A few results on table-based methods

Jean-Michel Muller

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 8512

CENTRE NATIONAL DE LA RECHERCHE SCIENTIFIQUE

SPI

A few results on table-based methods

Jean-Michel Muller

October 1998

Research Report N° 98-46

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`

INRIA

A few results on table-based methods

Jean-Michel Muller

October 1998

Abstract

Table-based methods are frequently used to implement functions. We examine some methods introduced in the literature, and we introduce a generalization of the bipartite table method, named the **multipartite table method**.

Keywords: Elementary functions, computer arithmetic, table-based methods

Résumé

Les méthodes à base de tables sont de plus en plus utilisées pour implanter des fonctions. Nous examinons quelques méthodes suggérées antérieurement, puis nous proposons une généralisation de la méthode des "tables bipartites", appelée méthode des "tables multipartites".

Mots-clés: Fonctions élémentaires, arithmétique des ordinateurs, méthodes à base de tables.

Abstract

Table-based methods are frequently used to implement functions. We examine some methods introduced in the literature, and we introduce a generalization of the bipartite table method, named the **multipartite table method**.

1 Introduction

Throughout the paper, f is the function to be evaluated. We assume n-bit, fixed-point arguments, between $1/2$ and 1 (that is, they are mantissas of floating-point numbers).

Table-based methods have frequently been suggested and used to implement some arithmetic (reciprocal, square root) and transcendental functions. One can distinguish three different classes of methods:

- **compute-bound methods:** these methods use table-lookup in a small table to find parameters used afterward for a polynomial or rational evaluation. The main part of the evaluation of f consists in arithmetic computations;
- **table-bound methods:** The main part of the evaluation of f consists in looking up in a generally rather large table. The computational part of the function evaluation is rather small (e.g., a few additions);
- **in-between methods:** these methods use the combination of table lookup in a medium-size table and a significant yet reduced amount of computation (e.g. one or two multiplications, or several "small multiplications" that use rectangular – fast and/or small – multipliers).

Many methods currently used on general-purpose systems belong to the first class (e.g. Tang's methods [7, 8, 9, 10]. The third class of methods has been widely studied since 1981 [2]. The use of small (e.g., rectangular) multipliers to fasten the computational part of the evaluation has been suggested by several authors (see for instance Wong and Goto's algorithms for double precision calculations [11], or Ercegovac et al.'s methods [1]).

In this paper, we examine some table-bound methods. Of course, the straightforward method, consisting in building a table with n address bits, cannot be used unless n is very small. The first useful table-bound methods have been introduced in the last decade: they have become implementable thanks to progress in VLSI technology. Wong and Goto [12] have suggested the following method. We split the binary representation of the input-number x into four k-bit numbers, where $k = n/4$. That is, we write[1]:

$$= x_1 + x_2 2^{-k} + x_3 2^{-2k} + x_4 2^{-3k}$$

where $0 \leq x_i \leq 1 - 2^{-k}$ is a multiple of 2^{-k}.

Then $f(x)$ is approximated by:

$$\begin{aligned}
& f(x_1 + x_2 2^{-k}) \\
& + \tfrac{1}{2} 2^{-k} \left\{ f(x_1 + x_2 2^{-k} + x_3 2^{-k}) - f(x_1 + x_2 2^{-k} - x_3 2^{-k}) \right\} \\
& + \tfrac{1}{2} 2^{-2k} \left\{ f(x_1 + x_2 2^{-k} + x_4 2^{-k}) - f(x_1 + x_2 2^{-k} - x_4 2^{-k}) \right\} \\
& + 2^{-4k} \left\{ \frac{x_3^2}{2} f^{(2)}(x_1) - \frac{x_3^3}{6} f^{(3)}(x_1) \right\}
\end{aligned}$$

[1] To make the paper easier to read and more consistent, we do not use Wong and Goto's notations here. We use the same notations as in the sequel of this paper.

The approximation error due to the use of this approximation is about 2^{-5k}.

The **bipartite table method** was first suggested by Das Sarma and Matula [4] for quickly computing reciprocals. A slight improvement, the **symmetric bipartite table method** was introduced by Schulte and Stine [5]. Due to the importance of the bipartite table method (BTM), we will present it in detail in the next section. Compared to Wong and Goto's method, it requires larger tables. And yet, the amount of computation required by the BTM is reduced to one addition.

The problem of evaluating a function given by a converging series can be reduced to the evaluation of a partial product array (PPA). Schwarz [6] suggested to use multiplier structures to sum up PPAs. Hassler and Takagi [3] use PPAs to evaluate functions by table look-up and addition.

2 Order-1 methods

The methods described in this section use an order-1 Taylor approximation of f. This leads to very simple computations (mere additions), but the size of the required tables may be quite large.

2.1 The bipartite table method

This method was first suggested by DasSarma and Matula [4] for computing reciprocals. We split the binary representation of the input number x into 3 k-bit numbers, where $k = n/3$. That is, we write:

$$= x_1 + x_2 2^{-k} + x_3 2^{-2k}$$

where $0 \leq x_i \leq 1 - 2^{-k}$ is a multiple of 2^{-k}.

$x =$	x_1	x_2	x_3

We then write the order-1 Taylor expansion of f at $x_1 + x_2 2^{-k}$. This gives:

$$f(x) = f(x_1 + x_2 2^{-k}) + x_3 2^{-2k} f'(x_1 + x_2 2^{-k}) + \epsilon_1 \tag{1}$$

with $\epsilon_1 = \frac{1}{2} x_3^2 2^{-4k} f''(\xi_1)$, where $\xi_1 \in [x_1 + x_2 2^{-k}, x]$. Now, we approximate the value $f'(x_1 + x_2 2^{-k})$ by its order-0 Taylor expansion at x_1 (that is, by $f'(x1)$. This gives:

$$f(x) = f(x_1 + x_2 2^{-k}) + x_3 2^{-2k} f'(x_1) + \epsilon_1 + \epsilon_2 \tag{2}$$

with $\epsilon_2 = x_2 x_3 2^{-3k} f''(\xi_2)$, where $\xi_2 \in [x_1, x_1 + x_2 2^{-k}]$. This gives the **bipartite formula**:

$$f(x) = \alpha(x_1, x_2) + \beta(x_1, x_3) + \epsilon \tag{3}$$

where

$$\left\{ \begin{array}{lcl} \alpha(x_1, x_2) & = & f(x_1 + x_2 2^{-k}) \\ \beta(x_1, x_3) & = & x_3 2^{-2k} f'(x_1) \\ \epsilon & \leq & \left(\frac{1}{2} 2^{-4k} + 2^{-3k}\right) \max f'' \approx 2^{-3k} \max f'' \end{array} \right.$$

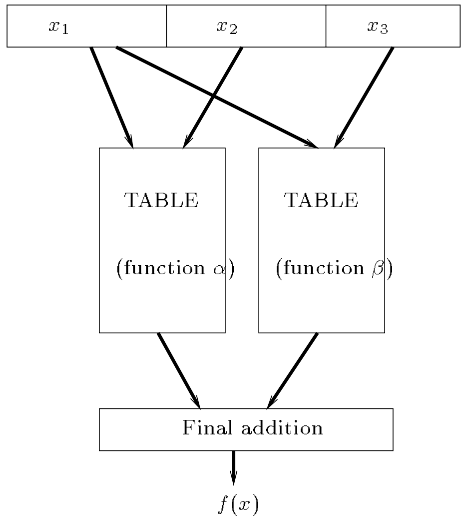

Figure 1: The bipartite table method

Hence, $f(x)$ can be approximated, with approximately n bits of accuracy (by this, we mean "with error $\approx 2^{-n}$") by the sum of two terms (α and β) that can be looked-up in $2n/3$-address bit tables, as illustrated by Figure 1.

The BTM still leads to large tables in single precision, and this. It is far from being implementable in double precision. And yet, this leads to another idea: we should try to generalize the bipartite method, by splitting the input word into more than three parts. Let us first try a splitting into five parts, we will after that generalize to an arbitrary odd number of parts.

2.2 The tripartite table method

Now, we split the input n-bit fixed-point number x into five k-bit parts x_1, x_2, ..., x_5. That is, we write:

$$= x_1 + x_2 2^{-k} + x_3 2^{-2k} + x_4 2^{-3k} + x_5 2^{-4k}$$

where $0 \leq x_i \leq 1 - 2^{-k}$ is a multiple of 2^{-k}.

$x =$	x_1	x_2	x_3	x_4	x_5

We use the order-1 Taylor expansion of f at $= x_1 + x_2 2^{-k} + x_3 2^{-2k}$:

$$\begin{array}{c} f(x) = f(x_1 + x_2 2^{-k} + x_3 2^{-2k}) \\ + \left(x_4 2^{-3k} + x_5 2^{-4k}\right) f'(x_1 + x_2 2^{-k} + x_3 2^{-2k}) \\ + \epsilon_3 + \epsilon_1 \end{array} \tag{4}$$

with $\epsilon_3 = \frac{1}{2}\left(x_4 2^{-3k} + x_5 2^{-4k}\right)^2 f''(\xi_3)$, with $\xi_3 \in [x_1 + x_2 2^{-k} + x_3 2^{-2k}, x]$, which gives $\epsilon_3 \leq \frac{1}{2} 2^{-6k} \max f''$.

In (4), we expand the term $\left(x_4 2^{-3k} + x_5 2^{-4k}\right) f'(x_1 + x_2 2^{-k} + x_3 2^{-2k})$ as follows:

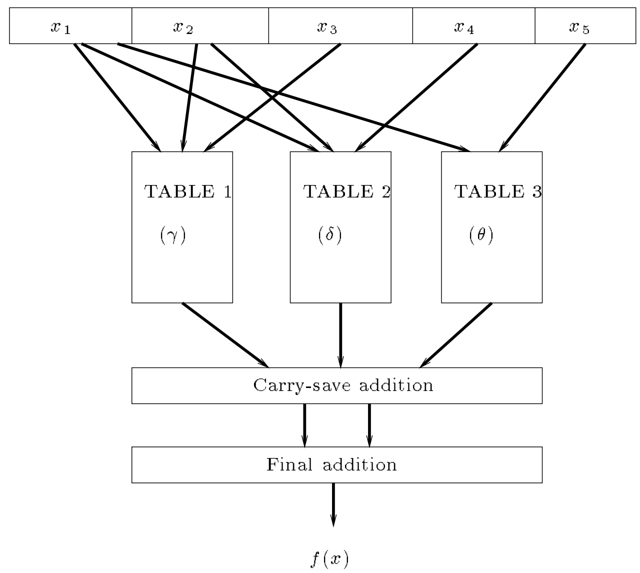

Figure 2: The tripartite table method

- $x_4 2^{-3k} f'(x_1 + x_2 2^{-k} + x_3 2^{-2k})$ is replaced by $x_4 2^{-3k} f'(x_1 + x_2 2^{-k})$. The error committed is $\epsilon_4 = x_3 x_4 2^{-5k} f''(\xi_4)$, where $\xi_4 \in [x_1 + x_2 2^{-k}, x_1 + x_2 2^{-k} + x_3 2^{-2k}]$. We easily get $\epsilon_4 \leq 2^{-5k} \max f''$.
- $x_5 2^{-4k} f'(x_1 + x_2 2^{-k} + x_3 2^{-2k})$ is replaced by $x_5 2^{-4k} f'(x_1)$. The error committed is $\epsilon_5 = \left(x_2 2^{-k} + x_3 2^{-2k}\right) x_5 2^{-4k} f''(\xi_5)$, where $\xi_5 \in [x_1, x_1 + x_2 2^{-k} + x_3 2^{-2k}]$. We get $\epsilon_5 \leq 2^{-5k} \max f''$.

This gives the **tripartite formula**:

$$f(x) = \gamma(x_1, x_2, x_3) + \delta(x_1, x_2, x_4) + \theta(x_1, x_5) + \epsilon \qquad (5)$$

where

$$\begin{array}{lcl}
\gamma(x_1, x_2, x_3) & = & f(x_1 + x_2 2^{-k} + x_3 2^{-2k}) \\
\delta(x_1, x_2, x_4) & = & x_4 2^{-3k} f'(x_1 + x_2 2^{-k}) \\
\theta(x_1, x_5) & = & x_5 2^{-4k} f'(x_1) \\
\epsilon & \leq & \left(\frac{1}{2} 2^{-6k} + 2 \times 2^{-5k}\right) \max f'' \approx 2^{-5k+1} \max f''
\end{array}$$

Hence, $f(x)$ can be obtained by adding three terms, each of them being looked-up in a table with (at most) $3n/5$ address bits. This is illustrated by Figure 2.

2.3 Generalization: the multipartite table method

The previous approach is straightforwardly generalized. We now assume that the n-bit input number x is split into $2p+1$ k-bit values x_1, x_2, $\ldots x_{2p+1}$. That is,

$$x = \sum_{i=1}^{2p+1} x_i 2^{(i-1)k}$$

where the x_i's are multiples of 2^{-k} and satisfy $0 \leq x_i < 1$. As in the previous sections, we use the order-1 Taylor expansion:

$$\begin{array}{rcl} f(x) & = & f\left(x_1 + x_2 2^{-k} + \ldots + x_{p+1}2^{-pk}\right) \\ & + & \left(x_{p+2}2^{(-p-1)k} + \ldots + x_{2p+1}2^{-2pk}\right) f'\left(x_1 + x_2 2^{-k} + \ldots + x_{p+1}2^{-pk}\right) \\ & + & \epsilon_{p+1} \end{array}$$

with $\epsilon_{p+1} \leq \frac{1}{2}2^{-2(p+1)k} \max f''$. We expand the term

$$\left(x_{p+2}2^{(-p-1)k} + \ldots + x_{2p+1}2^{-2pk}\right) f'\left(x_1 + x_2 2^{-k} + \ldots + x_{p+1}2^{-pk}\right),$$

and perform Taylor approximations to $f'\left(x_1 + x_2 2^{-k} + \ldots + x_{p+1}2^{-pk}\right)$. We then get:

$$\begin{array}{rcl} f(x) & = & f\left(x_1 + x_2 2^{-k} + \ldots + x_{p+1}2^{-pk}\right) + \epsilon_{p+1} \\ & + & x_{p+2}2^{(-p-1)k} f'\left(x_1 + x_2 2^{-k} + \ldots + x_p 2^{(-p+1)k}\right) + \epsilon_{p+2} \\ & + & x_{p+3}2^{(-p-2)k} f'\left(x_1 + x_2 2^{-k} + \ldots + x_{p-1} 2^{(-p+2)k}\right) + \epsilon_{p+3} \\ & + & x_{p+4}2^{(-p-3)k} f'\left(x_1 + x_2 2^{-k} + \ldots + x_{p-2} 2^{(-p+3)k}\right) + \epsilon_{p+4} \\ & & \ldots \\ & + & x_{2p+1}2^{-2pk} f'\left(x_1\right) + \epsilon_{2p+1} \end{array}$$

where $\epsilon_{p+2}, \epsilon_{p+3}, \ldots \epsilon_{2p+1}$ are less than $2^{(-2p-1)k} \max f''$.

This gives the **multipartite (or $(p+1)$-partite) formula**:

$$\begin{array}{rcl} f(x) & = & \alpha_1(x_1, x_2, \ldots, x_{p+1}) \\ & + & \alpha_2(x_1, x_2, \ldots, x_{p-1}, x_{p+3}) \\ & + & \alpha_3(x_1, x_2, \ldots, x_{p-2}, x_{p+4}) \\ & + & \alpha_4(x_1, x_2, \ldots, x_{p-3}, x_{p+5}) \\ & & \ldots \\ & + & \alpha_{p+1}(x_1, x_{2p+1}) \\ & + & \epsilon \end{array} \tag{6}$$

where

$$\left\{ \begin{array}{lcl} \alpha_1(x_1, \ldots, x_{p+1}) & = & f\left(x_1 + x_2 2^{-k} + \ldots + x_{p+1}2^{-pk}\right) \\ \alpha_i(x_1, \ldots, x_{p-i+2}, x_{p+i}) & = & x_{p+i}2^{-p-i+1} f'\left(x_1 + x_2 2^{-k} + \ldots + x_{p-i+2}2^{(-p+i-1)k}\right) \\ \epsilon & \leq & \left(\frac{1}{2}2^{(-2p-2)k} + p2^{(-2p-1)k}\right) \max f'' \\ & \approx & p2^{(-2p-1)k} \max f'' \end{array} \right.$$

Too large values of p are unrealistic: performing many additions to avoid a few multiplications is not reasonable.

3 Higher-order methods

In the previous section, we have used order-1 Taylor expansions only. Now, let us give an example of the use of an order-2 expansion. As in section 2.2, we split the input n-bit fixed-point number x into five k-bit parts x_1, x_2, $\ldots$, x_5. That is, we write:

$$= x_1 + x_2 2^{-k} + x_3 2^{-2k} + x_4 2^{-3k} + x_5 2^{-4k}$$

$2p+1$	n	nb bytes	nb of address bits
bipartite	24	262144	16
bipartite (larger n)	27	1179648	18
tripartite	25	143360	15
tripartite (larger n)	30	1376256	18
$2p+1=7$	28	327680	16
$2p+1=7$ (larger n)	35	6553600	20

Table 1: Table sizes for various order-1 methods

where $0 \leq x_i \leq 1-2^{-k}$ is a multiple of 2^{-k}.

$$\begin{array}{lcl}
\alpha_1(x_1,x_2) & = & f(x_1+x_2 2^{-k}) - 3f(x_1) - \frac{1}{2}(2^{-3k}+2^{-4k})x_2^2 f''(x_1) \\
\alpha_2(x_1,x_3) & = & f(x_1+x_3 2^{-2k}) - \frac{1}{2}2^{-3k}x_3^2 f''(x_1) - \frac{1}{6}2^{-4k}x_3^3 f'''(x_1) \\
\alpha_3(x_1,x_4) & = & f(x_1+x_4 2^{-3k}) - \frac{1}{2}2^{-4k}x_4^2 f''(x_1) \\
u & = & x_2+x_3 \\
v & = & x_2-x_3 \\
\beta_1(u,x_1) & = & \frac{1}{12}2^{-4k}u^3 f'''(x_1) \\
\beta_2(v,x_1) & = & -\frac{1}{12}2^{-4k}v^3 f'''(x_1)
\end{array} \tag{7}$$

Then

$$f(x) \approx \alpha_1(x_1,x_2)+\alpha_2(x_1,x_3)+\alpha_3(x_1,x_4)+\beta_1(u,x_1)+\beta_2(v,x_1)$$

Hence, with this method, we can use tables with $2n/5$ address bits. Two additions are used to generate u and v, and after the table-lookups, two carry-save additions and one carry-propagate addition suffice to get the final result. This method requires around 15Kbytes of table for single-precision.

Conclusion

Various table-based methods have been suggested during the last decade. When single-precision implementation is at stake, table-bound methods seem to be agood candidate for implementing fast functions. Unless there is a technology breakthrough, these methods are not suitable for double precision.

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. Technical Report RR97-47, LIP, École Normale Supérieure de Lyon, November 1997. available at ftp://ftp.lip.ens-lyon.fr/pub/Rapports/RR/RR97/RR97-47.ps.Z.

[2] P. M. Farmwald. High bandwidth evaluation of elementary functions. In In K. S. Trivedi and D. E. Atkins, editors, *Proceedings of the 5th IEEE Symposium on Computer Arithmetic*. IEEE Computer Society Press, Los Alamitos, CA, 1981.

[3] H. Hassler and N. Takagi. Function evaluation by table look-up and addition. In S. Knowles and W. McAllister, editors, *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, Bath, UK, July 1995. IEEE Computer Society Press, Los Alamitos, CA.

[4] D. Das Sarma and D. W. Matula. Faithful bipartite rom reciprocal tables. In S. Knowles and W. H. McAllister, editors, *Proceedings of the 12th IEEE Symposium on Computer Arithmetic*, pages 17–28, Bath, UK, 1995. IEEE Computer Society Press, Los Alamitos, CA.

[5] M. Schulte and J. Stine. Symmetric bipartite tables for accurate function approximation. In In T. Lang, J.M. Muller, and N. Takagi, editors, *Proceedings of the 13th IEEE Symposium on Computer Arithmetic*. IEEE Computer Society Press, Los Alamitos, CA, 1997.

[6] E. Schwarz. *High-Radix Algorithms for High-Order Arithmetic Operations*. PhD thesis, Dept. of Electrical Engineering, Stanford University, 1992.

[7] P. T. P. Tang. Table-driven implementation of the exponential function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 15(2):144–157, June 1989.

[8] P. T. P. Tang. Table-driven implementation of the logarithm function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 16(4):378–400, December 1990.

[9] P. T. P. Tang. Table lookup algorithms for elementary functions and their error analysis. In P. Kornerup and D. W. Matula, editors, *Proceedings of the 10th IEEE Symposium on Computer Arithmetic*, pages 232–236, Grenoble, France, June 1991. IEEE Computer Society Press, Los Alamitos, CA.

[10] P. T. P. Tang. Table-driven implementation of the expm1 function in IEEE floating-point arithmetic. *ACM Transactions on Mathematical Software*, 18(2):211–222, June 1992.

[11] W. F. Wong and E. Goto. Fast hardware-based algorithms for elementary function computations using rectangular multipliers. *IEEE Transactions on Computers*, 43(3):278–294, March 1994.

[12] 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.