Analyzed Binary Computing

NICHOLAS C. METROPOLIS

Abstract—A single format for the representation of numbers in a computer is proposed to accommodate both exact and inexact quantities. A consistent set of rules is described for addition (subtraction), multiplication, and division of such quantities, both within their separate types, as well as in combination. Error correlation aside, the propagation of inherent errors is monitored in operations with at least one imprecise value. A definitive algorithm must, of course take into account any correlations of inherent errors; these correlations must be recognized and incorporated into the algorithm by the numerical analyst, not by the logical designer of the computer.

Index Terms—Control of propagated errors, floating-point representation, representation error, significant digit arithmetic.

Introduction

EXPERIMENTALISTS have an established tradition of associating a careful estimate of error with a final result. This practice is less well developed for results of computational problems performed by theorists in various disciplines. Part of the neglect is due to the complicated nature of error correlation; in part, designers of arithmetic processors have not as a rule provided a convenient representation of inherent errors in the input quantities, nor an automatic monitoring scheme that takes into account the propagation of inherent error as the calculation proceeds. Such errors are present in experimental data used as inputs. Propagative effects can develop and become serious even in computations that have precise inputs, but where ill conditioning or instability causes considerable erosion of precision.

In a low key, some methods have been developed in the past for the representation and processing of inexact inputs; they have sometimes been described collectively as significant digit arithmetic (SDA). Integer arithmetic is commonplace so that exact numbers can be handled straightforwardly. When these two number types have interacted, one has simply used SDA throughout and represented the exact inputs with maximal available (single) precision. No awkwardness developed if an exact number was combined with an inexact one. But if part of a computation was concerned with exact numbers exclusively, the rules of SDA could lead to a misrepresentation of the results, primarily in the case of the subtraction process involving cancellation of leading digits.

A single representation is proposed here for both inexact and exact numbers and the rules for SDA are extended to properly accommodate both types interactively as well as separately. The discussion is based in part on an internal laboratory report [1]. In addition, attention is directed to implementation by software of such a system; we have been encouraged by our similar experiences with SDA on Maniac II and on a CDC 6600 in our laboratory [2], [3].

This analyzed binary computing is simply referred to as ABC arithmetic.

Manuscript received January 31, 1973. This work was performed under the auspices of the U.S. Atomic Energy Commission.

The author is with the University of California, Los Alamos Scientific Laboratory, Los Alamos, N. Mex. 87544.

Definitions and Format

Numerical quantities may be distinguished as being either exact or approximate. In computer work a further characterization is useful. If an exact number requires more than some preassigned number of digits, say P, for its representation, then this quantity is only approximately represented, owing to a nonzero rounding error, and belongs to the approximate class. All other exact numbers are said to be *precise.*

Approximate quantities are called *imprecise.*

For purposes of simplicity in hardware, systems, and problem preparation, a single format for the representation of both precise and imprecise numbers is desirable. The two types can, of course, be distinguished by some specified bit. Alternatively, and somewhat more efficiently, precise numbers are by fiat always in conventional, floating-point normalized form; whereas an imprecise quantity is always in an unnormalized form such that the form itself exhibits the number of meaningful digits.

Specifically, let x be represented in binary form by $x = 2^e \cdot f = (e, f)$ where *exponent* e is an integer, and *coefficient* f satisfies $0 \leqslant |f| < 1$. The operation of replacing (e,f) by $(e + a, 2^{-a} f)$ for integer a is called *adjustment.* Obviously for $a < 0$, $2^{-a}\ |f| < 1$ is required; for $a > 0$, the adjusted value may be only approximate if nonzero (meaningful) digits are truncated.

Let

$$|f| = \sum_{1}^{P} \alpha_i 2^{-i} \tag{1}$$

be the computer representation of $|f|$. If x is adjusted such that $\alpha_1 = 1$, x is in normalized form. If $\alpha_j = 0$ for $j > P$, x is always adjusted to normalized form.

For imprecise x, advantage may be taken of the one degree of freedom afforded by the adjustment process to arrange that the *least* significant digit of f resides in some selected position, say k (counting from the left), at user's option; a large number of significant digits implies a small number of leading zeros. (If $k < P$, the digits beyond k are called guard digits.) Clearly, k must be consistent with the precision of all imprecise input numbers, but is otherwise not a critical item. The magnitude of P is usually sufficiently large so that as a consequence of the imprecision in x, α_1 is usually zero. This unnormalized condition is imposed on all imprecise quantities; in the various arithmetic processes considered in what follows, the value of

α_1 is used to distinguish precise from imprecise quantities. If the number of meaningful digits exceeds $P - 1$, digits on the right are lost. This is an exceptional circumstance for most problems, especially those arising in the natural sciences.

For negative numbers, either a sign-magnitude form, $\alpha_0 = 1$ plays the role of minus sign, or the complement form, α_0 has "negative weight," may be used. This is a routine matter and need not be further considered here. For simplicity, consider only nonnegative quantities.

In the arithmetic developed in what follows, intermediate quantities may have lost all meaningful digits and become *relative* zero, 0_R. It is possible to give reasonable definitions for addition, multiplication, and (even!) division involving 0_R without the need to invoke interrupt conditions. However, it is planned to warn a user whenever an attempt to divide by 0_R is made.

It is useful to distinguish 0_R from absolute or true zero, 0_T, that interacts quite differently. A special symbol is needed to designate 0_T; it is convenient to reserve the smallest exponent for it. In the following, we will avoid making reference to this degenerate case; it is obvious what must be done.

A variant for the representation of exact quantities is not to restrict to P the number of bits in f, but to permit the number to expand as needed so that the exact quantities are representable, perhaps with some specifiable, reasonable upper limit for nonterminating rationals, irrationals, and transcendentals. This approach is not pursued further here.

Finally, m_x defines the number of leading zeros in $|f|$ associated with x; when needed write $|f_x|$. Clearly, $\frac{1}{2} \leq 2^{m_x} |f_x| < 1$ is satisfied.

Arithmetic

Consider the operations of addition (subtraction), multiplication, and division. Generally, if at least one of the inputs is imprecise, the output is imprecise. In such cases, the adjustment rules for SDA are appropriate [4]. Moreover, these rules have the property that in general, normalized inputs lead to a normalized result. Hence, it is convenient to initiate the arithmetic using SDA rules, and only afterwards detect if both inputs are precise. If the latter condition is satisfied, the low-order part of the result, assumed available, must be examined; if zero, the high-order part is precise; if nonzero, the high-order part must be adjusted to reflect the imprecise character and then rounded (with caution, to avoid some curious pitfalls).

Addition

If x_1, x_2 are the inputs, and x_3 the output, the adjustment rule is

$$e_3 = \max \{e_1, e_2\} \tag{2}$$

that is, achieve exponent match by adjustment of the operand with smaller exponent, then perform operation. In the event of overflow, e_3 is increased by unity with a corresponding right shift of f_3. The details are shown in the flow diagram of Fig. 1. The notation is shown in the caption. To reduce the number of subscripts in the flow diagram, let $a = x_1, b = x_2$, and $s, p = x_3$ (for sum and product, respectively). In box 1, addition is performed with rounding deferred. The first digits of the

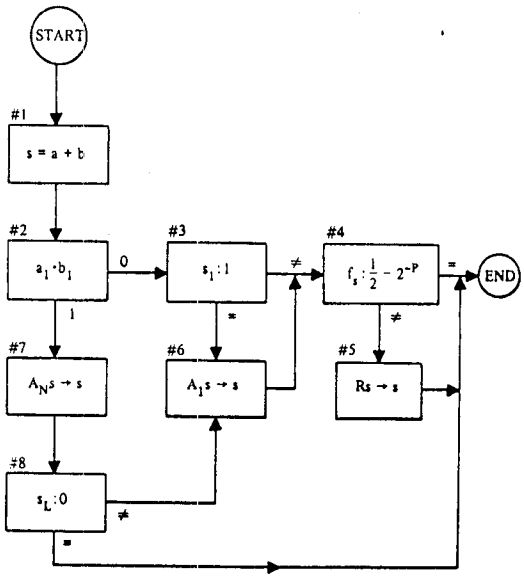

Fig. 1. Flow diagram for addition (subtraction) in ABC arithmetic. Numerical subscripts on operands identify bit positions; s_L is low-order part of s. Uppercase letters designate operators: A_k is adjust by k, A_N is normalize, and R is the round operator. The arrow is read "replaces," or simply "for." The colon indicates "compare"; control proceeds according to the outcome specified by the exit symbol, for example, "=" or "≠".

inputs are used to evaluate the Boolean AND function to determine whether both are precise. If not, "overflow" into the first digit position is checked; if no overflow, then rounding is effected in boxes 4 and 5. If the high-order part of s has a zero first digit followed by a sequence of ones, it is left unperturbed; all other possibilities are rounded. This exception, of course, avoids possible overflow owing to the rounding process and provides a more accurate representation (the exceptional case is usually overlooked in conventional normalized addition).

The other exit branch of box 3 indicates overflow; the tentative sum is adjusted by 1, using the operator A_1, and then rounded.

If the inputs are precise, the sum is normalized. The low-order part s_L is then examined; if $s_L \neq 0, s$ must be adjusted by 1 to represent an imprecise output; if $s_L = 0$, the operation is complete and no rounding is necessary.

Finally, it may be remarked that the most frequently traversed paths for both imprecise and precise results have been made short; the longer paths correspond to overflow or to a transition case from provisionally precise to imprecise.

Multiplication or Division

The flow diagram for multiplication is given in Fig. 2. Multiplication without rounding is effected in box 1. Recall that the adjustment rules are such that precise inputs at this stage lead to precise output; the simple test in box 2 discloses the (provisional) character of p. If imprecise, it is rounded and complete; otherwise, a test is made to determine whether the low-order product p_L is zero, If so, then p is precise and rounding is bypassed. If $p_L \neq 0, p$ is adjusted to indicate its imprecise character, and then it is rounded. It is easy to show that rounding cannot lead to overflow in contrast to addition.

The flow diagram for division is similar to that for multipli-

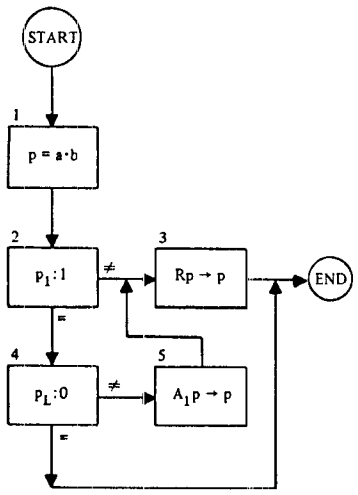

Fig. 2. Flow diagram for multiplication in ABC arithmetic. The notation is that given in Fig. 1.

cation and is omitted. Two remarks may, however, be appropriate: 1) rounding can lead to overflow; accordingly a test for potential overflow is made (cf. box 4 of Fig. 1). 2) The additional division step required for the rounding process in the conventional normalized method is not required here. For precise inputs, either the quotient is precise and rounding is vacuous, or the quotient is imprecise and the adjustment step yields the quotient digit for rounding. For imprecise inputs, the additional division step is, of course, never needed.

Physical Realization

The implementation of the present form of arithmetic processing in terms of solid-state circuitry is straightforward. For addition, with at least one imprecise input, the process would be similar to the conventional normalized approach except that here the postnormalization step is omitted. The digit s_1 must be examined to detect the overflow (cf. box 3 of Fig. 1); however, the standard word length (value of P) is usually large enough so that overflow is relatively infrequent. For precise inputs, an extra step is required to examine the low-order part of the sum; the presence of a 1 can, however, be detected during the exponent match operation, so that the extra time is almost negligible.

In multiplication or division, the quality of the result is established (provisionally) before the iterative procedure begins. If precise, the dichotomic character of p_L (box 4 of Fig. 2) is easily established in the course of shifting partial products in multiplication. For division, rapid detection of zero remainder is desired. Such zero detection is usually included for other purposes, in which case advantage may be taken to terminate in advance the iterative procedure of division (in an asynchronous processor).

For the present, ABC arithmetic is available on Maniac II by means of software. It has been incorporated as a distinct number type in the associated programming language Madcap [5]. A basic set of subroutines has been prepared that includes input/output, the logarithm, and trigonometric functions with their inverses.

The input/output routines have several convenient features. An exact quantity may be introduced as integer (without decimal point, of course), or as a ratio of integers, both cases with the option of adjoining exponent. If the quantity is not precisely representable, it is given maximal precision consistent with an imprecise quantity. For an imprecise quantity, the number of digits in the representation reflects its precision, i.e., every digit is regarded as meaningful. Briefly, the scientific notation is implied. For output, a precise quantity is either an integer (with exponent) or a ratio of integers (without exponent); the latter is irreducible, with a denominator a power of 2. An imprecise output exhibits only its meaningful decimal digits (rather than some uniform number of digits) coupled with an exponent and a separate decimal digit that reflects the precision in the last digit exhibited. This variation in the last digit stems from the conversion from one radix to another that is larger.

The operation times for addition and multiplication are an order of magnitude longer using a software approach; division is three times slower. Plans are being formulated for a hardware implementation. On the more modern CDC 6600, times for the software approach are only 40–70 percent longer depending upon the distribution of arithmetic operations. Obviously, the problem running time is affected much less.

Effect of Correlated Errors

The above adjustment rules for handling imprecise quantities are based upon their associated errors being statistically independent. If correlative effects develop among the intermediate and final quantities, the representation of an imprecise quantity may not be a reliable measure of significance. The assessment of significance based on representation can, in the presence of error correlation, be underestimated, in some cases, whereas in others it is overestimated. Given a task, one seeks an algorithm that takes account of the correlative effects and, as a consequence, provides a reliable assessment of precision, independent of the nature of the input. The fact that different algorithms for the some problem give different representations of the results is connected with the observation that mathematically equivalent algorithms are not computationally equivalent.

The issues here are similar to those encountered in the earlier SDA approach, but with the improvement in the handling of precise quantities. Moreover, the occurrence of imprecise output from precise inputs often suggests that word lengths be extended in certain parts of a computation.

A simple algorithm, such as the solution of a quadratic equation, has many pitfalls as Forsythe [6] has pointed out, if the detailed nature of arithmetic processing is not carefully considered. This example is discussed next.

Application

Forsythe has remarked that the seemingly simple algorithm for finding the roots of the quadratic equation $y = ax^2 + bx + c$ (a, b, c, real) can, under certain circumstances, give misleading results. He considers three sources of difficulty all connected with evaluating the discriminant $s = (b^2 - 4ac)^{1/2}$.

1) Scaling: The quantities a, b, c are in range, but the

products b^2, $4ac$ may be outside the range of computer representation (for a single word), and where a suitable change of scale for a, b, c would lead to a tractable solution.

2) Disparate Values: Specifically, if $b^2 >> 4ac$, then the smaller (in magnitude) root loses precision unduly if the conventional method is used. Following Kahan, Forsythe suggests alternative forms for the two roots; the appropriate pair is dictated by the sign of b.

3) Fantastic Cancellation: This awkwardness arises for precise a, b, c whenever $b^2 = 4ac \pm \epsilon$, where ϵ is sufficiently small so that greater precision is the representation of b^2 and $4ac$ is needed.

It is worth noting that Forsythe does not consider the case where a, b, c have inherent errors larger than that due to roundoff; this occurs frequently in problems arising in the natural sciences. Even in cases where a, b, c have only rounding errors, pitfalls occur and must be avoided. It would be quite restrictive to require that a, b, c always be precise, since the algorithm is viewed as a subroutine embedded in a larger problem where the coefficients could easily have rounding errors. On the other hand, if indeed a, b, c are precise, this fact should be distinguished from the imprecise case.

An algorithm has been proposed [1] that includes the imprecise case in a natural manner. Moreover, it handles the four examples suggested by Forsythe as illustrative of the pitfalls one may encounter in a less cautious version. Provision is made for linear scaling of the coefficients if intermediate quantities spill (overflow or underflow); the roots, of course, are invariant to such transformations. As mentioned before, a distinction is made between precise and imprecise coefficients. The former, but *not the latter* are treated using double-precision methods. In either case, the technique suggested by Kahan (cf. Forsythe [6]) is adopted. A detailed analysis shows that the method is appropriate for imprecise a, b, c in all instances (not merely for the disparate case) in the sense that it gives reliable estimates of the precision of the roots.

Finally, it may be remarked that it is possible to treat the case where the root(s) exceeds the range, although no intermediate quantity has. This is achieved by expressing the root as a product $r_i = \lambda_i r_i$, where the scale $\lambda_i > 1$ for excessively large r_i and $\lambda_i < 1$ for an excessively small root. An interrupt condition is used to transfer control to a root-scaling routine.

References

[1] V. Gardiner and N. Metropolis, "A comprehensive approach to computer arithmetic," Los Alamos Sci. Lab., Los Alamos, N. Mex., Rep. LA-4531, 1970.

[2] R. C. Blandford and N. Metropolis, "The simulation of two arithmetic structures," Los Alamos Sci. Lab., Los Alamos, N. Mex., Rep. LA-3979, 1968.

[3] V. Gardiner and N. Metropolis, "Significant digit arithmetic on a CDC 6600," Los Alamos Sci. Lab., Los Alamos, N. Mex., Rep. LA-4470, 1970.

[4] N. Metropolis and R. L. Ashenhurst, "Significant digit computer arithmetic," *IRE Trans. Electron. Comput.*, vol. EC-7, pp. 265–267, Dec. 1958.

[5] M. B. Wells, *Elements of Combinatorial Computing.* Oxford: Pergamon, 1971.

[6] G. E. Forsythe, "Solving a quadratic equation on a computer," in *The Mathematical Sciences.* Cambridge, Mass.: M.I.T. Press, 1969.

Nicholas C. Metropolis was born in Chicago, Ill., on June 11, 1915. He received the B.S. and Ph.D. degrees from the University of Chicago, Chicago, Ill., in 1936 and 1941, respectively.

In 1941 he became a Research Instructor at the University of Chicago. From 1946 to 1948 he was an Assistant Professor at the Institute for Nuclear Studies of the University of Chicago. From 1957 to 1965 he was a Professor at the Enrico Fermi Institute of Nuclear Studies. In 1942 he was a Physicist with the Manhattan Project, Columbia University, and with the Metallurgical Project, University of Chicago. From 1943 to 1946 and from 1948 to 1957 he was a Physicist with Los Alamos Scientific Laboratory, Los Alamos, N. Mex. From 1965 to the present he has been a Staff Member of the Los Alamos Scientific Laboratory. From 1958 to 1963 he was the Director of the Institute for Computer Research, University of Chicago. He also headed the groups that built electronic Computers Maniac I and Maniac II.

Dr. Metropolis is a fellow of the American Physical Society, and a member of the Association for Computing Machinery, the American Mathematical Society, and Sigma Xi.