Sparse-Coefficient Polynomial Approximations for Hardware Implementations

Nicolas Brisebarre[a,b], Jean-Michel Muller[a] and Arnaud Tisserand[a]
[a]Arénaire project (CNRS–ENS Lyon–INRIA–UCBL)
Laboratoire de l'Informatique du Parallélisme (LIP)
Ecole Normale Supérieure de Lyon (ENS Lyon)
46 allée d'Italie.
F–69364 Lyon Cedex 07, France
E-mail: {firstame.lastname}@ens-lyon.fr

[b]LArAl
Université J. Monnet
23, rue du Dr P. Michelon
F–42023 St-Étienne Cedex, France

***Abstract*— This paper presents a method for automatic generation of best polynomial approximations dedicated to hardware implementation. The generated polynomial approximations lead to high-speed and small hardware operators because of the use of sparse coefficients (i.e. we include fixed strings of zeros in the binary representation of the coefficients). Two different solutions have been investigated for the generation of the sparse-coefficient polynomial approximations. Our first results show up to 47% smaller coefficients compared to standard minimax approximations for comparable accuracy.**

INTRODUCTION

Polynomial approximations are widely used in digital systems. For instance, elementary functions (i.e. sine, cosine, exponential, logarithm, arctangent, *etc*) are often evaluated using polynomials. A presentation of elementary function algorithms that use polynomial approximation can be found in [1]. In some digital signal processing applications, such as frequency demodulation, low degree polynomials are often used for evaluating reciprocals. Other algebraic functions, such as square root or square root reciprocal can be efficiently approximated using polynomials.

In hardware implementation of polynomial approximations, the size of the multipliers is a major concern. Several solutions have been investigated to limit their size. For instance, in [2] a method based on argument reduction and series expansions, is used for the evaluation of reciprocals, square roots, reciprocal square roots, and some elementary functions using small multipliers and tables. A method based on a small table and a modified multiplication is presented in [3]. In [4] a method based on a degree-2 polynomial and a specialized squaring unit is proposed. That solution leads to 50% area savings compared to standard methods. Table and add methods, such as the multipartite table method [5], [6], [7], have been introduced to avoid the use of multipliers. Those methods are limited to moderate precision and use large silicon area for each function being evaluated.

In this work we focus on "small" multiplications with sparse coefficients (i.e. the multiplier operand can be written using fixed strings of bits stuck at 0) for the polynomial approximations. The sparse coefficients allow to replace the complete reduction tree of the multipliers by smaller ones. This leads to smaller and faster circuits.

This paper is organized as follows. A method [8] for generating polynomial approximation with exactly representable coefficients is summarized in Section I. The two proposed methods for generating sparse-coefficient polynomial approximations are presented in Section II. The first results of the proposed method are presented in Section III.

I. POLYNOMIAL APPROXIMATION WITH EXACTLY REPRESENTABLE COEFFICIENTS

When implementing a given function f on a given real compact interval $[a,b]$, one usually uses polynomial approximations, such as *minimax approximations* which minimize the distance

$$||p-f||_{\infty,[a,b]} = \sup_{a\leq x\leq b} |p(x)-f(x)|,$$

where $p \in \mathbb{R}_d[X]$, the vector space of polynomials with real coefficients and degree at most d, d being a given integer. Minimax approximations, that can be computed thanks to an algorithm due to Remez [9], have a major drawback: in most cases, their coefficients are not exactly representable with a finite number of bits. In [8], the authors propose an efficient method for computing a polynomial which minimizes the distance $||p-f||_{\infty,[a,b]}$ among the polynomials $p \in \mathbb{R}_d[X]$ that fulfill some given constraints on the size in bits of their coefficients.

More precisely, let $m_0,\ldots,m_d$ be a finite sequence of integers. Let

$$\mathcal{P}_d^m = \Big\{ q(x); a_i \in \mathbb{Z}, \forall i = 0,\ldots,d,$$

0-7803-8622-1/04/$20.00 ©2004 IEEE

where

$$q(x) = \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}}x + \cdots + \frac{a_d}{2^{m_d}}x^d \Big\}.$$

We look for for a polynomial $p \in \mathcal{P}_d^m$ which minimizes $||q - f||_{\infty,[a,b]}$ where q is any polynomial of $\mathcal{P}_d^m$. The key idea is to construct a rational polytope $\mathfrak{P}$ of $\mathbb{R}^{d+1}$, which the numerators of the coefficients of p belong to, such that $\mathfrak{P}$ contains a number as small as possible of points of $\mathbb{Z}^{d+1}$. Once this polytope is built, we perform an exhaustive search by computing the norms

$$\left\| \frac{a_0}{2^{m_0}} + \frac{a_1}{2^{m_1}}x + \ldots \frac{a_d}{2^{m_d}}x^d - f \right\|_{\infty,[a,b]}$$

with $(a_0, a_1, \ldots, a_d) \in \mathfrak{P} \cap \mathbb{Z}^{d+1}$. We use linear programming tools to efficiently run through all the points of $\mathfrak{P} \cap \mathbb{Z}^{d+1}$.

This method is flexible since it also applies if some other constraints (such as requiring some coefficients to be equal to some predefined constants, or minimizing relative error instead of absolute error) are required.

This method is implemented in the *machine-efficient polynomial* library (`MEPLib` [10]) by the authors (C library under LGPL license). This library generates polynomial approximations with exactly representable coefficients with respect to arbitrary precision requirements.

II. Sparse-Coefficient Polynomial Approximations

In this work, we deal with polynomial approximations in $\mathcal{P}_d^m$ with sparse coefficients. This means that in all the coefficients, there are several bits fixed to 0 such as coefficient examples presented in Figure 1. Our method generates the best possible approximations of f among the polynomials of $\mathcal{P}_d^m$ with sparse coefficients. The presented method ensures that all generated polynomials p are such that $||p - f||_{\infty,[a,b]}$ is minimal with p in $\mathcal{P}_d^m$ with sparse coefficients.

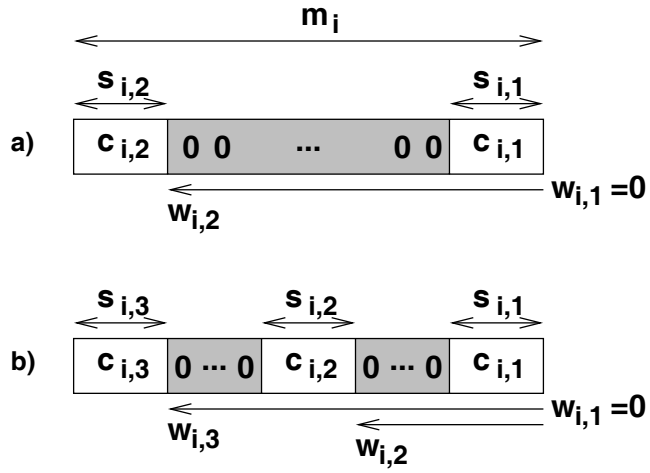

Fig. 1. Kinds of target coefficients. The grey areas are only composed of 0s. Cases a and b show respectively 2 and 3 chunks coefficients.

Each numerator a_i of the polynomial coefficient is a m_i-bit integer (with $i \in \{0, 1, \ldots, d\}$). The sparse-coefficient formulation splits the value a_i into k chunks $c_{i,j}$. The chunks are small signed integers. The size of the different chunks are the values $s_{i,j}$ with $j \in \{1, \ldots, k\}$. The weight of the chunk $c_{i,j}$ is the value $2^{w_{i,j}}$ as illustrated on Figure 1. The weight of the least significant chunk is $w_{i,1} = 0$. The value of a coefficient a_i is then:

$$a_i = \sum_{j=1}^{k} c_{i,j} \times 2^{w_{i,j}}.$$

The goal of this sparse-coefficient formulation is to find polynomials with all coefficients having a number of nonzero bits smaller than $\sum_{i=0}^{d} m_i$.

$$\sum_{i=0}^{d}\sum_{j=1}^{k} s_{i,j} << \sum_{i=0}^{d} m_i.$$

We have to modify the method presented in Section I. We investigated two different solutions. The first one is based on filtering the results from Section I. The second one is based on formulation of a new polytope.

A. Filtering

The results produced by `MEPLib` can be filtered. For each candidate polynomial, the filter verifies that all its coefficients can be represented using the target sparse format.

First of all, the list of the possible coefficients is built from the specifications of the target format. Let us consider a small example. On a 7-bit format, we consider 3 chunks with the following sizes and weights:

chunk	$c_{i,3}$	$c_{i,2}$	$c_{i,1}$
size	$s_{i,3} = 1$	$s_{i,2} = 1$	$s_{i,1} = 1$
weight	$w_{i,j} = 6$	$w_{i,j} = 3$	$w_{i,j} = 0$

Based on this format the list of the possible coefficients greater than 0 is:

$$0, 1, 7, 8, 9, 55, 56, 57, 63, 64, 65, 71, 72, 73.$$

The algorithm scans for each candidate polynomial if all its coefficient are in the list. If yes the polynomial is a sparse-coefficient polynomial. If not, the polynomial is discarded.

Among all sparse-coefficient polynomials filtered out by our method, the result is the polynomial with the smallest approximation error (this sorting is done using Maple).

B. New Polytope Formulation

Some additional constraints are required to handle this sparse-coefficient formulation. Indeed, we now have to build a rational polytope of $\mathbb{R}^{k(d+1)}$ which the $c_{i,j}$ belong to. In this new polytope formulation the $c_{i,j}$ are limited to $s_{i,j}$-bit integers. I.e., for all possible values of i and j, we have:

$$|c_{i,j}| \leq 2^{s_{i,j}} - 1.$$

The algorithm incrementally scans values for k and $s_{i,j}$ with respect to an arbitrary accuracy target. At each iteration the accuracy of the best generated polynomials is verified. In case

of a not precise enough result the values of k and $s_{i,j}$ are incremented.

Some other constraints should be verified (non-overlapping of the chunks, non-overflow of the chunk with respect to the m_i-bit words...). Those constraints are verified before the first production of polytope to avoid a memory increase. Those constraints are not useful to the polytope scanning but only to verify that the polytope represents a well formed polynomial.

C. Comparison of the Two Solutions

The two methods have been implemented. Our first results show that the current implementation of the second one (the new polytope formulation) leads to polytopes defined by higher dimension matrices and requires significantly more memory than the first one (filtering).

III. Examples

All the following examples have been tested on a 15-bit format and degree-2 polynomials. The target format is composed of 4 chunks (chunk number 4 is the most significant, and chunk number 1 is the least significant). All the following examples have been tested using the first method (filtering).

For each function, we compare the accuracy of the best sparse-coefficient polynomial with the accuracy of the minimax polynomial given Remez's algorithm (computed using Maple). This gives an idea of the "degradation" of the accuracy compared to the best theoretical polynomial (but with not representable coefficients).

A. Cosine Function

The cos function is approximated on the interval $[0, \pi/4]$. The upper bound $\pi/4$ is approximated using the value $351/452$.

The Remez algorithm gives the best polynomial (with real coefficients) with 10.01 bits of accuracy (this is the reference accuracy).

Our method returns the best sparse-coefficient polynomial with the following format:

chunk	4	3	2	1
size	2	3	2	1
weight	14	8	5	0

For this target format (illustrated on Figure 2), our method returns only 1 polynomial with an approximation error of 9.93 bits (which is very close to the best possible one). This leads to a 47% size reduction of the coefficient useful width. The corresponding best sparse-coefficient polynomial is:

$$p = \frac{32799}{32768} - \frac{609}{32768}x - \frac{14881}{32768}x^2.$$

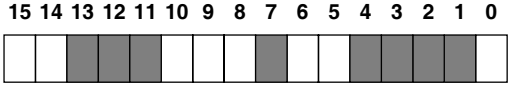

Fig. 2. Target format for cos function.

If we allow larger chunks, the accuracy improves but only in a very small amount. For instance, with respect to the target format:

chunk	4	3	2	1
size	3	3	3	3
weight	13	8	4	0

our method returns 134 sparse-coefficient polynomials. The best one has an approximation error up to 9.96 bits. This leads to a 20% size reduction of the coefficient useful width. The corresponding best sparse-coefficient polynomial is:

$$p = \frac{32800}{32768} - \frac{607}{32768}x - \frac{14887}{32768}x^2.$$

B. Sine Function

The sin function is approximated on the interval $[0, \pi/4]$.

The Remez algorithm gives the best polynomial with 8.76 bits of accuracy.

Our method returns the best sparse-coefficient polynomial with the following format:

chunk	4	3	2	1
size	1	2	4	4
weight	15	11	5	0

For this target format (illustrated on Figure 3), our method returns 62 polynomials with an approximation error up to 8.75 bits. This leads to a 27% size reduction of the coefficient useful width. The corresponding best sparse-coefficient polynomial is:

$$p = -\frac{75}{32768} + \frac{34538}{32768}x - \frac{6169}{32768}x^2.$$

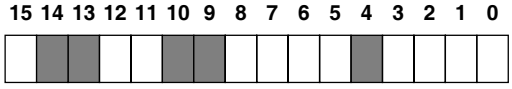

Fig. 3. Target format for sin function.

C. Exponential Function

The function exp is approximated on the interval $[0, 1/2]$.

The Remez algorithm gives the best polynomial with 10.21 bits of accuracy.

Our method returns the best sparse-coefficient polynomial with the following format:

chunk	4	3	2	1
size	2	3	4	3
weight	14	10	5	0

For this target format (illustrated on Figure 4), our method returns 1 polynomial with an approximation error of 10.17 bits (which is very close to the best possible one). This leads to a 20% size reduction of the coefficient useful width. The corresponding best sparse-coefficient polynomial is:

$$p = \frac{32793}{32768} + \frac{31836}{32768}x + \frac{21146}{32768}x^2.$$

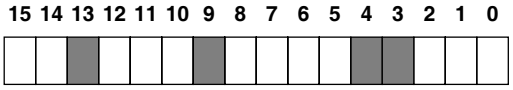

Fig. 4. Target format for $\exp$ function.

D. Reciprocal Square Root Function

The function $\frac{1}{\sqrt{1+x}}$ is approximated on the interval $[0, 1/4]$. The Remez algorithm gives the best polynomial (with real coefficients) with 13.25 bits of accuracy.

Our method returns the best sparse-coefficient polynomial with the following format:

chunk	4	3	2	1
size	3	1	3	2
weight	13	10	6	0

For this target format (illustrated on Figure 5), our method returns 1 polynomial with an approximation error of 13.20 bits (which is very close to the best possible one). This leads to a 40% size reduction of the coefficient useful width. The corresponding best sparse-coefficient polynomial is:

$$p = \frac{32765}{32768} - \frac{16131}{32768}x + \frac{9277}{32768}x^2.$$

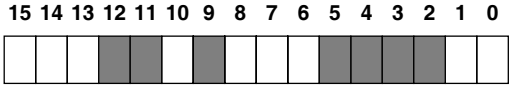

Fig. 5. Target format for $\frac{1}{\sqrt{1+x}}$ function.

IV. Conclusion and Future Prospects

Solutions for generating sparse-coefficient polynomial approximations have been investigated. The proposed methods produce coefficients with up to 47% of bits stuck at 0 with fixed positions (with an average value around 30%). These results will lead to smaller and faster circuits.

In a near future, we plan to work on the polytope scanning algorithms in order to reduce the memory requirements. We also plan to generate VHDL descriptions of polynomial evaluation circuits using our method.

Acknowledgment

The authors would like to thank the "*Ministère Français de la Recherche*" for his financial support (grant ACI-NIM-2004-212 "*ACI Nouvelles Interfaces des Mathématiques*").

References

[1] J.-M. Muller. *Elementary Functions: Algorithms and Implementation*. Birkhäuser, Boston, 1997.

[2] M.D. Ercegovac, T. Lang, J.-M. Muller, and A. Tisserand. Reciprocation, square root, inverse square root, and some elementary functions using small multipliers. *IEEE Transactions on Computers*, 49(7):627–637, July 2000.

[3] N. Takagi. Powering by a table look-up and a multiplication with operand modification. *IEEE Transactions on Computers*, 47(11):1216–1222, 1998.

[4] J. A. Pineiro, J. D. Bruguera, and J.-M. Muller. Faithful powering computation using table look-up and a fused accumulation tree. In *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, pages 40–47. IEEE Computer Society, 2001.

[5] D. A. Sunderland, R. A. Strauch, S. S. Wharfield, H. T. Peterson, and C. R. Role. CMOS/SOS frequency synthesizer LSI circuit for spread spectrum communications. *IEEE Journal of Solid State Circuit*, 19(4):497–506, August 1984.

[6] M. Schulte and J. Stine. Approximating elementary functions with symmetric bipartite tables. *IEEE Transactions on Computers*, 48(8):842–847, August 1999.

[7] F. de Dinechin and A. Tisserand. Some improvements on multipartite tables methods. In N. Burgess and L. Ciminiera, editors, *15th International Symposium on Computer Arithmetic ARITH15*, pages 128–135, Vail, Colorado, June 2001. IEEE.

[8] N. Brisebarre, J.-M. Muller, and A.Tisserand. Computing machine-efficient polynomial approximations. *submitted*, 2004.

[9] E. Remes. Sur un procédé convergent d'approximations successives pour déterminer les polynômes d'approximation. *C.R. Acad. Sci. Paris*, 198:2063–2065, 1934.

[10] N. Brisebarre, F. Hennecart, J.-M. Muller, A. Tisserand, and S. Torres. MEPLib. http://lipforge.ens-lyon.fr/, 2004.