Symmetric Bipartite Tables for Accurate Function Approximation

Michael J. Schulte and James E. Stine
Department of Electrical Engineering and Computer Science
Lehigh University
Bethlehem, PA 18015, USA

Abstract

This paper presents a methodology for designing bipartite tables for accurate function approximation. Bipartite tables use two parallel table lookups to obtain a carry-save (borrow-save) function approximation. A carry propagate adder can then convert this approximation to a two's complement number or the approximation can be directly Booth encoded. Our method for designing bipartite tables, called the Symmetric Bipartite Table Method, utilizes symmetry in the table entries to reduce the overall memory requirements. It has several advantages over previous bipartite table methods in that it (1) provides a closed form solution for the table entries, (2) has tight bounds on the maximum absolute error, (3) requires smaller table lookups to achieve a given accuracy, and (4) can be applied to a wide range of functions. Compared to conventional table lookups, the symmetric bipartite tables presented in this paper are 15.0 to 41.7 times smaller when the operand size is 16 bits and 99.1 to 273.9 times smaller when the operand size is 24 bits.

1. Introduction

Elementary function approximations are important for applications in several areas, including digital signal processing, computer graphics, and scientific computing. For applications that require low-precision elementary function approximations at high speeds, table lookups are often employed. However, as the required precision of the approximation increases (e.g., greater than 16 bits), the size of the memory needed to implement the table lookups becomes prohibitive. Several techniques have been devised for reducing the amount of memory required to approximate the elementary functions, while keeping their performance at an acceptable level. These techniques include parallel polynomial approximations [1] - [5], table-driven polynomial approximations [6], [7], and rational approximations [8]. Unfortunately, these techniques all require multiplication, which may not be available at high speeds.

One method that has recently been developed for approximating elementary functions is based on bipartite tables [9], [10]. With this technique, two tables are accessed in parallel. The outputs of the two tables provide an approximation to the function in carry-save (borrow-save) form. These two outputs can combined with a carry propagate adder [11] to produce a two's complement approximation to the function. Alternatively, if the carry-save (borrow-save) approximation is to be used in a subsequent multiplication, it can be Booth encoded, thus avoiding the need for the carry propagate addition [10].

Compared to conventional table lookups, bipartite tables require significantly less memory. The only increase in delay is caused by the time required to perform the addition or Booth encoding after the table lookup, and this increase in delay is offset by the fact that the smaller tables may have shorter access times. The concept of bipartite table lookups can also be extended so that more than two tables are employed [9]. Although this technique further reduces the amount of memory, it requires a multi-operand adder to sum the table outputs. Another technique, known as the ATA method, also reduces the amount of memory [12]. However, it requires four carry-propagate adders, six lookup tables, and a 6-input multi-operand adder to approximate functions to 24 bits of precision.

This paper presents an improved bipartite table method, called the Symmetric Bipartite Table Method (SBTM). This method uses symmetry in the entries of one of the tables to reduce the overall memory requirements. The SBTM has a closed form solution for the table entries and can be applied to a wide range of functions. Compared to previous bipartite table methods it requires less memory and has a tighter bound on the maximum absolute error. Section 2 introduces the fundamental concepts behind bipartite table approximations, describes the SBTM for selecting the table entries, and provides an error analysis for this method. Section 3 shows how symmetry in the entries of one of the tables is used to reduce the size of this table by a factor of two. Section 3 also gives the memory requirements of

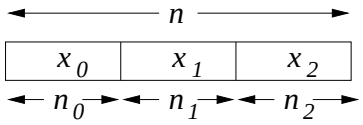

Figure 1. Dividing the Input Operand.

the SBTM for various functions. Section 4 compares the SBTM to previous bipartite table methods, and Section 5 presents conclusions. Additional literature on symmetric table methods and software tools for their implementation are available from the Internet URL

http://www.eecs.lehigh.edu/~caar/SBTM.html

2. Bipartite Table Function Approximation

To approximate a function $f(x)$ using bipartite tables, the input operand x is divided into three parts, as shown in Figure 1. These three parts are denoted as x_0, x_1, and x_2 and have lengths of n_0, n_1, and n_2, respectively. The value of the input operand is $x = x_0 + x_1 + x_2$ and it has a length of $n = n_0 + n_1 + n_2$. With this method, the function $f(x)$ is approximated as

$$f(x) = f(x_0 + x_1 + x_2) \approx a_0(x_0, x_1) + a_1(x_0, x_2) \quad (1)$$

The $n_0 + n_1$ most significant bits of x are inputs to a table that provides the coefficient $a_0(x_0, x_1)$, and the n_0 most significant and n_2 least significant bits of x are inputs to a table that provides the coefficient $a_1(x_0, x_2)$. The outputs from the two tables produce a carry-save (borrow-save) approximation to $f(x)$. If desired, the approximation can then be converted to a two's complement number using a carry-propagate adder. This is shown in Figure 2, where the lengths of $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ are denoted as p_0 and p_1, respectively. The approximation to $f(x)$, denoted as $\widetilde{f(x)}$, has a length of p. Although n and p are often equal, for a number of applications it is desirable to have different values for n and p.

The bipartite table algorithm has fairly simple hardware requirements. It has one n-bit register and one p-bit register to store the input and output operands. A $2^{n_0+n_1}$-word by p_0-bit table stores the values for $a_0(x_0, x_1)$ and a $2^{n_0+n_2}$-word by p_1-bit table stores the values for $a_1(x_0, x_2)$. If a two's complement result is desired, a p_0-bit carry-propagate adder is used to combine the table outputs. For the SBTM, $p_0 > p_1$ and $a_1(x_0, x_2)$ is sign-extended to p_0 bits prior to the addition.

2.1. Selecting the Coefficients for the SBTM

This section presents the method for selecting the coefficients $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ for the SBTM. Initially, it

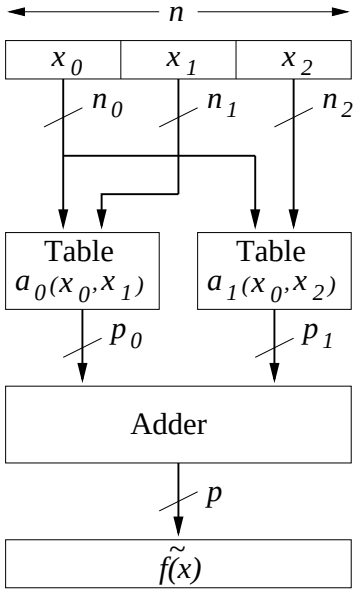

Figure 2. Bipartite Method Block Diagram.

is assumed that $0 \le x < 1$, which results in the following conditions

$$\begin{aligned} 0 &\le x_0 \le 1 - 2^{-n_0} \\ 0 &\le x_1 \le 2^{-n_0} - 2^{-n_0-n_1} \\ 0 &\le x_2 \le 2^{-n_0-n_1} - 2^{-n_0-n_1-n_2} \end{aligned} \quad (2)$$

The SBTM approximations are based on a two term Taylor series expansion of $f(x)$, about the point $x_0 + x_1 + \delta_2$, where

$$\delta_2 = 2^{-n_0-n_1-1} - 2^{-n_0-n_1-n_2-1} \quad (3)$$

is exactly halfway between the minimum and maximum values for x_2. This approximation takes the form

$$\begin{aligned} f(x) &= f(x_0 + x_1 + x_2) \\ &\approx f(x_0 + x_1 + \delta_2) + f'(x_0 + x_1 + \delta_2)(x_2 - \delta_2) \end{aligned} \quad (4)$$

and is illustrated in Figure 3. As described in Section 2.2, this approximation leads to a small error due to the omission of the high order terms of the Taylor series expansion.

From Equation 4, the first coefficient is selected as

$$a_0(x_0, x_1) = f(x_0 + x_1 + \delta_2) \quad (5)$$

Since the second coefficient cannot depend on x_1, it is selected as

$$a_1(x_0, x_2) = f'(x_0 + \delta_1 + \delta_2)(x_2 - \delta_2) \quad (6)$$

This corresponds to the second term of the Taylor series with x_1 replaced by the constant δ_1, where

$$\delta_1 = 2^{-n_0-1} - 2^{-n_0-n_1-1} \quad (7)$$

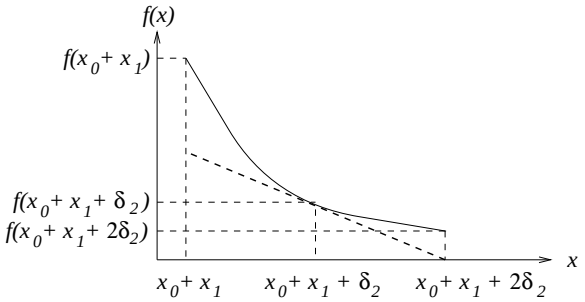

Figure 3. A 2-Term Taylor Series Expansion.

is exactly halfway between the minimum and maximum values for x_1.

One advantage of this method is that the magnitude of the second coefficient is much less than the magnitude of the first coefficient. Since $\mid x_2 - \delta_2 \mid < 2^{-n_0-n_1-1}$, the magnitude of the second coefficient is bounded by

$$\mid a_1(x_0, x_2) \mid < \mid f'(\xi_1) \mid 2^{-n_0-n_1-1} \tag{8}$$

where ξ_i is the point on [0, 1) at which $\mid f^i(x) \mid$ takes its maximum value. This results in approximately

$$n_0 + n_1 + 1 + \log_2(\mid f(\xi_0)/f'(\xi_1) \mid) \tag{9}$$

leading zeros (or leading ones if $a_1(x_0, x_2) < 0$). These leading zeros (or ones) do not need to be stored in memory, but can be obtained by sign-extending the most significant bit of $a_1(x_0, x_2)$, before performing the carry propagate addition.

2.2. Error Analysis for the SBTM

The SBTM has errors due to:

1. Approximating $f(x)$ with the first two terms of its Taylor series expansion (see Equation 11).

2. Replacing the second term of the Taylor series expansion with the coefficient $a_1(x_0, x_2)$ (see Equation 13).

3. Rounding the coefficients $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ to p_0 and p_1 bits, respectively (see Equation 14).

4. Rounding the final result $\widetilde{f(x)}$ to p bits (see Equation 15).

By controlling each of these errors, the SBTM can produce results that are faithfully rounded (i.e., the true result $f(x)$ and its approximation $\widetilde{f(x)}$ differ by at most one unit in the last place (ulp) [10]).

In Equation 4, leaving off the higher order terms in the Taylor series approximation results in an absolute error of

$$\begin{aligned} \epsilon_0 &= \mid \sum_{i=2}^{\infty} f^i(x_0 + x_1 + \delta_2)(x_2 - \delta_2)^i \mid \\ &\approx \mid f''(x_0 + x_1 + \delta_2) \mid (x_2 - \delta_2)^2 \end{aligned} \tag{10}$$

Since $\mid x_2 - \delta_2 \mid < 2^{-n_0-n_1-1}$, this error is bounded by

$$\epsilon_0 < \mid f''(\xi_2) \mid 2^{-2n_0-2n_1-2} \tag{11}$$

Replacing the second term of the Taylor series approximation with the value of $a_1(x_0, x_2)$ given in Equation 6, results in an absolute error of

$$\begin{aligned} \epsilon_1 &= \mid f'(x_0 + x_1 + \delta_2)(x_2 - \delta_2) - \\ &\quad f'(x_0 + \delta_1 + \delta_2)(x_2 - \delta_2) \mid \\ &= \mid \sum_{i=2}^{\infty} f^i(x_0 + \delta_2)(x_1^{i-1} - \delta_1^{i-1})(x_2 - \delta_2) \mid \\ &\approx \mid f''(x_0 + \delta_2)(x_1 - \delta_1)(x_2 - \delta_2) \mid \end{aligned} \tag{12}$$

Since $\mid x_1 - \delta_1 \mid < 2^{-n_0-1}$ and $\mid x_2 - \delta_2 \mid < 2^{-n_0-n_1-1}$, this error is bounded by

$$\epsilon_1 < \mid f''(\xi_2) \mid 2^{-2n_0-n_1-2} \tag{13}$$

which is 2^{n_1} times larger than the upper bound on ϵ_0.

To limit roundoff error, the coefficients $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ and the final result $\widetilde{f(x)}$ are rounded using a method similar to the one described in [10]. With this method, if $\widetilde{f(x)}$ has p_f fractional bits, and $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ each have $p_f + g$ fractional bits, then rounding is performed as follows:

1. $a_0(x_0, x_1)$ is rounded to the nearest number with $p_f + g$ fractional bits, and the $(p_f + g + 1)$-th fractional bit is set to zero.

2. $a_1(x_0, x_2)$ is truncated to $p_f + g$ fraction bits, and the $(p_f + g + 1)$-th fractional bit is set to one.

3. $\widetilde{f(x)}$ is rounded to the nearest number with p_f fraction bits.

The absolute error due to rounding both $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ is bounded by

$$\epsilon_2 \leq 2^{-p_f-g} \tag{14}$$

Since $a_1(x_0, x_2) + a_0(x_0, x_1)$ has $p_f + g + 1$ fractional bits and the least significant bit is guaranteed to be a one, the maximum error in rounding $\widetilde{f(x)}$ to p_f fractional bits is bounded by

$$\epsilon_3 \leq 2^{-p_f-1} - 2^{-p_f-g-1} \tag{15}$$

Since the least significant bits of $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ are known in advance (i.e., they are always 0 and 1, respectively), these bits are not actually stored in the tables. Instead, they are incorporated into the rounding logic of the adder.

To ensure that each approximations is faithfully rounded, the sum of the four errors should be no more than one ulp (i.e., $\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon_3 \leq 2^{-p_f}$). Replacing each error by its upper bound and combining terms gives the requirement

$$\mid f''(\xi_2) \mid 2^{-2n_0-n_1-2}(1 + 2^{-n_1}) + 2^{-p_f-g-1} \leq 2^{-p_f-1} \quad (16)$$

The first term on the left hand side of this inequality is controlled by selecting appropriate values for n_0 and n_1, and the second term is controlled by selecting an appropriate value for g, which effects the values of p_0 and p_1. Taking the $\log_2$ of Constraint 16 and combining terms gives

$$2n_0 + n_1 \geq p_f - 1 + \log_2(\frac{\mid f''(\xi_2) \mid (1 + 2^{-n_1})}{1 - 2^{-g}}) \quad (17)$$

Letting $g = 2$ and assuming $n_1 \geq 1$ yields the constraint

$$2n_0 + n_1 \geq p_f + \log_2(\mid f''(\xi_2) \mid) \quad (18)$$

As an example, suppose the goal is to approximate $\cos(x)$ for $0 \leq x < 1$, when the input and output operands are 24 bits (i.e., $n = p = p_f = 24$). Since $f(x) = \cos(x)$, $\mid f''(x) \mid = \mid \cos(x) \mid$, which takes a maximum value of 1 when $x = 0$. Constraint 18 then becomes

$$2n_0 + n_1 \geq 24 \quad (19)$$

To minimize the memory requirements, this is satisfied by choosing $n_0 = 8$, $n_1 = 8$, and $n_2 = 8$. To provide 2 guard digits, $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$ each require 26 fractional bits.[1] However, since $\mid a_1(x_0, x_2) \mid < 2^{-17}$, only 10 bits (including the sign bit) are stored and the remaining bits are obtained by extending the sign bit. This design requires a 2^{16}-word by 26-bit table to store $a_0(x_0, x_1)$ and a 2^{16}-word by 10-bit table to store $a_1(x_0, x_2)$.

3. Symmetric Bipartite Tables

The table that stores $a_1(x_0, x_2)$ is $2^{n_0+n_2}$ words by p_1 bits. However, by taking advantage of the symmetry of the coefficients, the size of this table can be reduced to $2^{n_0+n_2-1}$ words by $p_1 - 1$ bits. This is done by taking advantage of the following two properties:

1. $2\delta_2 - x_2$ is the one's complement of x_2
2. $a_1(x_0, 2\delta_2 - x_2)$ is the one's complement of $a_1(x_0, x_2)$

[1] When $a_0(x_0, x_1)$ takes its maximum value of 1, the value $1 - 2^{-26}$ is stored in memory instead. This still guarantees faithful rounding and eliminates the need to store an integer bit.

		$a_1(x_0, x_2)$	
x	$x_2 2^7$	decimal	binary
0.500000	000	+0.0166016	0.0000010001
0.507812	001	+0.0107422	0.0000001011
0.515625	010	+0.0068359	0.0000000111
0.523438	011	+0.0029297	0.0000000011
0.531250	100	-0.0029297	1.1111111101
0.539062	101	-0.0068359	1.1111111001
0.546875	110	-0.0107422	1.1111110101
0.554688	111	-0.0166016	1.1111101111

Table 1. Table Entries of $a_1(x_0, x_2)$.

These properties are demonstrated in Table 1, which gives the eight table entries for $a_1(x_0, x_2)$, when $f(x) = \cos(x)$, $x_0 = 0.5$, $x_1 = 0$, $n_0 = 2$, $n_1 = 2$, $n_2 = 3$, $p = 7$, and $g = 2$. To illustrate the symmetry in the table, the values for $x_2 2^7$ are given binary and the rounded values of $a_1(x_0, x_2)$ are given in both decimal and binary. Because of the method for selecting and rounding $a_1(x_0, x_2)$, the first four entries in the table are the one's complement of the last four entries. Although this table is for a specific instance, the results hold for the general case.

To reduce the size of the $a_1(x_0, x_2)$ table by a factor of two, the most significant bit of x_2 is examined. If this bit is zero, then the remaining bits of x_2, which are not changed, are used as the address to the $a_1(x_0, x_2)$ table, and the value read from this table is added to $a_0(x_0, x_1)$. Otherwise, the remaining bits of x_2 are complemented and used as the address to the $a_1(x_0, x_2)$ table, and the value read from this table is complemented and added to $a_0(x_0, x_1)$. This is illustrated in Figure 4. When the most significant bit of x_2 is a one, the row of $n_2 - 1$ exclusive-or gates before the table conditionally complement the remaining bits of x_2, and the row of p_1 exclusive-or gates after the table complement the bits of $a_1(x_0, x_2)$. Since all the values for $a_1(x_0, x_2)$ that are stored in memory now have the same sign, it is not necessary to store the sign bit in the table. Also, the least significant bits of $a_0(x_0, x_1)$ and $a_1(x_0, x_2)$, which are known to be 0 and 1, respectively, do not need to be stored in the table. Thus, in Table 1, only the four bits directly to the left of the least significant bit are actually stored in memory. These minor hardware modifications lead to a significant reduction in the amount of memory required to implement the bipartite table method. For example, for the 24-bit $\cos(x)$ example presented in Section 2.2, the size of the memory for $a_1(x_0, x_2)$ is reduced from 2^{16} words by 10 bits to 2^{15} words by 9 bits.

The Symmetric Bipartite Table Method can be applied to any differentiable function. Although the discussion presented in the previous sections assumes $0 \leq x < 1$, the general methodology can applied to operands outside of this

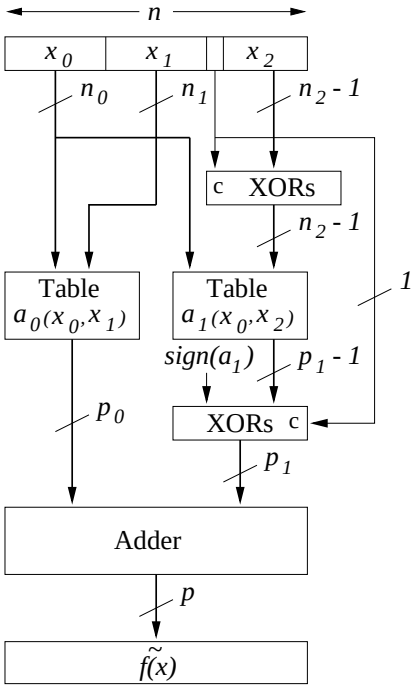

Figure 4. Symmetric Bipartite Block Diagram.

range. For example, reciprocals are often computed over the range $1 \leq x < 2$. For this range, the bit to the left of the radix point is known to be a one, so it does not need to be used in the table lookup and n_0 represents the number of *fractional bits* in the most significant portion of the number. The same equations that were presented in the previous sections are used, except that $n_0 + n_1 + n_2 = n - 1$.

Table 2 shows the input and output ranges for the functions discussed in this section. Well-known techniques, such as those presented in [13] and [14], can be used to bring the input operands within the specified range. Table 2 also shows the maximum values for the first and second derivative of the function over the input range. These values are used to help determine the table dimensions and the number of leading zeros in the second coefficient.

Tables 3, 4, and 5 show the dimensions of symmetric bipartite tables that produce faithfully rounded results. For comparison, these tables also show the required dimensions of a standard table lookup for each of the functions and the compression achieved by the SBTM. The compression is the amount of memory required by a standard table lookup divided by the amount of memory required by the SBTM. In these tables, the compression is rounded to the nearest tenth. For the functions $1/x$, and $\log_2(x)$, for which $1 \leq x < 2$, the leading bit of x is know to be one and does not need to be fed into the table. For the functions $1/x$, $\sqrt{x}$, and 2^x, the leading bit of $f(x)$ is known to be one and does not need to be produced by the table. Both the SBTM and the standard table lookup method take advantage of this. When computing $\sqrt{x}$ over [1,4) for normalized floating point number, the most significant bit of the significand is known to be a one and the least significant bit of the exponent is used to determine whether the exponent is even or odd. Thus, for $\sqrt{x}$, a total of n bits are used for the table lookups and n_0 includes the least significant bit of the exponent. For each of the functions, symmetric bipartite tables, with the dimensions given in Tables 3 to 5, were generated. These tables were then tested for all possible input values to ensure that they produce faithfully rounded results.

Compared to standard table lookups, the SBTM offers a significant reduction in memory. The compression factor achieved by the SBTM increases rapidly as the size of the input operand increases. For example, for 12-bit operands the SBTM uses 5.6 to 15.3 times less memory than standard table lookups; for 16-bit operands it uses 15.0 to 41.7 times less memory; and for 24-bit operands it uses 99.1 to 273.9 times less memory. In general, the highest compressions were achieved for $\sqrt{x}$, 2^x, and the trigonometric functions, while lower compressions were achieved for $1/x$ and $\log_2(x)$. This is because the functions $1/x$ and $\log_2(x)$ have larger values for $\mid f''(\xi_2) \mid$, which causes n_0 or n_1 to require larger values to achieve a given accuracy.

In [19], a method, known as the Two-Folds Technique, is presented. This technique uses symmetry to reduce the table sizes required for x^2. This method differs from the Symmetric Bipartite Table Method in that it (1) only uses a single table, (2) is used for computing the exact value of x^2, rather than an approximation, and (3) cannot be directly applied to other functions. The Two-Folds Technique is similar to the Symmetric Bipartite Table Method in that the input bits to the table lookup are conditionally complemented based on a bit from the input operand. Compared to a standard table lookup, the Two-Folds Technique reduces the number of table entries by a factor of two, but requires $n-1$ exclusive-or gates, $2(n-1)$ AND gates, and a $2(n-1)$-bit carry propagate adder to produce the $2n$-bit square of an n-bit number. The Two-Folds Technique can be extended to further reduce the memory requirements, however, this requires additional hardware, and is only applicable to the square function.

4. Comparison to Previous Bipartite Methods

This section compares the SBTM to previous bipartite table methods. Section 4.1 compares the SBTM to the 3-block method presented in [9]. This method is similar to the SBTM in that it assumes that all of the bits for x, except for bits whose values are known, are used in the table lookup. This is appropriate for lower precision elementary function approximations (e.g., 24 bits or less). Section 4.2 compares the SBTM to the bipartite reciprocal tables pre-

$f(x)$	Input Range	Output Range	$\mid f'(\xi_1) \mid$	$\mid f''(\xi_2) \mid$
$1/x$	[1, 2)	(0.5, 1]	1	2
$\sqrt{x}$	[1, 4)	[1, 2)	1/2	1/4
$\sin(x)$	[0, 1)	$[0, 1/\sqrt{2})$	1	$1/\sqrt{2}$
$\cos(x)$	[0, 1)	$(1/\sqrt{2}, 1]$	$1/\sqrt{2}$	1
$\tan^{-1}(x)$	[0, 1)	$[0, \pi/4)$	1	$0.6495\ldots$
$\log_2(x)$	[1, 2)	[0, 1)	$\log_2(e)$	$\log_2(e)$
2^x	[0, 1)	[1, 2)	$2 \ln 2$	$2(\ln 2)^2$

Table 2. Ranges and Maximum Derivative Values for Various Functions.

$f(x)$	Symmetric Bipartite Tables n_0, n_1, n_2	Sizes	Standard Table Sizes	Compression
$1/x$	5, 3, 3	$2^8 \times 13 + 2^7 \times 5$	$2^{11} \times 11$	5.7
$\sqrt{x}$	4, 3, 5	$2^7 \times 13 + 2^8 \times 5$	$2^{12} \times 11$	15.3
$\sin(x)$	4, 4, 4	$2^8 \times 14 + 2^7 \times 5$	$2^{12} \times 12$	11.6
$\cos(x)$	4, 4, 4	$2^8 \times 14 + 2^7 \times 5$	$2^{12} \times 12$	11.6
$\tan^{-1}(x)$	4, 4, 4	$2^8 \times 14 + 2^7 \times 5$	$2^{12} \times 12$	11.6
$\log_2(x)$	5, 3, 3	$2^8 \times 14 + 2^7 \times 6$	$2^{11} \times 12$	5.6
2^x	4, 4, 4	$2^8 \times 13 + 2^7 \times 6$	$2^{12} \times 11$	11.5

Table 3. Table Sizes and Compression for 12-Bit Operands.

$f(x)$	Symmetric Bipartite Table n_0, n_1, n_2	Sizes	Standard Table Sizes	Compression
$1/x$	6, 4, 5	$2^{10} \times 17 + 2^{11} \times 7$	$2^{15} \times 15$	15.5
$\sqrt{x}$	5, 5, 6	$2^{10} \times 17 + 2^{10} \times 6$	$2^{16} \times 15$	41.7
$\sin(x)$	6, 4, 6	$2^{10} \times 18 + 2^{11} \times 7$	$2^{16} \times 16$	32.0
$\cos(x)$	6, 4, 6	$2^{10} \times 18 + 2^{11} \times 7$	$2^{16} \times 16$	32.0
$\tan^{-1}(x)$	6, 4, 6	$2^{10} \times 18 + 2^{11} \times 7$	$2^{16} \times 16$	32.0
$\log_2(x)$	7, 3, 5	$2^{10} \times 18 + 2^{11} \times 8$	$2^{15} \times 16$	15.0
2^x	6, 4, 6	$2^{10} \times 17 + 2^{11} \times 8$	$2^{16} \times 15$	29.1

Table 4. Table Sizes and Compression for 16-Bit Operands.

$f(x)$	Symmetric Bipartite Tables n_0, n_1, n_2	Sizes	Standard Table Sizes	Compression
$1/x$	9, 7, 7	$2^{16} \times 25 + 2^{15} \times 9$	$2^{23} \times 23$	99.8
$\sqrt{x}$	8, 7, 9	$2^{15} \times 25 + 2^{16} \times 9$	$2^{24} \times 23$	273.9
$\sin(x)$	8, 8, 8	$2^{16} \times 26 + 2^{15} \times 9$	$2^{24} \times 24$	201.4
$\cos(x)$	8, 8, 8	$2^{16} \times 26 + 2^{15} \times 9$	$2^{24} \times 24$	201.4
$\tan^{-1}(x)$	8, 8, 8	$2^{16} \times 26 + 2^{15} \times 9$	$2^{24} \times 24$	201.4
$\log_2(x)$	9, 7, 7	$2^{16} \times 26 + 2^{15} \times 10$	$2^{23} \times 24$	99.1
2^x	8, 8, 8	$2^{16} \times 25 + 2^{15} \times 10$	$2^{24} \times 23$	196.3

Table 5. Table Sizes and Compression for 24-Bit Operands.

sented in [10]. This method differs from the SBTM in that it assumes that only the most significant fractional bits of x are used for the table lookup. This is appropriate when a low precision starting value is needed, such as the initial approximations employed in multiplication-based divide and square root algorithms [15] - [18]. In Section 4.2, the SBTM is extended to accurately handle this type of approximation.

4.1. The 3-Block Method

In [9], a method is presented for function approximation that uses table lookups followed by addition. With this method, the elementary functions are approximated based on the partial product arrays of their Taylor series expansions. In terms of hardware complexity, the 3-block method is similar to the SBTM, however, the SBTM yields smaller tables.

In Sections 3 and 4 of [9], 3-block approximations are presented for the functions $1/x$, $\ln(x)$, $\log_2(x)$, and 2^x for 24-bit operands, with $0.5 \leq x < 1$.[2] The memory requirements for bipartite tables using the 3-block method presented in [9] and the SBTM are given in Table 6. The goal is to have a maximum approximation error less than 2^{-25}. Since $0.5 \leq x < 1$, the most significant bit of x is guaranteed to be a one and only $n_0 - 1$ bits of x_0 are used as inputs to the tables. Both the SBTM and their method take advantage of this to reduce the size of the tables by a factor of two. The SBTM also takes advantage of the symmetry in $a_1(x_0, x_2)$ to reduce the size of the second table by an additional factor of two. Thus, the symmetric bipartite table for $a_0(x_0, x_1)$ table requires $n_0 + n_1 - 1$ words and the table for $a_1(x_0, x_2)$ requires $n_0 + n_2 - 2$ words. Following the notation presented in [9], l denotes the number of bits in the output operand, plus a designated number of guard digits (i.e., $l = p + g$). In the last column, the compression achieved by the SBTM compared to the method presented in [9] is given, assuming $l = 28$. Based on these values, the SBTM requires 2.5 to 5.2 times less memory than the method presented in [9]. The tables sizes for their method come from Tables 1 and 2 of [9]. The table sizes reported for the SBTM are the minimum tables sizes that will allow a maximum approximation error that is less than 2^{-25}, based on the error equations given in Section 2.2. These tables sizes were also verified by a computer program, which simulates the SBTM and exhaustively tests all possible 24-bit inputs on the interval $0.5 \leq x < 1$.

Compared to the 3-block method in [9], the SBTM requires less memory because it

1. Employs a more accurate method to select the coefficients
2. Does not store the leading zeros (or ones) of $a_1(x_0, x_2)$
3. Takes advantage of symmetry in $a_1(x_0, x_2)$ to reduce the size of the second table

The method presented in [9] is more general than the SBTM, because it allows more than two tables. However, unless the number of tables is quite large, the SBTM still requires less memory. In [20], the SBTM is extended to allow more than two tables.

4.2. Faithful Bipartite Reciprocal Tables

A second method for bipartite table approximations is presented in [10]. This method is designed for bipartite reciprocal tables with $1 \leq x < 2$, which corresponds to the range of normalized IEEE floating point numbers. Their paper presents an algorithmic method for designing bipartite reciprocal tables that takes a $(j+2)$-bit input, which correspond to the most significant fractional bits of x, and produces a j-bit output, which corresponds to the fractional bits of $1/x$, except the most significant bit which is guaranteed to be a one. Also, since $1 \leq x < 2$, the integer bit of x is guaranteed to be one and is not included in the table lookup.

The approach presented in [10] is fundamentally different from the SBTM in that their $(j+2)$-bit input is assumed to correspond to only the most significant fractional bits of x, whereas the SBTM assumes that all the bits of x, except the leading one, are used for the approximation. To allow a fair comparison between the two methods, the SBTM can be modified to take into account the error that occurs by leaving off the least significant bits of x. This is done by representing the n fractional bits of x as x_0, x_1, x_2, and x_3, where x_0, x_1, and x_2 correspond to the $n_0+n_1+n_2 = j+2$ most significant fractional bits of x, and x_3 corresponds to the remaining fractional bits. Since the bits of x_3 are unknown, they are approximated as

$$\delta_3 = 2^{-n_0-n_1-n2-1} - 2^{-n-1} \tag{20}$$

which is halfway between the minimum and maximum values of x_3.

The new values selected for the coefficients are

$$\begin{aligned} a_0(x_0, x_1) &= f(x_0 + x_1 + \delta_2 + \delta_3) \\ a_1(x_0, x_2) &= f'(x_0 + \delta_1 + \delta_2 + \delta_3)(x_2 - \delta_2) \end{aligned} \tag{21}$$

Replacing x_3 by δ_3 in the approximation results in an absolute error of

$$\begin{aligned} \epsilon_4 &= f(x_0 + x_1 + x_2 + x_3) - f(x_0 + x_1 + x_2 + \delta_3) \\ &= \mid \sum_{i=1}^{\infty} f^i(x_0 + x_1 + x_2)^i (x_3^i - \delta_3^i) \mid \\ &\approx \mid f'(x_0 + x_1 + x_2)(x_3 - \delta_3) \mid \end{aligned} \tag{22}$$

[2] In [9], the actual functions computed are $1/(1-x)$, $\ln(1-x)$, $\log_2(1-x)$, and the substitution $x = 1 - y$ is used to transform y, where $0.5 \leq y < 1$, to x, where $0 < x \leq 0.5$.

$f(x)$	Symmetric Bipartite Tables n_0, n_1, n_2	Sizes	Table Sizes From [9]	Compression
$1/x$	10, 7, 7	$2^{16}l + 2^{15}(l-16)$	$2^{17}l + 2^{16}l$	2.5
$\ln(x)$	9, 7, 8	$2^{15}l + 2^{15}(l-16)$	$2^{16}l + 2^{16}l$	2.8
$\log_2(x)$	10, 6, 8	$2^{15}l + 2^{16}(l-15)$	$2^{17}l + 2^{15}l$	2.6
2^x	8, 7, 9	$2^{14}l + 2^{15}(l-15)$	$2^{17}l + 2^{15}l$	5.2

Table 6. The SBTM and the 3-Block Method for 24-Bit Operands.

Output Bits (j)	Input Bits $(j+2)$ n_0, n_1, n_2	$a_0(x_0, x_2)$ Table Size	$a_1(x_0, x_2)$ Table Size	Total SBTM Size	Total Table Size [10]
$3k-2$	$k+1, k-1, k$	$2^{2k}(3k+1)$	$2^{2k}k$	$2^{2k}(4k+1)$	$2^{2k}(5k+2)$
$3k-1$	$k+1, k, k$	$2^{2k+1}(3k+2)$	$2^{2k}k$	$2^{2k}(7k+5)$	$2^{2k}(8k+4)$
$3k$	$k+2, k-1, k+1$	$2^{2k+1}(3k+3)$	$2^{2k+2}(k+1)$	$2^{2k}(10k+10)$	$2^{2k}(14k+10)$

Table 7. Table Sizes for $1/x$, with $k = \lceil j/3 \rceil$ and $n_0 + n_1 + n_2 = j + 2$.

Since $\mid x_3 - \delta_3 \mid < 2^{-n_0-n_1-n_2-1}$, this error is bounded by

$$\epsilon_4 < \mid f'(\xi_1) \mid 2^{-n_0-n_1-n_2-1} \tag{23}$$

To ensure faithful rounding, the sum of the errors must satisfy the constraint

$$\epsilon_0 + \epsilon_1 + \epsilon_2 + \epsilon_3 + \epsilon_4 \leq 2^{-p_f-1} \tag{24}$$

where ϵ_0, ϵ_1, ϵ_2 and ϵ_3 have the values defined in Section 2.2.

For reciprocal approximations on [1, 2), $\mid f'(\xi_1) \mid \leq 1$ and $\mid f''(\xi_2) \mid \leq 2$. Since the reciprocal table uses $j+2$ input bits and j output bits and the leading bits of the input and output are known to be one, $n_0 + n_1 + n_2 = j + 2$ and $p_f = j + 1$. Letting $g = 3$ and assuming $n_1 \geq 1$, gives the constraint

$$2^{-2n_0-n_1-1}(1 + 1/2) + 2^{-j-5} + 2^{-j-3} \leq 2^{-j-2} \tag{25}$$

Therefore, if the constraints

$$2n_0 + n_1 \geq j + 3 \tag{26}$$

and

$$n_0 + n_1 + n_2 = j + 2 \tag{27}$$

are simultaneously satisfied, faithful rounding is guaranteed. Constraint 26 limits the first term in Constraint 25 to be less than or equal to $2^{-j-4} + 2^{-j-5}$, which ensures that the total error is less than 2^{-j-2} and the final results are faithful. Constraint 27 ensures that the number of input bits to the table is $j + 2$. Since 3 guard bits are used, the number of bits for the coefficients are $p_0 = j + 3$ and $p_1 = j + 2 - n_0 - n_1 = n_2$.

Similar to [10], if $k = \lceil j/3 \rceil$, the relationship between j and k can be used to determine the values for n_0, n_1, and n_2, that minimize the table size. This is shown in Table 7, along with the total table sizes for the SBTM and those reported in Table 3 of [10]. Compared to their method, the SBTM requires less memory. For example, if $j = 15$ and $k = 5$, their technique requires a total of 81,920 bits, whereas the SBTM requires only 61,440 bits (a 25% reduction in memory). This reduction comes from the fact that the SBTM decreases the size of the table for $a_1(x_0, x_2)$ by a factor of 2, so that it requires only $2^{n_0+n_2-1}$ words. The SBTM also has the advantages that it provides a closed form solution for selecting the coefficients and can be applied to a wide range of functions. Their technique has the advantage that the coefficients can be more easily Booth encoded, since the coefficients are always in borrow-save form [10]. Also, their technique does not require the small amount of additional logic needed to take advantage of the symmetry in $a_1(x_0, x_2)$.

5. Conclusions

This paper introduces a new method for bipartite table approximations. Compared to previous bipartite table approximations, the SBTM requires less memory and has a tighter error bound. It takes advantage of symmetry in the coefficients to reduce the size of one of the tables by a factor of two, at the cost of a few exclusive-or gates. The SBTM can be used to approximate a wide range of elementary functions. It can also be used to obtain initial approximations for multiplication-based divide and square root algorithms, with the modifications presented in Section 4.2. Compared to conventional table lookup methods, the memory requirements are reduced by two *orders of magnitude* when the operand size is 24 bits or greater. In [20], the SBTM is extended to allow more than two lookup tables.

References

[1] P. Farmwald, "High Bandwidth Evaluation of Elementary Functions," in *Proceedings of the 5th Symposium on Computer Arithmetic*, pp. 139–142, 1981.

[2] M. J. Schulte and E. E. Swartzlander, Jr., "Exact Rounding of Certain Elementary Functions," in *Proceedings of the 11th Symposium on Computer Arithmetic*, pp. 138–145, 1993.

[3] M. J. Schulte and E. E. Swartzlander, Jr., "Hardware Designs for Exactly Rounded Elementary Functions," *IEEE Transactions on Computers, Special Issue on Computer Arithmetic*, vol. C-44, pp. 964–973, 1994.

[4] V. K. Jain and L. Lin, "High-Speed Double Precision Computation of Nonlinear Functions," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 107–114, 1995.

[5] V. K. Jain *et al.*, "DSP Coprocessor Cell for Systolic Arrays," in *VLSI Signal Processing, VI*, pp. 480–488, 1993.

[6] P. T. P. Tang, "Table-Lookup Algorithms for Elementary Functions and Their Error Analysis," in *Proceedings of the 10th Symposium on Computer Arithmetic*, pp. 232–236, 1991.

[7] W. E. Ferguson and T. Brightman, "Accurate and Monotone Approximations of Some Transcendental Functions," in *Proceedings of the 10th Symposium on Computer Arithmetic*, pp. 237–244, 1991.

[8] I. Koren, "Evaluating Elementary Functions in a Numerical Coprocessor based on Rational Approximations," *IEEE Transactions on Computers*, vol. C-40, pp. 1030–1037, 1990.

[9] H. Hassler and N. Takagi, "Function Evaluation by Table Look-up and Addition," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 10–16, 1995.

[10] D. D. Sarma and D. W. Matula, "Faithful Bipartite Rom Reciprocal Tables," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 17–29, 1995.

[11] I. Koren, *Computer Arithmetic Algorithms*, Prentice Hall, Englewood Cliffs, NJ, 1993.

[12] W. F. Wong and E. Goto, "Fast Evaluation of Elementary Functions in Single Precision," *IEEE Transactions on Computers*, vol. C-44, pp. 453–457, 1995.

[13] W. Cody and W. Waite, *Software Manual for the Elementary Functions*. Prentice-Hall, Englewood Cliffs, NJ, 1980.

[14] J. S. Walther, "A Unified Approach for Elementary Functions," in *Spring Joint Computer Conference*, pp. 379–385, 1971.

[15] M. D. Ercegovac, T. Lang, and P. Montuschi, "Very-High Radix Division with Selection by Rounding and Prescaling," *IEEE Transactions on Computers*, vol. C-43, pp. 909–918, 1994.

[16] T. Lang and P. Montuschi, "Very-High Radix Combined Division and Square Root with Prescaling and Selection by Rounding," in *Proceedings of the 12th Symposium on Computer Arithmetic*, pp. 124–131, 1995.

[17] M. J. Flynn, "On Division by Functional Iteration," *IEEE Transactions on Computers*, vol. C-19, pp. 702–706, 1970.

[18] C. V. Ramamoorthy and J. Goodman, "Some properties of Iterative Square-Rooting Methods Using High-Speed Multiplication," *IEEE Transactions on Computers*, vol. C-21, pp. 837–847, 1972.

[19] C.-L. Wey, "On Design of Efficient Square Generator," in *Proceedings of the International Conference on Computer Design*, pp. 506-511, 1996.

[20] M. J. Schulte and J. E. Stine, "Accurate Function Approximations by Symmetric Table Lookup and Addition," *submitted to the 11th International Conference on Application-specific, Systems, Architectures, and Processors*, 1997.