Programming of optical array logic. 2: Numerical data processing based on pattern logic

Jun Tanida, Masaki Fukui, and Yoshiki Ichioka

A new technique for space-variant processing with optical array logic and a new concept for parallel processing called pattern logic are proposed. Optical array logic is a technique for achieving any parallel neighborhood operation by simple coding, optical correlation, and parallel OR operation. Using pattern logic, various kinds of parallel processing can be realized, which can be implemented by optical array logic. Several kinds of numerical data processing are presented to verify the capability of pattern logic.

I. Introduction

Specialized algorithms are required for parallel computation on an optical computing system using the excellent capabilities of optical parallel processing. Optical processing has the capability of performing massively parallel computation, making good use of the inherent nature of light propagation. However, direct photon control is difficult, so that an optical computing system should be designed on a different basis from that of electronic computers. One reasonable architecture is that composed of a large number of simple processors capable of using the parallel nature of light propagation. In such a system, designing and executing operational procedures differ from those in electronic computers. Thus a new concept is required to use the optical computing system effectively.

One promising concept for optical parallel processing is coded pattern processing in which data to be processed are converted into patterns in images, which are processed with operations for the images. Since pattern manipulation is required as a basic operation for 2-D data, the coding technique is useful to increase the usable field of optical parallel processing. Coded pattern processing has many advantages such as flexibility and capability of parallel manipulation for a large amount of data.

Several techniques have been proposed to implement coded pattern processing. Symbolic substitution[1] provides a generalized concept of coded pattern processing. Whereas several practical procedures have been developed with symbolic substitution,[2,3] we aim to achieve coded pattern processing through another approach, i.e., using optical array logic (OAL).[4–7] OAL is a technique for parallel neighborhood operations by simple coding, optical correlation, and parallel processing. OAL can be executed effectively with the OPALS (optical parallel array logic system).[8–10] However, OAL is a SIMD (single instruction stream multidata flow) system, so that its field of use is restricted. Thus, we have considered one approach to break through the limitation.

In this paper we propose a new technique for space-variant processing with OAL and a new concept for parallel processing called pattern logic (PTL). In Sec. II we explain OAL and space-variant processing by OAL. In Sec. III we describe the concept of PTL and its implementation method by OAL. In Sec. IV we demonstrate some kinds of numerical processing to verify the capabilities of PTL. In Sec. V the possibility of data-driven processing is presented. In Sec. VI we discuss the computational advantages of our approach.

II. OAL and Space-Variant Processing

OAL is a technique to achieve any parallel neighborhood operation for two 2-D binary data.[6,7] Figure 1 shows the processing procedures of OAL. Two binary images to be processed are encoded according to the coding rule shown in Fig. 1 and converted into a coded image. The coded image is separately correlated with different operation kernels. The individual correlated images are spatially sampled at one-pixel size intervals (double the structural cell size) in the vertical and the horizontal directions. Parallel NAND (exactly, OR for data expressed by negative logic) operation for all the sampled images provides the result of a parallel neighborhood operation.

The authors are with Osaka University, Department of Applied Physics, Suita, Osaka 565, Japan.

Received 10 August 1987.
0003-6935/88/142931-09$02.00/0.

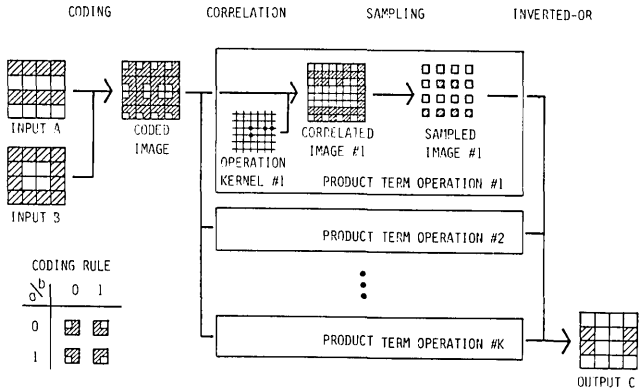


Fig. 1. Processing procedures of OAL for a parallel neighborhood operation including coding rule.

The features of OAL are as follows:

(1) Any parallel neighborhood operations can be designed, described, and executed systematically.

(2) Since OAL has a close relationship to both array logic[11] and cellular logic[12] in electronics, their programming resources can be used in programming OAL.

(3) SIMD type of parallel processing is achieved.

Although the third feature is attractive for global or space-invariant processing, e.g., some kinds of image processing,[5] it restricts the usefulness of OAL. For example, when multibit numerical data processing is attempted, localized or space-variant processing is required, which cannot be achieved by the original procedure of OAL. Although many bit-plane memories enable us to execute such processing,[13] we have devised a technique for space-variant processing with OAL to extend the application field of OAL.

The fundamental of the technique is that one of the 2-D inputs of OAL is used for data to be processed and the other is used for the selector of the operation to be executed. When OAL is considered as a system with two inputs and one output, it is regarded as a space-invariant system. However, if one of the inputs is assigned to the selector of operation, OAL is regarded as a space-variant system with one input and one output. Therefore, using the one 2-D input image for the selector, space-variant processing can be achieved with OAL.

Using logical expression, the parallel neighborhood operation by OAL is written as follows[7]:

$$c_{i,j} = \sum_{k=1}^{K} \prod_{m=-L}^{L} \prod_{n=-L}^{L} f_{m,n;k}(a_{i+m,j+n}, b_{i+m,j+n}), \qquad (i,j = 1, \ldots, N), \quad (1)$$

where a and b mean binary data in images A and B, respectively; subscripts denote the location of the data in the image. Here $f_{m,n;k}(a,b)$ refers to a two-variable binary logic function for pixels a and b; subscripts m and n indicate the relative address of a pixel in the neighborhood area centering on (i,j); and k is an identifier of product terms. Σ and Π denote logical sum and logical product operators, respectively; and N, L, and K are image size, neighborhood area size, and product term number, respectively. Since OAL executes SIMD processing, the same operation is used for all pixels in the images. Thus Eq. (1) can be written without specifying the pixel location, i and j,

$$c = \sum_{k=1}^{K} \prod_{m=-L}^{L} \prod_{n=-L}^{L} f_{m,n;k}(a_{m,n}, b_{m,n}). \qquad (2)$$

This abbreviation is used in this paper to describe OAL operations.

The proposed technique for space-variant processing with OAL is explained as follows: We assign image A to the image containing the selector of operation and image B to that containing the data. Then Eq. (2) can be rewritten as

$$c = \sum_{k=1}^{K} P_k(\mathbf{a},\mathbf{b}) Q_k(\mathbf{b}), \qquad (3)$$

where

$$P_k(\mathbf{a},\mathbf{b}) = \prod_{m=-L}^{L} \prod_{n=-L}^{L} p_{m,n;k}(a_{m,n}, b_{m,n}), \qquad (4)$$

$$Q_k(\mathbf{b}) = \prod_{m=-L}^{L} \prod_{n=-L}^{L} q_{m,n;k}(b_{m,n}), \qquad (5)$$

$$f_{m,n;k}(a_{m,n}, b_{m,n}) = p_{m,n;k}(a_{m,n}, b_{m,n}) q_{m,n;k}(b_{m,n}); \qquad (6)$$

$p(a,b)$ and $q(b)$ are also binary logic functions with two and one variables, respectively.

In Eq. (3) evaluation of $Q_k(\mathbf{b})$ is valid only when $P_k(\mathbf{a},\mathbf{b}) = 1$. This $P_k(\mathbf{a},\mathbf{b})$ is called a function selector, which becomes 1 when $\mathbf{a}$ and $\mathbf{b}$ match predefined patterns, respectively. Note that $\mathbf{a}$ and $\mathbf{b}$ express specific patterns of pixels in the neighborhood area. Then combining $P_k(\mathbf{a},\mathbf{b})$ and $Q_k(\mathbf{b})$, we can select operations by patterns of $\mathbf{a}$ and $\mathbf{b}$, so that space-variant processing can be achieved. The reason $\mathbf{b}$ is applied in the function selector is to use the value of the data for the selecting condition.

For convenient description and intuitive comprehension of OAL, a symbolic notation is used,[7] in which a neighborhood operation is expressed as follows:

$$\begin{bmatrix} \cdot\cdot & .0 & \cdot\cdot \\ \cdot\cdot & \underline{11} & \cdot\cdot \\ \cdot\cdot & \cdot\cdot & \cdot\cdot \end{bmatrix} + \begin{bmatrix} \cdot\cdot & \cdot\cdot & \cdot\cdot \\ \cdot\cdot & \underline{01} & \cdot\cdot \\ \cdot\cdot & .0 & \cdot\cdot \end{bmatrix}, \qquad (7)$$

where the symbols in the brackets denote the two-variable binary logical functions tabulated in Fig. 2 and the underscore indicates the origin of the neighborhood area ($m = 0, n = 0$). Expression (7) denotes

Function	Symbol	Kernel Unit	Function	Symbol	Kernel Unit
1	..	#	a + b	PP	#
$\bar{a} + \bar{b}$	NN	#	a ⊕ b	UU	#
$\bar{a} + b$	NP	#	b	.1	#
$\bar{a}$	0.	#	$\bar{a}$ b	01	#
$a + \bar{b}$	PN	#	a	1.	#
$\bar{b}$	.0	#	a $\bar{b}$	10	#
$\overline{a \oplus b}$	EE	#	a b	11	#
$\bar{a}\,\bar{b}$	00	#	0	00	#

Fig. 2. Symbols for symbolic notation of OAL.

$$c = a_{0,0}b_{0,0}\bar{b}_{-1,0} + \bar{a}_{0,0}b_{0,0}\bar{b}_{1,0}. \quad (8)$$

In Eq. (8), if $a_{0,0} = 1$, then $c = b_{0,0}\bar{b}_{-1,0}$ is evaluated, or else $c = b_{0,0}\bar{b}_{1,0}$ is executed. Thus $a_{0,0}$ and $\bar{a}_{0,0}$ work as operation selectors, and two different operations are selected and executed according to the value of $a_{0,0}$.

To execute OAL efficiently, we have proposed the OPALS. The OPALS is a parallel digital processing system, and several kinds of optical implementation have been discussed, for example, shadow-casting system with electronic parallel feedback loop,[6] multi-imaging system consisting of spatial light modulators and optical flip-flop devices.[8] Thus, the proposed technique for space-variant processing can be executed on the OPALS. In other words, this technique is addressed to one of the software techniques for the OPALS, which offers the means to execute space-variant processing on the OPALS.

The limitation of this method is the number of different operations to be executed. With a little thought, it can be realized that any attempt to increase the kinds of operations results in complication of the logical operation expressed in Eq. (3), i.e., increase of the product terms. In OAL, a product term corresponds to a sequence of correlations with an operation kernel (shown in Fig. 1), so that many sequences are required for such a complicated operation and the performance of OAL is significantly reduced. Therefore, the proposed method is only useful when the number of kinds of operation is few, i.e., MSIMD (multisingle instruction stream multidata flow) processing.

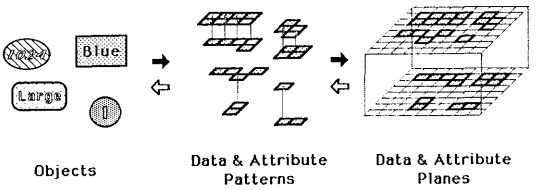


Fig. 3. Schematic diagram of pattern logic. Objects to be processed are converted into pixel patterns and placed in image planes. They are processed using parallel neighborhood operations.

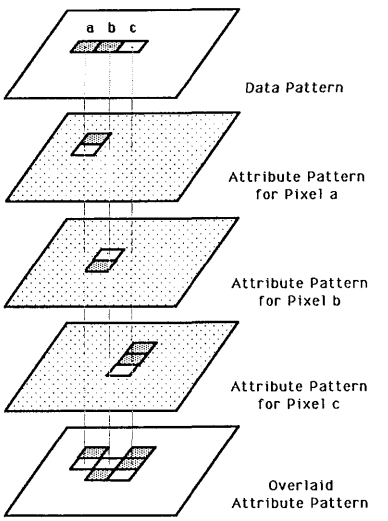


Fig. 4. Data and attribute patterns. An attribute is defined for each pixel in a data pattern. The projected pattern of all attribute patterns is called an overlaid attribute pattern.

III. Concept of PTL

PTL is a concept of parallel processing in which information is expressed as a coded pattern and processed by space-variant neighborhood operations. Figure 3 shows a conceptual diagram of PTL. In PTL, objects to be processed may have different data types, e.g., numerical data, character data. One object is coded into a binary data pattern and placed in the data plane. Every pixel in the data pattern has its own attribute pattern indicating its data type in the attribute plane. Both data and attribute patterns may be placed at an arbitrary location in the planes if the location of a pixel in the data plane corresponds to that of the origin of its attribute pattern. Many object patterns can be placed in a processing plane as long as the patterns do not overlap.

Here, we define the terminology for the pattern logic. A 2-D image used in PTL is called a processing plane and identified as a data plane or an attribute plane according to its use. A pixel is a primitive element of the processing plane. A set of localized pixels can express various kinds of information. We refer to the localized pixels as an object pattern, which is specifically called a data pattern (on a data plane) or an attribute pattern (on an attribute plane). An attribute pattern is defined for every pixel in a data pattern, so that every pixel in a data pattern has its own attribute pattern as shown in Fig. 4. The projected pattern of all attribute patterns is called an overlaid attribute pattern, and the pixels other than object patterns are called background pixels.

Processing in PTL is executed by space-variant neighborhood operations for the data patterns. The attribute pattern works as a tag assigning the operation to be executed to the pixel, namely, pixels with the same attribute pattern are processed by the identical operation. Then, even if various types of data are mixed in the data plane, they can be processed by different operations according to their attribute patterns. After the processing the data patterns are decoded into the original form of information such as numerical data or character data and so on.

PTL can be implemented by OAL with the technique for space-variant processing described in Sec. II. The data plane and attribute plane of PTL are assigned to two input images of OAL and the attribute pattern corresponds to the function selector. Thus the data patterns and the attribute patterns are places in images B and A, respectively. To program PTL, Eq. (3) is used. After the programming, PTL is executed according to the OAL procedure shown in Fig. 1. Thus, the OPALS can effectively execute PTL.

The advantage of PTL is that a large amount of data, which may be different types of data, can be collectively processed without care for its location and type.

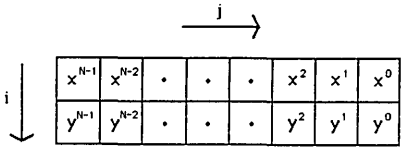


Fig. 5. Data structure for N-bit signed integers. For two-operand operation, two patterns are placed side by side.

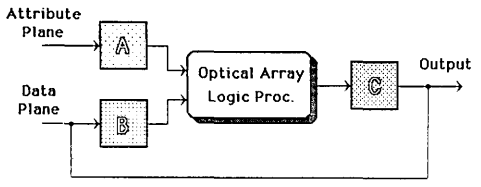


Fig. 6. Data flow in the OPALS. Algorithms presented in this paper are assumed to be processed according to this data flow.

Since each data point has its own information of what operation is applied to it, the PTL programmer does not care about the absolute location of the data. This becomes significant as the degree of parallelism increases, namely, in a massively parallel system, much effort is needed to calculate the addresses of data to be processed. In an electronic computer, a memory managing unit executes this troublesome task and the programmer usually does not worry about the memory address. However, if PTL is used with the OPALS, such a memory management mechanism can be simplified and more efficient parallel processing can be expected.

The limitation of PTL is the number of data types because of the limitation of OAL as discussed in Sec. II. However, some kinds of processing do not require so many data types and can effectively derive the capabilities of PTL as discussed below.

IV. Numerical Data Processing with PTL

To verify the capabilities of PTL, some kinds of numerical data processing are attempted. Numerical data are assumed to be N-bit signed integers and coded by strip patterns as shown in Fig. 5. The signed number is represented by twos complement. For a two-operand operation, the strip patterns are placed side by side in the processing plane. The data flow of the processing is assumed to be the same as that of the OPALS as shown in Fig. 6. A feedback loop is formed in the OPALS for efficient use of the system hardware; the output of the system is used as one of the system inputs at the successive processing stage.

Note that each algorithm consists of several processing steps. In this paper, we use the term step for a functional block of an algorithm and stage for each iteration of OAL.

A. Addition and Subtraction

One of the basic and important numerical operations is addition. When two N-bit signed integers, $\mathbf{x}{:}x^{N-1}x^{N-2}\ldots x^0_{(2)}$ and $\mathbf{y}{:}y^{N-1}y^{N-2}\ldots y^0_{(2)}$, are addends, addition is achieved by the following algorithm[1]:

(1) Execute the sum operation

$$s^i = x^i \oplus y^i, \tag{9}$$

and the carry operation

$$c^{i+1} = x^i y^i, \tag{10}$$

for $i = 0$ to $N-1$ in parallel, where s^i and c^i denote the ith bit of the sum and the carry, respectively; $\oplus$ is an XOR operator.

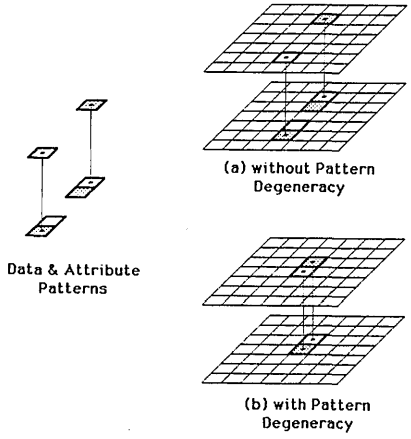


Fig. 7. Attribute patterns for addition and their pattern degeneracy; data arrangements (a) without pattern degeneracy and (b) with pattern degeneracy.

(2) Substitute $\mathbf{s}{:}s^{N-1}s^{N-2}\ldots s^0_{(2)}$ and $\mathbf{c}{:}c^{N-1}c^{N-2}\ldots c^0_{(2)}$ for $\mathbf{x}$ and $\mathbf{y}$, respectively.

(3) Repeat steps 1 and 2 until $\mathbf{x}$ becomes zero (maximum N iteration); then $\mathbf{y}$ provides the result.

In this algorithm two kinds of operation, sum and carry, are used. Referring to Eqs. (9) and (10) and to the data arrangement in Fig. 5, we can obtain the following neighborhood operations:

$$S(\mathbf{b}) = b_{0,0} \oplus b_{-1,0}, \tag{11}$$

$$C(\mathbf{b}) = b_{0,1}b_{1,1}. \tag{12}$$

These operations can be executed simultaneously with PTL.

Two attribute patterns are designed as shown in Fig. 7, which are used for the attribute patterns of x^is and y^is. While the attribute patterns consist of two pixels, they can be meshed without affecting the spacing of the data patterns and provide one overlaid attribute pattern as shown in Fig. 7. This is regarded as a sort of pattern degeneracy, which is useful for saving space in processing planes.

To detect the above attribute patterns, two operations are designed as operation selectors:

$$P_S(\mathbf{a},\mathbf{b}) = \bar{a}_{0,0}a_{-1,0}, \tag{13}$$

$$P_C(\mathbf{a},\mathbf{b}) = a_{0,0}\bar{a}_{1,0}. \tag{14}$$

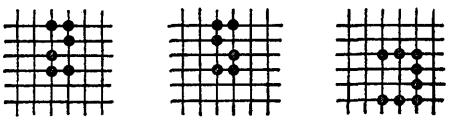
Then the neighborhood operation to be executed is

$$
\begin{aligned}
c &= P_S(\mathbf{a},\mathbf{b})S(\mathbf{b}) + P_C(\mathbf{a},\mathbf{b})C(\mathbf{b}) \\
&= \bar{a}_{0,0}a_{-1,0}(b_{0,0} \oplus b_{-1,0}) + a_{0,0}\bar{a}_{1,0}(b_{0,1}b_{1,1}) \\
&= \bar{a}_{0,0}b_{0,0}a_{-1,0}\bar{b}_{-1,0} + \bar{a}_{0,0}\bar{b}_{0,0}a_{-1,0}b_{-1,0} \\
&\quad + a_{0,0}\bar{a}_{1,0}b_{0,1}b_{1,1}. \qquad (15)
\end{aligned}
$$

This operation is developed into the sum of products for the symbolic notation.[7] Thus the following symbolic notation is obtained:

$$
\begin{bmatrix} .. & 10 & .. \\ .. & \underline{01} & .. \\ .. & .. & .. \end{bmatrix} + \begin{bmatrix} .. & 11 & .. \\ .. & \underline{00} & .. \\ .. & .. & .. \end{bmatrix} + \begin{bmatrix} .. & .. & .. \\ .. & \underline{1.} & .1 \\ .. & 0. & .1 \end{bmatrix}. \qquad (16)
$$

The advantage of the symbolic notation is that the operation kernels to be used in OAL can be generated using the rule in Fig. 2, namely,



(17)

are generated from Eq. (16). Using these operation kernels in the procedure of Fig. 1 and repeating the procedure, we can execute the parallel addition.

Figure 8 shows a simulated result of addition; two addends (a) are converted into the data object in the data plane (b), where the bit data of 1 and 0 are expressed by pixels with 1 and 0 values, respectively. Background pixels have 0 value. Figure 8(c) is the overlaid attribute pattern in the attribute plane. The data and attribute planes are coded into a coded image in OAL [Fig 8(d)], correlated with the above operation kernels [Figs. 8(e)–(g)], and spatially sampled [Figs. 8(h)–(j)]. A NAND operation for the sampled images provides the result of Eq. (15) [Fig. 8(k)], which is used as the data plane at the next stage. Repeating this sequence we can execute addition. Figures 8(l)–(n) show the results of Eq. (15) at individual processing stages. After four iterations, the result of the addition is obtained as Fig. 8(o).

While the above processing is for a data pair, two matrices can be added with the same procedure. Figure 9 shows the result of matrix addition. Addends X and Y are matrices of 7-bit signed integers and coded into the data plane. The attribute plane is also depicted, which contains sixteen overlaid attribute patterns corresponding to the data patterns coded from X and Y. The results of Eq. (15) at each processing stage are shown with labels of iteration numbers. After seven iterations, the desired result is obtained. Subtraction can also be implemented using the same algorithm for addition with twos complement representation.

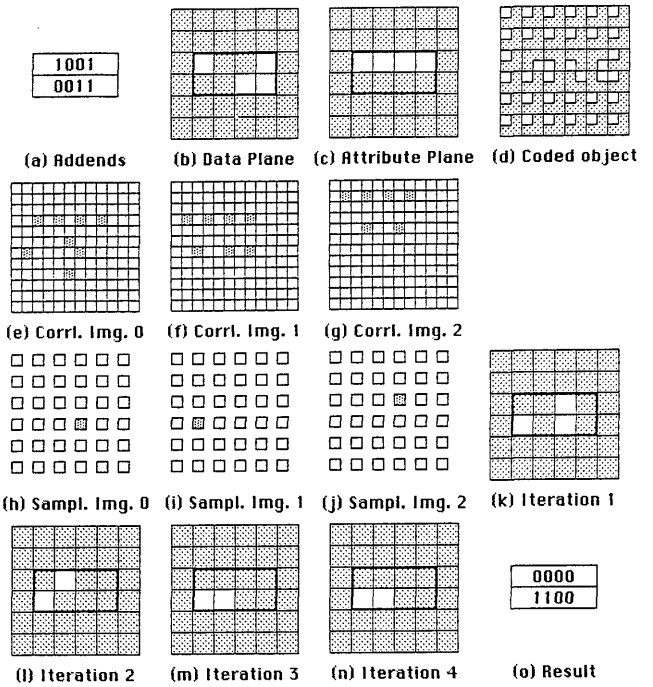


Fig. 8. Simulated result of addition; (a) two addends, (b) their data plane, (c) attribute plane, (d) coded image in OAL, (e)–(g) correlated and (h)–(j) sampled images for each product term operations in Eq. (15), (k) result of Eq. (15) at the first iteration, (l)–(n) results of Eq. (15) at every iteration after second, (o) final result of addition.

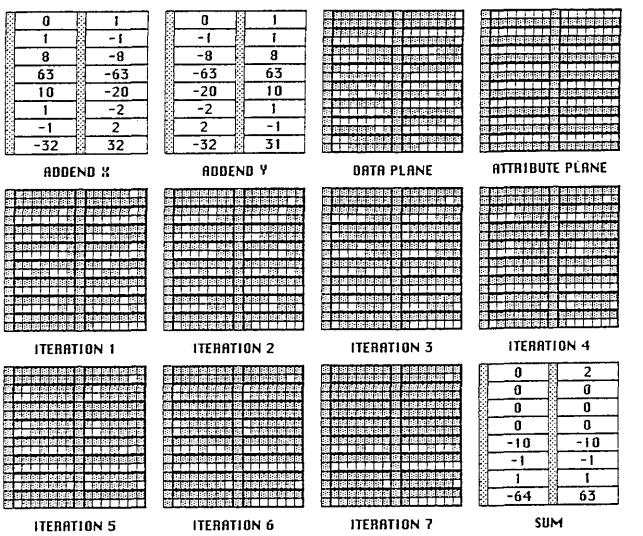


Fig. 9. Simulated result of matrix addition.

B. Multiplication

Multiplication requires a more complicated algorithm than addition. Here we explain the abstract of the algorithm using an example for 3-bit integers. Multiplication is composed of two fundamental operations: conditional duplication and addition. For 3-bit multiplication, $\mathbf{x}{:}x^2x^1x^0_{(2)} \times \mathbf{y}{:}y^2y^1y^0_{(2)}$, the following steps are used (Fig. 10):

(1) If $y^0 = 1$, duplicate $\mathbf{x}$ to the register Y located in the data plane.

(2) If $y^1 = 1$, duplicate $\mathbf{x}$ to the register X also located in the data plane with one pixel offset to the left direction.

(3) Add the data in the registers X and Y using the algorithm in Sec. IV.A.; then the result is obtained in the register Y.

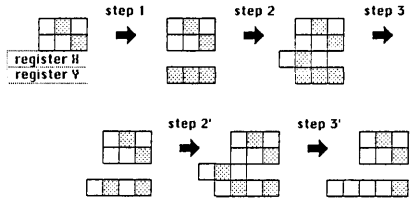


Fig. 10. Processing sequence of 3-bit multiplication.

(2′) If $y^2 = 1$, duplicate $\mathbf{x}$ to the register X with two pixels offset to the left direction.

(3′) Add the data in the registers X and Y; then the result is obtained in the register Y.

For multiplication of more-bits numbers, steps 2 and 3 are repeated with appropriate offset in duplicating $\mathbf{x}$.

The main problem in this processing is how to realize the conditional duplication. For this purpose we use attribute patterns as shown in Fig. 11. Then the following operation selectors are designed:

$$P_n(\mathbf{a},\mathbf{b}) = \begin{cases} \sum_{j=-2}^{2} a_{-2,j} b_{-2,j} & (n = 0), \\ \sum_{j=-2}^{2} a_{-1,j} b_{-1,j} & (n = 1,2), \end{cases} \tag{18}$$

where P_0, P_1, and P_2 are operation selectors used in steps 1, 2, and 2′, respectively. Equation (18) has information of the data patterns (e.g., $b_{-2,j}$ or $b_{-1,j}$), so that conditional operation can be executed.

The duplication with shifting is expressed as follows:

$$D_n(\mathbf{b}) = \begin{cases} b_{-3,0} & (n = 0), \\ b_{-2,n} & (n = 1,2). \end{cases} \tag{19}$$

where D_0, D_1, and D_2 are operations used in steps 1, 2, and 2′, respectively. Also the identity operation,

$$I(\mathbf{b}) = b_{0,0}, \tag{20}$$

is used to preserve the object patterns in the data plane.

Assuming that the pixels to which $\mathbf{x}$ is duplicated are always cleared (i.e., have 0 value), we can obtain the neighborhood operation to be executed as

$$C_n = P_n(\mathbf{a},\mathbf{b}) D_n(\mathbf{b}) + I(\mathbf{b}) = \begin{cases} b_{-3,0} \sum_{j=-2}^{2} a_{-2,j} b_{-2,j} + b_{0,0} & (n = 0), \\ b_{-2,n} \sum_{j=-2}^{2} a_{-1,j} b_{-1,j} + b_{0,0} & (n = 1,2). \end{cases} \tag{21}$$

Although Eq. (21) is sufficient for the processing, the inverse form enables us to reduce the number of processing stages.[7] Then, the sequence of

$$\bar{C}_n = \begin{cases} \bar{b}_{0,0}\bar{b}_{-3,0} + \bar{b}_{0,0} \prod_{-2}^{2} (\bar{a}_{-2,j} + \bar{b}_{-2,j}) & (n = 0), \\ \bar{b}_{0,0}\bar{b}_{-2,n} + \bar{b}_{0,0} \prod_{-2}^{2} (\bar{a}_{-1,j} + \bar{b}_{-1,j}) & (n = 1,2), \end{cases} \tag{22}$$

$$\bar{I}(\mathbf{b}) = \bar{b}_{0,0} \tag{23}$$

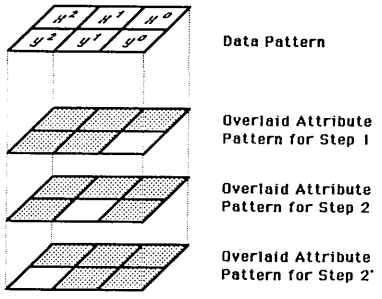


Fig. 11. Overlaid attribute patterns for 3-bit multiplication.

is used for practical processing. Operations in Eq. (22) are expressed by symbolic notation as follows:

$$\begin{bmatrix} .0 \\ .. \\ .. \\ .0 \end{bmatrix} + \begin{bmatrix} NN & NN & NN & NN & NN \\ .. & .. & .. & .. & .. \\ .. & .. & .0 & .. & .. \end{bmatrix} \quad (n = 0), \tag{24}$$

$$\begin{bmatrix} .. & .0 \\ .. & .. \\ .0 & .. \end{bmatrix} + \begin{bmatrix} NN & NN & NN & NN & NN \\ .. & .. & .0 & .. & .. \end{bmatrix} \quad (n = 1), \tag{25}$$

$$\begin{bmatrix} .. & .. & .0 \\ .. & .. & .. \\ .0 & .. & .. \end{bmatrix} + \begin{bmatrix} NN & NN & NN & NN & NN \\ .. & .. & .0 & .. & .. \end{bmatrix} \quad (n = 2). \tag{26}$$

Figure 12 shows a simulated result of multiplication for an array form of 4-bit unsigned integers. The figure depicts the attribute (left-hand side) and data (right-hand side) planes at individual processing stages. The result of a current stage is used as the data plane at the next stage. In this processing sequence, both operation kernels and attribute planes must be controlled by the program. In the twentieth stage, the desired result is obtained. Although this algorithm is not for matrix multiplication, the techniques used here are applicable in various other matrix operations.

V. Data-Driven Processing

In Sec. IV the system output is assumed to be fed back to the data plane as the input at the next processing stage. If the output is fed back to the attribute plane, a sort of data-driven processing[14] is possible. Namely, one result of processing determines the contents of the following processing, so that several processes can proceed without control from outside the system.

As a simple example of data-driven processing, we design an algorithm to calculate the absolute value of a signed integer, $\mathbf{x}$:$x^{N-1}x^{N-2}\ldots x^0_{(2)}$. The signed integer is assumed to be represented by twos complement. The algorithm is as follows:

(1) If the most significant bit of $\mathbf{x}$ (sign bit), x^{N-1}, is 0, set 1 to the pixels in the attribute plane corresponding to $\mathbf{x}$ in the data plane; or else set 0 to the pixels.

(2) If the pixel in the attribute plane is 1, inverse the value of the corresponding pixel in the data plane.

(3) If the pixel in the attribute plane is 1, set the pattern 00...01 side by side with $\mathbf{x}$.

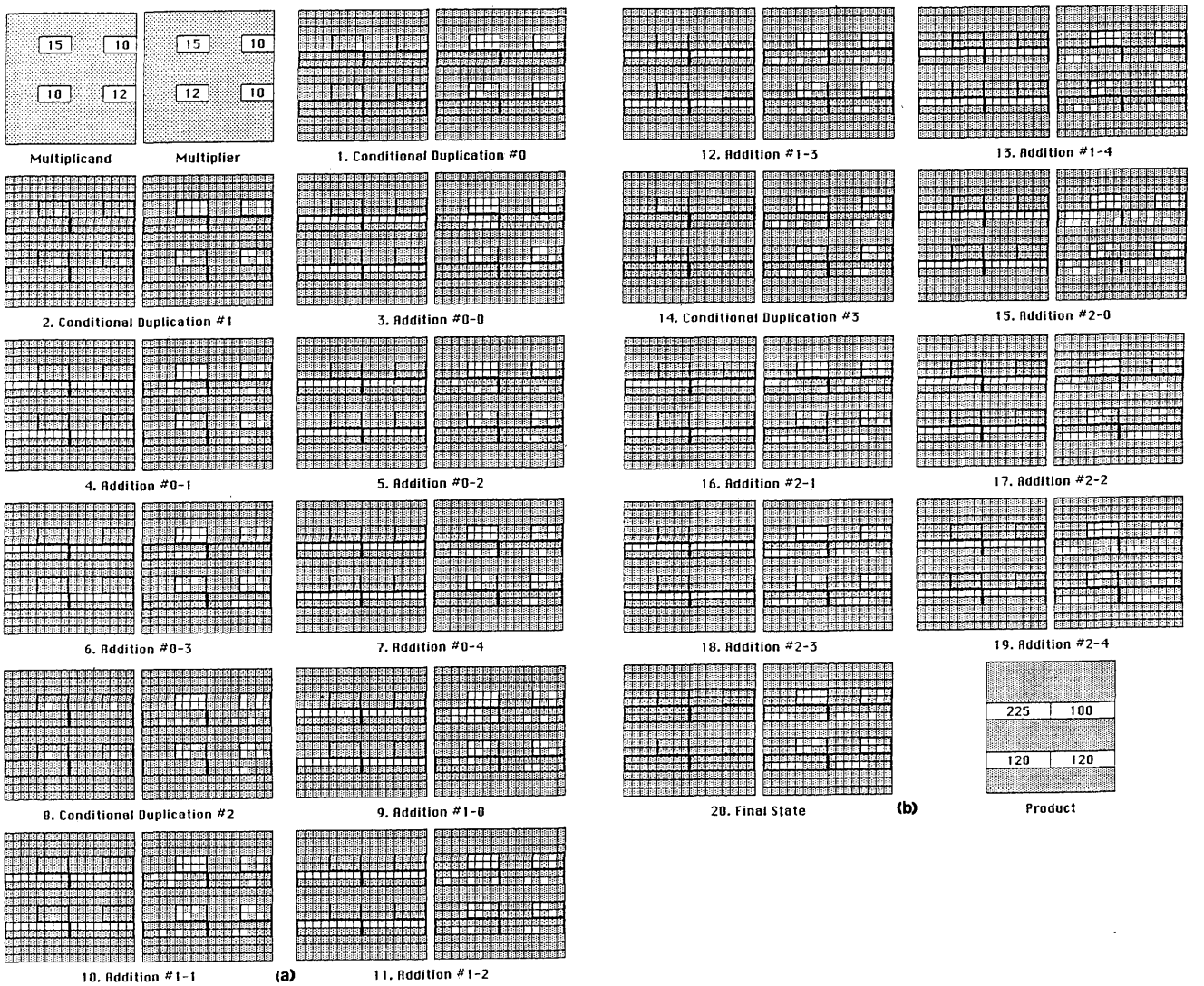


Fig. 12. Simulated result of multiplication for an array form of data. At each stage, attribute and data planes are shown at left- and right-hand sides, respectively.

(4) If the pixel in the attribute plane is 1, execute addition for **x** and the pattern 00...01. To obtain a uniform data pattern, the addition provides the sum and the carry upside down compared with the addition in Sec. IV.A.

The result of each stage is fed back as the data plane at steps other than step 1; at step 1 the result is fed back as the attribute plane.

A simulated result is shown in Fig. 13. Figure 13 depicts the attribute and data planes at individual stages with contents of the processing. In this processing both attribute and data planes do not have to be controlled from outside the system. The desired result can be obtained for matrix data.

Extending the technique of data-driven processing, we can effectively use the capacity of optical parallel processing. For this purpose an extended version of OPALS is considered. Figure 14 depicts a schematic diagram of the extended version of OPALS. This OPALS can execute two kinds of processing to provide data and attribute planes at a time, and feed them back as a data pair. The system is constructed by two optical array logic processors. Technically, a wavelength multiplexing technique[9] is used for the implementation.

VI. Discussion

OAL is regarded as an array of programmable logic arrays, so that it is needed to justify the computational capabilities of OAL in comparison with those of electronic implementation. Figure 15 shows a block diagram equivalent to the function of one processing unit in OAL and its optical implementation by a shadow-casting system. In OAL, pixel data in two input images, $a_{i,j}$, $a_{i+1,j}$, ..., $b_{i,j}$, $b_{i+1,j}$, ..., are the inputs, which are coded with the corresponding pixels, e.g., $a_{i,j}$ and $b_{i,j}$. The coded signals are selected according to the configuration signals, and the selected signals are logically produced. Finally, the product signals are logically summed, which will be the output of the processing unit. In this system, the configuration signals determine the contents of the processing.

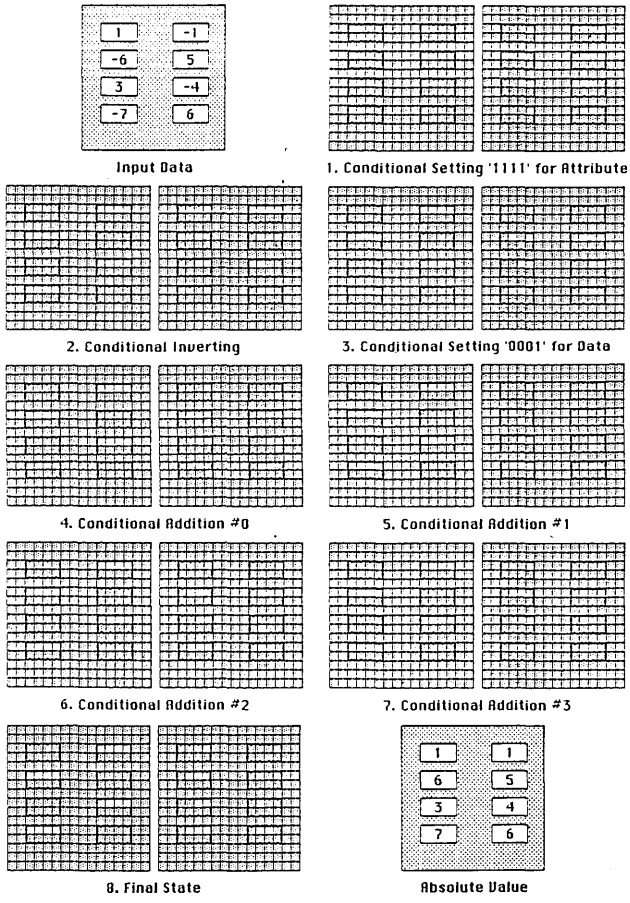


Fig. 13. Simulated result of absolute-value operation.

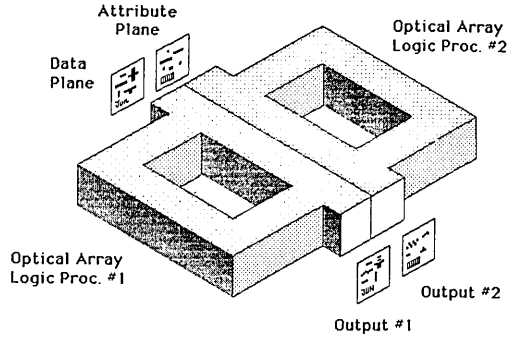


Fig. 14. Schematic diagram of the extended version of the OPALS for pattern logic.

The processing time for the operation in the processing unit is mainly determined by the time for coding, τ_{cd}, that for selecting, τ_{sl}, and that for logical sum, τ_{OR}. Note that the time for the logical product can be neglected because it is performed by overlapping of light signals, namely, when a dark signal is assigned to a logical one, the overlapped signal becomes a logical one only if all the signals to be overlapped are dark; this is nothing but the logical product. Thus, the processing time by OAL is estimated at $\tau_{cd} + \tau_{sl} + \tau_{OR}$.

The same functional unit can be implemented by electronics, which is known as a dynamically programmable logic array. In this case its processing time is estimated at $\tau_{cd} + \tau_{sl} + \tau_{AND} + \tau_{OR}$, where τ_{AND} is the time for the logical product. Therefore, if τ_{cd}, τ_{sl}, and τ_{OR} could be the same for optical and electronic implementation, OAL will be faster than the electronic by τ_{AND}.

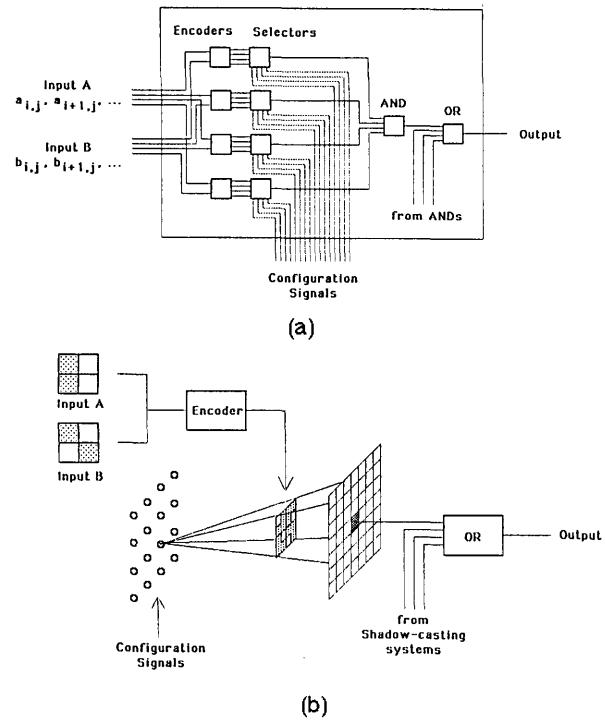


Fig. 15. Processing unit of OAL: (a) block diagram and (b) an optical implementation using a shadow-casting system.

Whereas OAL could have an advantage in the processing time, a more significant advantage is in data and configuration signal communication. As shown in Fig. 15(a), there are many signal lines in and around the processing unit. Figure 15 depicts only one unit, so that a huge number of signal lines are required to construct a parallel processing system with the units. In OAL these signal lines are realized by optical paths in free space, which are parallel and crosstalk-free lines. On the other hand, the electronic method is imposed on hard-wired communication, which causes a serious communication problem when many processing units are used and each processing unit has many inputs. Therefore, OAL has advantages in its computational capabilities especially for massively parallel processing.

In PTL a processing plane itself is regarded as a 2-D register which can be accessed randomly and in parallel. We estimate in Table I the required number of pixels and the iteration for three kinds of operation. The algorithm used in this paper is based on a ripple carry method and is not optimal. Other useful techniques, such as carry look-ahead addition, are also available for reducing the iteration number.[15]

For a hardware implementation of PTL, the required size of an operation kernel and the number of kernel units are important factors, because they deter-

Table I. Estimation of Required Numbers of Pixels and Iteration for Numerical Processing

Operation	Required Number of Pixels	Required Number of Iteration
Addition (n-bit Signed Integer)	2 (n + 1)	n
Multiplication (n-bit Unsigned Integer)	6 n	$n^2 + n - 1$
Absolute Value (n-bit Signed Integer)	2 n	n + 3

Table II. Required Size of Operation Kernel and Required Number of Kernel Units for Numerical Processing

Operation	Required Size of Operation Kernel	Required Number of Kernel Units
Addition (n-bit Signed Integer)	6 * 6	4
Multiplication (n-bit Unsigned Integer)	10 * 4n-2	2n
Absolute Value (n-bit Signed Integer)	6 * 6	4

mine the required specifications of the optical correlator for OAL. Thus we estimate their value for numerical processing (Table II). The result indicates that, for numerical processing, a large size of operation kernel is required, but the number of kernel units used is not so many. Namely, correlation in OAL is required to have large kernels but the number of elements in the kernel is small. Therefore, a specific correlator for OAL can be designed, which will be reported in a subsequent paper.[16]

The extended version of the OPALS can effectively execute PTL. In PTL information to be processed is expressed as a set of data and attribute patterns. Therefore, concurrent execution of OAL for both data and attribute patterns promises effective processing. We believe that the extended version of the OPALS is a basic architecture for PTL.

VII. Conclusions

We have proposed a new technique for space-variant processing with optical array logic and a new concept for parallel processing called pattern logic. The processing capability of pattern logic has been verified by several kinds of numerical data processing; a possibility of data-driven processing has also been presented. We have shown a system architecture suitable for pattern logic extended from the OPALS.

We are developing effective hardware to achieve pattern logic and constructing a systematic procedure for programming in pattern logic.

References

1. A. Huang, "Parallal Algorithms for Optical Digital Computers," in *Proceedings, Tenth International Optical Computing Conference* (MIT Press, Cambridge, 1983), pp. 13–17.
2. K.-H. Brenner, A. Huang, and N. Streibl, "Digital Optical Computing with Symbolic Substitution," Appl. Opt. **25,** 3054 (1986).
3. M. J. Murdocca, "Digital Optical Computing with One-Rule Cellular Automata," Appl. Opt. **26,** 682 (1987).
4. J. Tanida and Y. Ichioka, "Optical Logic Array Processor Using Shadowgrams," J. Opt. Soc. Am. **73,** 800 (1983).
5. J. Tanida and Y. Ichioka, "Optical-Logic-Array Processor Using Shadowgrams. II. Optical Parallel Digital Image Processing," J. Opt. Soc. Am. A **2,** 1237 (1985).
6. J. Tanida and Y. Ichioka, "Optical-Logic-Array Processor Using Shadowgrams. III. Parallel Neighborhood Operations and an Architecture of an Optical Digital-Computing System," J. Opt. Soc. Am. A **2,** 1245 (1985).
7. J. Tanida and Y. Ichioka, "Programming of Optical Array Logic. 1: Image Data Processing," Appl. Opt. **27,** 2926 (1988).
8. J. Tanida and Y. Ichioka, "OPALS: Optical Parallel Array Logic System," Appl. Opt. **25,** 1565 (1986).
9. J. Tanida and Y. Ichioka, "Optical Parallel Array Logic System. 2: A New System Architecture without Memory Elements," Appl. Opt. **25,** 3751 (1986).
10. J. Tanida and Y. Ichioka, "Modular Components for an Optical Array Logic System," Appl. Opt. **26,** 3954 (1987).
11. H. Fleisher and L. I. Maissel, "An Introduction to Array Logic," IBM J. Res. Develop. **19,** 98 (1975).
12. K. Preston, Jr., and M. J. B. Duff, *Modern Cellular Automata* (Plenum, New York, 1984).
13. T. J. Drabik and S. H. Lee, "Shift-Connected SIMD Array Architectures for Digital Optical Computing Systems, with Algorithms for Numerical Transforms and Partial Differential Equations," Appl. Opt. **25,** 4053 (1986).
14. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing* (McGraw-Hill, New York, 1985).
15. E. Swartzlander, "Digital Optical Arithmetic," Appl. Opt. **25,** 3021 (1986).
16. J. Tanida, J. Nakagawa, and Y. Ichioka, "Birefringent Encoding and Multichannel Reflective Correlator for Optical Array Logic," Appl. Opt. **27** (1988), submitted.