# Low Precision Table Based Complex Reciprocal Approximation


Pouya Dormiani
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90024, USA
Email: pouya@cs.ucla.edu

Miloš D. Ercegovac
4731H Boelter Hall
Computer Science Department
University of California at Los Angeles
Los Angeles, CA 90024, USA
Email: milos@cs.ucla.edu

Jean-Michel Muller
CNRS-Laboratoire CNRS-ENSL-INRIA-UCBL LIP
Ecole Normale Supérieure de Lyon
46 Allée d'Italie
69364 Lyon Cedex 07, France
Email: Jean-Michel.Muller@ens-lyon.fr



***Abstract*—A recently proposed complex valued division algorithm[1] designed for efficient hardware implementations requires a prescaling step by a constant factor. Techniques for obtaining this prescaling factor have been mentioned by the authors, which serves to justify the feasibility of the algorithm but is inadequate for obtaining efficient implementations. Table based solutions are formulated in this paper for obtaining the prescaling factor, a low precision reciprocal approximation for a complex value, using techniques adopted from univariate function approximations. Two separate designs are proposed, one using a single table (a reference design) and another using generalized multipartite tables. The main contribution of this work is the extension of generalized multipartite table methods to a function of two variables. The multipartite tables derived were up to 67% more memory efficient than their single table counterparts.**


## I. Introduction

Two classes distinguished in [2] are used in general to evaluate functions,

- **Compute-bound:** Evaluates the function computationally using, for example, iterative techniques such as digit-recurrence or multiplicative methods [3]. Combinational logic and registers dominate the hardware cost in these schemes.
- **Table-Bound:** Store the evaluated function in a table indexed via the function arguments. Multiple tables may be used in which the value could be directly stored or reconstructed. This method relies on memories such as ROMs (Read Only Memory) which constitute the majority of hardware costs. In the discussions that follow we will use the terms table and ROM interchangeably.

The class of compute-bound methods are more flexible as they can scale to perform high precision approximations by trading performance for accuracy through evaluating the approximation on the fly during execution. Indeed the division algorithm in [1] is itself a compute-bound approach for evaluating complex valued division. Hybrid schemes where logic and ROMs are intermixed to achieve a desired memory and latency tradeoff are also possible.

We denote $\hat{x}$ as an approximation to $x$ where $\hat{x}$ is a lower precision value of $x$ obtained by truncation, rounding etc. which has fewer significant digits and larger unit in the last position (ulp). We distinguish an approximation to a function $f$ using notation $\tilde{f}$, for example by an approximating polynomial obtained via Taylor expansion about a point. We use the notation $\imath = \sqrt{-1}$, and given $x = x^{\mathcal{R}} + \imath\, x^{\mathcal{I}}$, $\Re(x) = x^{\mathcal{R}}$, $\Im(x) = x^{\mathcal{I}}$, we define two norms $\| x \|_\infty = \max(| x^{\mathcal{R}} |, | x^{\mathcal{I}} |)$ and $| x | = \sqrt{(x^{\mathcal{R}})^2 + (x^{\mathcal{I}})^2}$. An approximation to a complex value is defined as an approximation to the real and imaginary parts respectively, i.e., $\hat{x} = \hat{x}^{\mathcal{R}} + \imath\, \hat{x}^{\mathcal{I}}$.

| $r$ | $a$ | $\sigma$ | $\epsilon_S \leq$ |
|---|---|---|---|
| 2 | 1 | 4 | 7/64 |
| 4 | 2 | 5 | 13/512 |
| 4 | 3 | 3 | 3/64 |
| 8 | 6 | 4 | 11/512 |
| 8 | 7 | 3 | 3/128 |

TABLE I: Upper bound for $\epsilon_S$ values given specific parameters of the complex division algorithm presented in [1].

Given a value $d = d^{\mathcal{R}} + \imath\, d^{\mathcal{I}}$ with reciprocal $f(d) = 1/d$, we define a prescaling factor $K$ which is a reciprocal approximation with error characterized by

$$\left\| K - 1/\hat{d} \right\|_\infty < \epsilon_K \tag{1}$$

with

$$K = \widehat{\tilde{f}(\hat{d})} \tag{2}$$

$$\frac{1}{\hat{d}} = \frac{\hat{d}^{\mathcal{R}} - \imath \hat{d}^{\mathcal{I}}}{(\hat{d}^{\mathcal{R}})^2 + (\hat{d}^{\mathcal{I}})^2} \tag{3}$$

i.e., $K$ is an approximation to the evaluation of $\tilde{f}(\hat{d})$. Complex reciprocal approximations play an important role in a digit-recurrence based complex division algorithm presented in [1] which computes $x/y$ by prescaling the computation with $K$ such that $\frac{Kx}{Ky}$ is the actual value computed. Since the purpose in finding $K$ is to build efficient tables for this algorithm, its desired accuracy is characterized here in the same manner as expressed in [1], which requires that

$$\| Kd - 1 \|_\infty < \epsilon_S \tag{4}$$

where $\epsilon_S$ is a known constant for a given choice of algorithm parameters. Table I shows various $\epsilon_S$ values for given parameters of the division algorithm which are derived from Eq. (5) given in [1] for a radix $r$ algorithm with digits in $\{-a, \ldots, a\}$ and $\sigma$ fractional bits in the residual estimate.

$$2\, a\, \epsilon_S + \frac{1}{2} + 2^{-\sigma} \leq \frac{1}{r}\left(a + \frac{1}{2} + 2^{-\sigma}\right) \tag{5}$$

Real valued reciprocal approximations have been studied in literature [4][5][6], but the complex reciprocal approximation is a bivariate function. This paper formulates some table based solutions for obtaining the complex reciprocal approximation $K$ by extending techniques developed for univariate functions. To the best of our knowledge table based methods for two variable functions have not been studied in computer arithmetic literature. Looking at Table I one can see that the desired accuracy of $K$ is very low. Our approach of using linear programs (LP) to obtain these tables does not scale to obtain high precision approximations and is only useful for low precision reciprocal approximations–the LP size grows exponentially.

### A. Problem Formulation

For a given value $d = d^{\mathcal{R}} + \imath d^{\mathcal{I}}$ with constraint

$$\frac{1}{2} \leq \| d \|_{\infty} < 1 \tag{6}$$

and precision $n$ such that $d$ is representable in two's complement form by

$$d^{\mathcal{R}} = d_0^{\mathcal{R}}.d_1^{\mathcal{R}}d_2^{\mathcal{R}}\dots d_n^{\mathcal{R}},\ d^{\mathcal{I}} = d_0^{\mathcal{I}}.d_1^{\mathcal{I}}d_2^{\mathcal{I}}\dots d_n^{\mathcal{I}} \tag{7}$$

Let $\hat{d}$ be obtained by rounding to nearest $d$ to the $q^{\text{th}}$ fractional position such that,

$$\hat{d}^{\,\mathcal{R}} = rnd(d^{\mathcal{R}}, q),\ \hat{d}^{\,\mathcal{I}} = rnd(d^{\mathcal{I}}, q) \tag{8}$$

$$\left\| d - \hat{d} \right\|_{\infty} \leq \frac{1}{2}2^{-q} \tag{9}$$

where $ulp(\hat{d}) = 2^{-q}$. Using $\hat{d}$ we would like to obtain a $K$ such that constraints (1) and (4) are satisfied for some given $\epsilon_S$ from Table I.

## II. Single Table Design

In a single table design the estimate $\hat{d}$ is used to address one large table which stores the corresponding reciprocal approximation rounded to $t$ fractional positions, i.e.,

$$K = rnd\left(\frac{1}{rnd(d,q)}, t\right)$$

Error analysis shown in [7] is used to derive the following constraint which relates $q$, $t$ and $\epsilon_S$,

$$2^{-t} + 2^{-q+1} < \epsilon_S \tag{10}$$

Because $\hat{d}$ is obtained by rounding to nearest it could possibly take on the value $\| \hat{d} \|_{\infty} = 1$, and its representation requires an extra bit to the left of the binary point. For now, assume that $\| \hat{d} \|_{\infty} < 1$ (we will later incorporate $\| \hat{d} \|_{\infty} = 1$ as a special case). First we look at some properties of the reciprocal function $f$–we will refer to $f$ interchangeably as a univariate function when its argument is a complex value i.e., $f(d)$, and also as a bivariate function with arguments $f(d^{\mathcal{R}}, d^{\mathcal{I}})$.

$$f(d) = \frac{1}{d^{\mathcal{R}} + \imath\, d^{\mathcal{I}}} = \frac{d^{\mathcal{R}} - \imath\, d^{\mathcal{I}}}{(d^{\mathcal{R}})^2 + (d^{\mathcal{I}})^2}$$

1) $f(-d) = -f(d)$
2) $f(d^{\mathcal{R}}, d^{\mathcal{I}}) = -[\Im(f(d^{\mathcal{I}}, d^{\mathcal{R}})) + \imath\, \Re(f(d^{\mathcal{I}}, d^{\mathcal{R}})]$
3) $f(-d^{\mathcal{R}}, d^{\mathcal{I}}) = -\Re(f(d^{\mathcal{I}}, d^{\mathcal{R}})) + \imath\, \Im(f(d^{\mathcal{I}}, d^{\mathcal{R}}))$, etc.

Two techniques are used to reduce the address width to the table (which is $2(q+1)$ bits wide in a naïve implementation),

- Using the first property, we eliminate the sign of $\hat{d}$ when addressing the ROM, i.e., the implementation now requires a $2q$ bit wide address,

$$|\, \hat{d}^{\,\mathcal{R}} \,| = 0.\alpha_1^{\mathcal{R}}\alpha_2^{\mathcal{R}}\dots\alpha_q^{\mathcal{R}},\ |\, \hat{d}^{\,\mathcal{I}} \,| = 0.\alpha_1^{\mathcal{I}}\alpha_2^{\mathcal{I}}\dots\alpha_q^{\mathcal{I}} \tag{11}$$

- Since $\| d \|_{\infty} \geq \frac{1}{2}$ we know that either $\alpha_1^{\mathcal{R}} = 1$ or $\alpha_1^{\mathcal{I}} = 1$ (or both)–as originally proposed in [1]. If the following address was formed: $\alpha_1^{\mathcal{R}}\alpha_2^{\mathcal{R}}\dots\alpha_q^{\mathcal{R}}\alpha_1^{\mathcal{I}}\alpha_2^{\mathcal{I}}\dots\alpha_q^{\mathcal{I}}$ then we could check if $\alpha_1^{\mathcal{R}} = 1$, in which case the address can be reduced by one bit by making $\alpha_1^{\mathcal{R}}$ implicit, i.e., addressing the ROM with $\alpha_2^{\mathcal{R}}\alpha_3^{\mathcal{R}}\dots\alpha_q^{\mathcal{R}}\alpha_1^{\mathcal{I}}\alpha_2^{\mathcal{I}}\dots\alpha_q^{\mathcal{I}}$. Otherwise, it must be true that $\alpha_1^{\mathcal{I}} = 1$, and the ROM is addressed with $\alpha_2^{\mathcal{I}}\alpha_3^{\mathcal{I}}\dots\alpha_q^{\mathcal{I}}\alpha_1^{\mathcal{R}}\alpha_2^{\mathcal{R}}\dots\alpha_q^{\mathcal{R}}$ and the second property of the reciprocal function is used to recover the result. This reduces the number of address bits to $2q-1$ (halving the memory required) while introducing little additional overhead.

As previously mentioned the scenario in which $\| \hat{d} \|_{\infty} = 1$ is incorporated as a special case by including a smaller ROM (called $\text{ROM}_s$), having $q$ address bits. One possible implementation could be that $\text{ROM}_s[A] = f(1, A)$, where $A = \alpha_1\alpha_2\dots\alpha_q$ and the square brackets denote performing a look-up in the ROM at address $A$–again note that all the different permutations of arguments, e.g. $f(\hat{d}^{\,\mathcal{R}}, -1)$, $f(1, -\hat{d}^{\,\mathcal{I}})$, etc. can be obtained by performing a conditional final swapping of real and imaginary parts and/or a negation.

So far we have limited our discussion to the addressing scheme of the ROM, however the width of the actual contents of the ROM is also of interest which we will now discuss. We already know that each entry in the ROM (including $\text{ROM}_s$) contains a real and imaginary part each with $t$ fractional bits, but how many bits are required to the left of the binary point? It is known that $K^{\mathcal{R}}$ is positive and $K^{\mathcal{I}}$ is negative for positive values of $d^{\mathcal{R}}$ and $d^{\mathcal{I}}$ respectively. Since $1/2 \leq \| \hat{d} \|_{\infty} \leq 1$ and the ROM only stores values for positive $d^{\mathcal{R}}$ and $d^{\mathcal{I}}$, then if we let $U^{\mathcal{R}}$ and $U^{\mathcal{I}}$ denote the values to be stored in the ROM (i.e., $K$ is obtained via conditional negation of $U^{\mathcal{R}}$ and $U^{\mathcal{I}}$), then we know that $0 \leq U^{\mathcal{R}} \leq 2$ and $-2 \leq U^{\mathcal{I}} \leq 0$. The imaginary part $U^{\mathcal{I}}$ is stored in magnitude form thus both $U^{\mathcal{R}}$ and $U^{\mathcal{I}}$ can be represented in $2+t$ bits. Each entry in the ROM (including $\text{ROM}_s$) stores $2(2+t)$ bits, of which the first $2+t$ bits is $U^{\mathcal{R}}$ and the second $2+t$ bits is $U^{\mathcal{I}}$. The cost in memory (bits) of the proposed implementation is therefore

$$\overbrace{2(2+t)\times 2^{2q-1}}^{\text{Main ROM}} + \overbrace{2(2+t)\times 2^{q}}^{\text{Special Case } (\text{ROM}_s)} \text{ bits} \tag{12}$$

The contents of the ROMs can be verified by perfect induction to ensure that the satisfy the error bound dictated in Eq. (10). Also note that even though there are two ROMs in the aforementioned design, this is a consequence of trying to

| $r$ | $a$ | $\sigma$ | $\epsilon_s$ | $q$ | $t$ | $\approx$ KBits |
|---|---|---|---|---|---|---|
| 2 | 1 | 4 | 7/64 | 5 | 5 | 7.44 |
| 4 | 2 | 5 | 13/512 | 7 | 7 | 146.25 |
| 4 | 3 | 3 | 3/64 | 6 | 6 | 33.0 |
| 8 | 6 | 4 | 11/512 | 7 | 8 | 162.5 |
| 8 | 7 | 3 | 3/128 | 7 | 7 | 146.25 |

TABLE II: Sampling of design space for single table memory requirements given specific parameters of the complex division algorithm presented in [1].

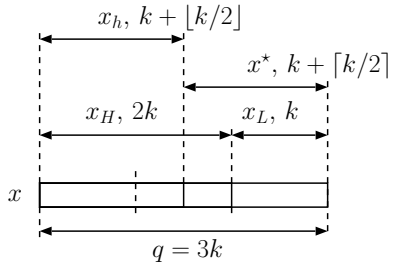


Fig. 1: Partitioning of input $x$ ($y$ partitioned similarly) into $x_H$, $x_L$, and $x_h$, such that $x = x_H + 2^{-2k}x_L$ and $x = x_h + 2^{-k-\lfloor k/2 \rfloor}x^\star$.

reduce the memory requirements and should not be confused with multiple table designs which are discussed next.

## III. Multiple Table Designs

In the single table design a precise value for $1/\hat{d}$ was calculated and rounded to $t$ fractional positions to obtain $K$. The reciprocal function itself however can be approximated, which can reduce the required memory size. Approximating the function via multiple tables was originally proposed in [4] as bipartite (two) tables, which was extended and formalized by [2], reformulated as a linearization problem in [8], and extended to multipartite (greater than two) tables in [9][2].

In this section we define $g$ as $g(x,y) = x/(x^2+y^2)$ such that, $f(d) = g(d^{\mathcal{R}}, d^{\mathcal{I}}) - \imath\, g(d^{\mathcal{I}}, d^{\mathcal{R}})$. Let $\| \tilde{f}(\hat{d}) - f(\hat{d}) \|_\infty < \epsilon_K$, then from [7], the error $\epsilon_K$ is related to $\epsilon_S$ such that if $\epsilon_K < \frac{1}{2}\epsilon_S - 2^{-q}$ is satisfied then the approximating function $\tilde{f}$ can replace $f$. The function $\tilde{f}$, which was originally implemented as one large table, can be approximated in a manner which decomposes it into a sum of terms where each term is a function of *fewer* address bits than required by $f$.

Note that $\tilde{f}$ can be obtained from $\tilde{g}$. Let $\tilde{g}$ be the first order Taylor expansion of $g$ about $(x_0, y_0)$, where $g_x$ denotes the first partial derivative of $g$ in $x$

$$\tilde{g} = g(x_0,y_0) + (x-x_0)g_x(x_0,y_0) + (y-y_0)g_y(x_0,y_0) \quad (13)$$

The arguments $x$ and $y$, which are both positive numbers with $q$ fractional bits were partitioned as shown in Fig. 1, and the point of expansion $(x_0, y_0)$ in Eq. (13) was chosen to be $(x_H, y_H)$, resulting in

$$\begin{aligned}\tilde{g} &= g(x_H,y_H) + x_L 2^{-2k} g_x(x_H,y_H) + y_L 2^{-2k} g_y(x_H,y_H) \\ &\approx g(x_H,y_H) + x_L 2^{-2k} g_x(x_h,y_h) + y_L 2^{-2k} g_y(x_h,y_h)\end{aligned} \quad (14)$$

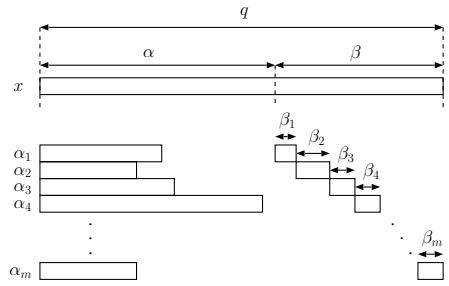


Fig. 2: Partitioning an argument $x$ into $x_\alpha$ and $m$ pairs $(x_{\alpha_1}, x_{\beta_1}), \ldots, (x_{\alpha_m}, x_{\beta_m})$.

with the assumption that $g_x$ and $g_y$ are approximately constant at the granularity of $x_h$.

Each term in the sum of Eq. (14) is implemented by a table. The value $\tilde{g}(x,y)$ is determined by addressing all tables in parallel and summing their outputs which requires a ternary addition. The cost in bits of this scheme is,

$$\begin{aligned}&\overbrace{(p_H + t_H) \times 2^{2k}}^{g(x_H,y_H)} + \overbrace{(p_x + t_x) \times 2^{2(k+\lfloor k/2 \rfloor)+k}}^{x_L 2^{-2k} g_x(x_h,y_h)} + \\ &\overbrace{(p_y + t_y) \times 2^{2(k+\lfloor k/2 \rfloor)+k}}^{x_L 2^{-2k} g_y(x_h,y_h)} + \overbrace{2(2+t) \times 2^q}^{\text{ROM}_s} \text{ bits}\end{aligned} \quad (15)$$

where $p_H$, $t_H$ is the number of integer and fractional bits (i.e., the width) of the table (same for $x$ and $y$ subscripts). This approximation is called bipartite in [1] even though it has three tables, which is a consequence of it being an approximation of a function in two variables–the first order Taylor approximation of a univariate function and a similar treatment yields two tables. Two techniques discussed in the aforementioned literature will be employed to further reduce the size of these tables.

### A. Generalized Partitioning

Looking back at the partitioning of the arguments in Fig. 1, one may wonder why this particular partitioning was chosen by the authors. This partitioning is based on work presented in [2] and gives algorithmic bounds on the table size. A much more general partitioning scheme presented in [8] can be used to obtain the optimal partitioning by exhausting all partitions. The partitioning scheme introduced by [8] is shown in Fig. 2, where the argument is partitioned first into two parts $x_\alpha$ and $x_\beta$ such that $x = x_\alpha + 2^{-\alpha}x_\beta$. Then $x_\beta$ is subsequently partitioned into $m$ parts, $x_{\beta_1}, \ldots, x_{\beta_m}$, where each $x_{\beta_i}$ has a prefix $x_{\alpha_i}$, i.e., they form a pair $(x_{\alpha_i}, x_{\beta_i})$.

$$o_1 = 0 \qquad \alpha_i < \alpha$$
$$o_i = \sum_{j=1}^{i-1} \beta_j,\ i > 1 \qquad x = x_\alpha + 2^{-\alpha} \sum_{i=1}^{m} \left( x_{\beta_i} 2^{-o_i} \right)$$

### B. Multipartite Approximations

In this section an approach is presented to obtain a linearized multipartite approximation $\tilde{g}$ using the generalized partitioning scheme in section III-A. Observing that $q$ was rather small for

single precision tables and ranged from 5 to 9, any useful multipartite scheme derived should be even smaller, which reduces computational complexity of determining such tables considerably. Because of the very moderate problem size, the problem is simply formulated as a linear program (LP) which is then solved via available LP solvers.

The approach to finding the optimal sized tables for a given $\epsilon_S$ is as follows: for each partition of the general partitioning performed on the arguments, generate an LP which minimizes the L1 norm (least absolute value, $|x|$) of all errors for all possible points $\hat{d}^{\mathcal{R}}$ and $\hat{d}^{\mathcal{I}}$. If the LP is infeasible then the current partitioning scheme can not yield the desired level of accuracy. Among the feasible choices we pick the partition which has the minimum memory requirements.

Let $\tilde{g}$ approximate the reciprocal function with the partitioning discussed in section III-A

$$\begin{aligned}\tilde{g} = g(x_\alpha, y_\alpha) + 2^{-\alpha}\sum_{i=1}^{m} x_{\beta_i} 2^{-o_i} g_x(x_{\alpha_i}, y_{\alpha_i}) \\ + 2^{-\alpha}\sum_{i=1}^{m} y_{\beta_i} 2^{-o_i} g_y(x_{\alpha_i}, y_{\alpha_i})\end{aligned} \tag{16}$$

from which $2m+1$ tables are derived: one table for the initial value denoted $TIV$ and $m$ tables in $x$ and $y$ providing offsets denoted $TO_i^x$ and $TO_i^y$ respectively–note that for each offset, $x_{\alpha_i}$ replaces $x_\alpha$.

$$TIV(x_\alpha, y_\alpha) = g(x_\alpha, y_\alpha) \tag{17}$$

$$TO_i^x(x_{\alpha_i}, y_{\alpha_i}, x_{\beta_i}) = 2^{-\alpha} x_{\beta_i} 2^{-o_i} g_x(x_{\alpha_i}, y_{\alpha_i}) \tag{18}$$

$$TO_i^y(x_{\alpha_i}, y_{\alpha_i}, y_{\beta_i}) = 2^{-\alpha} y_{\beta_i} 2^{-o_i} g_y(x_{\alpha_i}, y_{\alpha_i}) \tag{19}$$

The concept of using variable sized prefixes for each $x_{\beta_i}$ was originally proposed in [8] and could possibly reduce the table size for each offset table. As before, it is assumed that the derivatives are approximately constant over regions with granularity $\alpha_i$. In order to simplify the linearization procedure and speed up the run time performance a heuristic can be applied when determining the derivatives in Eq. (16): for each derivative $g_x(x_{\alpha_i}, y_{\alpha_i})$, the average derivative of the four corners of the approximating region is used instead, i.e., if we let $\overline{x_{\alpha_i}} = x_{\alpha_i} + 2^{-\alpha_i} - 2^{-q}$ (similarly define for $\overline{y_{\alpha_i}}$) then

$$\begin{aligned}TO_i^x(x_{\alpha_i}, y_{\alpha_i}, x_{\beta_i}) = \frac{2^{-\alpha} x_{\beta_i} 2^{-o_i}}{4}\Big[g_x(x_{\alpha_i}, y_{\alpha_i}) + \\ g_x(\overline{x_{\alpha_i}}, y_{\alpha_i}) + g_x(x_{\alpha_i}, \overline{y_{\alpha_i}}) + g_x(\overline{x_{\alpha_i}}, \overline{y_{\alpha_i}})\Big]\end{aligned}$$

Some notation must first be developed before expressing the LP. Let $x$ be represented as $i_x 2^{-q}$ where $i_x$ is a positive integer less than $2^q$ (similarly define $i_y$), where $i_x$ and $i_y$ are used as indices to distinguish variables. Also define $i_x[k]$ to be the integer formed by taking the most significant $k$ bits of the $q$-bit vector which represents $i_x$.

The error bounds for the LP must be carefully examined–if each term in the expansion of Eq. (14) is rounded to some fractional position, then these rounding errors must be incorporated. For a partitioning $\alpha, (\alpha_1, \beta_1), \ldots (\alpha_m, \beta_m)$, the required amount of memory bits can be expressed with

$$\begin{aligned}(p+t)\times 2^{2\alpha} + \sum_{i=1}^{m}(p_i^x + t_i^x)\times 2^{2\alpha_i+\beta_i} \\ + \sum_{i=1}^{m}(p_i^y + t_i^y)\times 2^{2\alpha_i+\beta_i} + \overbrace{2(2+t)\times 2^q}^{\text{ROM}_s}\end{aligned} \tag{20}$$

where $p$, $t$ denotes the number of integer and fractional bits respectively of $TIV$ entries, and $p_i$, $t_i$ denotes the number of integer and fractional bits respectively of $TO_i^x$ and $TO_i^y$ entries respectively. The total rounding error ($\epsilon_R$) of the terms is,

$$\| \tilde{f}(\hat{d}) - \widehat{\tilde{f}(\hat{d})} \|_\infty = \epsilon_R \leq \frac{1}{2}\left(2^{-t} + \sum_{i=1}^{m} 2^{-t_i}\right) \tag{21}$$

which affects the error bounds as such

$$\epsilon_K < \tfrac{1}{2}\epsilon_S - 2^{-q} - \epsilon_R \tag{22}$$

The value of $\epsilon_R$ is not known a priori. The number of fractional positions to which each table will be rounded to depends on the error slack ($\frac{1}{2}\epsilon_S - 2^{-q} - \epsilon_K$) which is dependent on the quality of the attainable linearization for a given partition. Minimizing Eq. (20) according to constraints in Eqs. (21) and (22) can be done by brute force because of the small problem size. Therefore, when formulating the LP the errors will not be bounded, instead the maximum error, $\epsilon_K$, obtained from the LP solution will be checked against $\frac{1}{2}\epsilon_S - 2^{-q}$. If $\epsilon_K$ is greater, then the current partition for the given precision $q$ is infeasible as it exhibits more than the permissible amount of error. If $\epsilon_K$ is less, then the difference ($\epsilon_R$) will be distributed into rounding errors for the tables in a manner which minimizes the total number of required bits.

The LP problem using the heuristic described to obtain the offsets can then be written as,

**Objective:**

$$\min\left(\sum_{i_x=0}^{2^q-1}\sum_{i_y=0}^{2^q-1}\epsilon_{i_x,i_y}\right)$$

**Variables:**

$$\begin{cases}\text{for } i_x[\alpha], i_y[\alpha] = 0, \ldots, 2^\alpha - 1 \\ t_{i_x[\alpha], i_y[\alpha]}\end{cases}$$

**Constraints:**

for $i_x, i_y = 0, \ldots, 2^q - 1$

$$\begin{aligned}\epsilon_{i_x,i_y} = \Big| g(i_x, i_y) - t_{i_x[\alpha],i_y[\alpha]} - \sum_{i=0}^{m} TO_i^x(x_{\alpha_i}, y_{\alpha_i}, x_{\beta_i}) \\ - \sum_{i=0}^{m} TO_i^y(x_{\alpha_i}, y_{\alpha_i}, y_{\beta_i})\Big|\end{aligned}$$

| $r$ | $a$ | $\sigma$ | $\epsilon_S$ | $\epsilon_K$ | $\epsilon_R$ | $q$ | $(p,t),[(p_1,t_1)^x(p_1,t_1)^y],\ldots,[(p_m,t_m)^x(p_m,t_m)^y]$ | $\alpha$ $(\alpha_1,\beta_1),\ldots,(\alpha_m,\beta_m)$ | $\approx$ KBits |
|---|---|---|---|---|---|---|---|---|---|
| 2 | 1 | 4 | 7/64 | 0.023 | 0.00092 | 5 | (2, **10**), $[(-2,12)^x(-4,12)^y]$ | **4** (3,1) | **5.688** |
| | | | | 0.036 | 0.0031 | 6 | (2, **8**), $[(-2,10)^x(-4,10)^y]$, $[(-3,13)^x(-4,13)^y]$ | **4** (3,1) (2, 1) | 5.844 |
| | | | | 0.028 | 0.011 | | (2, **6**), $[(-2,9)^x(-4,9)^y]$, $[(-3,11)^x(-5,11)^y]$ | **4** (3,1) (3, 1) | 6.25 |
| 4 | 2 | 5 | 13/512 | 0.0039 | 0.001 | 7 | (2, **9**), $[(-4,16)^x(-7,16)^y]$ | **6** (4,1) | **56.75** |
| | | | | 0.0016 | 0.0033 | | (2, **8**), $[(-4,10)^x(-5,10)^y$ | **6** (5,1) | 64.25 |
| 4 | 3 | 3 | 3/64 | 0.0062 | 0.0016 | 6 | (2, **9**), $[(-3,11)^x(-6,11)^y]$ | **5** (4,1) | **18.5** |
| 8 | 6 | 4 | 11/512 | 0.0016 | 0.0013 | 7 | (2, **9**), $[(-4,12)^x(-5,12)^y]$ | **6** (5,1) | **53.75** |
| 8 | 7 | 3 | 3/128 | 0.0039 | 2.3e-05 | 7 | (2, **15**), $[(-4,18)^x(-7,18)^y]$ | **6** (4,1) | 82.75 |
| | | | | 0.0016 | 0.0023 | 7 | (2, **8**), $[(-4,12)^x(-5,12)^y]$ | **6** (5,1) | **72.25** |

TABLE III: Multipartite results using derivative heuristic.

whose solution will yield the values of all $t_{i_x[\alpha],i_y[\alpha]}$ which will be used to populate the corresponding entry in the initial value table, $TIV$. By minimizing the sum of errors, each error $\epsilon_{i_x,i_y}$ is minimized. Some results using the aforementioned approach have been shown in Table III, which can again be verified by perfect induction.

The width of a table entry can be further reduced by finding the range of its values. For example if all entries in a table range from $[-0.12, 0.12]$ and the table is rounded to 10 fractional positions, then all numbers in the table can be represented with 8 bits. The reason behind this is the value can be represented with $s.sssx_4x_5x_6x_7x_8x_9x_{10}$ where $s$ is the sign bit and $x_i$ are fractional bits. It's obviously wasteful to store unnecessary sign bits as the value can be sign extended to the desired precision after table-lookup. Each table is analyzed in this manner to determine if any of the leading fractional positions can be omitted. If any fractional positions can be omitted, this is denoted by a negative $p$ value, for example, $p_1^x = -4$, $t_1^x = 10$ would mean that for offset table $TO_1^x$, only six bits need to be stored, where the 4 most significant fractional positions are omitted and the remaining 6, up to the desired rounding position, are retained.

## IV. Results

A comparison between the single table and multipartite tables derived is shown in the plot of Fig. 3. From the results it is clear that multipartite tables are much more memory efficient than single table designs, ranging in improvements from 23% up to 67% for the studied design points. The results verify the intuition behind the multipartite scheme and prove that they remain highly applicable to functions of two variables.

## V. Conclusion & Future Work

The objective of this research was to extend the multipartite scheme as developed by [8][2][9] to the complex valued reciprocal, which is a function of two variables. The contribution of this work is the extension of multipartite and linearization techniques and proving their viability via a concrete design problem. Multipartite schemes derived via the proposed extensions were found to be 23% to 67% more efficient than their single table counterparts for the design points studied.

The linearization problem can actually be posed in a manner which allows solving for the optimal offset values [7], as opposed to using the heuristic employed in obtaining the results presented in this paper, which is the topic of future work.

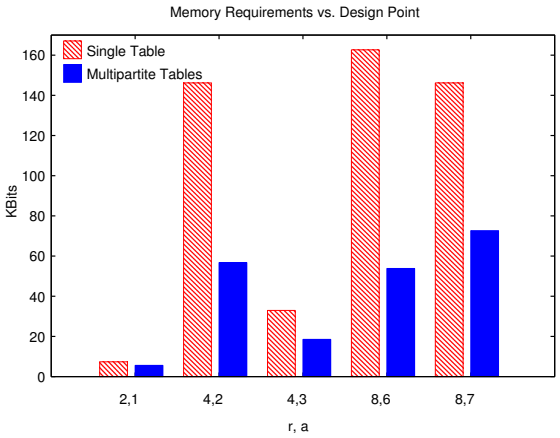


Fig. 3: Single table reciprocal approximation costs vs multipartite costs. The x-axis is the design point under question, which is specified via the radix $r$ and digit-set $a$ used by the division algorithm in [1].

## References


[1] M. D. Ercegovac and J. M. Muller, "Complex division with prescaling of operands," in *Application-Specific Systems, Architectures, and Processors, 2003. Proceedings. IEEE International Conference on*, Jun. 2003, pp. 304–314.

[2] J. M. Muller, "A few results on table based methods." in *Reliable Computing, 5(3)*, 1999, pp. 279–288.

[3] M. Ercegovac and T. Lang, *Digital Arithmetic*. Morgan Kaufmann Publishers, San Francisco, 2004.

[4] D. Das Sarma and D. W. Matula, "Faithful bipartite ROM reciprocal tables," in *Computer Arithmetic, 1995., Proceedings of the 12th Symposium on*, Bath, UK, Jul. 1995, pp. 17–28.

[5] D. DasSarma and D. W. Matula, "Measuring the accuracy of ROM reciprocal tables," *IEEE Transactions on Computers*, vol. 43, no. 8, pp. 932–940, Aug. 1994.

[6] P. Kornerup and D. W. Matula, "Single precision reciprocals by multipartite table lookup," *2005. ARITH-17 2005. 17th IEEE Symposium on Computer Arithmetic*, pp. 240–248, Jun. 2005.

[7] P. Dormiani, "Low precision table based complex reciprocal approximation," computer Science Department, UCLA, Internal Report 2009.

[8] F. de Dinechin and A. Tisserand, "Some improvements on multipartite table methods," in *Computer Arithmetic, 2001. Proceedings. 15th IEEE Symposium on*, Vail, CO, USA, 2001, pp. 128–135.

[9] J. E. Stine, Michael, and J. Schulte, "The symmetric table addition method for accurate function approximation," *Journal of VLSI Signal Processing*, vol. 21, pp. 167–177, 1999.