Complex Division with Prescaling of Operands.

Milos D. Ercegovac, Jean-Michel Muller

Laboratoire de l'Informatique du Parallélisme

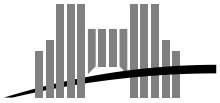
École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON n° 5668

Complex Division with Prescaling of Operands

Miloš D. Ercegovac
Jean-Michel Muller

Février 2003

Research Report N° 2003-10



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

Complex Division with Prescaling of Operands

Miloš D. Ercegovac
Jean-Michel Muller

Février 2003

Abstract

We adapt the radix-r digit-recurrence division algorithm to complex division. By prescaling the operands, we make the selection of quotient digits simple. This leads to a simple hardware implementation, and allows correct rounding of complex quotient. To reduce large prescaling tables required for radices greater than 4, we adapt the bipartite-table method to multiple-operand functions.

Keywords: Computer arithmetic, Complex division, Recurrence division

Résumé

On adapte l'algorithme de division itérative de base r à la division complexe. Par une mise à l'échelle préliminaire des opérandes, on fait en sorte que le choix, à chaque itération, des chiffres de quotient soit élémentaire. Ceci conduit à des implantations matérielles simples, et permet de fournir des divisions avec arrondi correct. Pour permettre la réalisation des tables nécessitées par la mise à l'échelle pour les itérations de base supérieure à 4, on adapte la méthode des tables bipartites aux fonctions à plusieurs opérandes.

Mots-clés: Arithmétique des ordinateurs, Division complexe, Division itérative

Complex Division with Prescaling of Operands

Miloš D. Ercegovac
Computer Science Department
3732 Boelter Hall
University of California at Los Angeles
Los Angeles, CA 90024, USA

Jean-Michel Muller
CNRS-Laboratoire CNRS-ENSL-INRIA-UCBL LIP
Ecole Normale Supérieure de Lyon
46 Allée d'Italie
69364 Lyon Cedex 07, FRANCE

Février 2003

1 Introduction

1.1 Complex division

Complex division is used in many applications such astronomy [4] and non-linear RF measurement [19].

A straightforward way to implement complex division is to use the conventional formula

$$\frac{a+ib}{c+id} = \frac{ac+bd+i(bc-ad)}{c^2+d^2}. \tag{1}$$

Using this formula, however, may lead to overflows during the intermediate computations: c^2+d^2 may overflow, even if the final result is representable in the floating-point format being considered. This severe drawback has been discussed by many authors (e.g., [1, 11]). R.L. Smith [14] suggested another, much more robust, algorithm. It uses the relations:

$$\frac{c+id}{e+if} = \begin{cases} \dfrac{c+d(f/e)}{e+f(f/e)} + i\dfrac{d-c(f/e)}{e+f(f/e)} & (\text{ if } |e| \geq |f|) \\ \dfrac{d+c(e/f)}{f+e(e/f)} + i\dfrac{c-d(e/f)}{f+e(e/f)} & (\text{ if } |e| \leq |f|) \end{cases} \tag{2}$$

Stewart [16] analyzes Smith's algorithm and suggests an even more robust, and even more complicated, algorithm. It is worth being noticed that none of these algorithms guarantees correct rounding of the real and imaginary parts of the obtained quotient. Moreover, these algorithms would require very costly hardware implementation.

Hardware implementation of complex division has been recently considered in [12]. A radix-2 on-line complex division algorithm is used for implementing (1) on a reconfigurable hardware.

The aim of this paper is to adapt to complex division a high-radix (real) digit-recurrence division algorithm with prescaling [9, 10]. For computing x/y, this division algorithm uses the recurrence

$$w[j+1] = rw[j] - q_{j+1}y \tag{3}$$

where $w[0] = x$, and the quotient digits q_j's are chosen in a radix-r redundant digit-set, so that the $w[j]$'s remain bounded. One of the main difficulties is to find a practical quotient-digit function for higher radices. Several solutions have been suggested. Among them, the *prescaling* technique consists in multiplying x and y by a constant K chosen so that Ky becomes close to 1, and then using the residual recurrence to compute $(Kx)/(Ky)$. By doing that, one can choose q_{j+1} by rounding $w[j]$ (or an approximation to $w[j]$ with a few digits only) to the nearest integer, provided that some conditions are satisfied. The idea of scaling operands to make the quotient-digit selection independent of the divisor was proposed in [17] for a decimal computer. This scheme was extended in [18] to an arbitrary radix with signed-digit adder and applied in signed-digit division algorithm. Svoboda's scaling is one-sided, i.e., the (positive) divisor is transformed into the range $1 + \epsilon$. A two-sided prescaling approach is discussed in [6, 7]. The prescaling technique and the radix-4 division are considered in [8, 2, 15].

1.2 Notation

Throughout the paper, i is $\sqrt{-1}$, and if z is a complex number, then $\Re(z)$ and $\Im(z)$ denote the real and imaginary parts of z. The norm $||z||_\infty$ denotes $\max\{|\Re(z)|, |\Im(z)|\}$, whereas $|z|$ denotes the usual complex absolute value

$$\sqrt{(\Re(z))^2 + (\Im(z))^2}.$$

2 Complex Division Algorithm

2.1 General scheme

Assume we wish to compute $q = z/d$, where z and d are complex numbers. We will perform the division as follows:

Prescaling Obtain (through table-lookup and, possibly, some interpolation) a complex scaling factor K such that

$$||Kd - 1||_\infty < 2^{-p}$$

where the integer p is a parameter of the algorithm. Then, compute

$$\begin{cases} w[0] &= Kz \\ y &= Kd \end{cases}$$

The major benefit of prescaling the divisor and dividend is that it allows separate selection of the real and imaginary parts of the complex quotient digits. These two selections can be performed in parallel.

Iterations Perform radix-r iterations

$$w[j+1] = rw[j] - q_{j+1}y \tag{4}$$

where r (in most cases a power of 2) is the radix of the division, and $q_{j+1} = q_{j+1}^{\mathcal{R}} + iq_{j+1}^{\mathcal{I}}$, where $q_{j+1}^{\mathcal{R}}$ and $q_{j+1}^{\mathcal{I}}$ belong to the digit set $\mathcal{S}$ of a redundant radix-r representation (a typical example is the *maximally redundant set* $\mathcal{S} = \{-r+1, -r+2, \ldots, r-2, r-1\}$).

A straightforward induction shows that

$$\frac{w[j]}{y} = r^j \left[\frac{w[0]}{y} - \left(q_1 r^{-1} + q_2 r^{-2} + \cdots + q_j r^{-j} \right) \right]. \tag{5}$$

Hence, any choice of the q_j's for which the $||w[j]||_\infty$'s remain bounded will suffice to give

$$0.q_1^{\mathcal{R}} q_2^{\mathcal{R}} q_3^{\mathcal{R}} \cdots q_n^{\mathcal{R}} + i0.q_1^{\mathcal{I}} q_2^{\mathcal{I}} q_3^{\mathcal{I}} \cdots q_n^{\mathcal{I}} \to z/d$$

2.2 Selection of the quotient digits

As for conventional higher radix division, the key point is to be able to easily select quotient digits. The benefit of division with prescaling is that, since the prescaled divisor becomes close to 1, the $j+1$st quotient digit can be obtained by rounding the residual to its integer part. Moreover, a short-precision estimate of the residual suffices for this rounding so that residuals can be obtained in redundant form (e.g., carry-save or signed-digit). Therefore, the selection is performed by rounding to the nearest integer the real and imaginary parts of $\hat{w}[j]$, where $\hat{w}[j]$ is obtained by truncating $w[j]$ after some fractional position. More precisely, we choose

$$\begin{cases} q_{j+1}^{\mathcal{R}} &= s\left(\Re(rw[j])\right) \\ q_{j+1}^{\mathcal{I}} &= s\left(\Im(rw[j])\right) \end{cases}$$

where the *selection function* s satisfies

$$|s(x) - x| < \frac{1}{2} + 2^{-\sigma}. \tag{6}$$

For instance, one can meet such a requirement by truncating a carry-save representation of x after the $\sigma+1$-st fractional position, and rounding the obtained result to the nearest integer. If x is represented in borrow-save, it suffices to truncate it after the σ-th fractional position before rounding it to the nearest.

We assume that the quotient digits, for the real part as well as for the imaginary part, will be chosen from the digit-set

$$\mathcal{S} = \{-a, -a+1, \ldots, 0, \ldots, +a\}$$

where $2a+1 > r$ and, in most cases, $a \leq r-1$ (and yet, we will later see that the "over-redundant" choice $a = r$ may sometimes be of interest). To make sure that the selection function (6) always returns elements of $\mathcal{S}$, we must have

$$||w[j]||_\infty \leq \frac{1}{r}\left(a + \frac{1}{2} - 2^{-\sigma}\right) \tag{7}$$

Let us call $\Omega = 1/r \times (a + \frac{1}{2} - 2^{-\sigma})$ this last bound. We now find conditions on the various parameters (i.e., a, p, r and σ) for the algorithm to converge.

We assume that $||w[0]||_\infty < \Omega$ (if needed, this can be achieved by a shift of the dividend and extra iterations), and we show that $w[j]$'s, for all j, have a norm less than Ω. As for conventional SRT division, this is shown by induction, assuming $||w[j]||_\infty < \Omega$, and trying to get $||w[j+1]||_\infty < \Omega$.

Define $\epsilon_y = y - 1$. From (4) and (6), we get

$$\begin{aligned} w[j+1] &= rw[j] - q_{j+1}y \\ &= rw[j] - \left(q_{j+1}^{\mathcal{R}} + iq_{j+1}^{\mathcal{I}}\right)(1+\epsilon_y) \\ &= rw[j] - \{\Re(rw[j]) + [s(\Re(rw[j])) - \Re(rw[j])]\}(1+\epsilon_y) \\ &\quad -i\{\Im(rw[j]) + [s(\Im(rw[j])) - \Im(rw[j])]\}(1+\epsilon_y) \\ &= -\epsilon_y s(\Re(rw[j])) - [s(\Re(rw[j])) - \Re(rw[j])] \\ &\quad -i\epsilon_y s(\Im(rw[j])) - i\,[s(\Im(rw[j])) - \Im(rw[j])] \\ &= -\epsilon_y q_{j+1}^{\mathcal{R}} - [s(\Re(rw[j])) - \Re(rw[j])] \\ &\quad -i\epsilon_y q_{j+1}^{\mathcal{I}} - i\,[s(\Im(rw[j])) - \Im(rw[j])] \end{aligned}$$

Therefore,

$$w[j+1] = \epsilon_y q_{j+1} + A + iB \tag{8}$$

r	a	p	σ	Ω
2	1	4	4	23/32
2	2	3	3	19/16
4	2	6	5	79/128
4	3	5	3	27/32
4	4	4	4	71/74
8	4	8	6	287/512
8	5	6	6	351/512
8	6	6	4	103/128
8	7	6	3	59/64
8	8	5	5	271/256
16	8	10	7	1087/2048
16	15	7	3	123/128
16	16	6	6	1055/1024

Table 1: Values of the parameters p, σ and Ω of the algorithm, depending on r and a. It is worth being noticed that p decreases when a increases, which is important, since the prescaling step (if implemented just by one table-lookup) requires to access a table with $2p+1$ address bits. Even *over-redundant* digit-sets (i.e., for which $a \geq r$) may therefore prove useful.

where A and B are real numbers of absolute value less than $\frac{1}{2}+2^{-\sigma}$, $||q_{j+1}||_\infty \leq a$ and $||\epsilon_y||_\infty < 2^{-p}$. From that, we immediately deduce

$$||w[j+1]||_\infty < 2^{-p+1}a + \frac{1}{2} + 2^{-\sigma}.$$

This gives the condition we were looking for

$$2^{-p+1}a + \frac{1}{2} + 2^{-\sigma} \leq \frac{1}{r}\left(a + \frac{1}{2} - 2^{-\sigma}\right). \tag{9}$$

Typical examples of parameters that meet these constraints are given in Table 1.

2.3 The complex residual recurrences

The new recurrences for computing complex residuals are given below. Note that the real and imaginary parts of $w[j+1]$ can be computed in parallel. This fact and the simplicity of the selection function (that only require to separately round the real and imaginary parts of $w[j+1]$) make the iterations very simple.

$$\left\{\begin{array}{rcl} \Re(w[j+1]) & = & r\Re(w[j]) - q^{\mathcal{R}}_{j+1}\Re(y) + q^{\mathcal{I}}_{j+1}\Im(y) \\ \Im(w[j+1]) & = & r\Im(w[j]) - q^{\mathcal{I}}_{j+1}\Re(y) - q^{\mathcal{R}}_{j+1}\Im(y) \end{array}\right. \tag{10}$$

3 Remarks on prescaling

Our method requires that, from a given divisor d, we obtain a scaling factor K such that $||Kd-1||_\infty < 2^{-p}$ by a table-lookup. Let us examine three approaches to obtaining the complex scaling factor K..

3.1 First approach: direct table-lookup

We assume that

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

This is obtained by a mere shift of the divisor d. Now, if we write $d = a+ib$, a and b can be represented as binary fixed-point numbers

$$\begin{cases} a &= 0.a_1a_2a_3a_4\cdots \\ b &= 0.b_1b_2b_3b_4\cdots, \end{cases}$$

where $a_1 = 1$ or $b_1 = 1$. Define $\hat{a}$ and $\hat{b}$ as a and b rounded to the nearest q-fractional-bit number. Our first solution consists in looking-up

$$K = \frac{1}{\hat{a} + i\hat{b}}$$

in a table with $2q-1$ address bits[1]. Now, by denoting $\hat{d} = \hat{a} + i\hat{b}$, we easily find

$$\left|\left|\frac{d}{\hat{d}} - 1\right|\right|_\infty \leq 2||d-\hat{d}||_\infty \left|\left|\frac{1}{\hat{d}}\right|\right|_\infty \leq 4||d-\hat{d}||_\infty \leq 2^{-q+1}$$

Therefore, to assure that $||y-1||_\infty$ will be less than 1, it suffices to choose $q = p+1$. Hence the lookup table will have $2p+1$ address bits.

3.2 Second approach: Prescaling using a multiple variable bipartite method

The previous prescaling method works for radix-2 division (for $a = 1$, the constraint $p = 4$ – hence, $q = 5$ – requires the use of a table with 9 address bits, which is feasible with current technology. If we use an over-redundant number system, with $a = 2$, the table will have 7 address bits, which is small). Radix 4 can be used as well (with the maximally redundant digit-set – i.e., a=3, the table will have 11 address bits, and if we use the over-redundant digit set with $a = 4$, the table we have 9 address bits). Radix-8 might be feasible (13 address bits with the maximally redundant digit-set, 11 with an over-redundant digit-set). For higher radices, the reciprocal table grows rapidly and becomes quickly impractical.

A possible solution to this problem is to generalize to functions of two variables the bipartite table method of [3, 13]. This is done as follows. Since

$$\frac{1}{\hat{a} + i\hat{b}} = \frac{\hat{a} - i\hat{b}}{\hat{a}^2 + \hat{b}^2}$$

we easily deduce that we have to quickly get good approximations to the function

$$\varphi(x, y) = \frac{x}{x^2 + y^2}$$

where x and y are q-bit numbers. To simplify, assume q is some multiple of 3, and define $k = q/3$. Define two $2k$-bit numbers x_H and y_H, two k-bit numbers x_ℓ and y_ℓ, and two $k + \lfloor k/2 \rfloor$-bit numbers x_h and y_h, all with absolute value less than 1 such that:

- x_H is x rounded to the nearest multiple of 2^{-2k};
- y_H is y rounded to the nearest multiple of 2^{-2k};
- $x = x_H + x_\ell 2^{-2k}$;
- $y = y_H + y_\ell 2^{-2k}$;

[1] A straightforward implementation would require $2q$ address bits, but one can notice that one can assume $a_1 = 1$: if $a_1 = 0$, then we know that $b_1 = 1$, and we can easily interchange $\hat{a}$ and $\hat{b}$.

- x_h is x rounded to the nearest multiple of $2^{-k-\lfloor k/2 \rfloor}$;
- y_h is y rounded to the nearest multiple of $2^{-k-\lfloor k/2 \rfloor}$;

If we define functions:

$$\begin{cases} \beta(x_h, y_h, x_\ell) &= x_\ell 2^{-2k} \frac{\partial}{\partial x} \varphi(x_h, y_h) \\ \gamma(x_h, y_h, x_\ell) &= y_\ell 2^{-2k} \frac{\partial}{\partial y} \varphi(x_h, y_h) \end{cases}$$

then an elementary manipulation of Taylor series shows that $\varphi(x, y)$ can be approximated by

$$\varphi(x_H, y_H) + \beta(x_h, y_h, x_\ell) + \gamma(x_h, y_h, x_\ell)$$

with error $\approx 2^{-3k}$. This shows that instead of using one table with $2q-1$ address bits (with the previous method), we can use three tables (one for ϕ, one for β and another one for γ), each of them with $4q/3$ address bits.

The functions that need be stored are

$$\begin{cases} \varphi(x_H, y_H) &= \dfrac{x_H}{x_H^2 + y_H^2} \\ \beta(x_h, y_h, x_\ell) &= x_\ell 2^{-2\,k} \left(\left({x_h}^2 + {y_h}^2 \right)^{-1} - 2 \dfrac{{x_h}^2}{\left({x_h}^2 + {y_h}^2 \right)^2} \right) \\ \gamma(x_h, y_h, x_\ell) &= -2 \dfrac{y_\ell 2^{-2\,k} x_h y_h}{\left({x_h}^2 + {y_h}^2 \right)^2} \end{cases}$$

Improvements to the bipartite method such as the one suggested in [5] might be adapted to functions of 2 variables, but this is outside the scope of this paper.

3.3 Third approach: hybrid method

The scaling factor K is an approximation to $1/d$ with p bits of relative accuracy. A simple solution consists in actually computing this scaling factor with a few steps of our complex recurrence (using a small radix, typically 2 or 4). As soon as we have obtained an accurate enough approximation (this will require $p+1$ radix-2 or $(p+1)/2$ radix-4 iterations), we can start the higher-radix iterations.

4 Rounding

One of the major advantages of a digit-recurrence division algorithm over techniques (such as Goldschmidt or Newton-Raphson methods) is that getting a correctly rounded result is straightforward. We now show that this remains true with our complex digit-recurrence algorithm. Note that the algorithm, by construction, always returns digits of a "valid" radix-r representation of the quotient (which means that whatever the number of steps we will continue to execute – i.e., whatever the final accuracy will be –), the digits that have already been output will never change. Thus, if we stop the computation after having computed digit q_j, then the maximum possible error on the real as well as on the imaginary part is obtained if all subsequent digits have the maximum possible value. Hence this maximum possible error is

$$\sum_{\ell=j+1}^{\infty} a \times r^{-\ell} = \frac{a r^{-j}}{r-1}.$$

If the digit-set $\mathcal{S}$ is not over-redundant (that is, if $a \leq r-1$) then this is less than or equal to the weight of the last computed digit q_j. Therefore our algorithm always provides faithfully rounded real and imaginary parts of the quotient.

Now, let us turn to the problem of getting *correctly rounded* quotients. Let us first neglect the prescaling step, and assume we stop the iterations at step j, where $j \log_2 r$ is greater than or equal to the number of bits of the target format.

From

$$\frac{w[j]}{y} = r^j \left[\frac{w[0]}{y} - \left(q_1 r^{-1} + q_2 r^{-2} + \cdots + q_j r^{-j} \right) \right].$$

We deduce that the sign of the real part (imaginary part) of $q - q_1 r^{-1} + q_2 r^{-2} + \cdots + q_j r^{-j}$ is the sign of the real part (resp. the imaginary part) of $w[j]/y$. Since y is very close to 1, in most cases, that sign will be the sign of the real part (imaginary part) of $w[j]$, but this will not always be the case. More precisely, from $||1 - y||_\infty < 2^{-p}$ we deduce $|1 - y| < \sqrt{2} \times 2^{-p}$ and $|y| \geq 1 - 2^{-p}$, hence,

$$\left| \frac{1-y}{y} \right| \leq \frac{2^{-p}\sqrt{2}}{1 - 2^{-p}},$$

hence,

$$\left|\left| 1 - \frac{1}{y} \right|\right|_\infty \leq \frac{2^{-p}\sqrt{2}}{1 - 2^{-p}}.$$

Define

$$\rho_p = \frac{2^{-p}\sqrt{2}}{1 - 2^{-p}}.$$

From

$$||w[j]||_\infty < \Omega,$$

and

$$\Re(w[j]/y) = \Re(w[j])\Re(1/y) - \Im(w[j])\Im(1/y)$$

and

$$\Im(w[j]/y) = \Im(w[j])\Re(1/y) + \Re(w[j])\Im(1/y),$$

we deduce

$$\begin{array}{lllll}
\text{if} & \Re(w[j]) \geq 0 & \text{then} & \Re\left(\frac{w[j]}{y}\right) & \geq \Re(w[j])(1-\rho_p) - \Omega\rho_p \\
\text{if} & \Re(w[j]) < 0 & \text{then} & \Re\left(\frac{w[j]}{y}\right) & \leq \Re(w[j])(1-\rho_p) + \Omega\rho_p \\
\text{if} & \Im(w[j]) \geq 0 & \text{then} & \Im\left(\frac{w[j]}{y}\right) & \geq \Im(w[j])(1-\rho_p) - \Omega\rho_p \\
\text{if} & \Im(w[j]) < 0 & \text{then} & \Im\left(\frac{w[j]}{y}\right) & \leq \Im(w[j])(1-\rho_p) + \Omega\rho_p
\end{array}$$

Hence, if

$$\Re(w[j]) \geq \frac{\Omega\rho_p}{1-\rho_p}$$

then $\Re(w[j]/y) \geq 0$, so that $\Re(q) \geq q_1^{\mathcal{R}} r^{-1} + q_2^{\mathcal{R}} r^{-2} + \cdots + q_j^{\mathcal{R}} r^{-j}$, and rounding of the real part of the quotient is done exactly as with conventional SRT division. If

$$\Re(w[j]) \leq -\frac{\Omega\rho_p}{1-\rho_p}$$

then $\Re(q) \leq q_1^{\mathcal{R}} r^{-1} + q_2^{\mathcal{R}} r^{-2} + \cdots + q_j^{\mathcal{R}} r^{-j}$. The same properties hold for the imaginary part if

$$\Im(w[j]) \geq \frac{\Omega\rho_p}{1-\rho_p}$$

or

$$\Im(w[j]) \leq -\frac{\Omega\rho_p}{1-\rho_p}$$

If these conditions are not met, then additional iterations may need to be performed to get a correctly rounded result. In all practical cases, $\frac{\Omega\rho_p}{1-\rho_p}$ is very small, so that, if we assume a uniform distribution of the real and imaginary parts of the residuals between $-\Omega$ and Ω, the probability that additional iterations should be necessary is very small. For instance, if $r = 8$ and $a = 4$, we find

$$\frac{\Omega\rho_p}{1-\rho_p} \approx 0.003126093949 < 2^{-8}$$

This means that for this case, each time the final real and imaginary parts of the residual are more than 2^{-8} we are able to correctly round the quotient.

It is important to notice that this possible rounding difficulty, in some extreme cases, is not due to our algorithm. Indeed, it is an intrinsic problem of complex division. Consider, for instance, the following problem. We wish to evaluate

$$\frac{1}{1+2^{-60}i}.$$

The exact result is

$$\frac{1329227995784915872903807060280344576}{1329227995784915872903807060280344577} + \frac{1152921504606846976}{1329227995784915872903807060280344577}\,i$$

whose decimal representation is

$$0.999999999999999999999999999999999999999247\cdots + 8.673617379\cdots \times 10^{-19}\,i.$$

It thus requires to approximate the real part of this value with a precision corresponding to roughly 38 decimal places to be able to safely decide whether this real part is less or more than 1. By replacing the term 2^{-60} by an even smaller term, one can build arbitrarily difficult cases.

Now, to take into account the prescaling, it is necessary that the scaling factor K is exactly representable with a (small) finite number of bits, so that no rounding error occurs during the prescaling step. This is done by adequately rounding the real and imaginary parts of the "theoretical" value of K.

5 Comparison of recurrence implementation with other methods

Implementing complex division by using the basic formula (1) requires two real divisions, two squarings and four multiplications. As indicated earlier, such a formula is not suitable for implementation.

A block diagram of implementing one of the two recurrences (10) is shown in Figure 1 suitable for radix-2 or radix-4. A similar scheme implements imaginary part of the residual recurrence. The selection of the real and imaginary parts of $q_{j+1}$and the real and imaginary parts of $w[j+1]$ can be computed in parallel. Compared to a conventional radix-4 SRT recurrence (3), a [4:2] carry-save adder is used instead of a [3:2] adder. Moreover, another multiple generator is needed. Hence, the delay of our division algorithm will be slightly more than the delay of a conventional radix-4 SRT division, and certainly less than an implementation based on (1). The recurrence implementation can be pipelined to allow sharing in computing complex recurrence.

Compared to Smith's iteration (2) which require significantly more arithmetic operations than (1), we conclude that our algorithm is much faster.

The complex division in [12] uses a radix-2 on-line algorithm. It implements the selection function using selection constants and consequently, unlike the proposed algorithm which uses rounding, it is restricted to small radices. In the case of radix 2, the critical path in the recurrence and its cost are similar in both schemes.

Prescaling requires a table lookup as discussed earlier and two complex (rectangular) multiplications to perform scaling of the operands. As described in the literature, these can be merged with the recurrence implementation.

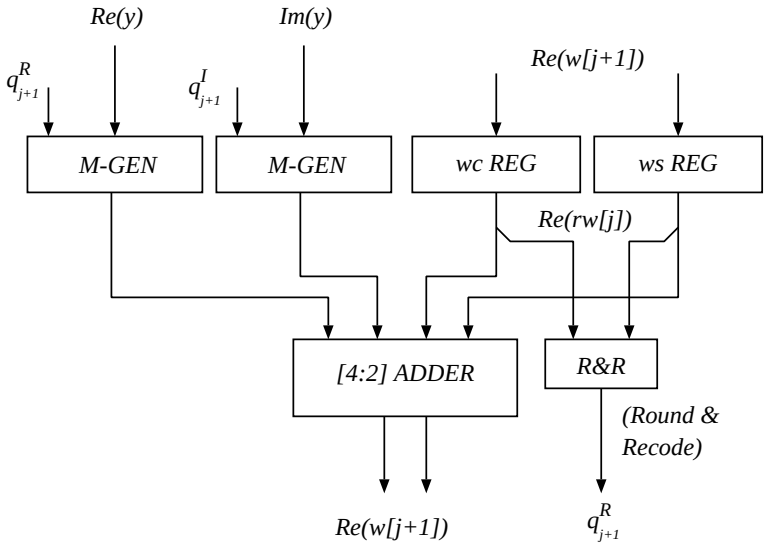


Figure 1: Implementation of recurrence for real part (Similarly for imaginary part). This corresponds to radix-2 or radix-4 schemes. For higher radices, multiple generators (M-GEN) are replaced by digit-vector multipliers producing redundant products, and the [4:2] adder is replaced by a [6:2] adder.

Summary

We have introduced a new algorithm for complex division. It is derived from the conventional (real) digit-recurrence algorithm and uses prescaling of the operands to allow quotient-digit selection by rounding. The recurrences are suitable for higher radix. The prescaling is more complicated than in the real case and the table size is the limiting factor for the choice of radix. The cost of implementing complex recurrence in radix-r is roughly twice the cost of a radix-r recurrence in the case of reals. The proposed algorithm is faster than the other known algorithms for complex division and it is suitable for hardware implementation. Moreover, it always allows easy faithful rounding while correct rounding can be performed at a reasonable cost.

References

[1] D. Bindel, J. Demmel, W. Kahan, and O. Marques. On computing Givens rotations reliably and efficiently. *ACM Transactions on Mathematical Software*, 28(2):206–238, 2002.

[2] N. Burgess . Prescaled maximally-redundant radix-4 SRT divider. *Electronics Letters*, 30(23):1926–8, 1994.

[3] D. Das Sarma and D. W. Matula. Faithful bipartite ROM reciprocal tables. 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] S.R. Dicker et al. Cbm observations with the Jodrell Bank - iac interferometer at 33 Ghz. *Mon. Not. R. Astron. Soc.*, 00:1–12, 2000.

[5] F. de Dinechin and A. Tisserand. Some Improvements on Multipartite Table Methods. In L. Ciminiera and N. Burgess, editors, *Proceedings of the 15th IEEE Symposium on Computer Arithmetic*, Vail, Colorado, June 2001. IEEE Computer Society Press, Los Alamitos, CA.

[6] M. D. Ercegovac. A general hardware-oriented method for evaluation of functions and computations in a digital computer. *IEEE Transactions on Computers*, C-26(7):667–680, 1977.

[7] M. D. Ercegovac. A higher radix division with simple selection of quotient digits. In *Proc. 6th IEEE Symposium on Computer Arithmetic*, pages 94–98, 1983.

[8] M. D. Ercegovac and Lang, T. Simple radix-4 division with operands scaling. *IEEE Transactions on Computers*, C-39(9):1204–1207, 1990.

[9] M. D. Ercegovac, Lang, T., and Montuschi, P. Very-high radix division with prescaling and rounding. *IEEE Transactions on Computers*, 43(8):909–918, 1994.

[10] M. D. Ercegovac and T. Lang. *Division and Square Root: Digit-Recurrence Algorithms and Implementations*. Kluwer Academic Publishers, Boston, 1994.

[11] X. Li et al. Design, implementation and testing of extended and mixed precision BLAS. *ACM Transactions on Mathematical Software*, 28(2):152–205, 2002.

[12] R.D. Mcilhenny. *Complex Number On-line Arithmetic for Reconfigurable Hardware: Algorithms, Implementations, and Applications*. PhD thesis, University of California at Los Angeles, 2002.

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

[14] R.L. Smith. Algorithm 116: Complex division. *Communications of the ACM*, 5(8):435, 1962.

[15] H. Srinivas and Parhi, K. A fast radix-4 division algorithm and its architecture. *IEEE Transactions on Computers*, 44(6):826–831, 1995.

[16] G.W. Stewart. A note on complex division. *ACM Transactions on Mathematical Software*, 11(3):238–241, 1985.

[17] A. Svoboda, A. An algorithm for division. *Information Processing Machines, (Stroje na Zpracovani Informaci)*, 9:25–34, 1963.

[18] C. Tung. A division algorithm for signed-digit arithmetic. *IEEE Transactions on Computers*, C-17(9):887–889, 1963.

[19] G. Vandersteen et al. Comparison of arithmetic functions with respect to Boolean circuits. In *58th ARFTG Conference Digest RF Measurements for a Wireless World*, 2001.

Appendix: Maple program that simulates our algorithm

This is a high-level simulation. We even did not use a redundant number system for representing the $w[j]$'s. The aim of the program is to allow the reader to quickly check the algorithm, and to understand how it works on examples.

Prescaling

```
prescale := proc(d,q);
# d is a complex number
# we assume 1/2 <= ||d|| < 1
hat_a := 2^(-q)*(round(2^q*Re(d)));
hat_b := 2^(-q)*(round(2^q*Im(d)));
denom := hat_a^2+hat_b^2;
K1 := hat_a/denom;
K2 := -hat_b/denom;
K := K1+I*K2
end;
```

Computation of the various parameters

```
getparameters := proc(r,a); # p is param[1]
 # sigma is param[2]
 # omega is param[3]
 bound1 := 1/r * (a+1/2)-1/2;
 p := 1;sigma := 1;
 while a*2^(-p+1) >= bound1 do p := p+1 od;
 bound2 := bound1 - a*2^(-p+1);
 while (1+1/r)*2^(-sigma) >= bound2 do sigma := sigma+1 od;
 omega := 1/r * (a+1/2-2^(-sigma));
 param[1] := p;
 param[2] := sigma;
 param[3] := omega;
 param;
 end;
```

Selection function

```
 selection := proc(x,sigma);
 hat_x := 2^(-sigma)*round(2^(sigma)*x);
 round(hat_x)
 end;
```

Main part

```
complexSRT := proc(z,d,r,a,n) # computes z/d, n radix-r iterations
 # digit set {-a ... a}
 param := getparameters(r,a);
 p := param[1];
 sigma := param[2];
 omega := param[3];
 print("p = ",p);
 print("sigma = ",sigma);
 print("omega = ",omega);
 q := p+1;
 K := prescale(d,q);
 y := K*d; print("y = ",evalf(y));
 x := K*z;
 scale_factor := 1;
 while max(abs(Re(x)),abs(Im(x))) > omega do
            x := x/2; scale_factor := scale_factor*2 od;
 w[0] := x;
 print("w[0] = ",evalf(x)) ; quotient := 0; PowerOfR := 1/r;
 for j from 0 to (n-1) do
 q[j+1] := selection(Re(r*w[j]),sigma)
      +I*selection(Im(r*w[j]),sigma);
 w[j+1] := r*w[j]-q[j+1]*y;
 print("q[",j+1,"] = ",q[j+1]);
 quotient := quotient+PowerOfR*q[j+1];
 PowerOfR := PowerOfR/r
 od;
 quotient := quotient*scale_factor;
 print("computed value: ",evalf(quotient));
```

```
print("exact value: ",evalf(z/d));
end;
```