A new scheme for table-based evaluation of functions

David Defour, Dinechin, Florent De, Jean-Michel Muller

Laboratoire de l'Informatique du Parallélisme

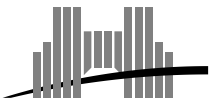
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

A new scheme for table-based evaluation of functions

David Defour, Florent de Dinechin, Jean-Michel Muller

Novembre 2002

Research Report N° 2002-45



École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : lip@ens-lyon.fr

A new scheme for table-based evaluation of functions

David Defour, Florent de Dinechin, Jean-Michel Muller

Novembre 2002

Abstract

This paper presents a new scheme for the hardware evaluation of functions in fixed-point format, for precisions up to 30 bits. This scheme yields an architecture made of four look-up tables, a multi-operand adder, and two small multipliers. This new method is evaluated and compared with other published methods.

Keywords: Bipartite tables, table-and-addition methods, multipartite, Elementary functions

Résumé

Cet article présente un nouvel algorithme pour l'évaluation de fonctions en virgule fixe, pour des précisions allant jusqu'à 30 bits. Cet algorithme est basé sur une architecture composée de 4 lectures de tables, un additionneur multi-opérandes et de deux petits multiplieurs. Cette nouvelle méthode est évaluée et comparée avec de précédents algorithmes.

Mots-clés: Tables biparties, méthodes à base de tables et d'additions, tables multiparties, Fonctions élémentaires

1 Introduction

Table-based methods for the hardware evaluation of functions have been the subject of much recent attention [1, 3, 4, 7, 9]. They allow to compute commonly used functions with low accuracy (currently up to 24 bits) with significantly lower hardware cost than that of a straightforward table implementation, while being faster than shift-and-add algorithms *à la* CORDIC or polynomial approximations. They are particularly useful in digital signal or image processing, and also for providing initial seed values to iterative methods such as the Newton-Raphson algorithms for division and square root [6] which are commonly used in the floating-point units of current processors.

This paper introduces a new table-based scheme for computing elementary functions. While the bi- or multi-partite methods are based on a first-order approximation of the function, this scheme uses fifth-order terms of the Taylor expansion, which should allow better accuracy with the same amount of hardware. The tradeoff is that this scheme involves small multipliers. We explore a range of implementation tradeoffs and compare them with previously published architectures for the evaluation of functions.

This paper is organized as follows: In Section 2, we examine some table-and-addition methods. Section 3 presents the principle and gives a bound on the mathematical approximation error of the proposed method. Section 4 discusses implementation issues such as rounding, and gives actual data for some functions which are compared to previously published results. Section 5 discusses future research directions.

2 Previous Work

2.1 Multipartite method

First presented by Das Sarma and Matula [1] in the specific case of the reciprocal function, this method has been generalised independently by Schulte and Stine [9] and Muller [7], and finally by Dinechin and Tisserand [3]. It consists in a piecewise first-order approximation to the function where the multiplications are avoided by storing approximated products in specific tables. It typically allows faithfully rounded functions for 16 bits of precision in a few Kbytes of tables and a small multi-operand adder.

2.2 Other table-and-addition methods

Hassler and Tagaki [4] decompose a polynomial approximation to the function as a sum of products of bits of the input word. The set of these partial products is heuristically decomposed into subsets which are stored in tables, and added in a multi-operand adder. Another method is given by Wong and Goto [10], where additions are performed before and after the table look-ups. Although these two methods consider higher-order terms, they have been shown to produce larger and slower architectures than the generalized multipartite method, and they are cited for reference only.

2.3 Higher-order methods

The exponential size of table-based architectures renders them impractical for precisions higher than 24 bits. Bruguera, Piñeiro and Muller have therefore presented an architecture based on a second-order approximation using a multiplier and a dedicated squarer unit [8]. Liddicoat and Flynn [5] similarly propose an architecture for computing the reciprocal (whose Taylor expansion

involves only unit coefficients) using several x^k units in parallel, followed by a multi-operand adder.

The method presented here is intermediate between table-and-addition methods and multiplication-based methods.

3 A table-based method with small multipliers

3.1 Mathematical approximation

Throughout this paper we want to evaluate $f(x)$. We assume $x \in [0,1[$ is an n-bit fixed point format with $n = 4k + p$. We can thus write $x = x_0 + x_1 2^{-k} + x_2 2^{-2k} + x_3 2^{-3k} + x_4 2^{-4k}$, where the x_i's are k-bit numbers belonging to $[0,1[$, except for x_4 that is a p-bit numbers such that $p < k$.

We have:

$x =$ | x_0 | x_1 | x_2 | x_3 | x_4 |

The method consists in writing an order-5 Taylor expansion of f at x_0, and keeping only the terms relevant to the target accuracy that is 2^{-4k+p}. We get:

$$\begin{array}{rcll} f(x) & = & f(x_0) & (T_0) \\ & + & [x-x_0]f'(x_0) & (T_1) \\ & + & \frac{1}{2}[x-x_0]^2 f''(x_0) & (T_2) \\ & + & \frac{1}{6}[x-x_0]^3 f'''(x_0) & (T_3) \\ & + & \frac{1}{24}[x-x_0]^4 f^{(4)}(x_0) & (T_4) \\ & + & \frac{1}{120}[x-x_0]^5 f^{(5)}(x_0) & (T_5) \\ & + & \epsilon_1 & \end{array} \tag{1}$$

The error committed is $\epsilon_1 = \frac{1}{720}([x_1 2^{-k} + x_2 2^{-2k} + x_3 2^{-3k} + x_4 2^{-4k}]^6) f^{(6)}(\zeta_1) < \frac{1}{720} 2^{-6k} \max |f^{(6)}|$

In equation (1), we expand the term $(T_1) = [x - x_0]f'(x_0)$ as follow:

$$\begin{array}{rcl} (T_1) & = & [x-x_0]f'(x_0) \\ & = & x_1 2^{-k} f'(x_0) + \\ & & x_2 2^{-2k} f'(x_0) + \\ & & x_3 2^{-3k} f'(x_0) + \\ & & x_4 2^{-4k} f'(x_0) \end{array}$$

Now let us focus on terms $(T_2) = \frac{1}{2}[x-x_0]^2 f''(x_0)$, $(T_3) = \frac{1}{6}[x-x_0]^3 f'''(x_0)$, $(T_4) = \frac{1}{24}[x-x_0]^4 f^{(4)}(x_0)$ and $(T_5) = \frac{1}{120}[x-x_0]^5 f^{(5)}(x_0)$ from equation (1). We have:

$$\begin{cases} (T_2) & = & \frac{1}{2}x_1^2 2^{-2k} f''(x_0) + \\ & & x_1 x_2 2^{-3k} f''(x_0) + \\ & & x_1 x_3 2^{-4k} f''(x_0) + \\ & & \frac{1}{2}x_2^2 2^{-4k} f''(x_0) + \\ & & x_2(1/2)2^{-5k} f''(x_0) + \\ & & \frac{1}{2}x_1(1/2)2^{-5k} f''(x_0) + \epsilon_2 \\ (T_3) & = & \frac{1}{6}x_1^3 2^{-3k} f'''(x_0) + \\ & & \frac{1}{2}x_1^2 x_2 2^{-4k} f'''(x_0) + \\ & & \frac{1}{2}x_1^2 x_3 2^{-5k} f'''(x_0) + \\ & & \frac{1}{2}x_1 x_2^2 2^{-5k} f'''(x_0) + \epsilon_3 \\ (T_4) & = & \frac{1}{24}x_1^4 2^{-4k} f^{(4)}(x_0) + \\ & & \frac{1}{6}x_1^3 x_2 2^{-5k} f^{(4)}(x_0) + \epsilon_4 \\ (T_5) & = & \frac{1}{120}x_1^5 2^{-5k} f^{(5)}(x_0) + \epsilon_5 \end{cases}$$

with $\epsilon_2 < \frac{1}{2}2^{-5k}\max|f''|, \epsilon_3 < \frac{1}{3}2^{-6k}\max|f'''|, \epsilon_4 < \frac{1}{24}2^{-6k}\max|f^{(4)}|$ and $\epsilon_5 < \frac{1}{120}2^{-6k}\max|f^{(5)}|$. We obtain the **multiplicative table-based formula** by rewriting Equation (1) as follows:

$$\begin{array}{rcl} f(x) & = & A(x_0, x_1) + B(x_0, x_2) + \\ & & C(x_0, x_3) + D(x_0, x_4) + \\ & & x_2 \times E(x_0, x_1) + \\ & & x_3 2^{-k} \times E(x_0, x_1) + \epsilon_f \end{array} \tag{2}$$

where

$$\begin{array}{rcl} A(x_0, x_1) & = & f(x_0) + x_1 2^{-k} f'(x_0) + \\ & & \frac{1}{2}x_1^2 2^{-2k} f''(x_0) + \\ & & \frac{1}{6}x_1^3 2^{-3k} f'''(x_0) + \\ & & \frac{1}{24}x_1^4 2^{-4k} f^{(4)}(x_0) + \\ & & \frac{1}{120}x_1^5 2^{-5k} f^{(5)}(x_0) + \\ & & \frac{1}{2}x_1(1/2)2^{-5k} f''(x_0) + \\ B(x_0, x_2) & = & x_2 2^{-2k} f'(x_0) + \\ & & \frac{1}{2}x_2^2 2^{-4k} f''(x_0) + \\ & & (1/2)x_2 2^{-5k} f''(x_0) \\ C(x_0, x_3) & = & x_3 2^{-3k} f'(x_0) \\ D(x_0, x_4) & = & x_4 2^{-4k} f'(x_0) \\ E(x_0, x_1) & = & x_1 2^{-3k} f''(x_0) + \\ & & \frac{1}{2}x_1^2 2^{-4k} f'''(x_0) + \\ & & \frac{1}{6}x_1^3 2^{-5k} f^{(4)}(x_0) + \\ & & \frac{1}{2}x_1(1/2)2^{-5k} f'''(x_0) \\ \epsilon_f & \leq & \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 + \epsilon_5 \\ & \leq & \frac{1}{720}2^{-6k}\max|f^{(6)}| + \frac{1}{2}2^{-5k}\max|f''| + \\ & & \frac{1}{3}2^{-6k}\max|f'''| + \frac{1}{24}2^{-6k}\max|f^{(4)}| \\ & & \frac{1}{120}2^{-6k}\max|f^{(5)}| \\ & \leq & 2^{-5k}\left[\frac{1}{2}\max|f''| + \frac{1}{3}\max|f'''| + \right. \\ & & \frac{1}{24}\max|f^{(4)}| + \frac{1}{120}\max|f^{(5)}| + \\ & & \left.\frac{1}{720}\max|f^{(6)}| + \right] \end{array}$$

Hence, $f(x)$ can be obtained by performing 2 multiplications and adding six terms with an

error less than ϵ_f. The size of the tables where all these terms are looked-up from, depend heavily of the function we consider. We will discuss about it in the Section 4.2.

It must be noticed that the proposed method exploits derivative properties of functions. The successive derivatives must decreased rapidly enough to make removed terms insignificant compared to the one we kept.

4 Implementation

An abstract architecture for the multiplicative method is presented on Figure 1. It is very similar to the multipartite architecture of [3], with the exception of the two multipliers.

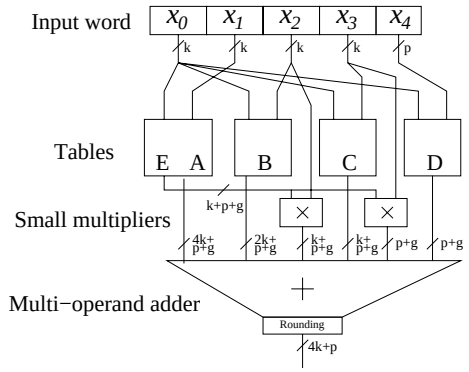


Figure 1: The abstract architecture for the proposed scheme

4.1 Rounding errors

In addition to the approximation errors considered in the previous section, this architecture involves several sources of rounding error:

- The tables have to be filled with fixed-point values which are the previous mathematical values rounded to some precision.
- Similarly, the output of the multipliers have to be rounded to the target precision.
- These approximation errors, and the approximation error, require us to compute with an internal precision which is higher than the final required precision. We classically use g additional bits of precision (or *guard bits*).
- Then the result has to be rounded to the final precision afer the final addition (with an error at worst of an half LSB).

Therefore, the table sizes needed are such that:

$$\begin{aligned} Size(A[x_0, x_1]) &\leq 2^{2*k}(4k + p + g) \\ Size(B[x_0, x_2]) &\leq 2^{2*k}(2k + p + g) \\ Size(C[x_0, x_3]) &\leq 2^{2*k}(k + p + g) \\ Size(D[x_0, x_4]) &\leq 2^{p+k}(p + g) \\ Size(E[x_0, x_5]) &\leq 2^{2*k}(k + p + g) \end{aligned}$$

The multiplier used to multiply E with x_2 is a $(k) \times (k+p+g)$ multiplier and the one used to multiply E with x_3 is a $(k) \times (p+g)$ one.

The number of guard bits is determined as the smallest value such that the sum of the approximation error and all the rounding errors is smaller than a LSB of the final result (faithful rounding).

Note that the rounding of the multiplier results, as well as the final rounding, may be performed by simple truncation (at no hardware cost), if corresponding correcting terms have been added to the value used to fill the A table.

4.2 General results

The method presented above has been implemented in a C program. This program fills the table and determines the total error (by enumeration) for increasing values of g, until faithful rounding is achieved.

The table sizes for various functions generated by the program are presented in table 1. In addition, we compared our results with the best known table size of generalized multipartite methods [3].

f	k	p	Table size	Number of correct bits	ref size
sin $[0, \pi/4[$	3	2	2768	14	3712
	4	3	15040	19	29440
	5	3	70528	23	138624
exp $[0, 1[$	3	2	3232	14	6272
	4	3	16256	19	56320
	5	4	82432	24	366080
$2^x - 1$ $[0, 1[$	3	2	3392	14	7168
	4	3	18048	19	56320
	5	4	89600	24	259584

Table 1: Table sizes and accuracy for 15-bit, 20-bit and 25-bit input operands

On tested functions we remark that the table size required by multiplicative table based method is less than the one required by generalized multipartite method. The tables A, B, C, E have $2k$ input bits, whereas table D have $p+k$ input bits. For all of them the number of output bits is variable and depends on the table, but also on the function. These tables are thus smaller than those in the multipartite method ($2^{2n/5}$ words versus $2^{n/2}$), which compensates for the multipliers.

4.3 Targetting FPGAs

This method represents an important improvement over the multipartite method when targetting last-generation FPGAs (Virtex II and Virtex Pro), which embed a large number of small ($18 \times 18 \rightarrow 35$ bits) multipliers. Their precision is more than enough for this scheme, and they allow a tree-fold reduction on the table size at no cost since they are available on the chip. For small k it may even be possible to use only one multiplier to compute the two products.

It should be noted that the tables can be further compressed by back-end logic optimization tools. Previous work [2] has shown that a 20% improvement on the table size can be expected..

Finally, it is unclear how this method would compare to a (more straightforward) implementation of a function using a piecewise degree 2 or 3 polynomial evaluation exploiting the same

multipliers. To our knowledge, such an approach has not yet been published, and it is the subject of current work.

4.4 Targetting VLSI

When targetting VLSI, the comparison with multipartite tables resumes to trading look-up table silicon for multiplier silicon. To be fair, it should be noted that an actual VLSI implementation will be somewhat different from what appears on Figure 1:

- A multiplier classically consists of a partial product array, a carry-save reduction tree, and a final adder. In our architecture there will be only one final adder at the end of the multi-operand adder. Besides, for a $k \times k$ or a $2k \times k$ multiplication, the depth of the reduction tree will be very small in the overall critical path. Altogether the additional latency due to the multipliers should be very small.
- Our approach of truncating the result of a full multiplier is certainly justified on FPGAs where these full multipliers are present. However, in VLSI, it is probably more area- and delay-efficient to truncate the partial product array. The rounding error will be higher, maybe requiring more guard bits, but the overall speed and area should be reduced. The study of this trade-off remains to be done.

5 Extensions and future work

One can remark that tables $C[x_0, x_3] = x_3 * 2^{-3k} * f'(x_0)$ and $D[x_0, x_4] = x_4 * 2^{-4k} * f'(x_0)$ store the same values. Therefore, we can reduce the overall table size required by storing only the C table. In this scheme, values previously stored in D will be look-up from table C address with (x_0, x_4) followed by a k-bit shift. The saving in space is small (about 1/13th of the table size) and will probably not be justified.

The multiplicative methods presented in Section 3 is based on an order-5 Taylor expansion with an input word equally split into 5 sub-word. This method can be tuned by increasing the order of the Taylor expansion, by relaxing the strict $4k + p$ splitting we have used, or by centering approximations. It will lead to an improvement in table size and/or in accuracy similar to that obtained by generalized multipartite methods [3] compared to previous multipartite methods [1, 7, 9]. The drawback is that the nice Taylor formulae have to be replaced with more general polynomial approximations (minimax or similar).

In general, there are many tradeoffs which can be obtained by increasing the number of small multipliers. It is part of our future investigation.

6 Conclusion

We have presented yet another method for approximating arbitrary functions up to single precision (24-bit) using moderate table size. The table size is reduced up to tree-fold compared to the best previously published table-and-addition methods, at the expense of two very small multipliers. This method has been implemented in a C program where the function, the input and output accuracy can be specified. This method is clearly well suited when targetting FPGAs embedding small multipliers, and probably offers good VLSI potential.

Future work include FPGA and VLSI synthesis and actual hardware comparisons, and the work described in section 5.

References

[1] D. Das Sarma and D.W. Matula. Faithful bipartite ROM reciprocal tables. In S. Knowles and W.H. McAllister, editors, *12th IEEE Symposium on Computer Arithmetic*, pages 17–28, Bath, UK, 1995. IEEE Computer Society Press.

[2] F. de Dinechin and J. Detrey. Multipartite tables in JBits for the evaluation of functions on fpgas. In *IEEE Reconfigurable Architecture Workshop, International Parallel and Distributed Symposium*, Fort Lauderdale, Florida, April 2002. Updated version of LIP research report 2001-44.

[3] F. de Dinechin and A. Tisserand. Some improvements on multipartite table methods. In Neil Burgess and Luigi Ciminiera, editors, *15th IEEE Symposium on Computer Arithmetic*, pages 128–135, Vail, Colorado, June 2001. Updated version of LIP research report 2000-38.

[4] H. Hassler and N. Takagi. Function evaluation by table look-up and addition. In S. Knowles and W.H. McAllister, editors, *12th IEEE Symposium on Computer Arithmetic*, pages 10–16, Bath, UK, 1995. IEEE Computer Society Press.

[5] A. A. Liddicoat and M. J. Flynn. High-performance floating point divide. In *Euromicro Symposium on Digital System Design*, pages 354–361, Warsaw, Poland, September 2001.

[6] J.M. Muller. *Elementary Functions, Algorithms and Implementation*. Birkhauser, Boston, 1997.

[7] J.M. Muller. A few results on table-based methods. *Reliable Computing*, 5(3):279–288, 1999.

[8] J.A. Piñeiro, J.D. Bruguera, and J.-M. Muller. Faithful powering computation using table look-up and a fused accumulation tree. In Neil Burgess and Luigi Ciminiera, editors, *15th IEEE Symposium on Computer Arithmetic*, pages 40–47, Vail, Colorado, June 2001.

[9] J.E. Stine and M.J. Schulte. The symmetric table addition method for accurate function approximation. *Journal of VLSI Signal Processing*, 21(2):167–177, 1999.

[10] W.F. Wong and E. Goto. Fast evaluation of the elementary functions in single precision. *IEEE Transactions on Computers*, 44(3):453–457, March 1995.