TECHNICAL NOTE

Computer Architecture and Systems

John P. Hayes*
Editor

Hitohisa Asai and C.K. Cheng are doing research into the implementation of a division process that uses an iterative multiplying operation instead of repeated subtractions.

SPEEDING UP AN OVERRELAXATION METHOD OF DIVISION IN RADIX-2^n MACHINE

Hitohisa Asai and C. K. Cheng
Christopher Newport College

ABSTRACT: ***For normalized floating point division, digital computers can take advantage of a division process that uses an iterative multiplying operation instead of repeated subtractions. An improvement of this division process by using accelerating constants in the overrelaxation has previously been proposed. Multiplication by a chosen accelerating constant accelerates the process of generating accurate digits of a quotient in division. We propose a further improvement by generalizing the accelerating constants in the overrelaxation method. Two benefits resulting from this improvement promise to yield faster division in digital computers.***

*This paper was processed by John P. Hayes, editor of Computer Architecture and Systems before Duncan Lawrie.

Authors' Present Addresses: Hitohisa Asai, and C.K. Cheng Computer Science Department, Christopher Newport College, 50 Shoe Lane, Newport News, Va 23606.

1. INTRODUCTION

It is well known that the division process can be performed using multiplication instead of subtraction as an iterative operation. This type of division is called the Wilkes–Harvard (W–H) scheme [6, 9, 13, 15, 20]. Because this type of iterative division allows the use of very fast multiplier devices, the W–H scheme may become a prevailing division process in commercial computer systems [2, 15, 18].

When the reciprocal of a divisor with a certain bit length is computed with the W–H scheme, a definite number of iterative multiplications must be performed to obtain a specified accuracy. A table look-up technique [2, 7, 10, 13, 17] is proposed for defining the accelerations to be used in the W–H scheme division process, although the hardware costs of using any table look-up technique must be kept in mind.

An accelerating method for the W–H scheme division method using overrelaxation constants chosen from a table [5] has previously been proposed. This method guarantees a small quantity q in the form of divisor $B = D'(1 + q/D')$ with at most four successive applications of the overrelaxation constants [3, 5], where D' denotes a power of a radix. The method's usefulness is demonstrated in the computation of the reciprocal using 32-bit arithmetic [3].

We have attempted a further low-cost improvement of division through this method by considering a continuity of the two ranges of the overrelaxation constants, D and D^2, where D is a radix. This extended range of the overrelaxation constants in a radix-2^n machine is a generalization of the accelerating constants in the method. The generalized process results in two benefits: (1) a smaller obtainable quantity q that is less than or equal to $1/(2mD - 3)$ in the worst case, where $m = 1, 2, 2^2, 2^3, \ldots, D$, and (2) a reduction in the use of the overrelaxation constants to at most three successive applications. In Sec. 2 we summarize the overrelaxation method. In Sec. 3 we present the speeding-up generalization.

2. THE ACCELERATION METHOD

A division A/B may be evaluated through the power series [4, 5]:

$$A/B = A(1 - P/D)(1 + P^2/D^2)(1 + P^4/D^4)(1 + P^8/D^8) \cdots /D \quad (2.1)$$

where D is the radix ($|D| > 1$), $B = D + P$ and $|P/D| < 1$. The power series involves no division operation ($1/D^n$ is a shift). The ratio $|P/D|$ must be small for fast convergence.

Let the divisor B be in the domain $[D^n, D^{n+1})$, where D and D^n are the base radix and pseudoradix, respectively. Then $B = D^n + q_0$, where $q_0 = P$ for $P > 0$ and $q_0 = D^{n+1} - D^n + P$ for $P < 0$. When $P = 0$, A/B is reduced to a shift operation. The ratio P/D^n to be used in Eq. (2.1) is represented by a function of q:

$$P/D^n = f_1(q) = q/D^n \quad \text{for } P > 0 \quad (2.2.a)$$

$$P/D^{n+1} = -f_2(q) = -[D^{n+1} - (D^n + q)]/D^{n+1} \quad \text{for } P < 0 \quad (2.2.b)$$

We introduce an overrelaxation parameter a in Eq. (2.2.b) as follows:

$$-g(q, a) = -[D^{n+1} - a(D^n + q)]/D^{n+1} \quad (2.3)$$

By imposing $-g(q^+, a + 1) = g(q^+, a)$ to determine the value of q^+, which is the boundary of subdivisions S_a for $a = 1, 2, 3, \ldots, D - 1$, the following results are obtained:

$$q^+(a) = (2D - 2a - 1)D^n/(2a + 1) \quad (2.4)$$

and

S_1: $[(2D - 3)D'/3, D'(D - 1)]$ (2.5.a)

S_a: $[(2D - 2a - 1)D'/(2a + 1), (2D - 2a + 1)D'/(2a - 1)]$ for $a = 2, 3, 4, \ldots, D - 1$ (2.5.b)

S_D: $[0, D'/(2D - 1)]$ (2.5.c)

where D' denotes an appropriate pseudoradix.

By substituting Eq. (2.4) into Eq. (2.3), the local maxima of $g(q, a)$ in the subdivisions (i.e., the maximum ratios of P/D') are

$$-g(q^+(a), a) = 1/(2a + 1) \text{ at } a = 1, 2, 3, \ldots, D - 1 \quad (2.6)$$

Figure 1 shows the area near the boundary between the subdivisions S_a and S_{a+1}.

An iterative contracting map has been introduced by starting with q_0 and recursively applying $q_{i+1} = h(q_i, a_i) = -g(q_i, a_i + 1)D^n$ [5]. The movement of q_0 through the contracting mapping results in the subdivision S_{D-1}, where the smallest ratio P/D', that is, $q_f = g(q^*, D - 1)$ in the domain $[D^{n+i}, D^{n+i+1})$, is obtained with i denoting the number of iterations. The q^* denotes the very last q value in the recursive mapping; we can find this smallest ratio from Eq. (2.6) with $a = D - 2$.

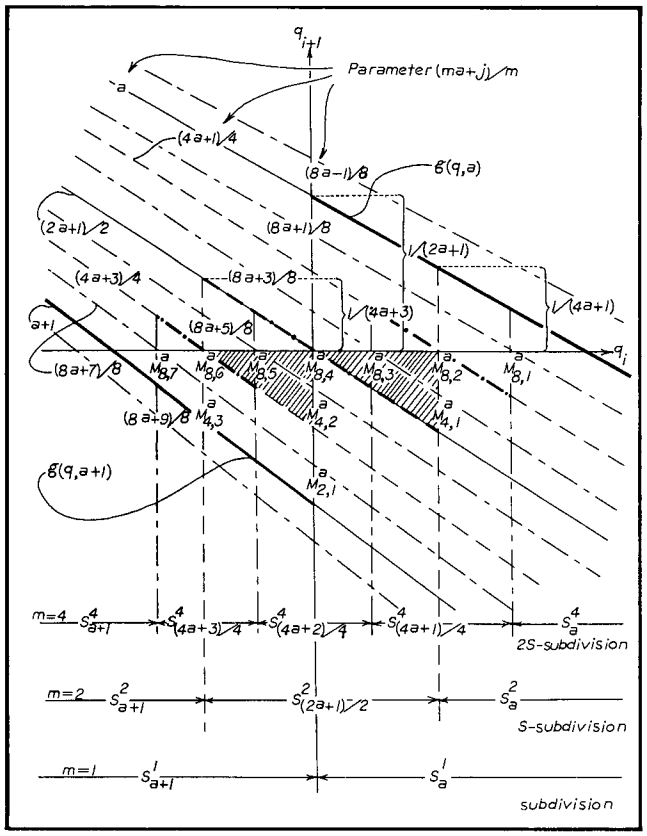

FIGURE 1. *l*s-Subdivisions and an Enlarged Boundary of Subdivisions.

3. GENERALIZED PROCESS OF THE METHOD

A generalized parameter $(ma + j)/m$ is proposed for the overrelaxation where $m = 1, 2, 2^2, 2^3, \ldots, D$ and j is an integer $m > j \geq 0$. When $m = 1$ and $j = 0$, or $m = 1$ and $j = 1$, the parameter $(ma + j)/m$ becomes a or $a + 1$, respectively. This generalization thus includes the method discussed in Sec. 2. Moreover, the multiplication ma and the division $1/m$ of the new parameter can be accomplished only by digit shift operations. Therefore, the use of the generalized parameter in the overrelaxation does not increase the number of multiplications in computing the reciprocal of divisor B.

Let us consider a boundary of subdivisions defined by Eqs. (2.5). By taking the average of parameters a and $a + 1$ (i.e., $(2a + 1)/2$), we find that the function $g(q, (2a + 1)/2)$ intersects the abscissa at the point marked by $M^a_{2,1}$, shown in Figure 1. Furthermore, the intersecting point found from $-g(q^+, a + 1) = g(q^+, a)$ is the boundary between subdivisions S_a and S_{a+1}. The number of subdivision intervals increases from D to $2D - 1$ when we adopt the averaged parameter. A new narrower interval like this is called a sub-subdivision. The lower/upper boundaries of a sub-subdivision are

$$q^+_L(a, j, m) = (2m(D - a) - 2j - 1)D^n/(2ma + 2j + 1) \quad (3.1)$$

and

$$q^+_U(a, j, m) = (2m(D - a) - 2j + 1)D^n/(2ma + 2j - 1) \quad (3.2)$$

from the conditions $-g(q^+_L, (ma + j + 1)/m) = g(q^+_L, (ma + j)/m)$ and $-g(q^+_U, (ma + j)/m) = g(q^+_U, (ma + j - 1)/m)$, respectively, where $m = 2^l$ and $m - 1 \geq j \geq 0$ for $a = 1, 2, 3, \ldots, D - 1$ where $l = 1$. There is, however, an exception in that the upper boundary of S_1 is $D'(D - 1)$. The leftmost sub-subdivision ($a = D$ and $j = 0$) is bounded by $[0, D'/(2mD - 1)]$. We shall abbreviate sub-subdivision as s-subdivision ($l = 1$) and sub-sub-subdivision as 2s-subdivision ($l = 2$), . . . , and a 2^l-subdivided sub- . . . sub-subdivision as ls-subdivision. Thus, the domain $[0, D'(D - 1)]$ of q is divided into the following ls-subdivisions:

$$\begin{aligned} S_1 &: [(2m(D - 1) - 1)D^n/(2m + 1), D^n(D - 1)] \\ S_{(ma+j)/m} &: [(2m(D - a) - 2j - 1)D^n/(2ma + 2j + 1), \\ &\quad (2m(D - a) - 2j + 1)D^n/(2ma + 2j - 1)] \\ &\quad \text{for } a = 1, 2, 3, \ldots, D - 1 \text{ and} \\ &\quad j = 0, 1, 2, \ldots, m - 1 \\ S_D &: [0, D^n/(2mD - 1)] \end{aligned} \quad (3.3)$$

with the parameter $(ma + j)/m = (2^l a + j)/2^l$. The boundary $q^+_L(a, j, m)$ is depicted in Figure 1 with points marked by $M^a_{2m,2j+1}$, where $m = 1$ and $j = 0$ for a subdivision; $m = 2$ and $j = 0, 1$ for an s-subdivision; and $m = 4$ and $j = 0, 1, 2, 3$ for a 2s-subdivision.

Next, consider the mapped value of q computed from $g(q, (ma + j)/m)$. By substituting (3.2) into $g(q, (ma + j)/m)$, we obtain the value of q as follows:

$$q = -g(q^+_U(a, j, m), (ma + j)/m) = 1/(2ma + 2j - 1) \quad (3.4)$$

As (3.4) shows, the value of q decreases as the value of m increases. The worst case of q_f for a fixed value of m is obtainable as

$$|q_f| = |1/(2mD - 3)| = |1/(2^{l+1}D - 3)| \quad (3.5)$$

by setting $a = D - 1$ and $j = m - 1$ in (3.4).

The recursive application of $q_{i+1} = h(q_i, (ma_i + j_i)/m) = -g(q_i, (ma_i + j_i + 1)/m)D^n$ by starting with the initial value q_0 in $S_{(ma_0+j_0)/m}$ assures the movement of successive mapping

images of q_0 into the ls-subdivision $S_{(mD-1)/m}$ (when $a = D - 1$ and $j = m - 1$) or S_D. As soon as the last contracting image q^* is reached in $S_{(mD-1)/m}$ or S_D, the smallest quantity q_f is evaluated from $q_f = -g(q^*, (mD - 1)/m)$ or $q_f = q^*$, depending on whether it has been reached in $S_{(mD-1)/m}$ or S_D, respectively.

We have been discussing the first benefit of the generalized method. The second benefit is described below. Consider the difference between two values of q computed from $g(q, (2ma + 2j + 1)/2m)$ and $g(q, (ma + j)/m)$. The difference is $(D^n + q)/2mD^{n+1}$, which can be seen in Figure 1 as $1/(4a + 3)$ and is indicated by a brace when $m = 2$ and $j = 1$ are used. From this, we may introduce a modified mapping function $h_m(q, a)$ where the suffix m denotes the modification:

$$h_m(q, (ma + j)/m) = -g(q, (ma + j)/m)D^n$$

if this is positive, or

$$\begin{aligned} h_m(q, (ma + j)/m) &= -g(q, (2ma + 2j + 1)/2m)D^n \\ &= -g(q, (ma + j)/m)D^n + (D^n + q)/2mD \end{aligned}$$

if $-g(q, (ma + j)/m)$ results in a negative number.

The mappings by $h_m(q, (ma + j)/m)$ for the cases $m = 2$ and $j = 1$, and $m = 2$ and $j = 0$ are indicated in Figure 1 by the shaded triangles. The largest mapped values for each case occur at the points of $M^a_{2,1}$ and $M^a_{4,1}$ on the abscissa. Since $h_m(q, (ma + j)/m)$ is a piecewise linear mapping, it is sufficient to consider only the largest possible mapped value of q that happens to be the upper boundary of each ls-subdivision obtained through Eq. (3.2).

Next, we demonstrate that the number of recursive applications for the worst case is three. By taking the worst initial $q_0 = q^+_0(a, j, m)$, the q_1 value is as follows:

$$q_1 = h_m(q_0, (ma_0 + j_0)/m) = D^n/(2ma_0 + 2j_0 - 1) \quad (3.6)$$

where q_0 belongs in $S_{(ma_0+j_0)/m}$ initially. It is obvious that if $a_0 = 1$ and $j_0 = 0$, q_1 is greater than the upper boundary $(3D^n/(2mD - 3)$ obtained by substituting $a = D - 1$ and $j = m - 1$ in (3.3)) of the ls-subdivision $S_{(mD-1)/m}$. So another contraction mapping of $h_m(q_1, (ma_1 + j_1)/m)$ is required to obtain a smaller value.

First, we must determine the corresponding a_1 and j_1 for the value q_1 from the following inequalities obtained from the ls-subdivision boundaries in Eq. (3.3):

$$\begin{aligned} &(2m(D - a_1) - 2j_1 - 1)D^n/(2ma_1 + 2j_1 + 1) \\ &\quad < D^n/(2ma_0 + 2j_0 - 1) \\ &\quad \leqslant (2m(D - a_1) - 2j_1 + 1)D^n/(2ma_1 + 2j_1 - 1). \end{aligned}$$

After a simple computation, we obtain

$$\begin{aligned} ma_1 + j_1 = \lfloor &((2mD + 1)(2ma_0 \\ &+ 2j_0 - 1) + 1)/(4ma_0 + 4j_0) \rfloor \end{aligned} \quad (3.7)$$

where $\lfloor R \rfloor$ denotes an integer in the range $R - 1 < \lfloor R \rfloor \leqslant R$, and R is a real number.

Next, by using the value q_1 and by determining a_1 and j_1 from the inequalities, $R - 1 < \lfloor R \rfloor \leqslant R$, the interval bound of q_2 is determined as follows:

$$\begin{aligned} q_2 = h_m(q_1, (ma_1 + j_1)/m) \leqslant \\ (ma_0 + j_0)D^n/(2ma_0 + 2j_0 - 1)mD \end{aligned} \quad (3.8)$$

and

$$q_2 > -(ma_0 + j_0)D^n/(mD(2ma_0 + 2j_0 - 1)). \quad (3.9)$$

The existence of the interval bound in $S_{(mD-1)/m}$ or S_D is proved in the Appendix.

After the second contraction mapping of $h_m(q_1, a_1)$, the value q_2 exists in $S_{(mD-1)/m}$ or S_D (see the Appendix). Then, the final value q_f is computed as follows:

$$|q_f| = |-g(q_2, (mD - 1)/m)| < |1/(2mD - 3)| \quad (3.10)$$

Here is an example of the generalized process. Let $A = 1$ be a dividend and $B = 54$, a divisor in decimal notation. By taking the base radix $D = 8 = 10_8$ and $n = 1$, the value of P is found to be 56_8 as follows: $B = D^n + P = 54_{10} = 66_8$. All computation in the example is carried out in octal with $m = 2$ by using the s-subdivision shown in Table I as computed from (3.3).

Step 1. Since $B = 10 + 56$, $q_0 = 56$. From Table I and the value $q_0/D^n = 56/10 = 5.6$, we find $a_0 = 1$ and $j_0 = 0$ in the s-subdivision. Set $i = 0$ and $\alpha = 1$.

Step 2. Since q_0 is not in $S_{(2D-1)/2}$ nor in S_D, go to the next step.

Step 3. Compute

$$\begin{aligned} q_1 &= -\{D^{n+i+1} - ((4a_0 + 2j_0 + 1)/4)(D^{n+i} + q_0)\}/D \\ &= -\{100 - (5/4)(10 + 56)\}/10 = 34/100 = 0.34. \end{aligned}$$

Since q_0 (=56) is less than the root, 7.0, of the s-subdivision as shown in Table I ($q_1 = -g(q_0, (ma_0 + j_0)/m) < 0$), the quantity $(D^{n+i} + q_i)/2mD$ is added to q_1, namely, the acceleration constant $(2ma_i + 2j_i + 1)/2m$ is used in the q_{i+1} and α (=$\alpha(2ma_0 + 2j_0 + 1)/2m = 1 \times (5/4) = 5/4$) computations. Increase the counter i by one, $i = i + 1 = 0 + 1 = 1$.

Step 4. Now the value q_1/D^n becomes $q_1/D^n = 0.34/10 = 0.034$. By searching the table, we find that q_1 is in the s-subdivision with $a_1 = D - 1$ and $j_1 = 1$. Since q_1 *is now in* $S_{(mD-1)/m}$, go to step 5.

Step 5. Compute the final value of q_f:

$$\begin{aligned} q_f &= -\{D^{n+i+1} - ((2a_1 + j_1)/2) \\ &\quad \cdot (D^{n+i} + q_1)\}/D^{n+i+1} \\ &= -\{1000 - (17/2)(100 + 3.4))/1000 \\ &= -56/10^4 \end{aligned}$$

and

$$\alpha = \alpha(2a_1 + j_1)/2 = (5/4)(17/2) = 113/10.$$

Step 6. Compute the power series:

$$\begin{aligned} \alpha &= \alpha/(1 + q_f)D^{n+i+1} = \alpha(1 - q_f)(1 + q_f^2) \\ &\quad \cdot (1 + q_f^4)(1 + q_f^8) \cdots /D^{n+i+1} \\ &= (113/10)(1 + 56/10^4)(1 + 56^2/10^8) \\ &\quad \cdot (1 + 56^4/10^{16})(1 + 56^8/10^{32}) \cdots /10^3 \\ &= 1136572(1 + 56^2/10^8)(1 + 56^4/10^{16}) \\ &\quad \cdot (1 + 56^8/10^{32}) \cdots /10^8 \\ &= 113664113422150(1 + 56^4/10^{16}) \\ &\quad \cdot (1 + 56^8/10^{32}) \cdots /10^{16} \\ &= 0.01136641136641136533620 6 \\ &\quad \cdots (1 + 56^8/10^{32}) \cdots . \end{aligned}$$

Comparing the real quotient, $1/66_8 = 0.0\overline{113664}$, with the approximate reciprocal, we find the results accurate up to 9 or 17 digits when the terms of the power series are evaluated up to q_f^2 or q_f^4, respectively.

If a parallel process provides the remainder of A/B where $A > B$, then the process is useful in integer division. The parallel process involves successive applications of Horner's scheme on the polynomial form $\{\cdots \{A_mP + A_{m-1}\}P + \cdots + A_1\}P + A_0$, where A_i for $i = 0, 1, 2, \ldots, m$ are digits of $A = \{\cdots \{A_mD' + A_{m-1}\}D' + \cdots + A_1\}D' + A_0$ and $P = D' - B$ [4], and the

Table I. S-Subdivision Intervals with $D^n = D = 10_8$ and $m = 2$.

S-subdivision Number		Lower boundary in octal	Upper boundary in octal	Root of $g(q,(ma+j)/m) = 0$ in S-subdivision
a	*j*			
1	0	5.31463...	7.00000...	7.00000...
1	1	3.44444...	5.31463...	4.25252...
2	0	2.43434...	3.44444...	3.00000...
2	1	1.72135...	2.43434...	2.14614...
3	0	1.35423...	1.72135...	1.52525...
3	1	1.10421...	1.35423...	1.22222...
4	0	0.703607..	1.10421...	1.00000...
4	1	0.536241..	0.703607..	0.616161..
5	0	0.414141..	0.536241..	0.463146..
5	1	0.310232..	0.414141..	0.350350..
6	0	0.217270..	0.310232..	0.252525..
6	1	0.136641..	0.217270..	0.166116..
7	0	0.0647562.	0.136641..	0.111111..
7	1	0.0204102.	0.0647562.	Not Applicable
8	0	0.00000...	0.0204102.	Not Applicable

applications continue on the evaluated result of Horner's scheme again and again until the very last result becomes less than D'. Suppose $A = 10000_8$ in the example where $P = 12_8$ and $D' = D^2 = 100_8$. A computation is shown below. The first and second applications of Horner's scheme result in

$$(1 \times P + 00)P + 00 = (1 \times 12) \times 12 = 144 \text{ and } 1 \times 12 + 44 = 56.$$

Then the last result 56 is less than 100 and is less than $B = 66$. So the remainder is $56_8 = 46_{10}$. When the last value is greater than B, the difference between the last value and the divisor (or a multiple of the divisor) becomes the remainder if P is less than B. If P is greater than B, *a smaller* P may be chosen by taking the difference between P and B.

4. CONCLUSION

The parameter of the overrelaxation in our method is generalized for a radix-2^n machine. Two benefits in expediting the computation from the generalization have been described: (1) the value of q_f computed from $g(q, a)$ decreases from $1/(2D - 3)$ to $1/(2mD - 3)$ for the worst case, and a rate of decrease is apparent as the value of m increases, and (2) the number of iterative computations of $q_{i+1} = -g(q_i, a_i)$ is reduced from four to three for the worst case, where the suffix i denotes the ith iteration.

The first benefit is faster convergence of the power series. The second benefit is a shortcut in the division process by eliminating a pair of the very last evaluations, $q_{i+1} = h(q_i, a_i)$ and $\alpha_{i+1} = \alpha_i(a_i + 1)$, which require the longest digit manipulations where α_i is a crude approximation of the quotient.

There is, however, a cost to be paid for the generalization which comprises (1) a preparation of the generalized parameter form, $(ma + j)/m$ $(= a + j/m)$; (2) the test whether the value of q_{i+1} computed from $q_{i+1} = -g(q_i, (ma_i + j_i)/m)$ is positive or negative; and (3) two additions of the quantity $(D^{n+i} + q_i)/2m$ to the value q_{i+1} and of the quantity $1/2m$ to the value $(ma_i + j_i)/m$ whenever q_{i+1} is negative.

The cost of item (1) is a bit-shift operation and an OR (or AND) operation, which is negligible. The cost of items (2) and (3), for which a sign bit check (of the term of $q_i - q_r$ where $q_r = (mD - (ma + j))D^n/(ma + j)$ is the root of $g(q, (ma + j)/m) = 0$ in an ls-subdivision), two 1-bit shifts, and one OR (or AND) operation are required, is also negligible. In other words, the estimated cost is far less than the cost of computing the third iteration of $q_{i+1} = h(q_i, a_i)$ and $\alpha_{i+1} = \alpha_i(a_i + 1)$, namely, $C + S(m) + 2S(1) + 2L(2D) \ll 2M(2D, tD)$, where C, $S(k)$, $L(k)$, and $M(k, l)$ denote a comparison, k-bit shift, k-bit OR (or AND) operation, and k-by-l bits multiplication, respectively, and t represents the digit length of dividend/divisor. Thus, the saving of multiplication steps could be maximized if these bit operations are implemented in the same machine cycle. Finally, although the generalized process is most suitable for normalized floating point division, we have shown that the process may be used for fast integer division when multiplying hardware computes the remainder in parallel.

APPENDIX

We consider two inequalities in order to obtain the interval bound of q_2.

(1) When the inequality $\lfloor R \rfloor \leq R$ is used, the upper bound of the interval range is

$$q_2 = h_m(q_1, (ma_1 + j_1)/m) \leq (ma_0 + j_0)D^n/(2ma_0 + 2j_0 - 1)mD \quad (3.8)$$

Since the last term of (3.8) is positive, there is no need to have the difference term, $(D^n + q_1)/2mD$, in $h_m(q, a)$. Then we compare the upper bound of q_2 with the upper bound of $S_{(ml)-1)/m}$:

$$3D^n/(2mD - 3) - (ma_0 + j_0)D^n/(2ma_0 + 2j_0 - 1)mD = ((4ma_0 + j_0) - 3)mD + 3(ma_0 + j_0))D^n/(2mD - 3)(2ma_0 + 2j_0 - 1)mD \quad (A.1)$$

Equation (A.1) results in a positive number, so q_2 must belong in $S_{(ml)-1)/m}$ since $4(ma_0 + j_0) - 3 > 0$, $2mD - 3 > 0$, and $(2ma_0 + 2j_0 - 1) > 0$ by taking the smallest values of $a_0 = 1$, $j_0 = 0$, $m = 1$, and $D = 2$, and no further mapping is needed.

(2) When the inequality $R - 1 < \lfloor R \rfloor$ is used, the lower bound of the interval range is found as follows:

$$q_2 > -(ma_0 + j_0)D^n/(mD(2ma_0 + 2j_0 - 1)) \quad (3.9)$$

Equation (3.9) results in a negative value when the smallest values of a_0, j_0, m, and D are taken. Therefore, this negative value is replaced by $(D^n + q_1)/2mD = (ma_0 + j_0)D^n/mD(2ma_0 + 2j_0 - 1)$, which is the largest value in $h_m(q_0, (ma_0 + j_0)/m)$, namely, $M^a_{2m,2j}$ for $m = 2, 4$, and $j = 1, 2$, respectively, as shown in Figure 1. It is sufficient to compare the largest value with the upper bound of $S_{(ml)-1)/m}$:

$$3D^n/(2mD - 3) - (ma_0 + j_0)D^n/mD(2ma_0 + 2j_0 - 1) = mD(4(ma_0 + j_0) - 3) + 3(ma_0 + j_0). \quad (A.2)$$

Equation (A.2) is positive when the smallest values of a_0, j_0, m, and D are taken. Thus, from these two cases we find the value q_2 is in $S_{(ml)-1)/m}$ or S_D.

REFERENCES

1. Aho, A.V., Hopcroft, J.E., and Ullman, J.D. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, Mass., 1974.
2. Anderson, S.F., et al. The IBM system/360 model 91: Floating-point execution unit. *IBM J. Res. Dev. 11*, 1 (Jan. 1967), 34–53.
3. Asai, H. A complexity measure of a division process—a comparison analysis of AMONIC. To appear.
4. Asai, H. A recursive radix conversion formula and its application to multiplication and division. *J. Comput. Math. Appl. 2*, 3/4 (1976), 255–265.
5. Asai, H. An overrelaxation for a numerical inverse of a constant. *Commun. ACM 23*, 9 (Sept. 1980), 503–510. U.S. patent #4364115.
6. Chen, T.C. Automatic computation of exponentials, logarithms, ratios and square roots. *IBM J. Res. Dev. 15*, 7 (July 1971), 380–388.
7. Ferrari, D. A division method using a parallel multiplier. *IEEE Trans. EC-16*, 2 (1967), 224–226.

8. Gilman, R.E. A mathematical procedure for machine division. *Commun. ACM 2*, 4 (April 1959), 10–12.
9. Goldschmidt, R.Z. Applications of division by convergence. M.S. thesis, MIT, Cambridge, Mass., June 1964.
10. Klir, J. A note on Svoboda's algorithm for division. *Inf. Process. Machine* (Prague, Czechoslovakia), *9* (1963), 35–39.
11. Knuth, D.E. *The Art of Computer Programming*, vol. 2. Addison-Wesley, Reading, Mass., 1969.
12. Knuth, D.E. *The Art of Computer Programming*, vol. 3. Addison-Wesley, Reading, Mass., 1973.
13. Krishnamurthy, E.V. On optimal iterative scheme for high-speed division. *IEEE Trans. Comput. C-19*, 3 (1970), 227–231.
14. Richards, R.K. *Arithmetic Operations in Digital Computers*. Van Nostrand, London, 1955.
15. Russell, R.M. The CRAY-1 computer system. *Commun. ACM 21*, 1 (Jan. 1978), 63–72.
16. Savage, J.E. *The Complexity of Computing*. Wiley, New York, 1976.
17. Svoboda, A. An algorithm for division. *Inf. Process. Machine* (Prague, Czechoslovakia), *9* (1963), 25–34.
18. TDM. ADP 6XXX. Large Information System Div., Honeywell Inc., Phoenix, Ariz., 1978.
19. Wallace, C.S. A suggestion for a fast-multiplier. *IEEE Trans. EC-13*, 1 (1964), 14–17.
20. Wilkes, M.V., Wheeler, D.J., and Gill, S. *The Preparation of Programs for an Electronic Digital Computer*. Addison-Wesley, Cambridge, Mass., 1951.

CR Categories and Subject Descriptors: B.2.1 [**Arithmetic and Logic Structures**]: Design Style—*calculator*; G.1.2 [**Numerical Analysis**]: Approximation—*rational approximation*; I.1.2 [**Algebraic Manipulation**]: Algorithms—*algebraic algorithms*
General Terms: Algorithms, Design
Additional Key Words and Phrases: Wilkes–Harvard scheme, power series, overrelaxation, convergence, iterative multiplication, truncation error, convergence division

Received 10/80; revised 10/81; accepted 5/82

Technical Correspondence

IMPROVING PROGRAM READABILITY AS A MODIFICATION AID

The article "Improving Computer Program Readability to Aid Modification" by Elshoff and Marcotty [1] represents an excellent discussion of techniques for enhancing the program modification process by improving source code readability. In addition, some of the transformations presented (e.g., in Sections 5.1, 5.2, 5.12) significantly improve the abilities of realtime computer programs to satisfy stringent execution speed (timing) and/or limited main and auxiliary memory (sizing) requirements. When read from the perspective of critical timing and sizing constrained realtime programs (e.g., air and space craft navigation, CAD/CAM tools, electrical power distribution control) several comments can be appended.

1. Critical timing and sizing constrained software is quite often coded in high order languages which are implemented by special purpose compilers. Even though the source language is familiar (e.g., often Fortran; sometimes Jovial, Pascal, CMS-2; and some day Ada®), the compilers are specifically constructed to produce object code only for special-purpose target processors (e.g., IBM 4PI CC-1, CC-2, AP-101; MIL-STD 1750; AN/UYK-7, -20). Since these compilers are generally "one shot" tools, with a few exceptions, only minimal attempts at object code optimizations are included. Consequently, a significant burden for tight object code is passed upstream to the source language programmer.

The transformations described in Section 5.1 (Move Single Entry Labeled Blocks) and 5.12 (Localize References) are particularly useful in this situation. Transformation 5.1 reduces both time and space by eliminating unnecessary code. Transformation 5.12 allows those easily implemented compiler local optimizations to function more effectively.

2. Transformation 5.2 (Duplicate Labeled Blocks) represents a double-edged sword. As indicated by the authors, replication of short (less than ten-statement) code blocks improves readability over branching to a multiple-referenced label. Ignoring those situations where use of a procedure becomes desirable, two major difficulties can occur when code blocks are repeated:

(a) Sizing problems can be aggravated. This is particularly the case when a short source code block causes long object code blocks to be generated. Mitigating this somewhat is loss of (a presumably) long range branch to the multiple-referenced label. (Long range branches are frequently indirect and slow on special purpose realtime processors).

(b) Installation of changes in all repeated short code blocks can fail. As with a procedure, one of the advantages of a multiple-referenced label is that the required code only exists in one place; thus corrections to that code need only be applied in one place. Replication of the code block requires that all changes must be applied precisely to each block. In pressure situations (which occur all too frequently in the realtime software world), it is easy to overlook one or more of the replicated code blocks requiring alteration.

Fortunately, two easy solutions to the latter are feasible. One is to number each replicated code block via comments (e.g., 1 of 7, 2 of 7, 3 of 7, etc.) This allows programmers to "check list" changes to each code block replica. The other is to code the short code block as a macro in those languages which allow it (e.g., [2]). This allows the code block to be written once, yet exist inline at as many locations as needed.

3. The statement in Section 7 that even though load module size of the example program increased 52 percent (8,800 to 13,400 bytes), general experience shows execution speed improvements of five to ten percent, seem in-