Optical multiplication and division using modified-signed-digit symbolic substitution

Kai Hwang
University of Southern California
Department of Electrical Engineering
Los Angeles, California 90089-0781

Ahmed Louri, MEMBER SPIE
University of Arizona
Department of Electrical and Computer Engineering
Tucson, Arizona 85721

Abstract. The modified-signed-digit (MSD) number system offers parallel addition and subtraction of any two numbers, with carry propagation constrained only between two adjacent digits. Based on MSD addition, parallel algorithms for multiplication and division are developed in this paper. The optical implementations of these MSD arithmetic algorithms are developed on the basis of symbolic substitution (SS). The space-invariant nature of SS matches well with the parallel nature of the MSD arithmetic algorithms presented. The potential advantages of using these algorithms for optical computing include the significant increase in speed, full exploitation of parallelism, and higher system throughput compared with existing electronic arithmetic processors. The performance of the proposed optical arithmetic system is analyzed and compared with that of state-of-the-art electronic counterparts.

Subject terms: optical computing; digital arithmetic; modified-signed-digit numbers; parallel processing; fast multipliers; convergence division; symbolic substitution; performance analysis.

Optical Engineering 28(4), 364-372 (April 1989).

CONTENTS

1. INTRODUCTION

This paper deals with digital optical arithmetic using modified-signed-digit (MSD) representation. The advantages of optics for computing have been expounded upon on numerous occasions.[1] These include massive parallelism, high temporal and spatial bandwidth, high processing speed, and noninterfering communications. The MSD representation was originally invented by Avizienis[2] and was recently introduced to the optical community by Drake et al.[3] for improving the precision and accuracy of optical computations. This number system (MSD) uses radix $r = 2$ and the digit set $\{\bar{1},0,1\}$, where $\bar{1}$ stands for -1. The introduction of redundancy provides a much weaker interdigit dependency compared with the strong dependency manifested by long carry propagation in a conventional number system using the digit set $\{0,1\}$. As a consequence, carry generated at any stage is confined within two adjacent digital positions in MSD code. This makes it possible to perform the addition/subtraction of any two numbers of arbitrary length in constant time.[4]

Based on the MSD addition, we have developed parallel algorithms for multiplication and division. The multiplication algorithm is generalized from the one introduced by Drake et al.[3] The major difference lies in the means of generating the partial products and summing them in parallel. We show that the multiplication of two n-digit numbers can be done in $O(\log_2 n)$ time by first generating all n partial products simultaneously and then adding them in a treelike fashion. The parallel generation of all partial products is done in constant time, independent of the word length n. It is the adder tree that requires $\log_2 n$ time. A similar multiplication scheme using the MSD code has been introduced for VLSI implementation.[5] The division algorithm is generalized from the quadratic convergence division method.[6] With the provision of high speed multiplication and parallel addition, the number of required iterations for convergence division is reduced to $O(\log_2 n)$, where n is the fraction length.

We use the symbolic substitution (SS) technique for the optical implementation of the arithmetic algorithms. Symbolic substitution is a parallel processing technique that was introduced by Huang[7] for performing digital computations on optical data.

Paper OC-110 received June 20, 1988; revised manuscript received Nov. 28, 1988; accepted for publication Jan. 16, 1989. Preliminary versions of this paper were presented in the 1988 International Conference on Parallel Processing and in the SPIE conference on High Speed Computing (Paper 880-10), Jan.11–12, 1988, Santa Clara, Calif.

It consists of two processing phases: a *recognition* phase, in which the presence of a specific pattern is detected within a binary image, and a *substitution* phase, in which the present pattern is replaced by another pattern according to a predefined SS rule. The left-hand-side pattern of the SS rule, called the *search* pattern, is first searched in the input image and then is replaced by the right-hand side, called the *replacement* pattern. Optical implementation of these two SS phases have been investigated by many researchers.[8–14] Substitution rules for conventional binary addition were suggested by Brenner et al.[10] However, in Brenner's scheme the addition time is a function of the operand length owing to carry propagation in the conventional binary number system.

In this paper Sec. 2 presents the SS rules for MSD addition and subtraction and Sec. 3 discusses the optical implementation of these SS rules. Sections 4 and 5 respectively describe the MSD multiplication and division algorithms and their optical implementation. Section 6 shows optical conversion between binary numbers and MSD numbers. Section 7 assesses the potential performance of the MSD arithmetic algorithms presented.

2. SS RULES FOR MSD ADDITION AND SUBTRACTION

Avizienis[2] defined three successive steps to perform the addition of two signed-digit numbers $X = x_{n-1}\cdots x_0.x_{-1}x_{-2}\cdots x_{-m}$ and $Y = y_{n-1}\cdots y_0.y_{-1}y_{-2}\cdots y_{-m}$. At the first step, $x_i + y_i = 2t_{i+1} + w_i$ is performed at the *ith* digit position, for $i = -m,\ldots,n-1$, where w_i and t_{i+1} are called the *interim sum* digit and the *transfer* digit, respectively. These digits assume the values

$$w_i = \begin{cases} 1 & \text{if } x_i + y_i = \bar{1} \\ 0 & \text{if } |x_i + y_i| \neq 1 \\ \bar{1} & \text{if } x_i + y_i = 1 \end{cases}, \quad t_{i+1} = \begin{cases} 1 & \text{if } x_i + y_i \geq 1 \\ 0 & \text{if } x_i + y_i = 0 \\ \bar{1} & \text{if } x_i + y_i \leq \bar{1} \end{cases}. \tag{1}$$

At the second step, $w_i + t_i = 2t'_{i+1} + w'_i$ is performed to produce another pair of digits w'_i and t'_{i+1}:

$$w'_i = \begin{cases} 1 & \text{if } w_i + t_i = 1 \\ 0 & \text{if } |w_i + t_i| \neq 1 \\ \bar{1} & \text{if } w_i + t_i = \bar{1} \end{cases}, \quad t'_{i+1} = \begin{cases} 1 & \text{if } w_i + t_i = 2 \\ 0 & \text{if } |w_i + t_i| \neq 2 \\ \bar{1} & \text{if } w_i + t_i = -2 \end{cases}. \tag{2}$$

The third step generates the final sum digit s_i:

$$s_i = w'_i + t'_i = \begin{cases} 1 & \text{if } w'_i + t'_i \geq 1 \\ 0 & \text{if } w'_i + t'_i = 0 \\ \bar{1} & \text{if } w'_i + t'_i \leq \bar{1} \end{cases}. \tag{3}$$

Drake et al.[3] proposed the use of holographic elements, prisms, and optical bistable devices to implement the MSD addition. Recently, Bocker et al.[15] proposed implementing the MSD addition and subtraction using optical symbolic substitution. In addition, several other optical techniques have been suggested for the implementation of MSD addition and subtraction.[16–19] In this paper, we derive the SS rules for optical MSD addition from Eqs. (1), (2), and (3). These SS rules are similar to those presented in Ref. 15, the major difference being in the third stage of the addition and in the use of light intensity for encoding.

Using light intensity, two pixels of different intensity levels are needed to encode the three digits $\{\bar{1},0,1\}$. A possible encoding scheme is to represent digit 1 by a bright pixel above a dark one, digit $\bar{1}$ by the reverse pattern, and digit 0 by two dark pixels, as shown in Fig. 1(a). The extra pattern, two bright pixels, can be used as a delimiter to denote the fraction point. The advantage of this encoding scheme is that the significant digits (1 and $\bar{1}$) are encoded in dual-rail coding, which simplifies the SS recognition phase.[10]

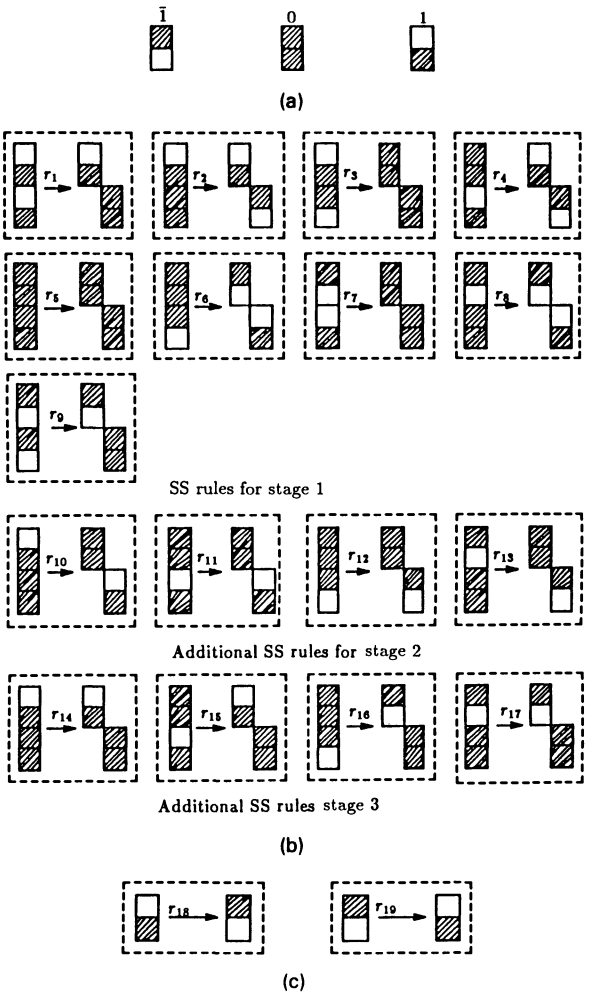


Fig. 1. Optical SS rules required for MSD addition and subtraction. (a) Light intensity encoding of the digit set $\{\bar{1},0,1\}$. (b) SS rules for optical MSD addition. (c) SS rules for MSD negation operation.

The light-intensity-coded SS rules for MSD addition are shown in Fig. 1(b). On the surface it seems that we need $3^3 = 27$ SS rules corresponding to the nine input combinations at each stage of the addition process. A closer look at the logic of the second and third stages reveals that fives entries are identical; furthermore, if we pad the last stage output with 0, five out of the nine entries of this last stage become identical to some entries in stages 2 and 3. Therefore, the total number of required SS rules becomes 17 using the encoding scheme presented in this paper. MSD subtraction is performed by first negating the nonzero digits of the subtrahend and then performing the addition of the two operands.[2] The digit negation requires two more SS rules, shown in Fig. 1(c).

3. OPTICAL IMPLEMENTATION OF SS RULES

The SS rules presented in this paper can be implemented using an *additive* process (image superimposition): the replicate-shift-

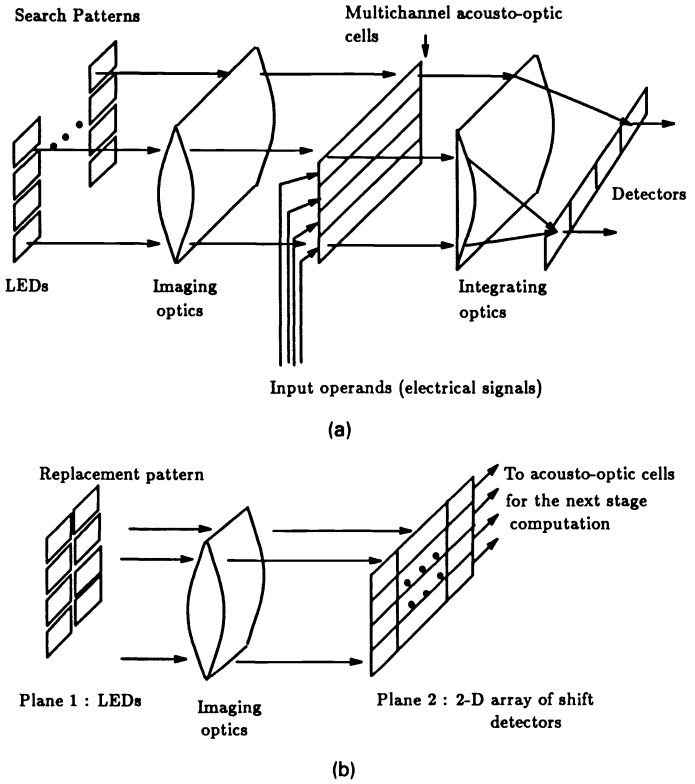


Fig. 2. Optical implementation of SS rules using multichannel acousto-optic cells. (a) Recognition phase using acousto-optic cells, LEDs, and detectors. (b) Substitution phase using LEDs and a 2-D array of shift detectors.

superimpose-threshold-mask-combine implementation method presented in Ref. 10 with a fan-in and fan-out of three.[10,20] Although it exploits space parallelism very effectively (several operands can be encoded on the input plane and processed at once), this implementation method relies on the use of 2-D arrays of optical logic gates (the inverting NOR-gate array in the recognition phase[10]), which should be reasonably large (1000×1000 gates). However, such components are at an embryonic stage,[21] at least for the desired size. Alternatively, one can implement the SS rules presented here using a *multiplicative* process and available optical devices. Botha et al.[8] have recently introduced an implementation method based on presently available optical devices such as light emitting diodes (LEDs) and multichannel acousto-optic (AO) cells and detectors. The method was intended for the implementation of the ripple carry addition rules.[10] We extend the optical setup to implement the optical MSD addition and subtraction.

The recognition phase is shown in Fig. 2(a). The search patterns are introduced through the LEDs. The reference plane consists of nine columns, corresponding to nine possible combinations of the digit set $\{\bar{1},0,1\}$. Each column contains four LEDs corresponding to the four pixels required to encode the operand digits. During the recognition, the input operands are fed to the multichannel AO cells, and the reference patterns are input through the LEDs. A multiplication of the reference patterns and the input operands takes place in the AO cells; the result is integrated onto the detectors. There are nine detectors, each corresponding to one of the nine search patterns at each stage of the optical addition. A peak signal on the output detectors indicates the presence in time of the searched pattern.

The substitution phase is implemented by the optical setup shown in Fig. 2(b). The output of the detectors of the recognition phase are used to scribe the desired substitution symbol on the LEDs situated in plane 1. The light emitted from the LEDs is imaged onto the array of shift detectors shown in plane 2. The size of this detector array is $n \times 4$, where n is the operand length and the factor 4 corresponds to the encoding scheme (two pixels per digit). The right end of the detector array is fed back to the AO cells for the next iteration. This implementation method does not rely on future optical components. All of the optical components required (LEDs, AO cells, detectors) are highly developed. With AO cells operating at gigahertz speed, this method may provide a very fast optical arithmetic processor, given that the response time of the detectors (for both the recognition and substitution phases) is made comparable with the rate of the AO cells. Its drawback is the limited number of operands that can be processed in parallel (within a reasonable hardware cost). Four AO cells are required to process two operands in parallel. To process N operands simultaneously, $N \times 4$ AO cells are needed. For the rest of the paper, the additive implementation method is assumed, for it offers much more parallelism than the multiplicative one, despite the unavailability of the devices (NOR-gate arrays) that are required for its implementation.

4. PARALLEL MSD MULTIPLICATION

The multiplication of two signed-digit (SD) numbers $X = x_{n-1}\cdots x_0.x_{-1}x_{-2}\cdots x_{-m}$ and $Y = y_{n-1}\cdots y_0.y_{-1}y_{-2}\cdots y_{-m}$ produces an SD product:

$$\begin{aligned} P &= p_{2n-1}p_{2n-2}\cdots p_0.p_{-1}p_{-2}\cdots p_{-2m+1}p_{-2m} \\ &= (y_{n-1} * X) \times 2^{n+m-1} + \cdots + (y_{-m+1} * X) \\ &\quad \times 2^1 + (y_{-m} * X) \times 2^0 , \end{aligned} \tag{4}$$

where y_i is the *ith* multiplier digit. The asterisk represents the signed AND operation that is defined below for any two SDs $x,y \in \{\bar{1},0,1\}$:

$$x * y = \begin{cases} 1 & \text{if } (x = y = 1) \vee (x = y = \bar{1}) \\ 0 & \text{if } (x = 0) \vee (y = 0) \\ \bar{1} & \text{if } (x = 1 \wedge y = \bar{1}) \vee (x = \bar{1} \wedge y = 1) \end{cases} . \tag{5}$$

The notations $\vee$ and $\wedge$ represent the conventional logical OR and logical AND operations. The notation $y_j * X$ defines the digitwise operations:

$$y_j * X = y_j * x_{n-1}, y_j * x_{n-2}, \ldots, y_j * x_{-m} . \tag{6}$$

We have previously developed a sequential algorithm for computing the product P in $n + m$ iterations using MSD additions and right shifts.[22] In the following, we present a parallel algorithm that computes the product of two MSD numbers in $\log_2(n+m)$ iterations, where $(n+m)$ is the word length including n integer digits and m fraction digits. For clarity, we consider only integer numbers where the fraction length $m = 0$.

4.1. MSD multiplication algorithm

Step 1. Given two signed n-digit numbers, generate all n partial products simultaneously, each having length n:

TABLE I. Example of parallel multiplication of 2 four-digit numbers.

Step 1: Generation of the partial products

$P_{0,0} = y_{-3} * X = 1.0\bar{1}\bar{1}$

$P_{0,1} = y_{-2} * X = \bar{1}.011$

$P_{0,2} = y_{-1} * X = 0.000$

$P_{0,3} = y_0 * X = 1.0\bar{1}\bar{1}$

Step 2: Shifting of the partial products

$(y_{-3} * X) \times 2^0 = 00010\bar{1}\bar{1}$

$(y_{-2} * X) \times 2^1 = 00\bar{1}0110$

$(y_{-1} * X) \times 2^2 = 0000000$

$(y_0 * X) \times 2^3 = 10\bar{1}\bar{1}000$

Step 3: Summation of the shifted partial products

$00010\bar{1}\bar{1}$ and $00\bar{1}0110$ → $000001\bar{1}\bar{1}1$

0000000 and $10\bar{1}\bar{1}000$ → $1\bar{1}\bar{1}01000$

$000001\bar{1}\bar{1}1$ and $1\bar{1}\bar{1}01000$ → $000.10010\bar{1}$

$X \times Y = Z = (000.10010\bar{1})_{SD} = (0.546875)_{10}$

$$P_{0,j} = y_j * X \quad \text{for } j = 0,\ldots,n-1 , \tag{7}$$

where the term $P_{0,j}$, an n-digit number, represents the *jth* partial product.

Step 2. Introduce the necessary shifts for each partial product. Each initial partial product $P_{0,j}$ will be shifted j digits to the left, corresponding to the weight factor 2^j shown in Eq. (4):

$$P_{0,j} = y_j * X \times 2^j \quad \text{for } j = 0,\ldots,n-1 , \tag{8}$$

Step 3. Pairwise add all partial products by means of an adder tree. With a total of n partial products at the leaves of the tree, the summation process takes $\log_2 n$ steps. At each step i, we perform $n/2^i$ SD additions in parallel:

$$P_{i,j} = P_{i-1,2j-2} + P_{i-1,2j-1} \quad \text{for } j = 1,2,\ldots,n/2^i , \; i = 1,\ldots,\log_2 n . \tag{9}$$

The final product is produced at the root of the binary tree. Steps 1 and 2 are carried out in constant time. For a multiplier of length n, step 3 requires $\log_2 n$ iterations. Since each MSD addition takes constant time, the multiplication of two n-digit MSD numbers can be carried out in $O(\log_2 n)$ time. Table I shows the parallel multiplication of two four-digit MSD numbers, $X = (1.0\bar{1}\bar{1})_{SD} = (0.375)_{10}$ and $Y = (1.0\bar{1}1)_{SD} = (0.875)_{10}$. In step 1, we generate all of the partial products using Eq. (7). In step 2, we introduce the necessary shifts. Finally, we add all of the shifted partial products using Eq. (9) and use a tree of SD adders to produce the final product $P = 000.10010\bar{1} = (0.546875)_{10}$.

4.2. Optical implementation of the MSD multiplication

The MSD multiplication algorithm uses the signed AND operation (*) to generate all partial products simultaneously and a tree of MSD additions to sum them. Using Eq. (5), we derive the SS rules needed for implementing the * operation, as shown in Fig. 3. Let us consider the optical implementation of the computations involved in Table I. The multiplicand and multiplier are arranged in 1-D arrays, as shown at the left in Fig. 4. The multiplicand is shown horizontally, and the multiplier is shown vertically. The generation of all partial products $P_{0,j}$ for $j = 0,\ldots,3$ is carried out in three stages. First, the multiplicand is spread out vertically by anamorphic optics (represented by lens L1 in Fig. 4) to fill the 4×4 data plane M1. Similarly, the multiplier is spread out horizontally so that each digit of the multiplier is duplicated vertically four times to fill the 4×4 plane M2. Next, planes M1 and M2 are 2-D perfect shuffled[12] and then stored in an 8×4 plane R. For clarity, the optics required for the 2-D perfect shuffle permutations are omitted from Fig. 4. The 2-D shuffle permutations intended here affect only the row position, leaving the column position of the data unchanged. Optical means of implementing the 2-D perfect shuffle have been suggested elsewhere.[12,23,24]

The resulting image R has alternating rows from M1 and M2 such that odd rows contain the multiplicand and even rows contain a replicated digit of the multiplier. Therefore, rows 1, 3, 5, ..., $n-1$ contain the multiplicand X; rows 2, 4, 6, ..., n contain the replicated digits $y_1, y_2, y_3, \ldots, y_{n-1}$ of the multiplier,

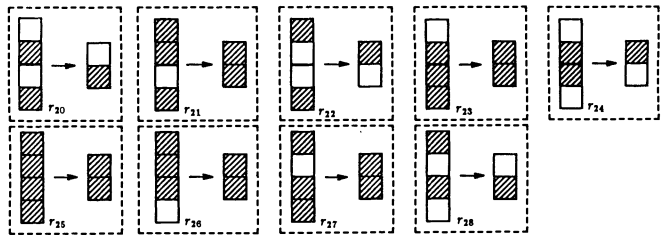


Fig. 3. Optical SS rules needed for the MSD AND operation.

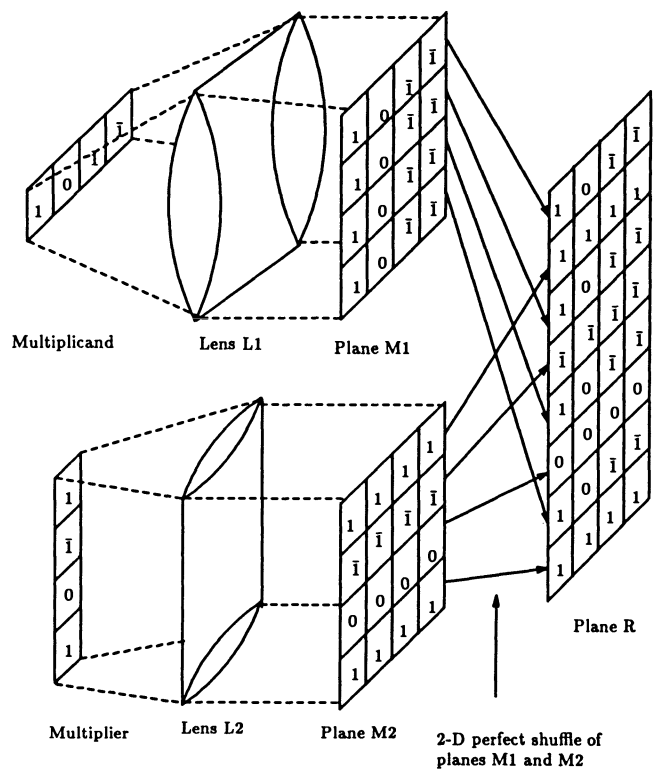


Fig. 4. Optical spreading of the operands for parallel MSD multiplication.

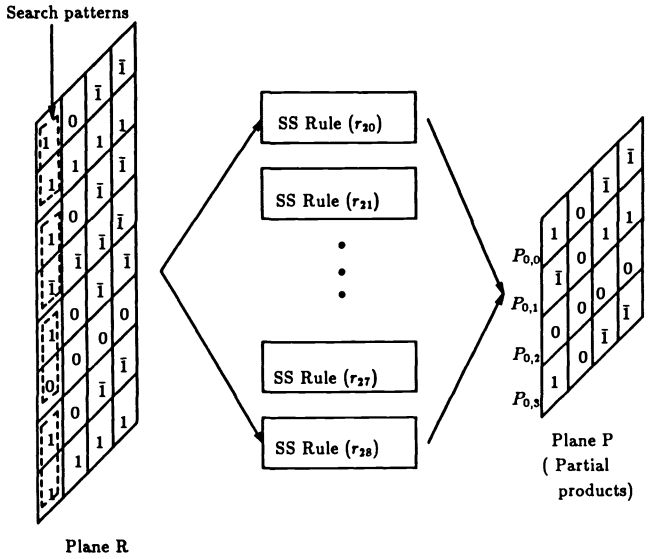


Fig. 5. Parallel generation of all partial products using the SS rules of the MSD AND operation.

respectively. In the third stage, plane R is replicated nine times; each copy is used for applying one SS rule of the $*$ operation. Therefore, every combination of the input operands is searched and replaced in parallel. Finally, the output planes of all of the SS rules applied are optically superimposed. To this end, all of the partial products have been generated in parallel as shown in plane P of Fig. 5.

Step 2 of the SD multiplication algorithm, shifting of the initial partial products, can now be performed. There are a variety of ways one can perform spatial shifts in optics. Figure 6(a) shows four possible methods for spatially shifting an optically encoded SD number. All of these methods can be extended to 2-D arrays, considering the ease with which optical systems can spread their hardware into two and three dimensions. From a reliability point of view, the holographic method may be the most reliable one for our application.[25]

Using the holographic method, we need a holographic element consisting of four subholograms. One subhologram with a fan-out of one can be used to shift one partial product. Within each subhologram, the input/output interconnections are space invariant since every pixel of a particular partial product is shifted by the same amount to the left. The holographic element as a whole is considered space variant since the interconnection patterns differ for each row. Figure 6(b) illustrates the parallel shifting of all partial products via the holographic element H, which consists of four subholograms. When illuminated by its corresponding partial product, each subhologram of H will reconstruct a shifted partial product on the plane S as shown. The plane S, consisting of all shifted partial products, is then fed to the MSD adder described in the previous section to perform the last step of the multiplication algorithm. This is accomplished by applying the MSD addition rules for $\log_2 4 = 2$ iterations.

In general, with a multiplicand of length n and a multiplier of length m, the planes M1, M2, and P are all $m \times n$ arrays, R is a $2m \times n$ array, and S is an $m \times (m+n)$ array. It should be noted that if the 1-D arrays used to input the operands in Fig. 4 are replaced by 2-D arrays and associated optics for spreading and shuffling, many operand pairs can be multiplied in parallel using the same set of SS rules.

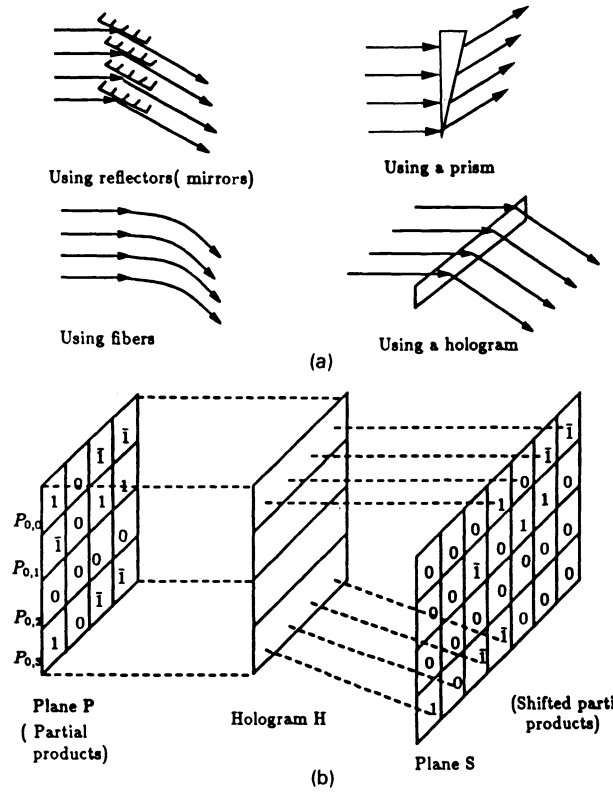


Fig. 6. Optical means for achieving spatial shifts and the shift of the partial products using holographic element H. (a) Optical methods for achieving spatial shifting. (b) Parallel shift of partial products using a hologram.

5. MSD CONVERGENCE DIVISION

The conventional restoring and nonrestoring division methods require knowledge of the sign of the partial remainder for exact selection of the quotient digits.[4] However, in MSD representation the sign of a partial remainder is not readily available if several most-significant digits are zero. This difficulty prevents the use of conventional methods for MSD division. In searching for an effective division algorithm for SD numbers with radix $r = 2$, we have to meet the following requirements: (1) The algorithm should overcome the difficulty of testing the polarity of the remainder after each iteration, (2) it should make effective use of the parallel MSD addition and multiplication schemes developed in previous sections, and (3) it should take full advantage of the massive parallelism and high speed of optics.

An MSD division algorithm satisfying these goals is developed here based on a convergence approach. Division by convergence is not a new idea. In fact, several convergence division schemes have been implemented with conventional binary numbers.[4,26,27] We extend the convergence method to achieve parallel division of signed-digit numbers.

Let us consider a dividend X and a divisor Y, both MSD fractions in normalized form; that is,

$$\tfrac{1}{2} \leq |X| < Y < 1 \ . \tag{10}$$

We want to compute the quotient $Q = X/Y$ without a remainder. The algorithm uses a sequence of multiply factors $m_0, m_1, m_2, \ldots, m_n$ such that $Y \times (\Pi_{i=0}^{i=n} m_i)$ converges to 1 (within

TABLE II. MSD convergence division example.

Iteration step	Multiply factor	Accumulated denominator	Accumulated numerator
i = 0	$m_0 = 2 - Y_0$	$Y_1 = Y_0 \times m_0$	$X_1 = X_0 \times m_0$
	$(1.01)_{SD}$	$(1.000\bar{1})_{SD}$	$(0.\bar{1}\bar{1}10)_{SD}$
	$= (1.25)_{10}$	$= (0.9375)_{10}$	$= (-0.625)_{10}$
i = 1	$m_1 = 2 - Y_0 \times m_0$	$Y_2 = Y_0 \times m_0 \times m_1$	$X_2 = X_0 \times m_0 \times m_1$
	$(1.001\bar{1})_{SD}$	$(1.0000001)_{SD}$	$(\bar{1}.1\bar{1}10\bar{1}\bar{1}10)_{SD}$
	$= (1.0625)_{10}$	$= (0.99509)_{10}$	$= (0.66406)_{10}$
i = 2	$m_2 = 2 - Y_0 \times m_0 \times m_1$	$Y_3 = Y_0 \times m_0 \times m_1 \times m_2$	$X_3 = X_0 \times m_0 \times m_1 \times m_2$
	$(1.0000001)_{SD}$	$(1.0000000000000001)_{SD}$	$Q = (\bar{1}.1\bar{1}10\bar{1}\bar{1}10\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}1)_{SD}$
	$= (1.00390625)_{10}$	$Y_3 \rightarrow 1$	$= (-0.6666..)_{10}$

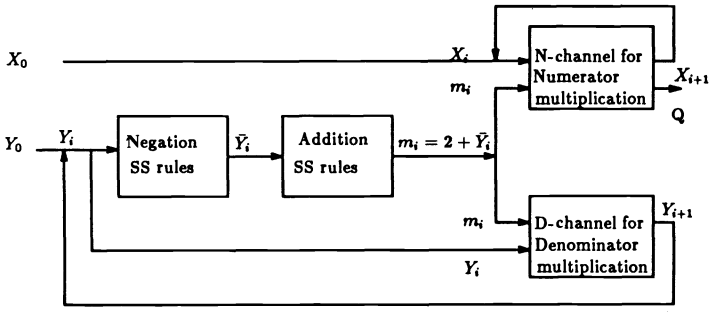


Fig. 7. Optical MSD convergence divider using two channels of optical multipliers.

an acceptable error criterion). Initially, we set $X_0 = X$ and $Y_0 = Y$. The algorithm repeats the following recursions,

$$X_{i+1} = X_i \times m_i \ , \quad Y_{i+1} = Y_i \times m_i \ , \tag{11}$$

such that for small n

$$Y \times \left(\prod_{i=0}^{i=n} m_i\right) \rightarrow 1 \ , \quad Q = X \times \left(\prod_{i=0}^{i=n} m_i\right) \ . \tag{12}$$

The effectiveness of this convergence method relies on the ease of computing the multiply factors m_i using only MSD addition and multiplication operations. The recursive formula of Eq. (11) can be rewritten as

$$Y_{i+1} = Y_i \times m_i = f(Y_i) \ . \tag{13}$$

We desire the function $f(Y_i)$ to converge to 1, starting from an initial value $Y_0 = Y$. Equation (13) can be rewritten in a polynomial form:

$$f(Y_i) - Y_i = 0 \ . \tag{14}$$

Flynn[26] described several iterative methods to enable such a polynomial to converge to a given value, say k. We are interested only in the quadratic convergence since this appears more convenient for optical realization.[20] To achieve this, let us rewrite Eq. (14) in a more general quadratic form:

$$f(Y_i) - Y_i = (Y_i - k_1)(Y_i - k_2) = 0 \ . \tag{15}$$

One of the roots of Eq. (15) should be 1. Krishnamurthy[27] found that for $f(Y_i)$ to converge quadratically to 1 the factors m_i should be selected as

$$m_i = 2 - Y_i \ , \quad 0 < Y_i < 2 \ . \tag{16}$$

Equation (16) implies that the multiply factor for each iteration can be easily obtained as the two's complement of the previous denominator Y_i. In MSD code, the arithmetic expression $2 - Y_i$ can be computed in constant time using SS rules for MSD negation and addition. Since the convergence is quadratic, the accumulated denominator length is doubled after each iteration. Hence, for a desired quotient of length n, the maximum number of iterations needed is $\log_2 n$. The complete MSD division algorithm is as follows:

begin
for i := 0 **to** $(\log_2 n) - 1$ **do**
  $m_i := 2 - Y_i$;
  $X_{i+1} := X_i \times m_i$;
  $Y_{i+1} := Y_i \times m_i$;
endfor;
$Q := X_{(\log_2 n)-1}$;
end.

Table II illustrates the MSD convergence division of $X = (0.\bar{1}0)_{SD} = (-0.5)_{10}$ by $Y = (0.11)_{SD} = (0.75)_{10}$. For 16-digit precision, the method generates the quotient after three iterations, $Q = (\bar{1}.1\bar{1}10\bar{1}\bar{1}10\bar{1}\bar{1}\bar{1}\bar{1}\bar{1}\bar{1})_{SD} = (-0.66664)_{10}$.

Each iteration of the MSD division consists of three major operations: a pair of MSD multiplications, $Y_{i+1} = Y_i \times m_i$ and $X_{i+1} = X_i \times m_i$, and a two's complement operation $m_i = 2 - Y_i$. Each MSD multiplication can be carried out optically as described in Sec. 4. The two's complement is carried out by a negation operation followed by an addition operation. The subtrahend Y_i is negated using the SS rules in Fig. 1(c). All nonzero digits of Y_i are negated in parallel. The expression $2 - Y_i$ then becomes $2 + \bar{Y}_i$, which is computed using the MSD addition rules in Fig. 1(b). The two MSD multiplications required to generate X_{i+1} and Y_{i+1} can be computed concurrently by replicating the MSD multiplication hardware into two channels, one for the numerator and one for the denominator, as shown in Fig. 7.

6. CONVERSION BETWEEN BINARY AND MSD REPRESENTATIONS

When the operands are in binary form, we must (1) convert them to MSD representation, (2) perform arithmetic operations using the parallel algorithms presented here, and (3) convert the result back to binary representation, as shown in Fig. 8(a).

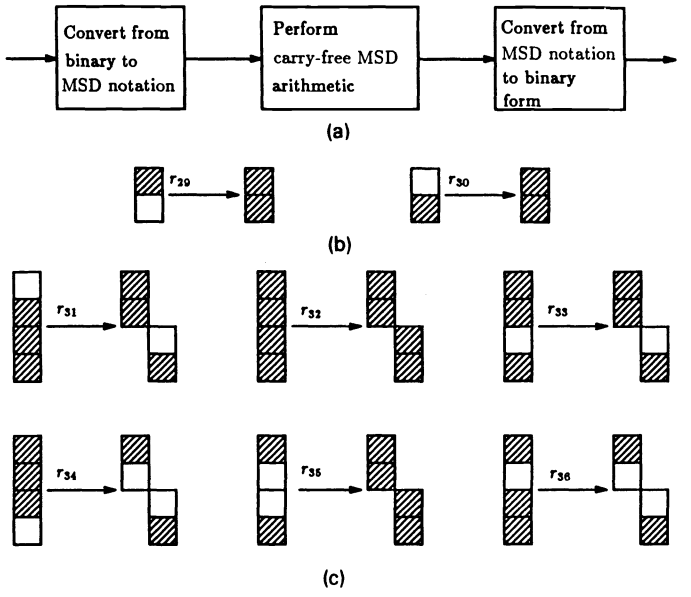


Fig. 8. Conversion from binary to MSD code and vice versa. (a) Logic diagram showing the sequence of operations. (b) SS rules for converting an MSD number to two components: Y^+ and Y^-. (c) SS rules for converting an MSD number to a two's complement number.

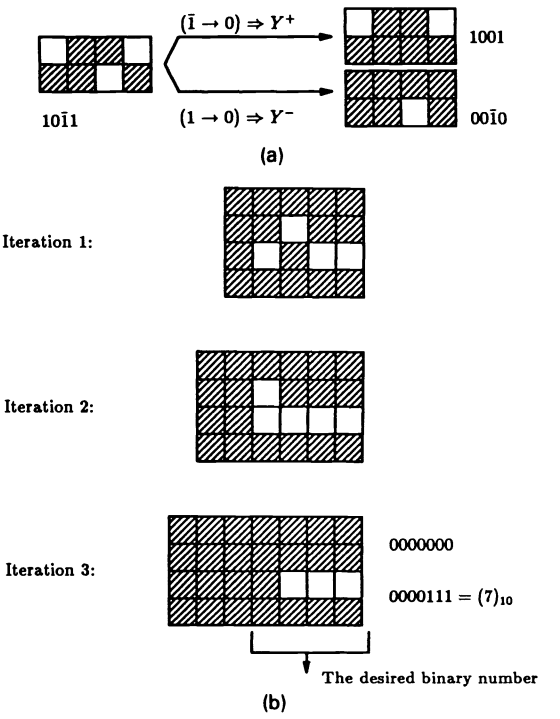


Fig. 9. Example of optical conversion from an MSD number to a two's complement number. (a) Splitting Y into two parts: Y^+ for positive digits and Y^- for negative digits. (b) Application of the SS conversion rules of Fig. 8 for three iterations.

6.1. From binary code to MSD code

Let us consider two representations of the binary number system, namely, the unsigned notation and the two's complement notation.[4] An n-bit unsigned binary number $x_{n-1},x_{n-2},\ldots,x_0$, where $x_i \in \{0,1\}$, is equivalent to an MSD number $y_{n-1},y_{n-2},\ldots,y_0$, where $x_i = y_i$ and the equivalence means same algebraic value. Therefore, no computation is involved in converting the unsigned number to an MSD equivalent. The same number can be fed to the MSD processor. An n-bit two's complement number $x_{n-1},x_{n-2},\ldots,x_0$ is equivalent to an n-digit SD number $y_{n-1},y_{n-2},\ldots,y_0$, where $y_i = x_i$ for all $i = 0,\ldots,n-2$ and

$$y_{n-1} = \begin{cases} 0 & \text{for } x_{n-1} = 0 \\ \bar{1} & \text{for } x_{n-1} = 1 \end{cases} . \tag{17}$$

Therefore, the conversion of a two's complement number to an MSD equivalent can be done easily by changing the most significant bit x_{n-1} to 0 or $\bar{1}$. This can be carried out in constant time, independent of the word length n.

6.2. From MSD code to two's complement code

The conversion from MSD code to binary format can be carried out as follows: we separate an MSD number Y into two components Y^+ and Y^-, where Y^+ is an unsigned number formed by the positive digits of Y, and Y^- is that formed by the negative digits of Y; the binary number X, equivalent to Y, is then obtained by subtracting Y^- from Y^+: $X = Y^+ - Y^-$. The number X is an $(n+1)$-bit integer in two's complement form. The optical implementation of this conversion method can be carried out in two steps. First, the MSD number Y is replicated into two identical copies. In one copy, we replace all of the occurrences of negative digits ($\bar{1}$) by a zero symbol using SS rule r_{29} of Fig. 8(b), leaving the 0s and 1s unchanged. This produces the positive part Y^+. In the other component, we replace all of the occurrences of 1 by a zero symbol using SS rule r_{30} of Fig. 8(b), leaving the 0s and $\bar{1}$s unchanged. This produces the negative part Y^-. We put Y^+ above Y^- and apply the SS rules shown in Fig. 8(c) repeatedly until the top word becomes all zeros and the bottom word becomes the converted number. The total number of iterations is equal to the total number of nonzero digits (1s and $\bar{1}$s) in the original MSD number to be converted.

The above method is best explained by an example. Let us consider the conversion of the MSD number $Y = 10\bar{1}1$. Figure 9 shows the different stages involved. Initially, the MSD number Y is converted to two numbers Y^+ and Y^-, as shown in Fig. 9(a). Next, the SS rules of Fig. 8(c) are applied to the two words for three iterations, corresponding to the total number of significant digits in Y. The final two's complement number is equal to $(7)_{10}$, as depicted in Fig. 9(b). The conversion from MSD to binary code using the above method can be performed in a time proportional to the number of nonzero digits in the MSD number. The conversion can be performed in a time proportional to $\log_2 n$ for an operand of length n, using the carry-lookahead technique.[28] However, the hardware complexity of the conversion grows in proportion to n.

7. PERFORMANCE ANALYSIS

We evaluate the performance of the optical arithmetic algorithms presented in this paper by estimating the processing time of each operation. The optical implementation of symbolic substitution is assumed to be the additive method based on replication, spatial shifts, superimposition, thresholding, masking, and combining as described in Ref. 10. The key parameters used in the analysis are T_p, the propagation time of a light beam through passive optical devices such as lenses, beamsplitters, and holograms; T_s, the switching time of the optical memory devices used to

hold the data; T_f, the feedback time for light propagation through the feedback interconnect (the MSD addition requires three successive steps, where the result of one stage is fed back to the next stage logic as described in Sec. 2); and T_{activ}, the response time of an optical NOR-gate array used for inversion and thresholding.

7.1. Optical addition time

The MSD addition is performed in three stages. The total time to perform each stage is attributed to the time needed to (1) hold the input image, (2) replicate the input image, (3) propagate the image through the first hologram to provide the shifts, (4) activate the optical NOR-gate array for inverting the superimposed image, (5) propagate light through the second hologram for substitution, (6) superimpose the output of all of the rules, and (7) feed back the intermediate result. Therefore, the total SD addition time is estimated as

$$T_{add} = 3(\overbrace{T_s}^{(1)} + \overbrace{T_p}^{(2)} + \overbrace{T_p}^{(3)} + \overbrace{T_{activ}}^{(4)} + \overbrace{T_p}^{(5)} + \overbrace{T_p}^{(6)}) + \overbrace{2T_f}^{(7)} . \quad (18)$$

The numbers over the braces indicate the times needed to accomplish each subtask as enumerated above. T_p and T_f can be approximated by 0.1 ns (light propagates at 1 ft/ns in free space). The dominant limitations to speed are the switching times of the optical NOR-gate array and the optical memory elements, representing the only active elements in the addition path.[29] Therefore, the total MSD addition time would be $T_{add} \approx 6T_{activ}$ (assuming $T_s = T_{activ}$). An n-digit MSD addition requires $(n+1) \times 4$ pixels, where the factor 4 is introduced by the encoding scheme used (two light pixels for each digit). Therefore, for an optical gate array of size $\ell \times \ell$ pixels and a switching time T_{activ}, the optical MSD adder is able to perform Θ_a n-digit additions per second, where

$$\Theta_a = \frac{\ell \times \ell}{6T_{activ} \times [(n+1) \times 4]} . \quad (19)$$

Optical gate arrays of very small sizes (say 2×2 to 6×6) have been demonstrated recently.[30] These arrays offer the possibility of achieving a 10^{-12} s switching time. However, these optical gate arrays cannot be used in a practical system owing to their small size and high power consumption. If we were to use a commercial spatial light modulator (SLM) such as the liquid crystal light valve (LCLV) with a 500×500 pixel resolution and 20 ms switching time,[31] we could perform about 16 $\times 10^3$ 32-digit MSD additions per second. This yields an average of $1/\Theta_a = 62$ μs per MSD addition. This speed is slower than today's electronic adders. However, faster SLMs are being produced in research laboratories.[21] If the response time of the SLM were reduced to 0.01 μs (10 ns), a 500×500 resolution would bring the 32-digit MSD addition time down to 30 ps, which represents a 100 times improvement over electronic adders of the same precision.

7.2. Optical multiplication time

Referring to the optical implementation model in Sec. 4, the MSD multiplication time is attributed to the time needed to (1) generate the partial products, (2) shift them, and (3) add up the shifted partial products. This time is expressed as

$$T_{mult} = \overbrace{T_{sp} + T_s + 4T_p + T_{activ}}^{(1)} + \overbrace{T_p}^{(2)} + \overbrace{T_{add} \times \log_2 n}^{(3)} , \quad (20)$$

where T_{sp} represents the time needed to spread and shuffle the operands. This time corresponds to light propagation through passive devices, which can be estimated by 0.1 ns. Since $T_{sp} \approx T_p << (T_{activ} = T_s)$ and $T_{add} \approx 6T_{activ}$, then $T_m \approx 2T_{activ}(1+3\log_2 n)$, where n is the precision of the multiplier. An n-digit MSD multiplication requires $4 \times (n \times 2n)$ pixels, where the factor 4 is related to the light encoding of the digit set $\{\bar{1},0,1\}$. Using an SLM with $\ell \times \ell$ pixel resolution and T_{activ} switching time, we obtain Θ_m n-digit MSD multiplications performable per second:

$$\Theta_m = \frac{\ell \times \ell}{4 \times (n \times 2n) \times 2T_{activ} \times (1+3\log_2 n)} . \quad (21)$$

If we were to use a standard off-the-shelf SLM with 500×500 resolution and 20 ms switching time, there could be 48 MSD multiplications per second. This corresponds to a speed of $1/\Theta_m = 20$ ms per one 32-digit SD result. This looks very slow; however, if the switching time of the SLM were reduced to 0.01 μs, the 32-digit MSD multiplication time would be reduced to 10 ns, which is about 10 times faster than today's electronic multipliers of the same word length.

7.3. Optical division time

Consider the optical implementation shown in Fig. 7. The time required to perform one iteration of the SD convergence division consists of the time needed to (1) generate the multiplicative factor m_i and (2) produce the next numerator and denominator X_{i+1}, Y_{i+1}. This time is then multiplied by the logarithm of the fraction length to obtain the total SD division time T_{div}:

$$T_{div} = \overbrace{(T_s + 4T_p + T_{activ} + T_{add} + T_f}^{(1)} + \overbrace{T_{mult} + T_f)}^{(2)} \times \log_2 n . \quad (22)$$

Substituting into Eq. (22) T_{add} and T_{mult} from Eqs. (18) and (20), respectively, we obtain $T_{div} \approx T_{activ}\log_2 n(10+6\log_2 n)$. An important feature of the MSD division algorithm is that several dividends can be divided simultaneously by the same divisor. This is because the multiply factors and the convergence rate depend only on the magnitude of the divisor. An n-digit MSD division requires $4 \times (n \times 2n)$ pixels to hold the accumulated numerators or denominators (assuming that we are truncating the intermediate products by n digits after each iteration). Therefore, for an optical gate array of $\ell \times \ell$ resolution and T_{activ} switching time, we obtain Θ_d MSD divisions per second:

$$\Theta_d = \frac{\ell \times \ell}{4 \times (n \times 2n) \times T_{activ} \times \log_2 n(10+6\log_2 n)} . \quad (23)$$

For a resolution $\ell \times \ell = 500 \times 500$ and a switching time $T_{activ} = 0.01$ μs, the time needed for a 32-digit MSD division would be $1/\Theta_d$, which is around 65 ns, or 100 times faster than electronic dividers.

It can be seen from the analysis that to exploit the potential advantages of optics (speed and parallelism), much research is needed to develop better nonlinear optical devices (optical gate

arrays and SLMs) with faster switching time and significantly reduced switching energy. A significant speedup of an optical arithmetic computer over the electronic counterpart depends heavily on the maturity of optical switching technology in the years to come.

8. CONCLUSIONS

The MSD code allows parallel addition-subtraction to be performed in constant time. In this paper, we have developed parallel algorithms for MSD multiplication and division based on parallel MSD addition. The execution times of the proposed multiplication and division algorithms are both proportional to $\log_2 n$, where n is the length of the multiplier and of the divisor. The optical implementation is based on symbolic substitution. We have presented new SS rules and discussed available optical setups for implementing the parallel algorithms.

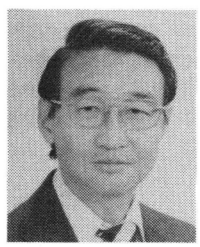
We have assessed the performance of optical arithmetic based on state-of-the-art optical devices. We conclude that the speedup over electronic counterparts is rather limited owing to today's slow switching time of optical gate arrays. If the switching time of the optical gate arrays were reduced to nanosecond range, we could perform 32-digit optical addition, multiplication, and division with a speedup ranging from O(10) to $O(10^3)$ over existing electronic counterparts. Therefore, the potential of building future supercomputers with optical arithmetic units looks very promising and encouraging.

9. ACKNOWLEDGMENTS

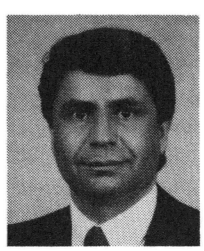
This research was supported in part by Office of Naval Research contract 14-86-K-559 and in part by National Science Foundation grant DMC-84-21022. The authors thank the referees for providing constructive suggestions.

10. REFERENCES

1. A. A. Sawchuk and T. C. Strand, "Digital optical computing," Proc. IEEE 72(7), 758–779 (1984).
2. A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," Trans. Elect. Comput. EC-10, 389–398 (1961).
3. B. L. Drake, R. P. Bocker, M. E. Lasher, R. H. Patterson, and W. J. Miceli, "Photonic computing using the modified signed-digit number representation," Opt. Eng. 25(1), 038–043 (1986).
4. K. Hwang, *Computer Arithmetic: Principles, Architecture, and Design*, Wiley, New York (1979).
5. N. Takagi, H. Yasuura, and S. Yajima, "High-speed VLSI multiplication algorithm with a redundant binary addition tree," IEEE Trans. Comput. C-34, 789–796 (1985).
6. R. Z. Goldschmidt, "Applications of division by convergence," master's thesis, MIT, Cambridge (1964).
7. A. Huang, "Parallel algorithms for optical digital computers," in *Proc. IEEE Tenth Int. Optical Computing Conf.*, pp. 13–17 (1983).
8. E. Botha, D. Casasent, and E. Barnard, "Optical symbolic substitution using multichannel correlators," Appl. Opt. 27(5), 817–818 (1988).
9. K.-H. Brenner, "New implementation of symbolic substitution logic," Appl. Opt. 25(18), 3061 (1986).
10. K.-H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," Appl. Opt. 25(18), 3054 (1986).
11. Y. Li, G. Eichmann, R. Dorsinville, and R. R. Alfano, "An AND operation-based optical symbolic substitution, "Opt. Commun. 63(6), 375–379 (1987).
12. A. Louri and K. Hwang, "A bit-plane architecture for optical computing with two-dimensional symbolic substitution," in *Proc. 15th Int. Symposium on Computer Architecture*, Honolulu, IEEE Computer Society Press, pp. 18–29 (1988).
13. J. N. Mait and K.-H. Brenner, "Optical symbolic substitution: system design using phase-only holograms," Appl. Opt. 27(9), 1692–1700 (1988).
14. F. T. S. Yu, C. Zhang, and S. Jutamulia, "Application of one-step holographic associative memories to symbolic substitution," Opt. Eng. 27(5), 399–402 (1988).
15. R. P. Bocker, B. L. Drake, M. E. Lasher, and T. B. Henderson, "Modified signed-digit addition and subtraction using optical symbolic substitution," Appl. Opt. 25(15), 2456–2455 (1986).
16. A. K. Cherri and M. A. Karim, "Modified-signed digit arithmetic using an efficient symbolic substitution," Appl. Opt. 27(18), 3824–3827 (1988).
17. Y. Li and G. Eichmann, "Conditional symbolic modified signed-digit arithmetic using optical content-addressable memory logic elements," Appl. Opt. 26(12), 2328–2333 (1987).
18. M. M. Mirsalehi and T. K. Gaylord, "Logical minimization of multilevel coded functions," Appl. Opt. 25(18), 3078–3088 (1986).
19. P. A. Ramamoorthy and S. Anthony, "Optical modified signed digit adder using polarization-coded symbolic substitution," Opt. Eng. 26(8), 821–825 (1987).
20. K. Hwang and A. Louri, "Optical arithmetic using symbolic signed-digit substitution," in *17th Int. Conf. on Parallel Processing* (St. Charles, Ill.), Pennsylvania State Univ. Press, pp. 55–64 (1988).
21. A. D. Fisher and J. N. Lee, "Current status of two-dimensional spatial light modulator technology," in *Optical and Hybrid Computing*, H. H. Szu, ed., Proc. SPIE 634, 352–371 (1987).
22. K. Hwang and A. Louri, "New symbolic substitution algorithms for optical arithmetic using signed-digit representation," in *High Speed Computing*, D. P. Casasent, ed., Proc. SPIE 880, 90–99 (1988).
23. K.-H. Brenner and A. Huang, "Optical implementations of the perfect shuffle interconnections," Appl. Opt. 27(1), 135–137 (1988).
24. A. W. Lohmann, W. Stork, and G. Stucke, "Optical perfect shuffle," Appl. Opt. 25(10), 1530 (1986).
25. B. K. Jenkins, P. Chavel, R. Forchheimer, A. A. Sawchuk, and T. C. Strand, "Architectural implications of a digital optical processor," Appl. Opt. 23(19), 3465–3474 (1984).
26. M. J. Flynn, "On division by functional iteration," IEEE Trans. Comput. C-19(8), 702–706 (1970).
27. E. V. Krishnamurthy, "On optimal iterative schemes for high-speed division," IEEE Trans. Comput. C-19(3), 227–231 (1970).
28. S. Unger, "Tree realization of iterative circuit," IEEE Trans. Comput. C-26, 365–383 (1977).
29. T. J. Cloonan, "Performance analysis of optical symbolic substitution," Appl. Opt. 27(9), 1701–1707 (1988).
30. G. Livescu, D. A. B. Miller, J. E. Henry, A. C. Gossard, and J. H. English, "Spatial light modulator and optical dynamic memory using a 6×6 array of self-electro-optic-effect devices," Opt. Lett. 13, 297–299 (1988).
31. M. T. Fatehi, K. C. Wasmundt, and S. A. Collins, Jr., "Optical logic gates using a liquid crystal light valve: implementation and application example," Appl. Opt. 20(13), 2250–2256 (1981).



Kai Hwang received the Ph.D. degree from the University of California at Berkeley in 1972 and is now a professor of electrical engineering and computer science at the University of Southern California. He has authored and coauthored more than 100 scientific papers on computer organizations, parallel algorithms, supercomputers, multiprocessors, and neural and optical computing. He is the author of *Computer Arithmetic* (Wiley, 1979) and *Computer Architecture and Parallel Processing* (McGraw-Hill, 1984) and edited *Supercomputer Design and Applications* (IEEE Computer Society Press, 1984) and *Parallel Processing for Supercomputers and Artificial Intelligence* (McGraw-Hill, 1989). He is the editor-in-chief of the *Journal of Parallel and Distributed Computing*. Dr. Hwang was elected an IEEE Fellow for contributions to digital arithmetic, system architectures, and parallel processing. He chaired the IEEE/ACM 1985 Seventh Symposium on Computer Arithmetic and the 1986 International Conference on Parallel Processing.



Ahmed Louri received the Diplome d'Ingenieur in Electrical Engineering from the University of Science and Technology (USTO), Oran, Algeria, in 1982 and the master's and the doctorate degrees in computer engineering from the University of Southern California in 1985 and 1988, respectively. From 1982 to 1988, he was the recipient of a six-year National Talent Search Scholarship from the government of Algeria. From 1986 to 1988, he worked as a researcher with the Computer Research Institute, University of Southern California, where he conducted extensive research in parallel processing and multiprocessor system design. Since graduating in 1988, he has been on the faculty of the Department of Electrical and Computer Engineering, University of Arizona, where he is currently an assistant professor. His areas of research include computer architecture, parallel processing, and optical computing. Dr. Louri is a member of the IEEE, OSA, and SPIE.