Gray-image processing using optical array logic

Sunao Kakizaki, Jun Tanida, and Yoshiki Ichioka

A method for digital image processing that uses optical array logic (OAL) is presented. Parallel thresholding and digital filtering are demonstrated. OAL is a promising computational paradigm for digital optical computing based on parallel neighborhood operations for two two-dimensional binary images. In the proposed method, virtual processing elements are assumed on an image plane and a gray pixel in an original gray image is stored in each processing element. Efficient gray-image processing can be achieved by data manipulation in the virtual processing elements and in the data communication among them by using OAL. Several simulation results are presented. Finally, hardware requirements for the developed algorithms and their capabilities are discussed.

I. Introduction

Many architectures for digital optical computers have recently been proposed.[1-11] A digital optical computer is expected to be a powerful computing system, making good use of both the flexibility of digital processing and the excellent advantages of optical parallel processing. We are attempting to find the optimal architecture for a digital optical computer. There are several factors hindering development of such a computing system, e.g., the difficulty of producing optical logic devices, the lack of appropriate materials, and problems in parallel programming. Problems concerned with programming are of special interest to us because the architectures of the optical digital computers have a close relationship to processing algorithms, and the two elements cannot be considered separately.

To solve the problems, we have studied optical array logic (OAL) as a computational paradigm for digital optical computing.[12,13] OAL can perform any parallel neighborhood operation for two binary images. In OAL, an identical operation is executed on all pixels in the input images in parallel. Thus OAL provides a single-instruction-stream multiple-data-flow type of parallel processing. In addition, if one input image of OAL is used as control signals specifying operations to be executed for the other input image, multiple single-instruction-stream multiple-data-flow (MSIMD) processing can be implemented.[13] Such processing in OAL can be described by a simple programming language.

These features of OAL are especially useful and attractive for digital image processing, in which large numbers of data must be processed efficiently. To demonstrate the advantages of OAL, we have presented algorithms for template matching,[14] edge detection,[12] and skeletonization.[15] However, these algorithms are designed for binary images, so those for gray images should be developed to extend the applications of image processing by OAL. Here we present a method of efficient gray-image processing with OAL.

A simple method of extracting the parallelism involved in image processing is to assign an individual gray pixel to a processing element and to process all data simultaneously. In the new method, we assume that the processing elements are set virtually upon an image plane of OAL and that programs for the desired processing are constructed in terms of operations for the individual processing element. Gray pixels in the original image are developed into spatial patterns and stored in the virtual processing elements upon an image. The resultant image is used for the inputs of OAL, with an additional image specifying the content of operations, pixel by pixel. With this technique, gray-image processing can be designed and executed effectively by OAL.

In this paper we present an effective method for gray-image processing in OAL. In Section II we explain OAL and a method for space-variant processing with OAL. In Section III we describe procedures for assigning gray images to virtual processing elements and the methods for performing gray-image processing. In Sections IV and V we present proce-

When this work was done, the authors were with the Department of Applied Physics, Osaka University, 2-1 Yamadaoka, Suita, Osaka 565, Japan. S. Kakizaki is now with the Central Research Laboratory, Hitachi, Ltd., 1-280, Higashi-koigakubo, Kokubunji-shi, Tokyo 185, Japan.

Received 2 July 1990.
0003-6935/92/081093-10$05.00/0.

dures for pixel-by-pixel operations and digital filtering with OAL. In Section VI we demonstrate successive operations for processing gray images by computer simulation. In Section VII, we discuss hardware requirements for the developed algorithms and their capabilities.

II. Optical Array Logic and Space-Variant Processing

OAL is a computational paradigm for digital optical computing based on parallel neighborhood operations for two two-dimensional binary images.[12,13] Figure 1 shows the processing procedure of OAL. OAL is implemented by four operations: coding, correlation, sampling, and inverted-OR operation. Two binary images to be processed are encoded with the coding rule shown in Fig. 1 and converted into a coded image. The coded image is separately correlated with different operation kernels. Individual correlated images are spatially sampled at 1-pixel intervals in the vertical and the horizontal directions. Parallel inverted-OR operation for all the sampled images provides the final result. The operations are specified by a set of operation kernels used in correlations. Thus a sequence of operation kernels is regarded as a program in OAL.

The processing procedure of OAL is simple and easy to implement by optical techniques such as optical shadow casting.[14] If the parallelism of optics is fully exploited, we can execute the operations in OAL in parallel. In this case an identical operation is carried out for all pixels in the images. Therefore single-instruction-stream multiple-data-flow processing can be performed in OAL. In other words, OAL executes space-invariant processing for two binary images.

Moreover, OAL can achieve space-variant processing with a specific programming technique.[13] The fundamental idea of the technique is that one of the inputs is used for data and the other is used for specifying operation imposed on the data. We call the former a data plane and the latter an attribute plane. A spatial pattern used for specification is called an attribute pattern.

Using the programming technique, we process all pixels with the same attribute pattern by an identical operation. Thus MSIMD processing can be achieved by OAL. This technique is not suitable for multi-instruction-stream multiple-data-flow processing because of processing inefficiency. Fortunately, MSIMD processing is sufficient for numerical operations required in gray-image processing. In this case, a large number of grouped data are processed effectively by an identical operation.[13]

III. Gray-Image Processing with Optical Array Logic

To execute gray-image processing efficiently with OAL, we assume virtual processing elements in an image plane, and individual pixels in an original gray image are processed in the processing element. That is, gray pixels in gray image are developed into spatial patterns with multiple bits and stored in the virtual processing elements on an input image for OAL. Figure 2 shows the assignment of gray-image Ga, consisting of 4-bit gray pixels, onto virtual processing elements constructed in an image plane. In this scheme, a gray level of a pixel in the original gray image is represented by binary pixels (a_{ij}^4, a_{ij}^3, a_{ij}^2, a_{ij}^1, a_{ij}^0) arranged in rows. Five pixels are prepared for 4-bit data to take care of overflow. This group of pixels is called a register and is used like a register in an electronic computer.

If we prepare several registers, multiple gray images can be stored in the same image plane. Since shift operations can easily be executed in OAL, the use of multiple registers makes image processing effective. A set of registers for pixels at the same position is called a register block. With this arrangement, most of the operations required in gray-image processing can be executed efficiently. In this case, the register block is regarded as a virtual processing element in which a gray pixel in an original gray image is processed.

Figure 3 shows an example of two input images of OAL. Nine register blocks are set in input B, or the data plane. Row-coded numbers are stored in registers A and B of individual register blocks. To identify individual register blocks, we set supplementary pat-

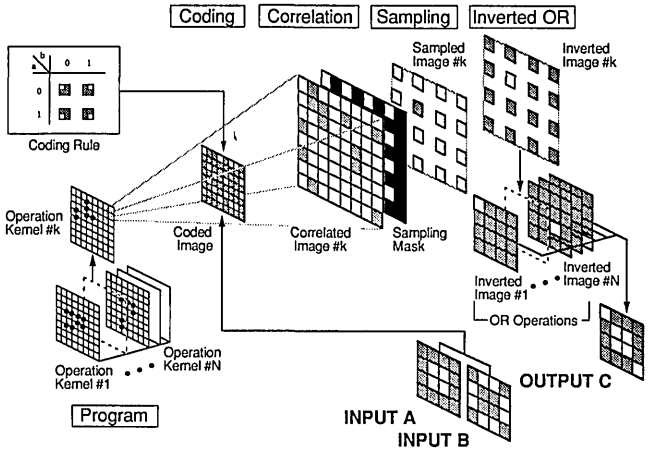

Fig. 1. Processing of optical array logic.

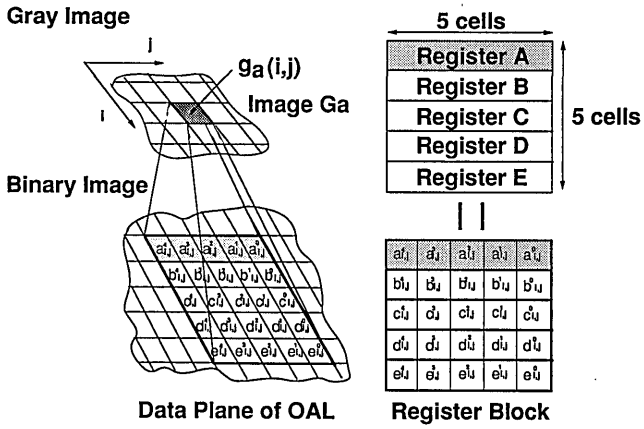

Fig. 2. Assignment of gray-image data for virtual elements constructed in an image plane of OAL.

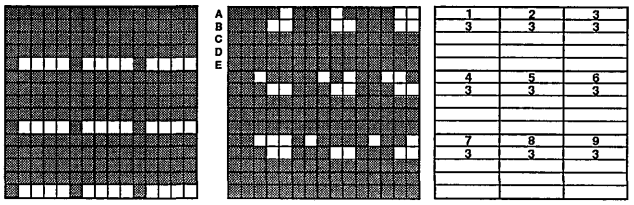

Fig. 3. Two input images of OAL in gray-image processing.

terns in input A, or the attribute plane. An identical operation is applied to the data in the neighborhood that have the same attribute patterns.

Using these two input images, we can perform gray-image processing efficiently with OAL. The assumed processing flow is depicted in Fig. 4. Input B, which stores the data, and input A, which specifies operations for the data, are processed by an OAL processor, and output C is produced. The output image is used as input B in the successive processing stage. For efficient processing, several kinds of attribute plane are used, according to the processing sequence.

We consider two classes of operation in gray-image processing. The first class is a pixel-by-pixel operation such as thresholding. This operation is achieved by data manipulation among the registers in one register block. The second class is a neighborhood operation such as digital filtering, in which a pixel's value is calculated relative to those of its neighboring pixels. In this operation, pixel-by-pixel operation and data communication among the neighboring register blocks are required. Using these techniques, we can execute various kinds of gray-image processing.

IV. Pixel-by-Pixel Operations with Optical Array Logic

A. Addition between Two Registers

As we mentioned in Section II, operations in OAL are specified by a set of operation kernels used in correlation. Thus a sequence of operation kernels can be used as a program of OAL. However, for convenience of programming we use a kernel expression[12,13] instead of a sequence of operation kernels. In the kernel expression, an operation kernel is expressed in matrix-like notation.

As an example of a program in OAL, we consider parallel addition for pairs of row-coded numbers in a binary image. The input images are assumed to have the same arrangement as in Fig. 3. Addition between registers A and B is designed with the space-variant technique described in Section II. In this algorithm, sum-and-carry operations are executed iteratively, and their results are sent back to registers A and B until no carry signal arises. These operations are written by the following logical operations:

$$a^n_{i,j} \leftarrow a^n_{i,j}\bar{b}^n_{i,j} + \bar{a}^n_{i,j}b^n_{i,j} \tag{1}$$

$$b^{n+1}_{i,j} \leftarrow a^n_{i,j}b^n_{i,j} \tag{2}$$

where $a^n_{i,j}$ and $b^n_{i,j}$ denote nth-bit signals in registers A and B, respectively, the subscript is the identifier of the register block, and + and $\bar{}$ mean OR and NOT operators, respectively.

Considering the arrangement of the individual registers shown in Fig. 3, the operations in expressions (1) and (2) are designed with a kernel expression as follows:

$$\begin{bmatrix} \underline{01} \\ 00 \\ 0. \\ 0. \\ 1. \end{bmatrix} + \begin{bmatrix} \underline{00} \\ 01 \\ 0. \\ 0. \\ 1. \end{bmatrix} + \begin{bmatrix} .. & 01 \\ \underline{..} & 01 \\ .. & 0. \\ .. & 0. \\ .. & 1. \end{bmatrix}, \tag{3}$$

where one set of brackets corresponds to one operation kernel, an underscore indicates the origin of the neighborhood area, and a + denotes an OR operation for the results obtained by the attached operation kernels. Each symbol inside brackets indicates an operation for the pixels at the same position as shown in Table I.

As an advantage of using register blocks, operations of addition can easily be specified by the combination of attribute patterns and operations. For example, if the attribute patterns shown in Fig. 5(a) and the operation described in Eq. (4) below are used, addition between registers C and D as well as between registers A and B can be achieved simultaneously:

$$\begin{bmatrix} \underline{01} \\ 10 \end{bmatrix} + \begin{bmatrix} \underline{00} \\ 11 \end{bmatrix} + \begin{bmatrix} .. & 01 \\ \underline{..} & 11 \end{bmatrix}. \tag{4}$$

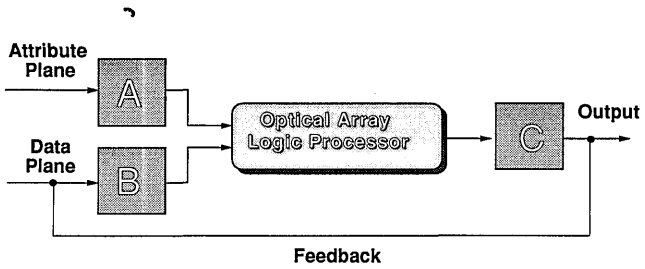

Fig. 4. Data flow in OAL. The algorithms presented in this paper are to be processed according to this data flow.

B. Thresholding

Here we consider thresholding as an example of pixel-by-pixel processing. Thresholding is an operation converting the value of a gray pixel into either 0 or −1, according to the relation between pixel and threshold values.

Let us assume that pixel and threshold values are set to registers A and B, respectively, in a register block. Each number has 4-bit length, represented by 2's complement. The algorithm for thresholding consists of two steps, i.e., subtraction of the number in

Table I. Kernel Units Corresponding to a Two-Variable Binary Logic Function[a]

Kernel Unit	Function	Symbol	Kernel Unit	Function	Symbol
	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		ab	11
	$\bar{a}\,\bar{b}$	00		0	DD

[a]Function symbols used for symbolic notation of OAL are also tabulated.

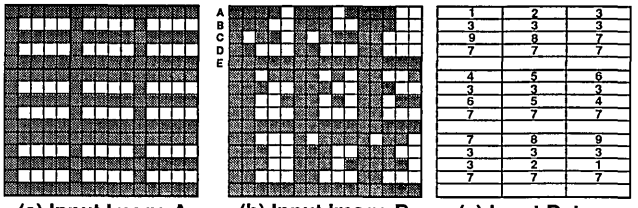

Fig. 5. Two input images of OAL used in addition between registers C and D and between registers A and B.

registers B from that in register A and testing of the status of the most significant bit, or sign bit. The subtraction is achieved as follows:

(1) Preserving a pixel value stored in register A, inverting a threshold value in register B, and assigning value 1 to register C as a forced carry.

(2) Preserving a pixel value stored in register A and executing addition between registers B and C to obtain 2's complement of the threshold. The result is stored in register B.

(3) Executing addition for registers A and B. This result is equal to the subtraction from the pixel value of the threshold value.

The operation kernels for the steps (1), (2), and (3), respectively, are

$$\begin{bmatrix}\underline{01}\\0.\\0.\\0.\\1.\end{bmatrix}+\begin{bmatrix}0.\\\underline{00}\\0.\\0.\\1.\end{bmatrix}+\begin{bmatrix}..&..&..&..\\..&..&..&..\\..&..&..&\underline{..}\\..&..&..&..\\1.&1.&1.&1.\end{bmatrix}, \tag{5}$$

$$\begin{bmatrix}\underline{01}\\0.\\0.\\0.\\1.\end{bmatrix}+\begin{bmatrix}0.\\\underline{01}\\00\\0.\\1.\end{bmatrix}+\begin{bmatrix}0.\\\underline{00}\\01\\0.\\1.\end{bmatrix}+\begin{bmatrix}..&0.\\..&01\\\underline{..}&01\\..&0.\\..&1.\end{bmatrix}, \tag{6}$$

$$\begin{bmatrix}\underline{01}\\00\\0.\\0.\\1.\end{bmatrix}+\begin{bmatrix}\underline{00}\\01\\0.\\0.\\1.\end{bmatrix}+\begin{bmatrix}..&01\\\underline{..}&01\\..&0.\\..&0.\\..&1.\end{bmatrix}. \tag{7}$$

These kernel expressions can easily be converted into corresponding operation kernels by referring to Table I. Using the obtained operation kernels in the procedure of Fig. 1, we can execute thresholding optically in parallel. The operations in expressions (6) and (7) must be repeated five times to complete addition for 4-bit numbers.

Figure 6 shows a simulation result of thresholding. Images 1A and 1B are the attribute and the data planes, respectively; five registers are prepared in a register block on image 1B, and the decimal expressions of the data stored in individual registers are indicated in image 1B′. The other images in Fig. 6 are arranged in the same manner as in the first row. Images 2B, 3B, and 4B are the results of steps (1), (2), and (3), respectively. After step (3), by testing the status of the most significant bit of register A we can obtain the result of thresholding as shown in image 5B′.

V. Digital Filtering with Optical Array Logic

A. Data Transmission among Register Blocks

As an example of neighborhood operation, digital filtering for image processing is considered. Digital filtering is categorized into two classes, recursive filtering and nonrecursive filtering. Nonrecursive filtering is useful and sufficient for many applications in digital image processing. Thus in this paper we consider only nonrecursive linear digital filtering.

To implement nonrecursive linear digital filtering by OAL, we must add data transmission to the sequence of pixel-by-pixel operation. That is, row-coded numbers in neighboring register blocks are transmitted to the register block at the origin of the

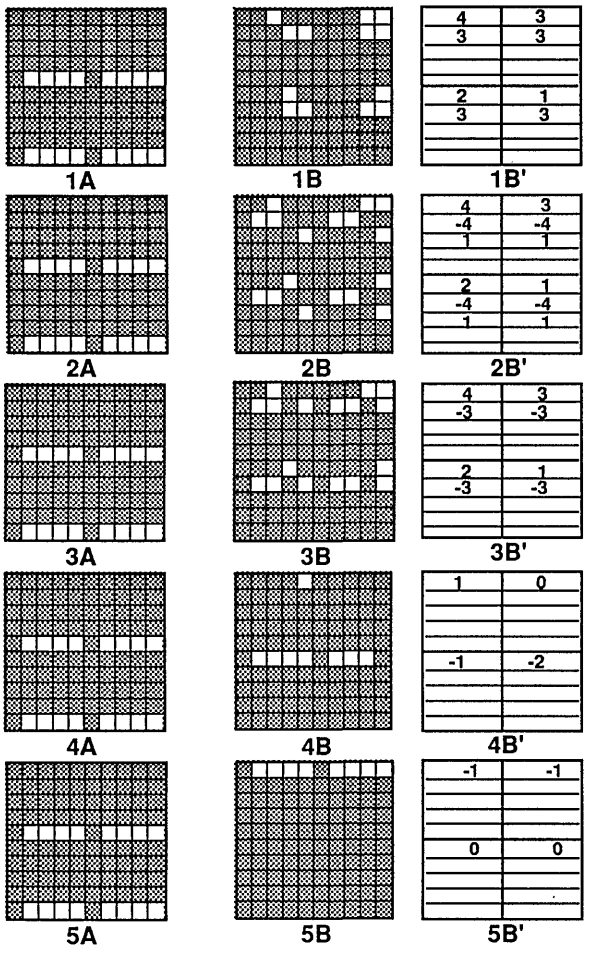

Fig. 6. Simulation result of thresholding: 1A–5A, attribute planes; 1B–5B, data planes; 1B′–4B′, decimal expressions of stored data in data planes.

neighborhood area, multiplied by a kernel of the filter, and accumulated.

Data transmission among the register blocks is achieved by shift operations in OAL. For ease of description of data transmission among the registers, we use identifiers, such as $A_{1,1}$ amd $A_{1,0}$, in considering a specific register block and the relative positions of the data. Figure 7 shows the identifiers of registers around one register block, where A–E indicate individual registers and the subscripts refers to the local coordinates of the register blocks in the neighborhood area. Note that a set of operations for one resister block can describe whole processing for a given image because of the single-instruction-stream multiple-data-flow characteristics of OAL. In addition, we sometimes use the identifier without any subscript, which indicates a composite set of a specific register in all register blocks.

Let us consider data transmission from register $A_{0,-1}$ to register $B_{0,0}$ and that from $A_{0,1}$ to $C_{0,0}$. Assuming that the number of registers in a register block is five and that the attribute plane is as shown in Fig. 8(a), we can achieve data transmission by performing

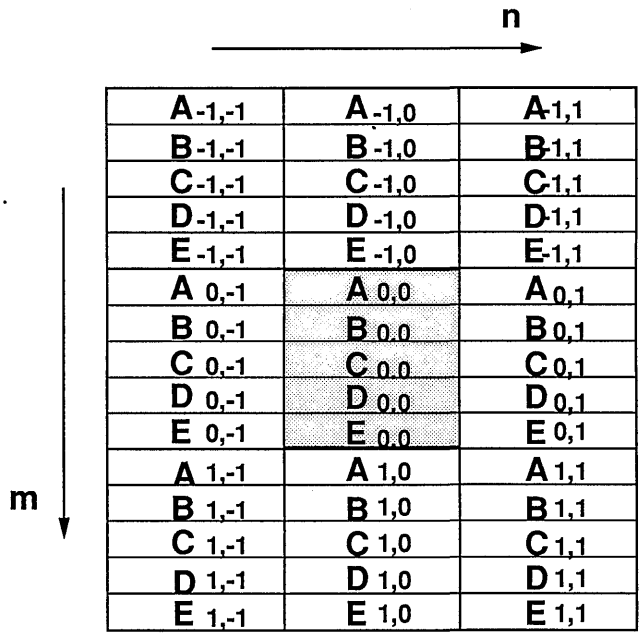

Fig. 7. Identifiers of registers around one resister block.

the following operation:

$$\begin{bmatrix} .1 & \cdots & 0. \\ \cdots & \cdots & \underline{0.} \\ \cdots & \cdots & 0. \\ \cdots & \cdots & 0. \\ \cdots & \cdots & 1. \end{bmatrix} + \begin{bmatrix} 0. & \cdots & .1 \\ 0. & \cdots & \cdots \\ \underline{0.} & \cdots & \cdots \\ 0. & \cdots & \cdots \\ 1. & \cdots & \cdots \end{bmatrix}, \quad (8)$$

Figure 8 shows a simulation result of data transmission. Combining this technique with the calculation method described in Subsection IV.A, we can carry out any kind of nonrecursive linear digital filtering.

B. Filtering with Robert's Gradient Operator

As an example of digital filtering, we consider Robert's gradient operator.[16] Robert's gradient operator is a linear filter for detecting the difference between a pixel and its adjacent diagonal pixel values. The filtering operation is performed by convolution of an input image with the digital filter, as shown in Fig. 9.

Let us assume that a value of a gray pixel is set in register A of each register block and represented by 2's complement. Digital filtering with Robert's gradi-

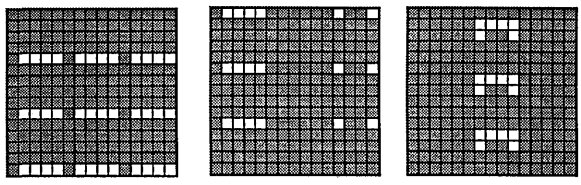

Fig. 8. Simulation result of data transmission: (a) attribute plane, (b) data plane, (c) the resultant image.

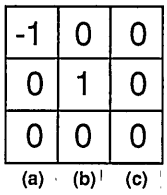

Fig. 9. Kernels of the digital filter: Robert's gradient operator.

ent operator is achieved by the following three steps:

(1) Preserving the value of a gray pixel in register $A_{0,0}$, negating all bits in register $A_{-1,-1}$, transmitting the result to register $B_{0,0}$, and setting 1 to register $C_{0,0}$ as a forced carry.

(2) Preserving the data in register $A_{0,0}$, executing addition between registers $B_{0,0}$ and $C_{0,0}$ to obtain 2's complement of the value of the gray pixel in register $A_{-1,-1}$, and setting the result at register $B_{0,0}$.

(3) Executing addition between registers $A_{0,0}$ and $B_{0,0}$. As a result, subtraction between the adjacent diagonal pixels is performed.

The operation kernels for steps (1), (2), and (3) are designated, respectively by

$$\begin{bmatrix} \underline{01} \\ 0. \\ 0. \\ 0. \\ 1. \end{bmatrix} + \begin{bmatrix} .0 & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & .. \\ .. & .. & .. & .. & .. & 0. \\ .. & .. & .. & .. & .. & \underline{0.} \\ .. & .. & .. & .. & .. & 0. \\ .. & .. & .. & .. & .. & 0. \\ .. & .. & .. & .. & .. & 1. \end{bmatrix} + \begin{bmatrix} .. & .. & .. & .. \\ .. & .. & .. & .. \\ .. & .. & .. & \doteq \\ .. & .. & .. & .. \\ 1. & 1. & 1. & 1. \end{bmatrix}, \quad (9)$$

$$\begin{bmatrix} \underline{01} \\ 0. \\ 0. \\ 0. \\ 1. \end{bmatrix} + \begin{bmatrix} 0. \\ \underline{01} \\ 00 \\ 0. \\ 1. \end{bmatrix} + \begin{bmatrix} 0. \\ \underline{00} \\ 01 \\ 0. \\ 1. \end{bmatrix} + \begin{bmatrix} .. & 0. \\ .. & 01 \\ \doteq & 01 \\ .. & 0. \\ .. & 1. \end{bmatrix}, \quad (10)$$

$$\begin{bmatrix} \underline{01} \\ 00 \\ 0. \\ 0. \\ 1. \end{bmatrix} + \begin{bmatrix} \underline{00} \\ 01 \\ 0. \\ 0. \\ 1. \end{bmatrix} + \begin{bmatrix} .. & 01 \\ \doteq & 01 \\ .. & 0. \\ .. & 0. \\ .. & 1. \end{bmatrix}. \quad (11)$$

Figure 10 shows a simulation result of filtering with Robert's gradient operator. Images 1A and 1B are the attribute and the data planes, respectively. Image 1B′ shows the values of the gray pixels to be processed, which are stored in register A. Image 2B shows the result of step (1) above, which is used for step (2) with image 2A. Image 2B′ indicates the decimal expression of the data stored in the individual registers of image 2B. Images 3B and 4B show the results of steps (2) and (3). The expected results are obtained in image 4B′.

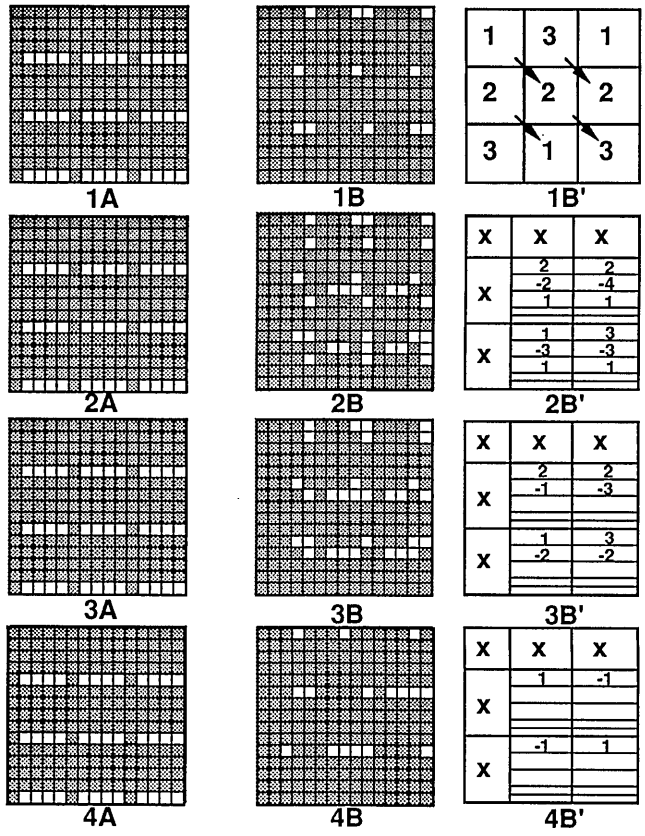

Fig. 10. Simulation result of digital filtering with Robert's gradient operator: 1A–4A, attribute planes; 1B–4B, data planes; 1B′–4B′, decimal expressions of stored data in the corresponding data planes.

Note that the processing explained in this section is independent of the number of register blocks. OAL executes an identical operation for all pixels in a given image in parallel. Therefore, if a large number of register blocks are set in the input image, great processing capabilities can be obtained.

VI. Successive Execution of Gray-Image Processing with Optical Array Logic

We have described basic ideas for gray-image processing in OAL. Here we show a series of gray-image

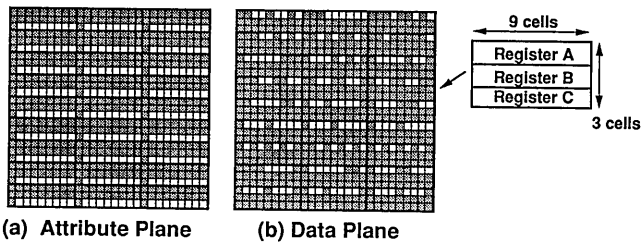

Fig. 11. Arrangement of attribute plane and data plane in the experiment.

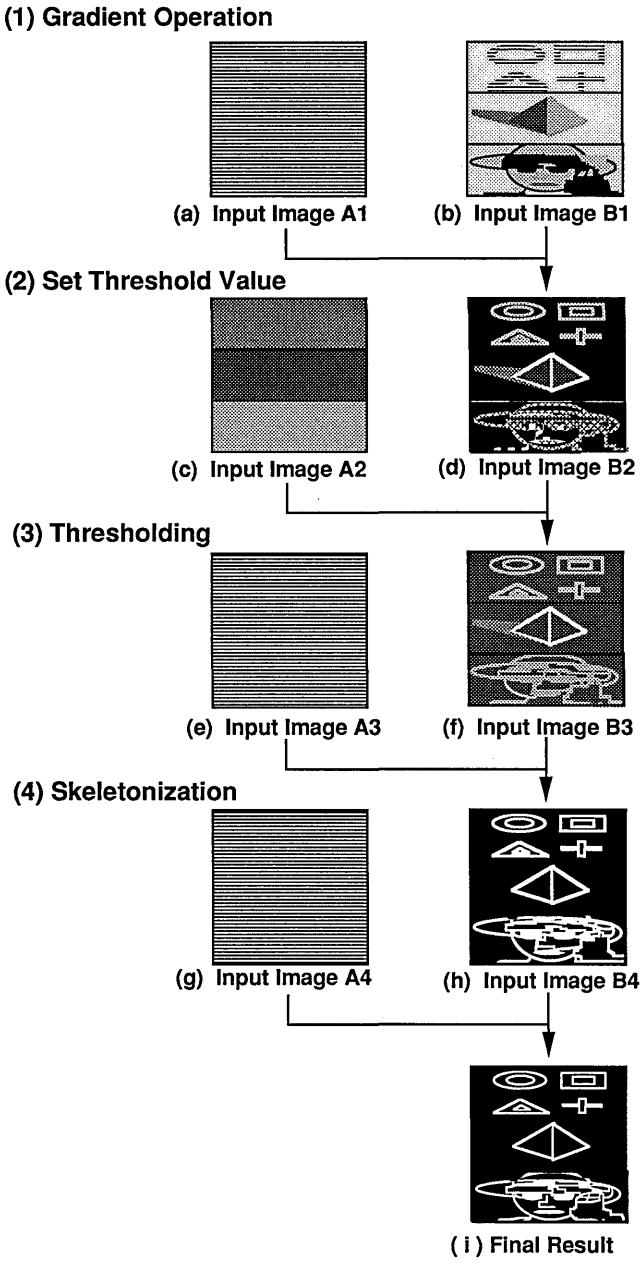

Fig. 12. Data flow of successive steps in gray-image processing: (a)–(i); the input and output data in four stages, i.e., gradient operation, setting threshold value, thresholding, and skeletonization.

processing steps to clarify the flexibility of our scheme. The image processing to be demonstrated is extraction of edge pixels with 1-pixel width from input gray images. The processing is achieved by a sequence of edge detection by Robert's gradient operator, binarization by thresholding, and skeletonization.

Figure 11 shows the arrangement of the attribute and the data planes for the processing. In this arrangement, three registers, A, B, and C, composed of 9 pixels to treat an 8-bit number, are set in a register block. Although Fig. 11 shows only 9 × 3 register blocks, 192 × 64 register blocks are prepared to process three images consisting of 64 × 64 pixels with

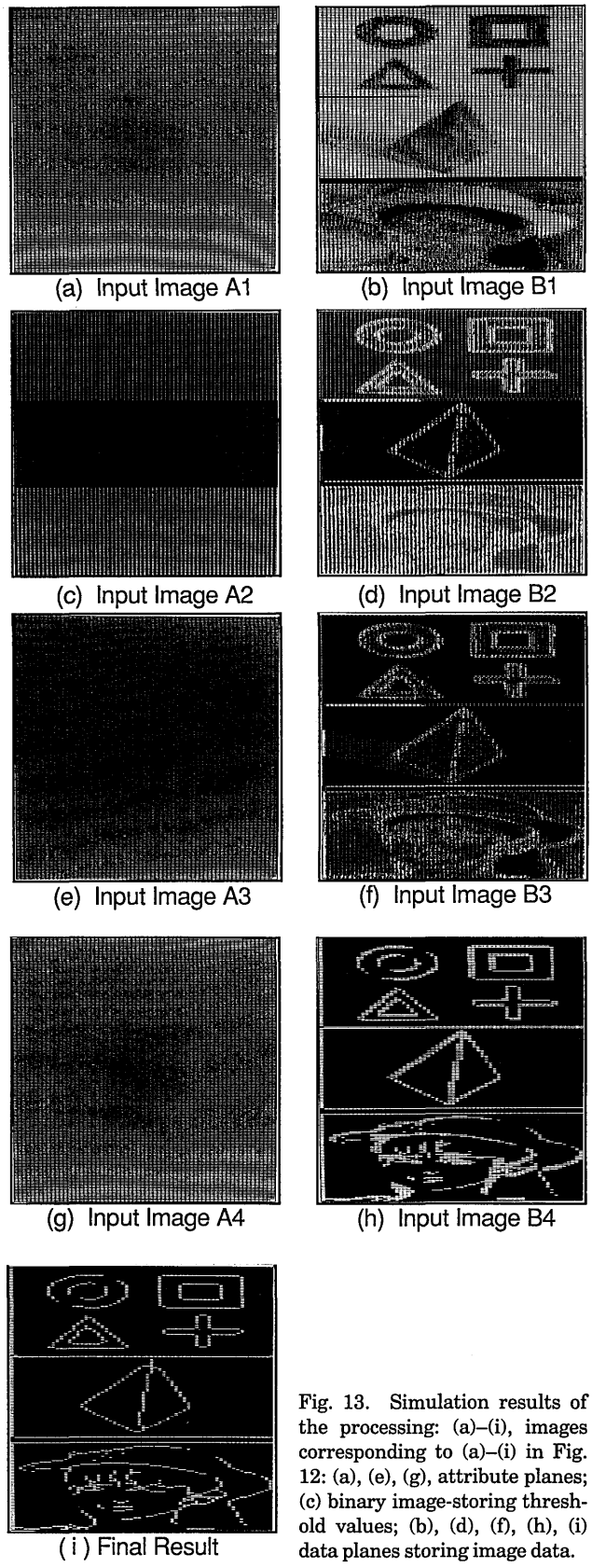

Fig. 13. Simulation results of the processing: (a)–(i), images corresponding to (a)–(i) in Fig. 12: (a), (e), (g), attribute planes; (c) binary image-storing threshold values; (b), (d), (f), (h), (i) data planes storing image data.

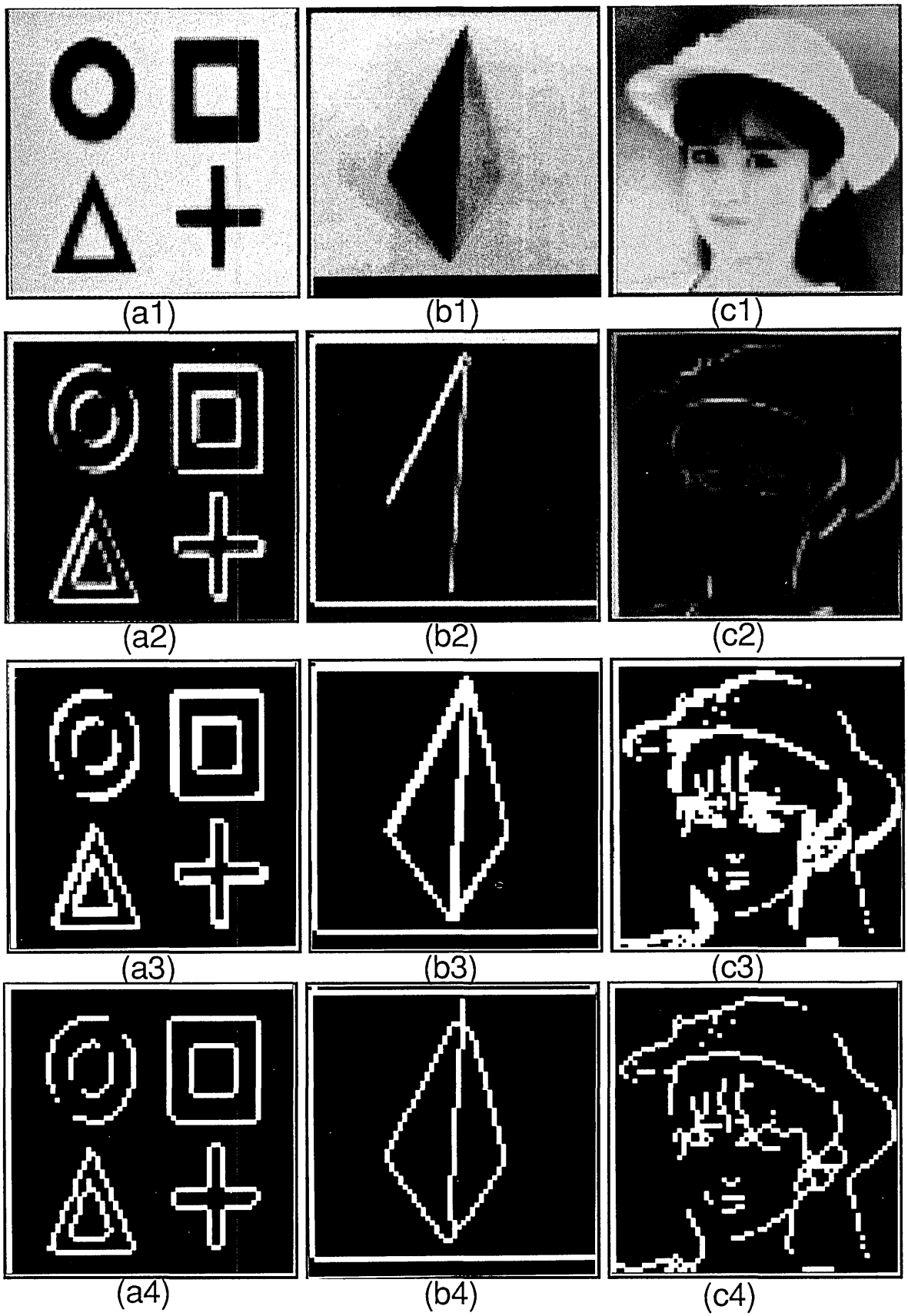

Fig. 14. Gray images converted from binary images (b), (d), (h), and (i) in Fig. 13.

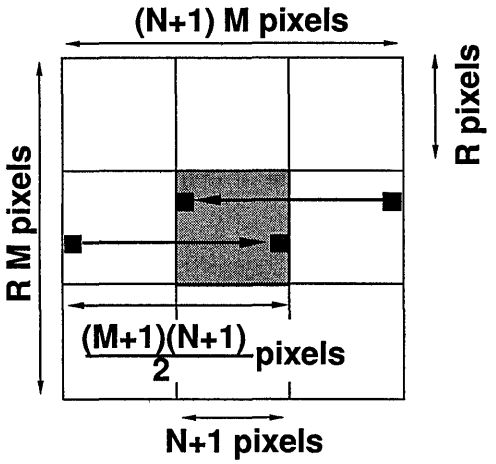

Fig. 15. Required cells of data shift in digital filtering: N, number of bits; R, number of the register; M, size of the neighborhood area.

8-bit data. Therefore 576 × 576 pixels are used in total.

Figure 12 is a processing flow chart. The processing consists of four steps: gradient operation, setting threshold value, thresholding, and skeletonization. Input B, or the data plane, has the image data and is processed in successive steps. On the other hand, input A, or the attribute plane, is changed to modify the operations at the individual steps. Note that three kinds of data are set in input B and processed simultaneously. The capability of such parallel processing for different objects is a notable advantage provided by OAL.

Figure 13 shows the result of computer simulation performed on a SUN3 workstation. Individual images show the contents of the input and the output data of the OAL processor. The letters under the images correspond to those in Fig. 12. Figure 14 depicts three kinds of gray image stored in input images of OAL. They are converted from images (b), (d), (h), and (i) of Fig. 13. As the figure shows, the desired results are obtained.

VII. Discussion

A. Hardware Requirements

As was shown previously,[14,17] OAL can be implemented with several types of optical system. The developed programs can easily be executed with those systems. However, two requirements for the optical system must be satisfied for implementation.

As the first requirement, the optical system must handle large sizes of images. When a gray image consists of $P \times P$ pixels, $(N + 1)P \times RP$ pixels are required for conversion of a gray image into a binary image, where N is the number of bits in the data representation and R is the number of registers in one register block. For example, if $P = 256$, $N = 8$, and $R = 9$, the binary image must have 2304×2304 pixels. Usually P is a large number, so we must handle quite a large number of pixels in an optical system. In a practical system, to overcome the limitation of hardware, input images may be partitioned into smaller sections and processed. In this case a virtual storage mechanism to support the hardware is required in the system.

As the second requirement, there must be a correlation of the input data with large sizes of kernels in the optical system. Consider digital filtering with a filter composed of $M \times M$ pixels. In this case, as Fig. 15 shows, at most $(M + 1)(N + 1)/2$ pixels of the data shift are required in the horizontal direction, and $(M + 1)R/2$ pixels in the vertical direction. Consequently the necessary size of an operation kernel is $2[(M + 1)(N + 1)1] \times 2[(M + 1)R - 1]$. If $M = 3$, $N = 8$, and $R = 9$, we need an operation kernel consisting of 70×70 points.

B. Computational Capabilities of the Developed Programs

The execution time of a program in OAL is affected primarily by hardware implementation, which we consider here in detail. Let the processing time for coding, correlation with sampling, and inverted OR be t_e, t_c, and t_d, respectively, and let the required numbers of times for these operations be n_e, n_c, and n_d, respectively. The processing time in OAL can be estimated to be $t_e n_e + t_c n_c + t_d n_d$ for the sequential execution of correlation with a single correlator and $n_e(t_e + t_c + t_d)$ for parallel execution with multiple correlators. Although t_e, t_c, and t_d depend heavily on hardware performance, n_e, n_c, and n_d are good measures for evaluation of processing efficiency of developed programs.

In Table II the calculated values of n_e, n_c, and n_d for programs presented in this paper are tabulated. N is

Table II. Processing Capabilities of the Developed Programs[a]

Title	n_e	n_c	n_d
Addition: N bits	$N + 1$	$3(N + 1)$	$3(N + 1)$
Subtraction: N bits (complement value)	$2N + 3$	$7N + 10$	$7N + 10$
Multiplication: N bits	$N^2 + 2N - 2$	$3N^2 + N - 2$	$3N^2 + N - 2$
Thresholding: N bits (complement value)	$2N + 6$	$8N + 12$	$8N + 12$
Robert's gradient operator N bits (complement value)	$2N + 3$	$7N + 10$	$7N + 10$
Skeletonization (erasing only the border of objects)	8	16	8

[a] N is the bit number of the pixel datum; n_e, n_c, and n_d are the numbers required for coding, correlation and sampling, and inverted-OR operation, respectively, for N-bit lengths of data.

the bit number of a gray pixel. As Table II shows, these values are independent of the size of a gray image or of the number of pixels. This means that processing time is invariant even if the number of data is increased. Therefore the developed algorithm can extract the parallel nature of OAL successfully.

On the other hand, the processing time depends on the bit number, N. Since the ripple carry algorithm is used for addition, the sum-and-carry operation must be repeated $N + 1$ times. For this problem, fruitful results in digital processing such as modified signed digit number representation[18] and the carry–look-ahead algorithm[19] can be used. Applying these algorithms to our scheme, we can achieve addition more effectively.

References

1. A. A. Sawchuk and T. C. Strand, "Digital optical computing," Proc. IEEE **72,** 758–779 (1984).
2. A. Huang, "Architectural considerations involved in the design of an optical digital computer," Proc. IEEE **72,** 780–786 (1984).
3. J. Tanida and Y. Ichioka, "OPALS: optical array logic system," Appl. Opt. **25,** 1565–1570 (1986).
4. K.-H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," Appl. Opt. **25,** 3054–3060 (1986).
5. K.-H. Brenner and G. Stuke, "Architectures for digital optical image processing using morphological filters," in *Optical Computing '88,* P. Chavel, J. W. Goodman, and G. Roblin, eds., Proc. Soc. Photo-Opt. Instrum. Eng. **963,** 657–662 (1988).
6. A. Huang, "Computational origami—the folding of circuits and systems," in *Optical Computing,* Vol. 6 of OSA 1991 Technical Digest Series (Optical Society of America, Washington, D.C., 1989), pp. 132–135.
7. M. J. Murdocca, A. Huang, J. Jahns, and N. Streibl, "Optical design of programmable logic arrays" Appl. Opt. **27,** 1651–1660 (1988).
8. K. S. Huaung, A. A. Sawchuk, B. K. Jenkins, P. Chavel, J. M. Wang, and I. Glaser, "Implementation of a prototype digital optical cellular image processor (DOCIP)," in *Optical Computing '88,* P. Chavel, J. W. Goodman, and G. Roblin, eds., Proc. Soc. Photo-Opt. Instrum. Eng. **963,** 687–695 (1988).
9. F. Kiamilev, Sadik C. Esener, R. Paturi, Y. Fainman, P. Mercier, C. C. Guest, and S. H. Lee, "Programmable optoelectronic multiprocessors and their comparison with symbolic substitution for digital optical computing," Opt. Eng. **28,** 396–408 (1989).
10. G. Stucke, "Parallel architecture for a digital optical computer," Appl. Opt. **28,** 363–370 (1989).
11. V. P. Heuring, H. F. Jordan, and J. P. Pratt, "A bit serial architecture for optical computing," Tech. Rep. 88-01a (Optoelectronic Computing System Center, University of Colorado, Boulder, Colo., 1988).
12. J. Tanida and Y. Ichioka, "Programming of optical array logic. 1: Image data processing," Appl. Opt. **27,** 2926–2930 (1988).
13. J. Tanida, M. Fukui, and Y. Ichioka, "Programming of optical array logic. 2: Numerical data processing based on pattern logic," Appl. Opt. **27,** 2931–2939 (1988).
14. J. Tanida and Y. Ichioka, "Optical-logic-array processor using shadowgrams. II. Optical parallel digital image processing," J. Opt. Soc. Am. A **2,** 1237–1244 (1985).
15. J. Tanida, J. Nakagawa, E. Yagyuu, M. Fukui, and Y. Ichioka, "Experimental verification of parallel processing on a hybrid optical parallel array logic system," Appl. Opt. **29,** 2510–2521 (1990).
16. M. P. Ekstrom, *Digital Image Processing Techniques* (Academic Press, Orlando, Fla., 1984).
17. J. Tanida, J. Nakagawa, and Y. Ichioka, "Birefringent encoding and multichannel reflective correlator for optical array logic," Appl. Opt. **27,** 3819–3823 (1988).
18. A. Avizienis, "Signed-digit number representations for fast parallel arithmetic," IRE Trans. Electron. Comput. **EC-10,** 389–398 (1961).
19. E. Swartzlander, "Digital optical arithmetic," Appl. Opt. **25,** 3021–3032 (1986).