Hardware operators for function evaluation using sparse-coefficient polynomials

Jean-Michel Muller, Arnaud Tisserand, Serge Torres, Nicolas Brisebarre

HARDWARE OPERATORS FOR FUNCTION EVALUATION USING SPARSE-COEFFICIENT POLYNOMIALS

N. BRISEBARRE, J.-M. MULLER, A. TISSERAND AND S. TORRES

Abstract: This article presents dedicated hardware arithmetic operators for function evaluation. The proposed solution uses polynomial approximations with sparse coefficients which leads to efficient hardware implementations. Up to 2× faster and 8× smaller operators are reported compared to standard implementations.

Introduction: Polynomial approximations are widely used in digital systems for function evaluation (e.g. $1/x$, $\sqrt{x}$, sin, cos, exp, log). In some digital signal processing applications, such as frequency demodulation, low-degree polynomials are often used for evaluating reciprocals. In hardware implementation of polynomial approximations, the size of the multipliers is a major concern. Several solutions have been investigated to limit their size: argument reduction and series expansions in [1], small table and a modified multiplication in [2], or the multipartite tables method [3, 4].

In this work we focus on polynomial approximations with sparse coefficients (i.e. the multiplier operand can be written so that it contains predefined strings of bits stuck at 0). Sparse coefficients allow us to replace the complete reduction tree of the multipliers by smaller ones. This letter is an improved and extended version of the paper presented in [5] and uses a new recursive coefficient filtering.

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

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

where $p \in \mathbb{R}_d[X]$, the set of polynomials with real coefficients and degree at most d. Minimax approximations, that can be computed thanks to an algorithm due

to Remez have a major drawback: in most cases, their coefficients are not exactly representable using a finite number of bits. In [6], 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 format of the coefficients. The result polynomials q are of the form:

$$q(x) = \frac{q_0}{2^{m_0}} + \frac{q_1}{2^{m_1}}x + \frac{q_2}{2^{m_2}}x^2 + \ldots + \frac{q_d}{2^{m_d}}x^d.$$

The degree d and the integer sequence $m_0, \ldots, m_d$ are input parameters of the method. The coefficients are such that $q_i \in \mathbb{Z}$. The method presented in [6] provides result polynomials q such that $||q - f||_{\infty,[a,b]}$ is minimal among the polynomials that fulfill the constraints. Those polynomials can be represented as the integer points of a polytope (cf. [6]) and efficient scanning of all the polytope points is performed using linear programming tools.

Here, we look for polynomials with sparse coefficients, i.e. there are several bits predefined to 0 as illustrated in Figure 1. Each coefficient $q_i/2^{m_i}$ is decomposed into k chunks $c_{i,j}$ with $j \in \{1, \ldots, k\}$. The chunks are small signed s_j-bit integers. The number of useful bits in coefficients is $S = \sum_{j=1}^{k} s_j$. The weight of the chunk $c_{i,j}$ is the value 2^{w_j} as illustrated on Figure 1. Notice that the size and the weight of the chunks are the same for all the coefficients. The value of a coefficient numerator is then $q_i = \sum_{j=1}^{k} c_{i,j} \times 2^{w_j}$. For a given number of chunks k, there are several possible sparse decompositions (the values s_j and w_j) with non-overlapping chunks. Polynomials with sparse coefficients are interesting for circuit implementation as soon as the number of nonzero bits is much smaller than the total format width. This corresponds to:

$$\sum_{i=0}^{d}\sum_{j=1}^{k} s_j = (d+1) \times S \;<<\; \sum_{i=0}^{d} m_i.$$

Finding Sparse-Coefficient Polynomials: In [5], we proposed two methods to find such sparse-coefficient polynomials. The first method is a very simple filter applied

to results from [6]. For each candidate polynomial, the filter verifies that all its coefficients can be represented using the target sparse format. This solution is limited to degree-2 polynomials in practice (huge memory requirements and computation times).

The second proposed solution uses a new polytope formulation. Instead of having a $d+1$-dimensional polytope, each coefficient introduces k chunks, then the dimension of the polytope is $k \times (d+1)$. Using this solution, up to degree-3 polynomials can be reached.

Our new solution is based on a recursive coefficient filtering strategy. As in the first solution proposed in [5], we build a $d+1$-dimensional polytope but we do not scan all its integer points. For each possible sparse decomposition (i.e. the values s_j and w_j), we scan only the integer points that fit this decomposition. This last scanning is recursively performed: if coefficient q_i can be decomposed into the sparse format then the next dimension corresponding to q_{i+1} is checked. If not, we scan another decomposition. This new solution leads to efficient programs up to degree-5 polynomials and 6 chunks.

FPGA Implementation Results: Implementations have been done on Xilinx XC3S400-5 FPGAs using ISE7.1i tools. The reported area (number of slices) and delay (ns) are post-P&R values with standard effort.

Approximations to the cosine function on $[0, \pi/4]$ with 24-bit format for input and output, with a degree-4 polynomial have been implemented. The number of chunks is $2 \leq k \leq 6$ and number of useful bits in the coefficients is $7 \leq S \leq 12$. The obtained operators are compared to the reference implementation with 24-bit full-width standard multiplication. Figure 2 summarizes the results while Table 1 presents the decompositions for which we get an absolute accuracy larger than 18.5 bits. The column named "acc." is the accuracy expressed in number of correct bits. The last line of this table presents the result for the reference solution.

The results show that significant area and speed improvements are possible using our method. For instance, a 19-bit accuracy for the cosine function on $[0, \pi/4]$ can be reached using 6 chunks and only 12 useful bits (S) compared to the 24-bit format. The corresponding decomposition is defined by $s_{6\ldots1} = (3, 2, 2, 2, 1, 2)$ and $w_{6\ldots1} = (4, 9, 12, 15, 18, 22)$. It corresponds to an operator that requires 39 slices and a delay of 7.3 ns while the standard solution (last line of Table 1) requires 332 slices and 16.2 ns. Hence, our method provides an 8× area reduction and a 2× speed improvement. Notice that due to the huge impact of multipliers on the circuit area, a 2× reduction on the number of useful bits leads to an 8× area reduction.

Conclusion: This letter presents a new solution for the design of hardware operators dedicated to function evaluation. It uses polynomial approximations with sparse coefficients. The proposed solution improves the results from [5] using an efficient recursive filtering of the polynomial coefficients that are viewed as points of rational polytopes. The results obtained using this solution lead to up to 2× faster and 8× smaller results have been reported compared to standard implementations.

Acknowledgment: This work was partly supported by the French Ministry Grant ACI "New interfaces of mathematics".

Author's affiliations:

Nicolas Brisebarre (*LaMUSE, Université J. Monnet, 23 rue du Dr P. Michelon, 42023 St-Etienne, France*)

Jean-Michel Muller and Serge Torres (*LIP, CNRS-ENSL-INRIA-UCBL, 46 allée d'Italie, 69364 Lyon, France*)

Corresponding author: Arnaud Tisserand (*LIRMM, CNRS-UM2, 161 rue Ada, 34392 Montpellier, France*)

Email: arnaud.tisserand@lirmm.fr

References

[1] 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.

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

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

[4] F. de Dinechin and A. Tisserand. Multipartite table methods. *IEEE Transactions on Computers*, 54(3):319–330, March 2005.

[5] N. Brisebarre, J.-M. Muller, and A. Tisserand. Sparse-coefficient polynomial approximations for hardware implementations. In *Proc. 38th Asilomar Conference on Signals, Systems and Computers*, pages 532–535, Pacific Grove, California, U.S.A., November 2004.

[6] N. Brisebarre, J.-M. Muller, and A. Tisserand. Computing machine-efficient polynomial approximations. *ACM Transactions on Mathematical Software*, 32(2):236–256, June 2006.

Figure captions:

Fig. 1: 2 and 3-Chunk Decompositions

Fig. 2: FPGA Implementation results for cos on $[0, \pi/4]$

Table captions:

Tab. 1: Results for cos on $[0, \pi/4]$

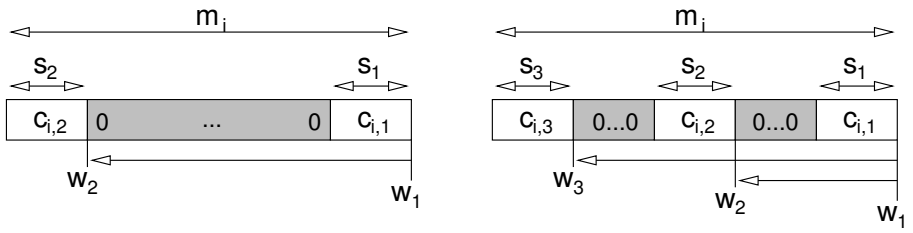

FIGURE 1

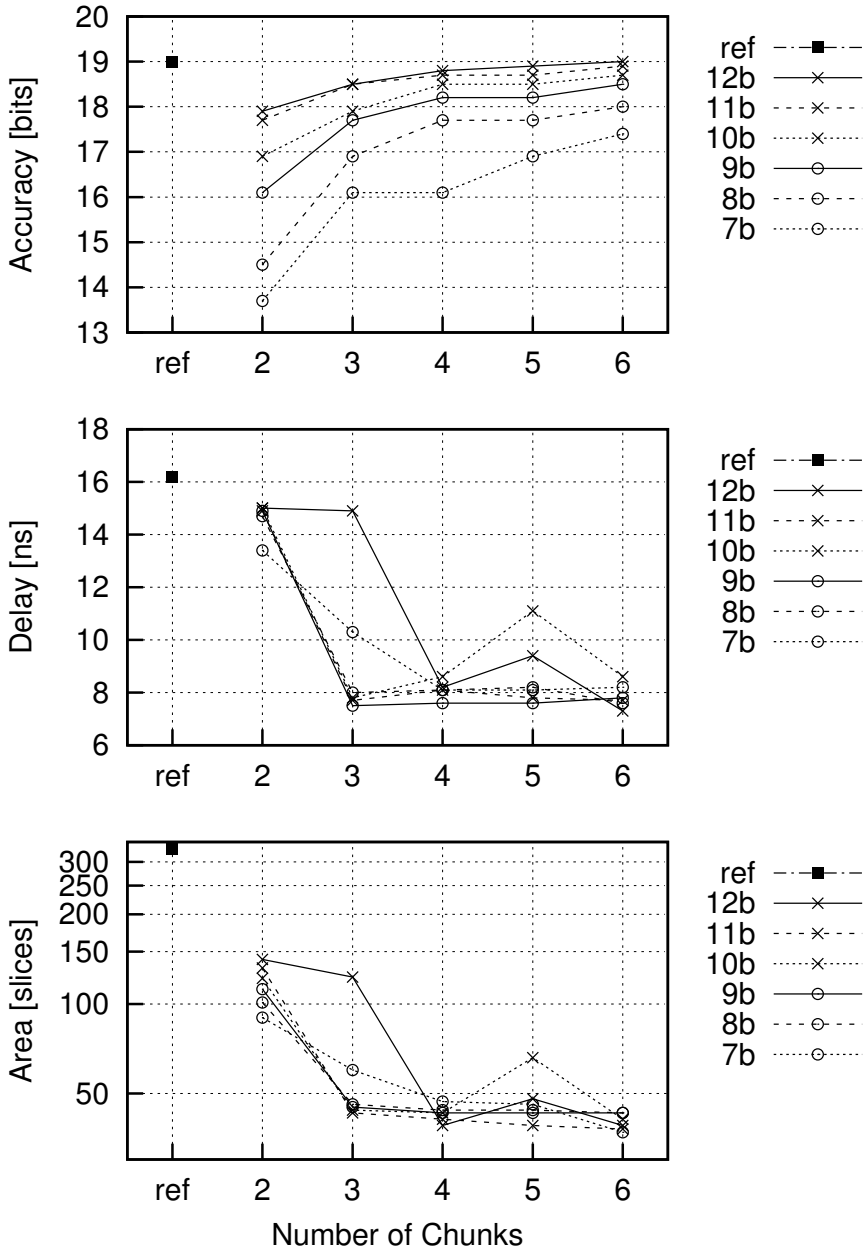
Accuracy [bits]
20
19
18
17
16
15
14
13
ref
2
3
4
5
6
ref
12b
11b
10b
9b
8b
7b
Delay [ns]
18
16
14
12
10
8
6
ref
2
3
4
5
6
ref
12b
11b
10b
9b
8b
7b
Area [slices]
300
250
200
150
100
50
ref
2
3
4
5
6
Number of Chunks
ref
12b
11b
10b
9b
8b
7b

FIGURE 2

S	k	acc.	s_j	w_j	area	delay
9	6	18.5	1,2,1,2,1,2	8,10,13,15,18,22	43	7.8
10	5	18.5	1,2,1,4,2	8,10,13,15,22	66	11.1
10	6	18.7	1,3,2,1,1,2	6,8,13,16,18,22	41	8.6
11	3	18.5	8,1,2	8,18,22	43	7.7
11	4	18.7	5,3,1,2	6,13,18,22	41	8.1
11	5	18.7	1,5,2,1,2	4,9,15,18,22	39	7.8
11	6	18.9	2,3,2,1,1,2	3,8,13,16,18,22	38	7.7
12	3	18.5	1,9,2	8,10,22	123	14.9
12	4	18.8	2,7,1,2,	4,9,18,22	39	8.2
12	5	18.9	2,3,2,3,2	3,8,13,16,22	48	9.4
12	6	19.0	3,2,2,2,1,2	4,9,12,15,18,22	39	7.3
24	-	19.0	24	0	332	16.2

TABLE 1