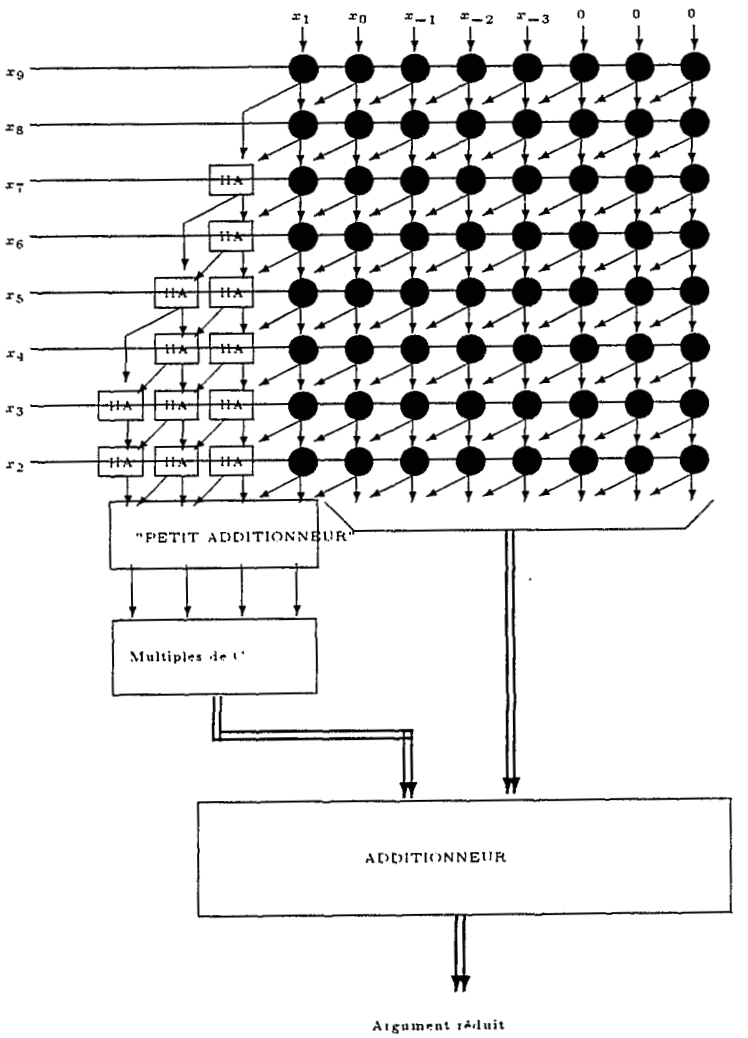

Fig. 1 - Architecture cellulaire de réduction d'argument

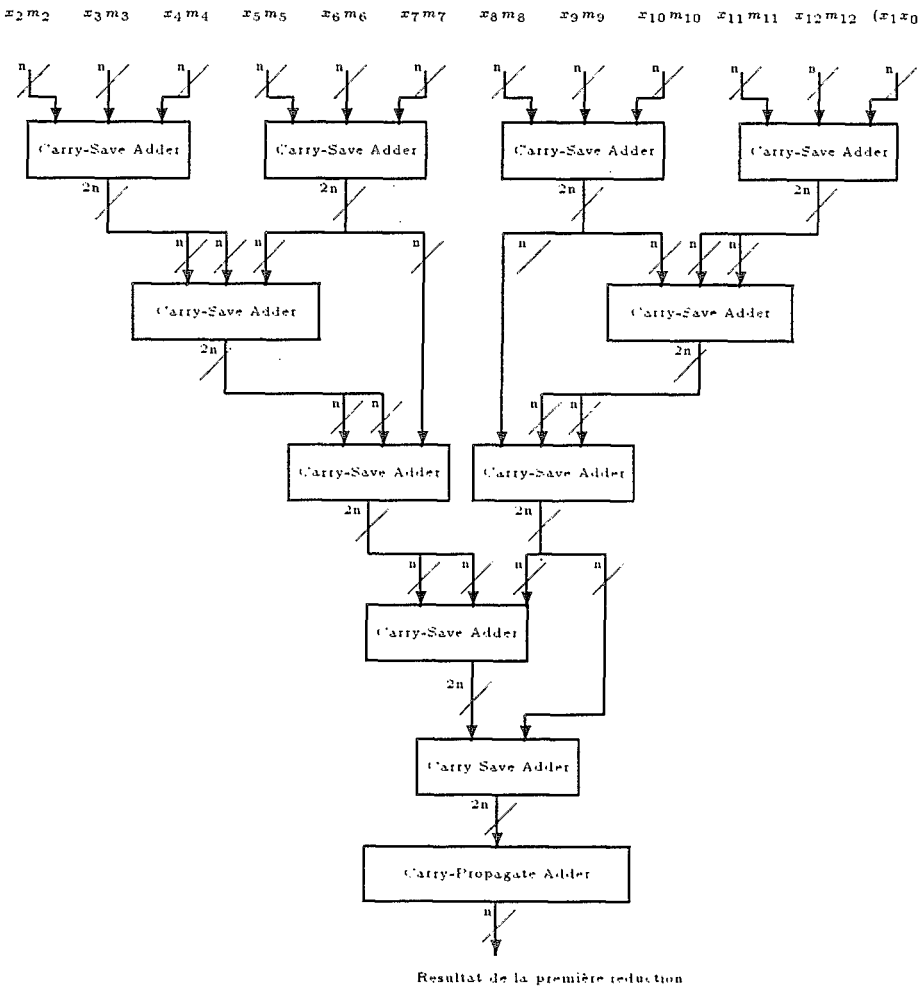

Fig. 2 - Réduction d'argument en temps logarithmique

Références

[1] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on electronic computers*, 10 : pp. 389-400, 1961. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 2, IEEE Computer Society Press Tutorial, 1990.

[2] T. Asada, N. Takagi, and S. Yajima. A hardware algorithm for computing sine and cosine using redundant binary representation. *Systems and computers in Japan*}, 18(8), 1987.

[3] A. Avizienis. Signed-digit number representations for fast parallel arithmetic. *IRE Transactions on Electronic Computers*, 10, 1961.

[4] M.D. Ercegovac, and K.S. Trivedi. *On line algorithms for division and multiplication. IEEE Trans. Comp.*, C-26(7) : 681-687, 1977.

[5] J.C. Bajard. Evaluation de fonctions dans des systèmes redondants d'écriture des nombres. Thèse de doctorat, ENS Lyon, 1993.

[6] C.R. Baugh and B.A. Wooley. A two's complement parallel array multiplication algorithm. *IEEE Transactions on Computers*, C-22 : 1045-1047, 1973. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[7] A.D. Booth. A signed binary multiplication technique. *Quarterly journal of mechanics and applied mathematics*, 4(2) : 236-240, 1951. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[8] E.L. Braun. *Digital computer design*. New York academic, 1963.

[9] P.R. Cappello and K. Steiglitz. A vlsi layout for a pipelined Dadda multiplier. *ACM Transactions on Computer Systems*, 1(2) : 157-174, May 1983. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 2, IEEE Computer Society Press Tutorial, 1990.

[10] W. Cody and W. Waite. *Software Manual for the Elementary Functions*. Prentice-Hall Inc, 1980.

[11] L. Dadda. Some schemes for parallel multipliers. *Alta Frequenza*, 34 : 349-356, March 1965. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[12] M.D. Ercegovac. Radix 16 evaluation of certain elementary functions. *IEEE Transactions on Computers*, C-22(6), June 1973. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[13] Y. Harata, Y. Nakamura, H. Nagase, M. Takigawa, and N. Takagi. A high-speed multiplier using a redundant binary adder tree. *IEEE journal of solid-state circuits*, SC 22(1) : 28-34, February 1987. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 2, IEEE Computer Society Press Tutorial, 1990.

[14] J.F. Hart. *Computer Approximations*. Wiley, 1968.

[15] G.H. Haviland and A.A. Tuszinsky. A cordic arithmetic processor chip. *IEEE Transactions on Computers*, C-29(2), February 1980.

[16] K. Hwang. *Computer Arithmetic Principles, Architecture and design*. Wiley & Sons Inc, 1979.

[17] B. De Lugish. *A Class of Algorithms for Automatic Evaluation of Functions and Computations in a Digital Computer*. PhD thesis, Dept. of Computer Science, University of Illinois, Urbana, 1970.

[18] J.E. Meggitt. Pseudo division and pseudo multiplication processes. *IBM Journal of Research and Development*, 6 : 210-226, 1962.

[19] S. Nakamura. Algorithms for iterative array multiplication. *IEEE Transactions on Computers*, C-35(8), August 1986.

[20] E. Remes. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *C.R. Acad. Sci. Paris*, 198, 1934.

[21] W.H. Specker. A class of algorithms for ln(x), exp(x), sin(x), cos(x), $\tan^{-1}(x)$ and $\cot^{-1}(x)$. *IEEE Transactions on Electronic Computers*, EC-14, 1965. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[22] N. Takagi, H. Yasukura, and S. Yajima. High speed multiplication algorithm with a redundant binary addition tree. *IEEE Transactions on Computers*, C-34(9), September 1985.

[23] J. Volder. The cordic computing technique. *IRE Transactions on Electronic Computers*, 1959. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[24] C.S. Wallace. A suggestion for a fast multiplier. *IEEE Transactions on Electronic Computers*, pages 14-17, February 1964. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.

[25] J. Walther. A unified algorithm for elementary functions. In *Joint Computer Conference Proceedings*, 1971. Reprinted in E.E. Swartzlander, Computer Arithmetic, Vol. 1, IEEE Computer Society Press Tutorial, 1990.