$$Q^* = Qr^{s-t}$$
$$R^* = Rr^s.$$

The scale factor exponent $s-t$ will, however, probably provide less than optimal accuracy for the quotient. Therefore, we pick up quotient digits by rescaling the dividend by r^λ before the division operation. This is accomplished by a left shift of λ positions of the dividend in the $2n$-digit register before dividing. The scale factor exponent of the quotient resulting from the operation will then prove to be $s-t+\lambda$ and that of the remainder $s+\lambda$. To determine the range of permissible values of $s-t+\lambda$ we estimate

$$|Q| < r^{p'}$$

and use (1) to get

$$\lambda \leqq t - p' - s + n - 1 \tag{7}$$

where everything on the right is either known or estimated. Equality in (7) gives the maximum permissible left shift before dividing. If $p'=p$, the maximum shift yields the optimal quotient scaling. If $p'>p$, this and the other shifts λ in the range (7) will yield permissible scale factor exponents. If $p'<p$, (7) may permit too large a left shift thus leading to quotient overflow.

In practical applications of scaling it will be necessary to take account of many combinations of the four arithmetic operations. No new conditions are involved over and above those we have already discussed. The important thing is the correct estimate of bounds for the magnitudes of the results of the operations at each stage. In general, we will attempt to use optimal scaling, when possible, to minimize the error due to using a fixed number of digits. In some cases, due to the way operations combine, optimal scaling may be impossible. In other cases we may be willing to sacrifice some accuracy for the sake of scaling expediency.

Connections with Floating Point

In the discussion above we represented a number X by its scaled machine representation $X^*=Xr^s$. The number X^* was assumed stored in the registers of the machine but not the scale factor exponent s. This was assumed to be handled externally by the programmer. It is then the responsibility of the programmer to do the accounting necessary to associate an appropriate scale factor exponent with each machine number and record it so that the machine number can be properly interpreted in terms of the true number, for example, as $X=X^*\cdot r^{-s}$.

In floating point arithmetic we represent the true number X as

$$X = X^*r^x$$

where for $X\neq 0$, X^* is a normalized machine number. If, for easier correspondence with our previous discussion, we assume X^* is an n-digit complement integer, we recall that

$$r^{n-2} \leqq |X^*| < r^{n-1}. \tag{8}$$

The normalization condition (8) implies that the leading digit of X^* is a sign digit and that the next digit to it is not. In floating point operation we store both the normalized part X^* and the exponent part x internally and they are then dealt with automatically by the hardware and/or the program. Thus the responsibility for accounting for the scale factor exponent becomes that of the machine and not the programmer. Reviewing (5), (6), and (7) we see that this is a matter of very simple integral arithmetic on exponents provided an appropriate and consistent way of defining p' can be set down. The normalization condition on X^*, in effect, does this and, in fact, yields $p'=p$, and so optimal scaling at each step. To see this we note that if X^* is a normalized n-digit integer such that $X=X^*r^s$, then

$$X^* = \text{integral part of } [X\cdot r^{-x}].$$

That is, from the external scaling point of view, X^* is the machine representation of X with scale factor exponent $-x$. Therefore, if we determine p by the usual inequality

$$r^{p-1} \leqq |X| < r^p$$

and bear in mind the normalization condition (8) we get

$$r^{n-2} \leqq |X^*| = |Xr^{-x}| < r^{p-x}$$

and it follows that $x\leqq p-n+1$. Similarly, $x\geqq p-n+1$. Therefore, $x=p-n+1$. That is, $-x=n-1-p$. We see from (1) that this last is the optimal value for the scale factor exponent $-x$. This shows that normalizing always picks up the unique optimal scale factor exponent for X. The *negative* of this exponent is then stored internally along with X^* as the floating point machine representation.

In spite of the fact that the process of normalization theoretically produces optimal scaling in floating point operation, this is not always the case. It may be that the optimal exponent is too large or too small to be represented by the number of digits allocated to it. Although most computers can detect this overflow or underflow, it can usually be avoided by at least a preliminary rough scaling of the problem by the programmer himself.

A Binary Multiplication Scheme Based on Squaring

TIEN CHI CHEN, MEMBER, IEEE

Abstract—Using the formula $A\cdot B=[(A+B)/2]^2-[(A-B)/2]^2$, the binary multiplication problem is reducible to that of decomposing the square of $P\,0\cdot p_1p_2\cdots p_k$ into a sum of two or three quantities. For the eight-bit case, a study of the multiplication parallelogram suggests $p^2=R+S+T$, where p_1 and p_8 appear only in R, and p_2, p_7 appear only in R and S. Each bit in T involves the ORing of no more than four terms, each involving no more than four Boolean variables. For a two-input adder, S and T are combined into a six-variable problem, each bit may have up to 14 terms. The six- and four-bit problems are degenerate cases with $R=0$ and $R=S=0$, respectively.

Index Terms—Computer arithmetic, high-speed binary logic, multiplying technique, quarter-square multiplying technique, squaring technique.

Manuscript received September 29, 1970; revised January 18, 1971.
The author is with the IBM San Jose Research Laboratory, San Jose, Calif. 95114.

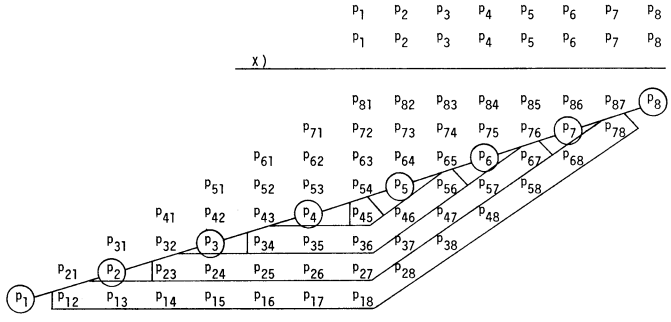

Fig. 1. The analysis of the squaring parallelogram.

In a recent communication Ling [1] described a binary multiplication scheme using $A \cdot B = 2[2^{2n}i - 2^{2m}j - 2^{2n}S(i) + 2^{2m}S(j)]$, with i, j being normalized fractions, $i = 2^{-n-1}(A+B)$, $j = 2^{-m-1}(A-B)$. If P is a k-bit fraction, then each bit in $P - P^2/2 \equiv S(P)$ can be specified by multiple ORing of terms involving the k bits in P. The case $k=8$ was then analyzed; the number of terms required to specify each bit of $S(P)$ vary from 1 to 26.

This is potentially a very high speed scheme that can exploit the inherent efficiency and fan-in fan-out characteristics of large-scale integrated (LSI) hardware. Furthermore, the interconnection problem is relieved in the sense that the original $2n$-bit problem is transformed into two well-defined problems each with essentially n bits.

The method is reminiscent of the quarter-square multiplication method in analog computation [2], with

$$A \cdot B = \left(\frac{A+B}{2}\right)^2 - \left(\frac{A-B}{2}\right)^2. \tag{1}$$

An alternative scheme based on this method can have a simpler, more efficient hardware implementation. The development of such a scheme is the subject of this note.

If A, B are fractions, so are the quantities $(A+B)/2$ and $(A-B)/2$. Because adders with two or three inputs are generally available, our main interest is to decompose the square of an arbitrary k-bit fraction $P = 0 \cdot p_1 p_2 \cdots p_k$ into a sum of two or three terms with minimum logic complexity; thus we are seeking Y, Z, R, S, and T such that

$$P^2 = Y + Z = R + S + T. \tag{2}$$

The squaring of P leads to a multiplication parallelogram (Fig. 1), a typical term being $p_{ij} \equiv p_i \wedge p_j$. Exploiting symmetry, one can write the result as follows for the case $k=8$:

$$P^2 = J+K+L+M+N$$

$$J = 0.0\ p_1\ 0\ p_2\ 0\ p_3\ 0\ p_4\ 0\ p_5\ 0\ p_6\ 0\ p_7\ 0\ p_8$$

$$K = 0.0\ p_{12}p_{13}p_{14}p_{15}p_{16}p_{17}p_{18}p_{28}p_{38}p_{48}p_{58}p_{68}p_{78}$$

$$L = 0.0\ 0\ 0\ p_{23}p_{24}p_{25}p_{26}p_{27}p_{37}p_{47}p_{57}p_{67}$$

$$M = 0.0\ 0\ 0\ 0\ 0\ p_{34}p_{35}p_{36}p_{46}p_{56}$$

$$N = 0.0\ 0\ 0\ 0\ 0\ 0\ 0\ p_{45} \tag{3}$$

where J is the antidiagonal, K is the southeast border of the parallelogram, and L, M, N are successive layers enveloped by K. We note that p_1, p_8 do not appear in L, M, N and p_2, p_7 are absent in M, N. Further, the dependence of J on p_1, p_2, p_7, and p_8 can be removed through trivial adjustments of K and L. We have

$$J + K + L = \hat{J} + \hat{K} + \hat{L} \tag{4}$$

with

$$\hat{J} = 0.0\ \ 0\ \ 0\ \ 0\ \ 0\ \ p_3\ \ 0\ \ p_4\ \ 0\ \ p_5\ \ 0\ \ p_6$$

$$\hat{K} = 0.p_{12}\ \ (p_1p_2')\ p_{13}\ \ p_{14}\ \ p_{15}\ \ p_{16}\ \ p_{17}\ \ p_{18}\ \ p_{28}\ \ p_{38}\ \ p_{48}\ \ p_{58}\ \ p_{68}\ \ p_{78}\ \ 0\ \ p_8$$

$$\hat{L} = 0.0\ \ 0\ \ p_{23}\ \ (p_2p_3')\ p_{24}\ \ p_{25}\ \ p_{26}\ \ p_{27}\ \ p_{37}\ \ p_{47}\ \ p_{57}\ \ p_{67}\ \ 0\ \ p_7.$$

We obtain $P^2 = Y + Z$ by identifying $Y = \hat{K}$ and $Z = \hat{J} + \hat{L} + M + N$. Note that Z involves only variables p_2 through p_7. The analysis of this expression is demonstrated in Table I, where each bit z_m involves no more than 14 terms, and each term involves no more than six variables. This is to be compared with the Ling algorithm, where a bit in $S(P)$ may involve 26 terms, and each term may involve seven variables (not eight, because p_1 can be normalized to unity).

With the use of three-input adders, we write $P^2 = R + S + T$ with the identification $R = Y = \hat{K}$, $S = \hat{L}$. Then $T(= \hat{J} + M + N)$ is easily analyzed; as shown in Table II, each bit t_m involves no more than four terms and each term no more than four variables.

The squaring problem for a six-bit fraction reduces to the case above with two of the input bits set to zero. However, the result is obtained more simply if the symmetry of the problem is maintained. We write

$$Q = 0 \cdot q_1\ q_2\ q_3\ q_4\ q_5\ q_6 = 2P \tag{5}$$

with $p_1 = p_8 = 0$, $p_k = q_{k-1}$ otherwise. In this formulation $\hat{K} = 0$, and Z alone gives the complete decoding of $P^2 = Q^2/4$. We can also write $P = S + T$; or $P = U + V + W$ with

$$U = 0.0\ \ 0\ \ 0\ \ p_{23}p_{24}p_{25}p_{26}p_{27}p_{37}p_{47}p_{57}p_{67}$$

$$V = 0.0\ \ 0\ \ 0\ \ 0\ \ 0\ \ p_{34}p_{35}p_{36}p_{46}p_{56}$$

$$W = 0.0\ \ 0\ \ 0\ \ 0\ \ 0\ \ p_3\ \ p_{45}(p_4p_5')0\ p_5\ \ 0\ \ p_6\ \ 0\ \ p_7.$$

In a similar vein the squaring of a four-bit fraction can be

TABLE I

THE ANALYSIS OF $p^2 = Y + Z$

$Y = 0.p_{12}\,(p_1p_2')\,p_{13}\,p_{14}\,p_{15}\,p_{16}\,p_{17}\,p_{18}\,p_{28}\,p_{38}\,p_{48}\,p_{58}\,p_{68}\,p_{78}\,0\;p_8 = \hat{K}$
$Z = 0.0\;\;0\;\;z_3\,z_4\,z_5\,z_6\,z_7\,z_8\,z_9\,z_{10}\,z_{11}\,z_{12}\,z_{13}\,z_{14} = \hat{J} + \hat{L} + M + N$

	p_2	p_3	p_4	p_5	p_6	p_7
$z_3 =$	1	0	1	1	1	X
	1	1	X	X	X	X
$z_4 =$	1	0	0	X	X	X
	1	0	1	0	X	X
	1	0	1	1	0	X
	1	1	1	X	X	X
$z_5 =$	0	1	0	1	1	1
	0	1	1	X	X	X
	1	0	1	0	X	X
	1	0	1	1	0	X
	1	1	X	1	X	X
	1	1	0	0	1	1
$z_6 =$	0	1	0	0	0	X
	0	1	0	1	0	X
	0	1	0	1	1	0
	0	1	1	1	X	X
	1	0	0	1	X	X
	1	0	1	0	1	1
	1	0	1	1	0	X
	1	1	0	0	0	X
	1	1	0	0	1	0
	1	1	0	1	1	X
	1	1	1	X	1	X

	p_2	p_3	p_4	p_5	p_6	p_7
$z_7 =$	X	1	0	1	0	X
	0	0	1	1	X	X
	0	1	0	1	1	0
	0	1	1	X	1	X
	1	0	0	X	1	X
	1	0	1	0	0	1
	1	0	1	0	1	0
	1	0	1	1	0	X
	1	0	1	1	1	1
	1	1	0	0	1	0
	1	1	0	1	1	1
	1	1	1	X	X	1
$z_8 =$	0	0	1	0	X	X
	0	0	1	1	1	X
	0	1	0	0	1	X
	0	1	0	1	1	0
	0	1	1	0	0	X
	0	1	1	0	1	1
	0	1	1	1	X	1
	1	0	0	X	X	1
	1	0	1	0	X	0
	1	0	1	1	0	1
	1	1	0	X	0	1
	1	1	0	X	1	0
	1	1	0	1	1	X
	1	1	1	0	0	0

	p_2	p_3	p_4	p_5	p_6	p_7
$z_9 =$	X	X	0	1	1	0
	X	X	1	0	1	0
	X	0	1	0	1	1
	X	0	1	1	0	1
	X	1	X	1	1	1
	X	1	0	0	1	1
	X	1	0	1	0	1
	X	1	1	0	0	1
$z_{10} =$	X	X	0	1	0	X
	X	X	0	1	1	1
	X	X	1	1	X	1
	X	X	1	1	0	0
$z_{11} =$	X	X	X	0	1	1
	X	X	X	1	0	1
$z_{12} =$	X	X	X	X	1	0
$z_{13} =$	X	X	X	X	X	$X(=0)$
$z_{14} =$	X	X	X	X	X	$1\ (=p_7)$

Example: $z_3 = p_{24}p_3'p_{56} \vee p_{23}$.

TABLE II

THE ANALYSIS OF $p^2 = R + S + T$

$R = Y = \hat{K}$
$S = 0.0\;\;0\;\;p_{23}\,(p_2p_3')\,p_{24}\,p_{25}\,p_{26}\,p_{27}\,p_{37}\,p_{47}\,p_{57}\,p_{67}\,0\,p_7 = \hat{L}$
$T = 0.0\;\;0\;\;0\;\;0\;\;t_5\,t_6\,t_7\,t_8\,t_9\,t_{10}\,t_{11}\,t_{12} = \hat{J} + M + N$

	p_3	p_4	p_5	p_6
$t_5 =$	1	1	X	X
$t_6 =$	1	0	X	X
	1	1	1	X
$t_7 =$	X	0	1	1
	0	1	1	0
	1	0	1	0
	1	1	X	1
$t_8 =$	X	1	0	0
	0	1	X	1
	1	0	X	1

	p_3	p_4	p_5	p_6
$t_9 =$	X	0	1	1
	X	1	0	1
$t_{10} =$	X	X	1	0
$t_{11} =$	X	X	X	$X\ (=0)$
$t_{12} =$	X	X	X	$1\ (=p_6)$

Example: $t_8 = p_4p_5'p_6' \vee p_3'p_{46} \vee p_{36}p_4'$.

obtained by examining $16P^2$ with $p_1 = p_2 = p_7 = p_8 = 0$. Here P^2 is equal to $V + W$, or T alone.

ACKNOWLEDGMENT

The author is indebted to Dr. H. Ling for discussions and an early perusal of Dr. Ling's paper. He wishes to acknowledge discussions with Dr. C. V. Freiman, who had considered approaches exploiting the quarter-square algorithm in 1965, and conversations with Dr. C. Tung and Dr. W. G. Tuel, Jr.

REFERENCES

[1] H. Ling, "High-speed computer multiplication using a multiple-bit decoding algorithm," *IEEE Trans. Comput.*, vol. C-19, Aug. 1970, pp. 706–709.

[2] A. S. Jackson, *Analog Computation.* New York: McGraw-Hill, 1960, p. 477.

A Distance Criterion for Figural Pattern Recognition

ALEXANDRU VINEA AND VLADIMIR VINEA

Abstract—The set of transformations which preserve the equivalence classes of figural patterns is considered. The simple figure and its abstract description in a metric space are defined. An appropriate distance is proposed to be applied to pattern recognition.

A trainable classifier using the nearest neighbor method is described. Some experiments with printed or handwritten letters are presented.

No statistical results are given, but the visual examination of figures that were recognized reveals that the method successfully imitates human performance.

Index Terms — Classifier, complex plane, contour following, distance criterion, Euclidean transformations, nearest neighbor recognition, pattern recognition, standard figure, training procedure.

Manuscript received November 10, 1969; revised October 1, 1970.

A. Vinea is with the Centre of Economic Computation and Economic Cybernetics, Bucharest, Romania.

V. Vinea is with the Institute for Electronics and Computing Technique, Bucharest, Romania.