On Division by Functional Iteration

MICHAEL J. FLYNN, MEMBER, IEEE

Abstract—**In order to avoid the time delays associated with linearly convergent division based on subtraction, other iterative schemes can be used. These are based on 1) series expansion of the reciprocal, 2) multiplicative sequence, or 3) additive sequence convergent to the quotient. These latter techniques are based on finding the root of an arbitrary function at either the quotient or reciprocal value. A Newton–Raphson iteration or root finding iteration can be used.**

The most useful techniques are quadratically convergent (i.e., $\text{error}_{i+1}=0\ (\text{error}_i)^2$). These techniques generally require two arithmetic operations (add or multiply) to double the precision of the quotient.

Index Terms—**Error bias, iterative division, Newton–Raphson division, quadratic convergence, series division.**

Introduction

With the advent of very-high-speed algorithms for performing multiplication and addition, the performance of the divide algorithm has become increasingly anomalous. The primary difficulty with the conventional subtractive divide is its linear rate of convergence with respect to the divisor and required testing of the remainder sign at the end of each iteration. This latter requirement effectively prevents overlapping of operations or iterative cycles.

As long as the multiply had convergence rate not significantly better than that of the divide (out of proportion to relative usage), there was no need for improved divide operation. With the advent of one-step multipliers, such as the Wallace [1] algorithm, and pipeline multipliers, such as Systems 360 Model 91 [2], the relative performance of simple divide algorithms (retiring 2 or 3 divisor bits per iteration) becomes quite unacceptable and prompts investigation of alternate approaches to divide.

One such technique—which is not really new—is the iterative divide which is most commonly based on some variation of the Newton–Raphson equation.

Iterative Divide

We presume the existence of a high-speed multiply and add. We desire to find q, the quotient of the division operative a/b. Now three different approaches to finding iterative expressions to ascertain q, the quotient, are the following.

Method 1: Series Expansion—Use a Taylor series to find q:

$$q = g_0(a, b) + g_1(a, b) + \cdots + g_n(a, b). \tag{1}$$

Manuscript received December 10, 1969; revised February 6, 1970. This work was supported in part by the U. S. Atomic Energy Commission at the Argonne National Laboratory, Argonne, Ill.

The author was with the Department of Industrial Engineering, Northwestern University, Evanston, Ill. 60201. He is now with Johns Hopkins University, Baltimore, Md. 21218.

Method 2: Additive Iteration—Let $f(x)$ have root at $a/b=q$; then find root with convergent sequence $x_0, x_1, \cdots, x_i, \cdots, x_n$ such that

$$x_n \Rightarrow a/b \quad \text{and} \quad x_{i+1} = x_i + H[f(x_i)]. \tag{2}$$

Method 3: Multiplicative Iteration—If $q=a/b$, suppose one can find a product sequence, $m_0, m_1 \cdots, m_n$, such that

$$b \prod_{i=1}^{n} m_i \Rightarrow 1$$

then

$$q \doteq a \prod_{i=1}^{n} m_i. \tag{3}$$

The results of applying these three different approaches to the iterative divide problem do not always yield unique results. It is quite possible that the same iterative expression for division can have three separate interpretations: as a series, as an additive iterative expression, or as a multiplicative iteration. The three distinct approaches are quite useful, however, in getting added insight into the definition of iterative functions. In all of these approaches one wishes to converge either towards a/b or towards some convenient alternate value. Thus, $1/b$ would also be suitable since we could terminate the operation with simple multiplication by a.

Series Expansion

Given the familiar Taylor series expansion of the function $g(x)$ at point p,

$$g(x) = g(p) + (x - p)g'(p) + \frac{(x - p)^2}{2!} g''(p) + \cdots + \frac{(x - p)^n}{n!} g^{(n)}(p) + \cdots. \tag{4}$$

We could let $g(x)=1/x$ and $p=1$ (or any convenient nonzero value) and solve. For computational ease, we let $p=0$ (Maclaurin series) and $g(x)=1/1+x$. Then (see [2]–[4]),

$$g(x) = \frac{1}{1 + x} = 1 - x + x^2 - x^3 + x^4 - \cdots. \tag{5}$$

Thus, if $x=b-1$, then

$$q = a(1 - x + x^2 - x^3 + \cdots) \tag{6}$$

or, in factored form,

$$q = a[(1 - x)(1 + x^2)(1 + x^4)(1 + x^8)\cdots]. \tag{7}$$

The 2's complement of $(1+x^n)$ is $[2-(1+x^n)]=1-x^n$, and the product term is

$$(1 + x^n)(1 - x^n) = (1 - x^{2n}). \tag{8}$$

The recomplementation $(1+x^{2n})$ continues the development.

For $x<1$ (i.e., b suitably normalized), each correction factor, $1+x^{2^i}$, doubles the precision of the quotient. Thus, the sequence converges quadratically with *error term* $\varepsilon_0^{2^i}$ or error per factor application $0(\varepsilon^2)$.

We avoid the use of the term "iteration" since the form of (6) and (7) allows direct implementation in one "iteration" or multiple ones. This distinction stresses the difference between the series and iterative method to be discussed later. It is perhaps a moot point since the factors in (6) and (7) are dependent—being most naturally developed sequentially. (Ferrari [15] provides an interesting discussion of multiterm iterations of this type.) If (7) is written in an iterative form, we get

$$x_{i+1} = x_i[1 + (b - 1)^{2^i}]. \tag{9}$$

Since x_{i+1} is a direct function of i (rather than simply x_i), (9) is not easily interpreted as an additive iteration [of the form of (2)].

The multiplicative interpretation is direct: the factor $(1+x^{2^i})$ corresponds to m_i in (3).

The amount of computation to double the precision of a partial quotient is an important measure of the efficiency of an algorithm. If we exclude the complementation operation, (9) requires two multiples to double the result precision.

In addition to the work of Goldschmidt [3] and Anderson [2], which concerns the hardware realization of the series representation of divide, Laughlin [5] and Knuth [6] apply a variation to produce multiple precision divide.

Multiplicative Iteration

This form of divide has been particularly well covered by Lehman and Senzig [7] and Krishnamurthy [8]. Given

$$q = \frac{a}{b} = \frac{a \prod_{i=1}^{n} m_i}{b \prod_{i=1}^{n} m_i} = \frac{\alpha}{k}, \tag{10}$$

k is usually selected as equal to 1, but any $k=2^n$ is equally attractive—in fact, certain other values ($k=2/3$) can also be used [8].

If we limit our choice of recursion (selection of sequence of m_i's) to simple expressions of the form

$$x_{i+1} = x_i \cdot f(x_i) = x_i \cdot m_i,$$

we exclude (9). More significantly, we have a factored dual of the additive iteration. Krishnamurthy has studied this form when the iteration was derived from the quadratic

$$P(x) = (x - k_1)(x - k_2),$$

one of whose roots equals k.

Additive Iteration

Since additive and multiplicative iterative schemes are basically duals, the remainder of the paper will concentrate mainly on the additive type of scheme. Thus, in the proposition $q=a/b$ the basic iterative divide scheme will consist of the following ingredients:

1) a continuous and differentiable equation of the form $f(x)=0$,
2) a root of the equation at $x=a/b$ (or some equivalent),
3) an iterative method for finding said root,
4) a constraint on the iteration that it itself be free of divide (i.e., it uses only add, subtract, multiply operations).

While any function that has a root at a reciprocal value can be used with any iterative system for finding that root, we will restrict our attention to relatively promising equations and iterative schemes of quadratic order or higher. For simplicity of discussion, we will call the iterative system the iteration and the function with root at the reciprocal value the priming function.

The Iteration

The best known quadratically convergent iteration scheme is the Newton–Raphson equation:

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \tag{11}$$

Fig. 1 is a simple illustration of its operation. The initial approximation, x_0, is known; $f(x)$ and its derivative $f'(x)$ are also known. $f'(x_0)$ is evaluated and its value is equal to the slope of the tangent line at x_0. Thus,

$$f'(x_0) = \frac{f(x_0)}{x_0 - x_1}.$$

The error term associated with the Newton–Raphson method is (for root at $x=p$)

$$\varepsilon_{i+1} = (x_i - p)^2 \frac{f''(\xi)}{2f'(x_i)} = \varepsilon_i^2 \frac{f''(\xi)}{2f'(x_i)} \tag{12}$$

where

$$|x_i| \geq \xi \geq |p|.$$

In general, three conditions must be satisfied for the Newton–Raphson method to converge.

1) Our initial approximation x_0 is sufficiently close to the root.
2) The second derivative f'' must not be excessively large in the region $|x_0-p|$.
3) The first derivative must be known, must be able to be evaluated, and it must exist over the range. Further, the first derivative should not be excessively small. This method is also called the Newton–Raphson first-order method since the tangent predictor is linear and the method can be derived from the linear terms in the Taylor series expansion. Higher order Newton–

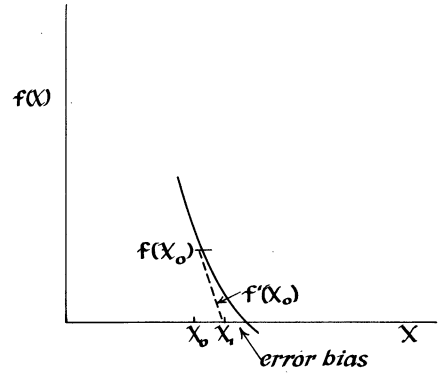

Fig. 1. Newton–Raphson iteration.

Raphson methods can be developed which allow arbitrary curves to be fit for predictives. In particular, the Newton–Raphson second-order method would be

$$x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)} - \frac{1}{2}\cdot\left[\frac{f(x_i)}{f'(x_i)}\right]^2 \cdot \frac{f''(x_i)}{f'(x_i)}. \tag{13}$$

Its error term would be

$$\varepsilon_{i+1} = \varepsilon_i^3 \frac{f'''(\xi)}{6f'(x_i)}. \tag{14}$$

Of course, this may be continued indefinitely. Another potentially promising iterative form (although the author knows no successful use of this form as yet) is Halley's iteration (read f as $f(x_i)$):

$$x_{i+1} = x_{i'} - \frac{f/f'}{1 - f/f' \cdot f/f''}. \tag{15}$$

With error term of the third order,

$$\varepsilon_{i+1} = 0(\varepsilon_i^3). \tag{16}$$

Halley's is basically a hyperbolic predictive fit and would seem to naturally suit the reciprocal curve—too well, it seems from the author's personal experience; most of his attempts have yielded $x_i = 1/b$ (i.e., the iteration requires the division we were trying to avoid). See Traub [9] for fuller discussion of general iterative solutions.

Root Targets

The following seem to be the most logical choices for target roots.

$q = a/b$	targets:	$1/b^2$
		a/b
		$1/b$
		$1 - 1/b$
		$1 - a/b$

(17)

This includes the true quotient, the reciprocal value which needs to be multiplied by the numerator to find the true quotient, a complement of the reciprocal, a complement of the quotient, and the square of the reciprocal which must be multiplied by the numerator and the denominator product. The choice of root target is quite arbitrary. One selects it merely out of convenience with respect to the basic iterative form, its convergence rate, its lack of divisions, and the overhead involved if a target root other than the true quotient is used.

Priming Functions

In dealing with priming functions, one may start with a particularly attractive iteration for its general form, and work backwards to find the priming function from whence it came, or one may work directly from promising priming functions. The problem with the former approach is that frequently, after analysis, the iteration either fails to converge or converges at an unattractive rate. In the second approach the difficulty is that the resulting iteration frequently involves a divide itself. For example, with the first approach, consider the simple iteration

$$x_{i+1} = x_i - (bx_i - a).$$

This can be interpreted as a Newtonian iteration. And the priming function would be found by solving the following differential equation:

$$\frac{y}{y'} = \frac{1}{bx - a}. \tag{18}$$

Solution to this differential equation is of the form

$$f(x) = Y = (bx - a)^{1/b}. \tag{19}$$

Now one can select a and b such that as $x \Rightarrow a/b$

$$f(x) = (ba/b - a)^{1/b} \Rightarrow 0,$$

thus giving the priming function a root quotient value. Unfortunately the iteration converges slowly $[\varepsilon_{i+1} = \varepsilon_i(1-b)]$ since the second derivative is poorly behaved in the region of the root.

Working directly from the priming function is basically a straightforward proposition. Consider the function

$$f(x) = \frac{1}{x} - b. \tag{20}$$

This has a resulting iterative form

$$x_{i+1} = x_i(2 - bx_i) \tag{21}$$

and its error term is

$$\varepsilon_{i+1} = \varepsilon_i^2 \cdot (b).$$

Notice that this converges to the reciprocal value $1/b$. This is the most widely known iterative scheme, being discussed in earlier books on numerical analysis (e.g., Kunz [14]). It apparently was implemented on the Harvard Mark IV—although information on this machine is scarce (see Richards [10] for a note on this). Both Lehman [7] and Krishnamurthy [8] discuss the equivalent (or dual) of this scheme.

Notice carefully the result of the use of $x_0 = 1$ in (21) and the substitution of $(b-1)$ for x in (7), the factored form of the

TABLE I
COMPARISON OF SOME TECHNIQUES

Type of Iteration	Priming Function	Iteration: x_{i+1}	Error Term: ε_{i+1}	Operations (ave) Required* to Double Quotient Precision	
				Adds	Multiplies
#1 Series Expansion of $1/b$	$\frac{1}{b} = (2-b)(1+(b-1)^2)(1+(b-1)^4)\cdots$ identical to #2 and #4 with $x_0 = 1$.		$b\varepsilon_i^2$		2
#2 Additive Newton–Raphson first order	$f(x) = \frac{1}{x} - b$	$x_i(2 - bx_i)$	$b\varepsilon_i^2$		2
#3 Additive Newton—Raphson first order	$f(x) = \frac{x - 1 + 1/b}{x - 1}$	$2x_i - 1 + b(x_i - 1)^2$	$b\varepsilon_i^2$	1	2
#4 Additive Newton–Raphson nth order	$f(x) = \frac{1}{x} - b$	$x_i[1 + (1 - bx_i) + (1 - bx_i)^2 + \cdots + (1 - bx_i)^n]$	$b\varepsilon_i^{n+1}$	(per order)	1

* Ignore shift and complement operations.

series expansion of the reciprocal. The results are identical iteration for iteration. Thus, the series expansion is identical to the Newton–Raphson iteration of (20) and (21) for $x_0 = 1$.

An alternative function which converges to the complement of the reciprocal value is the priming function

$$f(x) = \frac{x - 1 + 1/b}{x - 1}. \tag{22}$$

This has iterative form

$$x_{i+1} = 2x_i - 1 + b(x_i - 1)^2. \tag{23}$$

Although it has the same error term as before,

$$\varepsilon_{i+1} = \varepsilon_i^2 \cdot b.$$

This iterative forms appears in Wilkes [11] as a rather terse statement in the Appendix. The analysis of the priming function is due to Fieg [12]. Priming function of the general form $e^{-f(x)}$ appears quite interesting, especially when $f(x)$ has a pole at a target root which drives the overall function to 0. Thus, consider the function

$$f(x) = \exp\left[-\frac{1}{b(1 - bx)}\right] \Rightarrow 0 \quad \text{as} \quad x \Rightarrow 1/b. \tag{24}$$

This has iteration

$$x_{i+1} = x_i - (1 - x_i b)^2 \quad \text{and} \quad \varepsilon_{i+1} = \varepsilon_i(1 - \varepsilon_i b^2). \tag{25}$$

Thus, its convergence rate is not as attractive as the earlier schemes.

Priming functions may use the more general form or the nth-order Newtonian to develop arbitrarily high convergence rates (see Rabinowitz [13]). In applying the nth-order Newtonian to the first priming function discussed.

$$f(x) = \frac{1}{x} - b, \tag{26}$$

we get the following generalized iterative form:

$$x_{i+1} = x_i[1 + (1 - bx_i) + (1 - bx_i)^2 + (1 - bx_i)^3 + \cdots + (1 - bx_i)^n]. \tag{27}$$

This has convergence rate

$$\varepsilon_{i+1} = b\varepsilon_i^{n+1}.$$

Notice that here also, if $x_0 = 1$, n terms of (27) will be identical to an n-term iteration using the series expansion of the form of (6).

Since both the series expansion and the Newton–Raphson iteration are based on the Taylor series and the form of the function used in (5) and the priming function (20) are similar, the equivalence of the resulting iteration is not surprising.

The above is summarized in Table I.

Error Bias and Starting Tables

Clearly, the objective of all of these schemes is to develop the true quotient as quickly as possible, i.e., with as few iterations as possible, and with few operations per iteration. While the number of operations per iteration are basically intrinsic to the iteration itself, by carefully establishing a table of starting values one can minimize the initial error (i.e., work with an initial quotient precise up to m bits), thus shortening the number of iterations required. One accomplishes this by means of a starting table. This is basically a table in which high-order bits of the divisor (or reciprocal) form the address in a table of high-order bits of the quo-

tient. If m bits are used from the reciprocal the table requires 2^m entries and establishes m bits of quotient.

Since the table doubles in size for each additional bit of accuracy required in the initial quotient guess, small changes in the convergence rate per iteration may reflect substantial changes in the size of the starting table. Notice in almost all schemes the error is biased, hence it (or part of it) can be subtracted from the quotient, slightly reducing the average error. Referring back to Fig. 1, since we are approaching the root from the left side uniformly, we may predict ahead part of the distance for the next iteration. While this is attractive, the error bias may serve a useful function when left in the iterant. In certain cases it will serve to protect the integrity of integers (i.e., integer quotients will be preserved in their usual representation).

Conclusion

The problem of finding complexity or efficiency bounds for division is much more difficult than for add or multiply because of the multiplicity of approaches. The best known techniques require two basic arithmetic operations (add or multiply) to double the precision of the quotient. Even relatively small improvements in the convergence rate of a scheme can result in considerable hardware savings in the area of a starting table. The development of these techniques remains an open problem as does the application of non-Newtonian higher order iterations.

References

[1] C. S. Wallace, "A suggestion for a fast multiplier," *IEEE Trans. Electronic Computers*, vol. EC-13, pp. 14–17, February 1964.

[2] S. F. Anderson, J. G. Earle, R. E. Goldschmidt, and D. M. Powers, "IBM System/360 Model 91: floating-point execution unit," *IBM J. Res. Develop.*, vol. 11, pp. 34–53, January 1967.

[3] R. E. Goldschmidt, "Applications of division by convergence," M.S. thesis, Dept. of Electrical Engineering, Massachusetts Institute of Technology, Cambridge, Mass., June 1964.

[4] M. J. Flynn, "Very high-speed computing systems," *Proc. IEEE*, vol. 54, pp. 1901–1909, December 1966.

[5] H. H. Laughlin, "Large-number division by calculating machine," *Am. Math. Monthly*, vol. 37, pp. 287–293, 1930.

[6] D. E. Knuth, "Seminumerical algorithms," in *The Art of Computer Programming*, vol. 2. Reading, Mass.: Addison-Wesley, 1969, p. 215.

[7] M. Lehman, D. Senzig, and J. Lee, "Serial arithmetic techniques," *1965 Fall Joint Computer Conf., AFIPS Proc.*, vol. 27. Washington, D. C.: Spartan, 1965, pp. 715–725.

[8] E. V. Krishnamurthy, "On optimal iterative schemes for high-speed division," *IEEE Trans. Computers*, vol. C-19, p. 227–231, March 1970.

[9] J. F. Traub, *Iterative Methods for the Solution of Equations*. Englewood Cliffs, N. J.: Prentice-Hall, 1964.

[10] R. K. Richards, *Arithmetic Operations in Digital Computers*. New York: Van Nostrand Rheinhold, 1955.

[11] M. V. Wilkes, D. J. Wheeler, and S. Gill, *The Preparation of Programs for an Electronic Digital Computer*. Cambridge, Mass.: Addison-Wesley, 1951.

[12] R. J. Fieg, "Analysis of a computer divide algorithm," unpublished communication.

[13] P. Rabinowitz, "Multiple precision division," *Commun. ACM*, vol. 4, p. 98, February 1961.

[14] K. Kunz, *Numerical Analysis*. New York: McGraw-Hill, 1957.

[15] D. Ferrari, "A division method using a parallel multiplier," *IEEE Trans. Electronic Computers*, vol. EC-16, pp. 224–226, April 1967.

High-Speed Computer Multiplication Using a Multiple-Bit Decoding Algorithm

H. LING, MEMBER, IEEE

Abstract—**This paper presents a method of performing the binary multiplication beyond the scheme of multiple ADD and SHIFT. The binary multiplication algorithm will be discussed first, followed by block decoding method, logic implementation, hardware consideration, and two examples which are at the end of the discussion.**

Index Terms—**Block decoding technique, fast multiplication, high-speed computer logic, high-speed multiplication, parallel multiplication.**

Manuscript received May 20, 1969; revised December 12, 1969, and February 22, 1970.

The author is with the Information Sciences Department, IBM Research Laboratory, San Jose, Calif.

Introduction

ONE problem which the computer field has been concerned with for many years is how to improve the process of binary multiplication beyond the technique of repetitive ADD and SHIFT.

Some methods have been proposed, all of which have some disadvantages. It was pointed out by Lamdan and Aspinall [1], for example, that the realization of simultaneous multipliers necessitates a large number of components. Recently, carry save adders have generally been used to increase the speed of multiplication. However, due to the re-