Experimental verification of parallel processing on a hybrid optical parallel array logic system

Jun Tanida, Jun Nakagawa, Eiji Yagyu, Masaki Fukui, and Yoshiki Ichioka

A hybrid digital optical computing system is constructed, a variation of the optical parallel array logic system (OPALS). The OPALS is a general-purpose digital optical computing system based on optical array logic, in which image coding and 2-D correlation are used to achieve parallel logical operations. In the constructed system, 2-D correlation for optical array logic is performed optically with a modified multireflective correlator; the other procedures in optical array logic are achieved by electronics including a TV feedback system. We have verified correct execution of programs written by optical array logic on the system. Although the processing speed of the system is still slow because of the sequential process in electronics, it can be drastically improved by replacing the sequential processing devices with parallel ones. *Key words:* Optical computing, digital processing, optoelectronics, computers.

I. Introduction

A digital optical computing system is a promising candidate for a massively parallel computer in the future. The system will fully utilize excellent features of light for computation and communication, such as parallelism, high speed propagation, and crosstalk free interconnection. A number of methods and techniques have been proposed for digital optical computing as well as digital optical logic gates.[1]

Several architectures for digital optical computing systems have been proposed.[2–7] Most of them depend on specific design concepts suitable for optical parallel processing. Although much effort is devoted to construct the optical systems, implementation of quite basic circuits alone, e.g., parallel logic gates and flip-flops, has been actually demonstrated. This depends on the lack of appropriate optical devices.

To compensate the delay of development of optical and/or optoelectronic parallel devices, computer simulation is effective for studies on system behavior and evaluation of processing algorithms. The authors are investigating parallel processing algorithms based on optical array logic (OAL) and have demonstrated their capabilities.[8–10] Nevertheless, construction of a preliminary system is still important because it will provide useful information for designing practical devices and stimulating researchers on optical parallel computing. For this purpose, we have constructed a preliminary optical computing system capable of implementing parallel programming.

The architecture we adopt is that of the OPALS (optical parallel array logic system).[11] The OPALS is a conceptual digital optical computing system based on OAL. Many kinds of processing written by OAL can be executed on the OPALS, e.g., parallel image processing and parallel numerical processing. Several types of OPALS have been considered as available hardware. Among the systems, an optoelectronic hybrid one is easy to construct with current technology, and it is suitable for a prototype of the preliminary system. Thus we choose the hybrid OPALS as the target of our research.

In this paper we describe the hybrid OPALS composed of currently available devices and present some experimental results of parallel processing on the system. In Sec. II we briefly explain OAL and the OPALS. In Sec. III we describe details of the constructed system. In Sec. IV some results of the OAL program executed on the hybrid OPALS are presented. Finally, we discuss the performance of the hybrid OPALS and required devices for future extension.

II. Optical Array Logic (OAL) and OPALS

OAL is a technique to execute parallel logical operation for 2-D image data. Figure 1 illustrates the processing procedures of OAL. Essentially, OAL is achieved by several parallel processing procedures for 2-D data, i.e., encoding, 2-D correlation, sampling, and inverted OR. These procedures are easily implemented with optical methods in parallel, so that OAL is

When this work was done, all authors were with Osaka University, Department of Applied Physics, 2-1 Yamadaoka, Suita, Osaka 565, Japan; Jun Nakagawa is now with Konica Corporation, 2970 Ishikawa-cho, Hachioji, Tokyo 192, Japan; Eiji Yagyu is now with Mitsubishi Electric Corporation, Central Research Laboratory, 8-1-1 Tsukaguchi-Honmachi, Amagasaki, Hyogo 661, Japan; and Masaki Fukui is now with NTT Transmission Systems Laboratories, 1-2356 Take, Yokosuka, Kanagawa 238-03, Japan.

Received 24 July 1989.

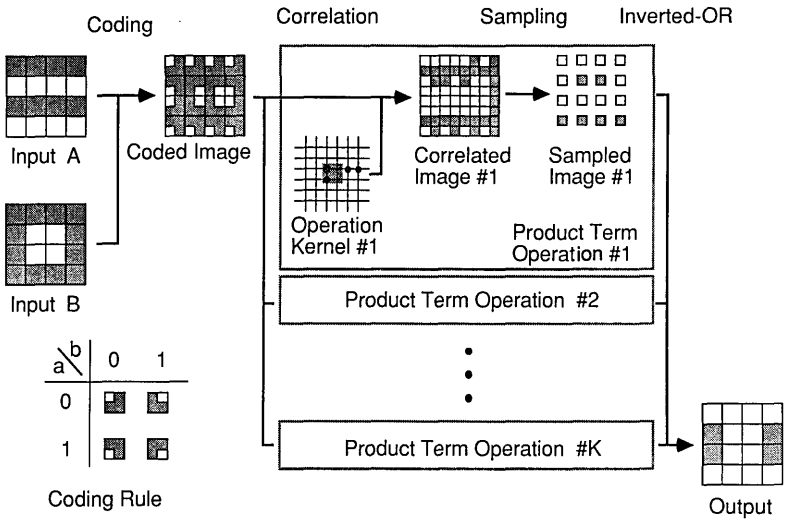


Fig. 1. Processing procedures of OAL.

promising as a basic technique for parallel digital optical computing.

Among the features of OAL, programmability is most important and attractive because it offers great flexibility for processing. In OAL, programming is realized by specifying operation kernels used in 2-D correlation. By selecting the operation kernels, any parallel neighborhood logical operation can be achieved.

OAL programs are described by kernel expression which is designated by the sum of a matrixwise expression. Hereafter we call an individual term in this matrixlike expression a matrix. Figure 2 shows the correspondence between a kernel expression and a set of operation kernels. Each matrix in Fig. 2(a) corresponds to one operation kernel in Fig. 2(c). Symbols, such as .1 and .0, in the matrix in Fig. 2(a) are expressed by specific dot patterns in small grids called kernel units in Fig. 2(b). The kernel units form the operation kernel shown in Figs. 2(b) and (c). Any processing can be described explicitly with the kernel expression. We have developed various ranges of applications written in this notation. In the Appendix, the notation rule is described briefly.

The OPALS is a general-purpose parallel processing system based on OAL.[7] OAL has two inputs and produces one output. As shown in Fig. 3, the OPALS involves a feedback line connecting the output of the OAL processor to one of the inputs for effective iterative operation. Whereas the feedback path is fixed in the original OPALS, the current system has a port selector to increase processing flexibility.

Several versions of the OPALS are considered for matching to various levels of technology. For example, optoelectronic,[7] all-optical,[11] and modularized[12] versions of OPALS are presented. Among these versions we chose the optoelectronic one as a preliminary system. The optoelectronic version of OPALS is a hybrid system in which optics is basically used for processing that requires complicated signal flow, i.e.,

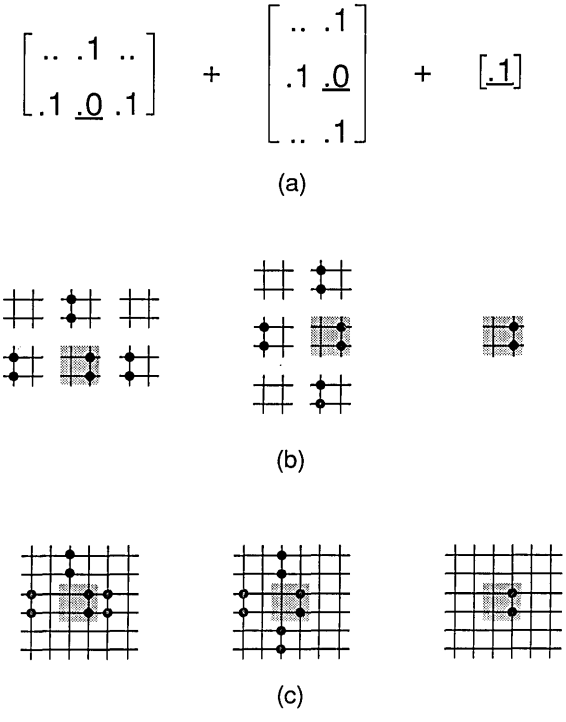


Fig. 2. OAL description by kernel expression: (a) kernel expression; (b) sets of kernel units; (c) operation kernels. The shaded squares in the operation kernels indicate the origin of a neighborhood area.

2-D correlation, and electronics is utilized for nonlinear operations. The hybrid system can be easily assembled with currently available devices. Although the processing speed of the hybrid system is quite slow, flexibility is more important than speed to investigate architecture and to evaluate production feasibility.

III. System Specification of Constructed H-OPALS

Figure 4 shows a block diagram and a photograph of the constructed hybrid OPALS (H-OPALS). The

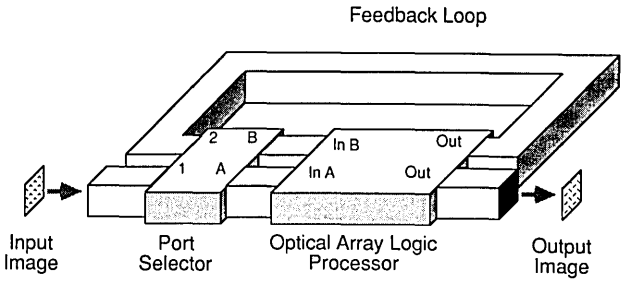
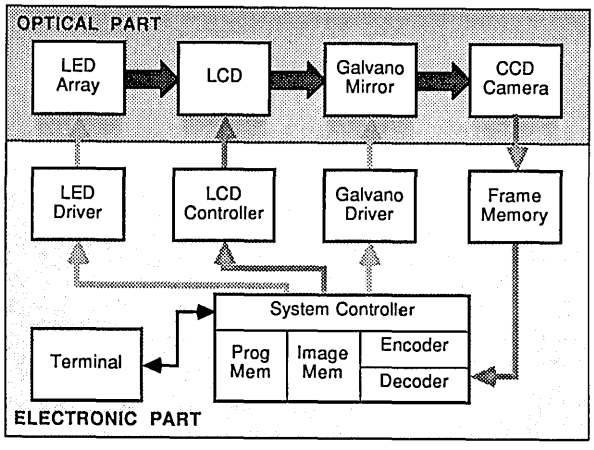
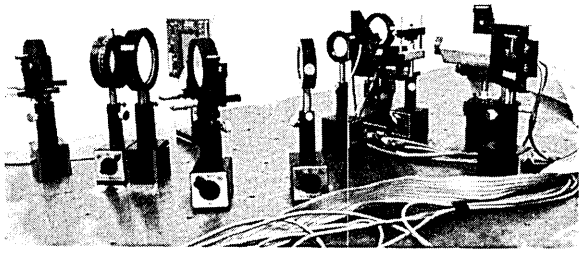


Fig. 3. OPALS involving a port selector.



(a)



(b)

Fig. 4. Constructed H-OPALS: (a) block diagram and (b) photograph of the system.

system is divided into two parts: optical and electronic parts. In the optical part, 2-D correlation is executed and in the electronic part, encoding, sampling, and inverted OR are achieved. Control of the whole system is also implemented by electronics. In the following, we explain individual functional blocks.

The correlator is the main block of the H-OPALS and it serves as 2-D correlation of a coded image and an operation kernel. As pointed out in the previous paper,[9] capability of a correlator determines performance and flexibility of an OAL processing system. The larger size of operation kernels a correlator can operate, the better performance the system attains. We have designed a correlator to treat operation kernels consisting of up to 20 × 20 grids.

To satisfy the above specifications with currently available devices, a multichannel reflective correlator[13] followed by a 2-D galvanometer mirror is used. Figure 5 shows the optical setup of the correlator. For simplicity, optical paths are unfolded at the segmented mirror and the galvanometer mirror. The reason a galvanometer mirror is introduced to the system is that individual segmented mirror pieces cannot be dynamically controlled. Therefore, we use the galvanometer mirror as an active control device for shifting the image by the amount of any pixel sizes.

We use visible red light emitting diodes (LEDs), Toshiba TLRA130-C, as a 2 × 2 light source array and a galvanometer mirror, General Scanning G-100PD, with a drive amplifier, A-601. As an input device, we use a transparent type of liquid crystal display (LCD), EDM-IG127801C made by Matsushita Electric Industrial with controller EDP-LCDC07C. The LCD can display images consisting of 128 × 128 cells, and each cell size is 0.45 × 0.45 mm.

Unfortunately, the LCD has poor contrast ratio (~4:1), so that some compensation techniques must be used. Figure 6 shows the procedure executing 2-D correlation. The 2-D correlation is divided into a sequence of subcorrelations. Each subcorrelation is made for a copy of the coded image and a kernel unit. A kernel unit consists of 2 × 2 grid points. This subcorrelation is executed in the front part of the correlator composed of lenses L_1, L_2, L_3, and L_4 in Fig. 5. In this processing, individual segmented mirror pieces are adjusted to produce images shifted by a half-amount of a pixel to each other vertically and horizontally. Although the segmented mirror is fixed, a change of switching configurations of the LED array

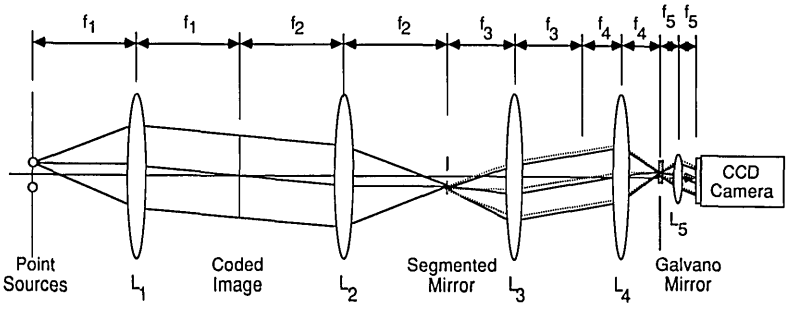


Fig. 5. Optical setup of the multichannel reflective correlator with a 2-D galvanometer mirror.

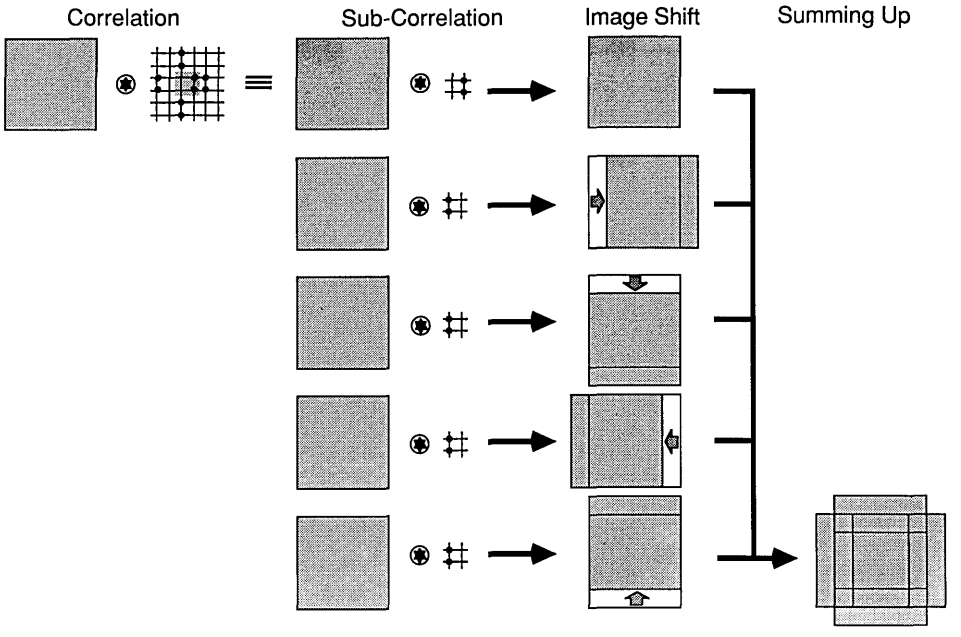


Fig. 6. Sequence of correlation in the multichannel reflective correlator with a 2-D galvanometer mirror.

enables us to achieve any of sixteen possible subcorrelations. Note that the result of subcorrelation is an overlapped version of at most four coded images. Thus we can avoid problems caused by low contrast of the LCD.

Individual results of subcorrelation are shifted by the galvanometer mirror by the amount of any pixel size and they are summed up. An operation of summation is achieved in the electronic system after detection by a charge coupled device (CCD) camera. As long as the beam deflected by the galvanometer mirror passes through lens L_5 in Fig. 5, correlation with any size of operation kernel can be realized in this manner.

The correlated, or subcorrelated, images are detected by the CCD camera, NEC TI-23A. Although the CCD has 380 × 350 resolution points, the optical system is adjusted such that the signal from one cell in the correlated image can be detected in the area of 8 × 8 resolution points. To reduce the mismatching effect between the detected spot size of the image and the cell size of the CCD, median filtering is used for the detected image. The LCD displaying a coded image and the CCD camera have different scanning frequencies, so that flicker results. In addition, image distortion caused by aberration of the optical system makes it difficult to reduce the number of resolution points for one cell. Consequently, a coded image consisting of 40 × 40 cells, equal to 20 × 20 pixels, is treated in the constructed system. This size seems to be too small to implement parallel processing, but it matches well to the capability of the electronic system used.

Encoding, sampling, and inverted OR operations are executed by a personal computer (PC), NEC PC-9801VM, with a frame memory, Edec's Image PC ED-1181. These procedures are programmed by pascal and assembly languages and implemented through a CRT terminal. Radiating configurations of the LEDs

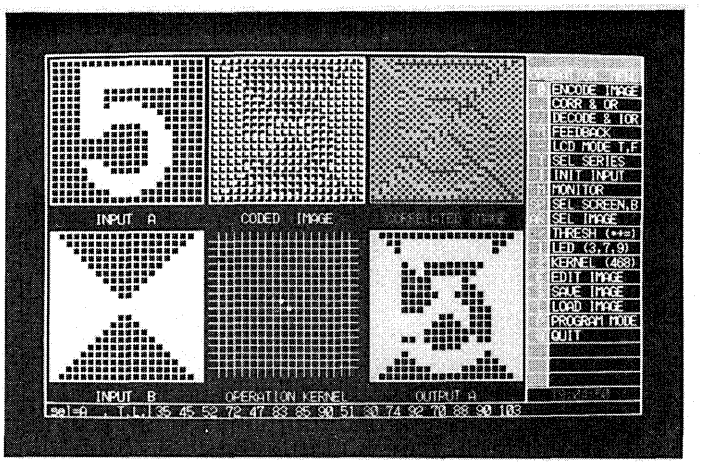


Fig. 7. Terminal display monitoring the H-OPALS.

and the position of the galvanometer mirror are also controlled by the program through input/output ports of the PC, so that all the operations of the constructed system can be controlled from the terminal of the PC. The status of the system is monitored through a CRT as shown in Fig. 7. The control program can execute a sequence of commands written in OAL, or an OAL program.

IV. Experimental Results

Many kinds of processing can be written in OAL. We execute four kinds of parallel processing on the constructed H-OPALS and demonstrate the capabilities of the system.

A. Parallel Logic Operation

We execute simple parallel logic operations. Figures 8(a) and (b) show input images; Figs. 8(c) and (d)

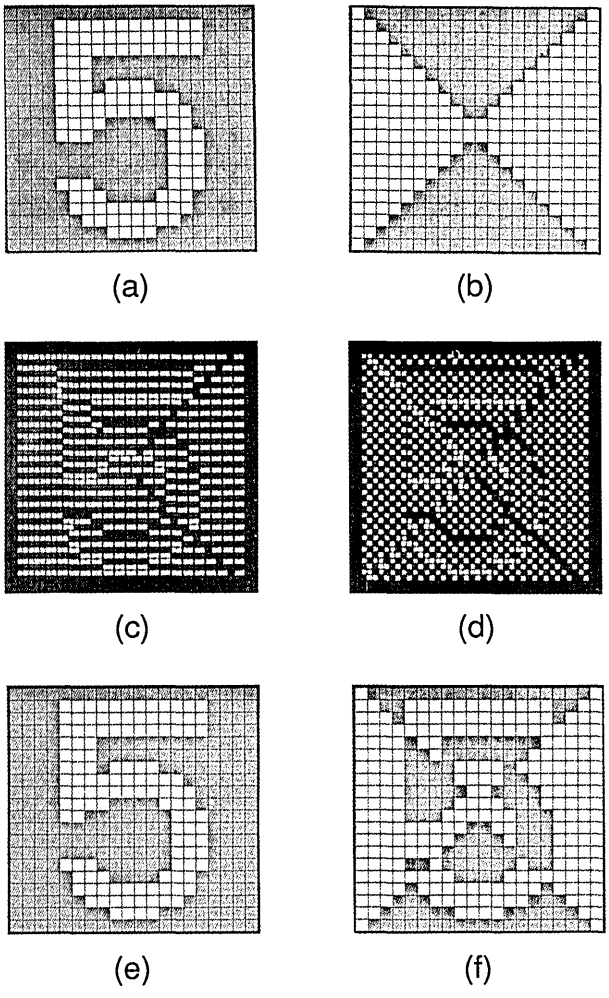


Fig. 8. Experimental result of parallel logic operations: (a) and (b) input images; (c) and (d) optical correlated images in operations A and A XOR B; (e) and (f) outputs of A and A XOR B.

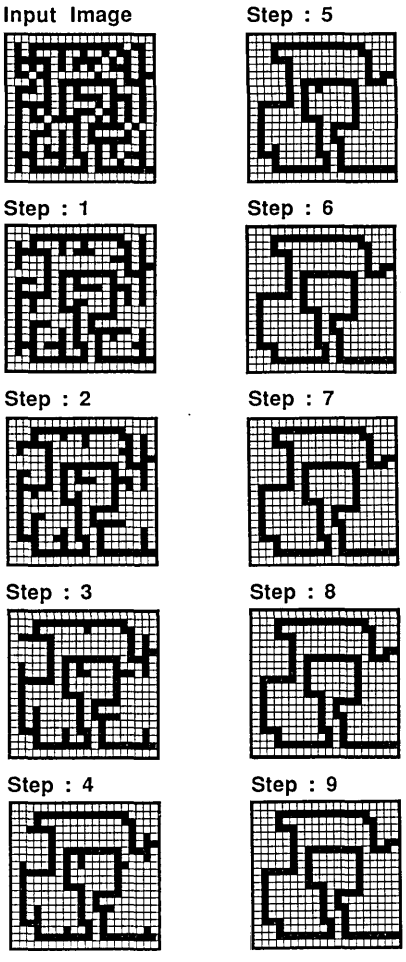


Fig. 9. Experimental results of the maze solution.

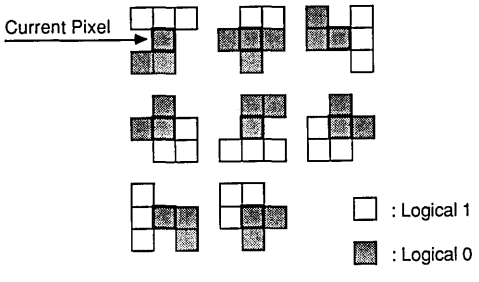


Fig. 10. Pixel patterns used for image thinning.

are optically correlated images for logical operations A and A XOR B, respectively. These images are sampled in the PC to produce the results of logical operations. The final results are shown in Figs. 8(e) and (f). Note that all the output results shown in this paper are not the result of computer simulation but are those obtained by parallel processing using the constructed H-OPALS; they are printed out by a terminal printer connected to the H-OPALS.

B. Binary Image Processing

As examples of binary image processing, maze solution and image thinning are demonstrated. The algorithm used to solve a maze is iteratively to find dead ends of paths by template matching and to delete them. The OAL program is shown in the Appendix. Figure 9 denotes the intermediate processed results and the final result of the maze solution. In the processing, dark pixels in input B indicate paths. After seven iterations the correct path is extracted.

Image thinning can be executed with template matching.[14] If a current pixel surrounded by the boxed area has one of the specific neighborhood patterns shown in Fig. 10, the pixel can be deleted without changing the topology of the input image. This search and deletion algorithm is iteratively executed for thinning. Figure 11 indicates the iterative operation, and the correct result is obtained after seventeen iterations.

C. Numerical Processing

Several algorithms for numerical processing such as addition, subtraction, and multiplication have been written in OAL and presented in previous papers.[9,10]

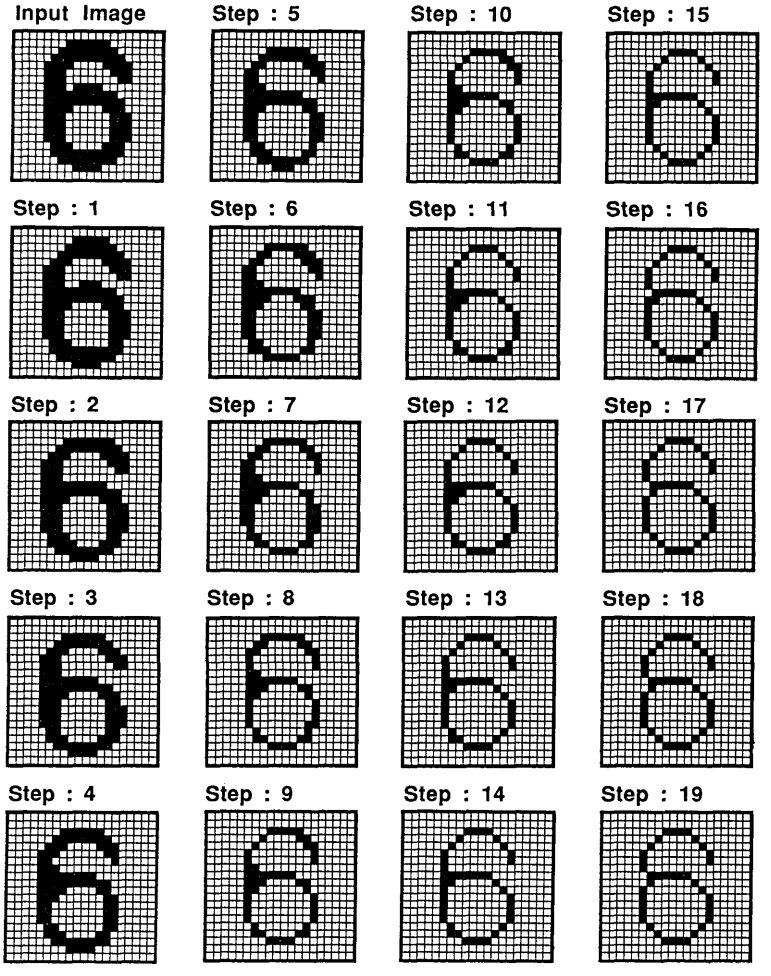


Fig. 11. Experimental results of image thinning.

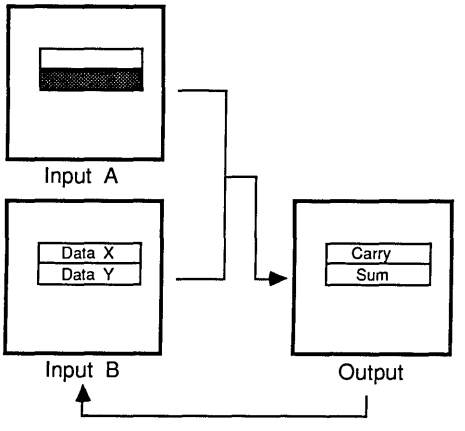


Fig. 12. Algorithm for binary addition with pattern logic.

Here, we demonstrate binary addition with pattern logic.[9] Figure 12 illustrates the algorithm in which images B and A are used to hold two operands and their discriminating tags, respectively. Feeding back output B into input B and iteratively executing the same processing, the result of addition is obtained at the sum location in the output.

Figure 13 shows the experimental results of binary addition for six sets of 8-bit numbers and four sets of 18-bit numbers. Figure 13(a) is input data. Input signals for ten independent binary additions are placed in input image B as shown in Fig. 13(b). The discriminating tags are set in input image A as shown in Fig. 13(c). Intermediate processed results of the first four steps and the final result after eighteen iterations are displayed in Figs. 13(d)–(h). The numerical data decoded from Fig. 13(h) is shown in Fig. 13(i), which indicates the correct answers expressed in a decimal system.

D. Turing Machine

A Turing machine is simulated on the constructed H-OPALS. A Turing machine is categorized into a class of the most discriminating automation.[15] As a result, if a Turing machine can be simulated on the H-OPALS, computational capabilities of the H-OPALS can be proved. In addition, the parallel drive of multiple Turing machines can reveal parallelism inherent in the OPALS.

A Turning machine is composed of a tape and a reading head. Several kinds of symbol are written on the tape. The head moves on the tape, reads the symbol at the head position, and rewrites a symbol at

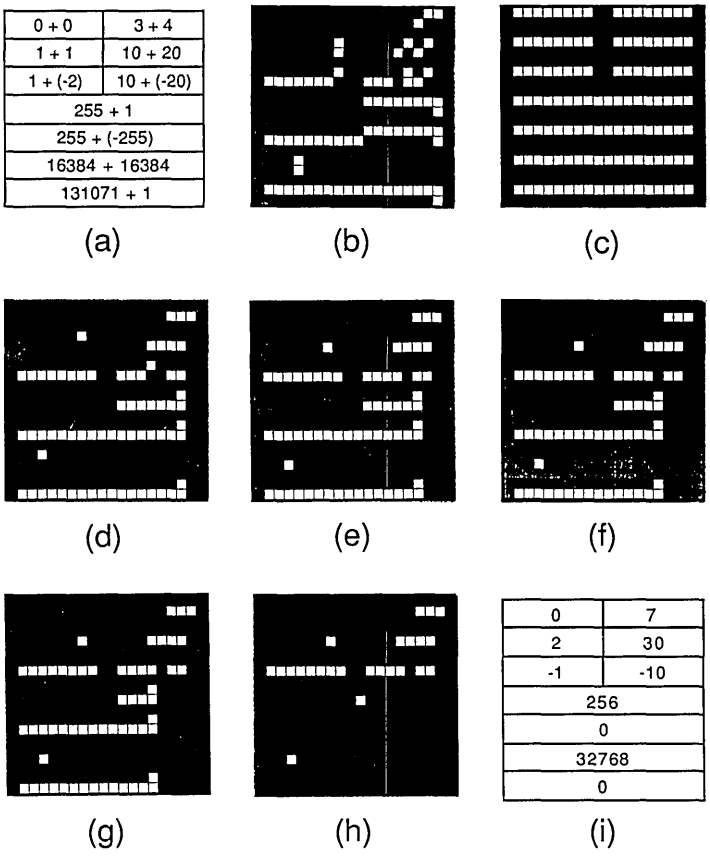


Fig. 13. Experimental results of addition: (a) input data; (b) image form of the input data; (c) attribute patterns for the input data; (d)–(h) output images at first, second, third, fourth, and final steps; (i) the decoded numbers in a decimal system.

the same position. The head has an internal state, which determines the moving direction and the symbol to be rewritten according to the symbol at the head. The operation of a Turing machine is described by a transfer function such as $Q_iT_iQ_oT_oM$, where Q_i is the state of the head when reading and T_i is the read symbol; Q_o and T_o are the rewritten symbol and the state of the head at the next step, and M is the moving direction of the head.

The characteristics of the simulated Turing machine are denoted in Fig. 14(a), and pixel patterns for symbols and internal states are shown in Fig. 14(b). The manner of data arrangement in a Turing machine is shown in Fig. 14(c). The transfer function is assigned by parallel neighborhood operations in OAL. The parallel neighborhood operation is applied in parallel to all the Turing machines placed in the image plane. As a result, we can operate multiple Turing machines in parallel.

Figure 15 shows the result executed on the H-OPALS. In this example, four Turing machines are driven in parallel. Observing the movement of each head, we can verify the capabilities of the constructed H-OPALS as a parallel computer.

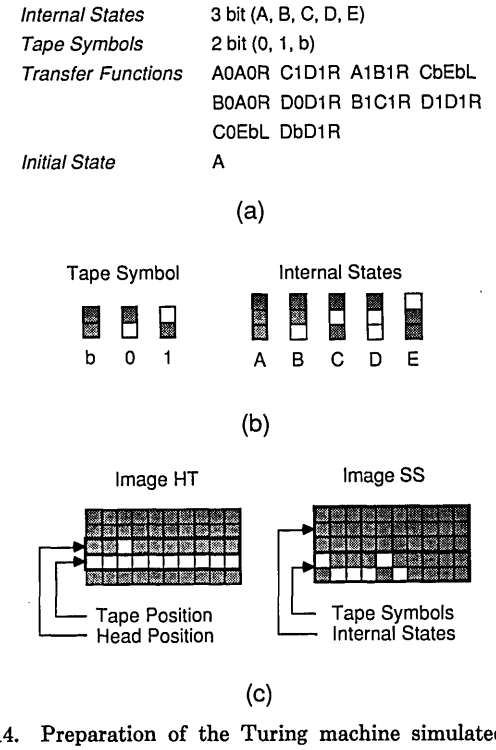


Fig. 14. Preparation of the Turing machine simulated on the OPALS: (a) characteristics of the target Turing machine; (b) pixel patterns for coding symbols and interval status; (c) data arrangement.

V. Discussion

We have presented a preliminary system of the hybrid version of OPALS and demonstrated capabilities

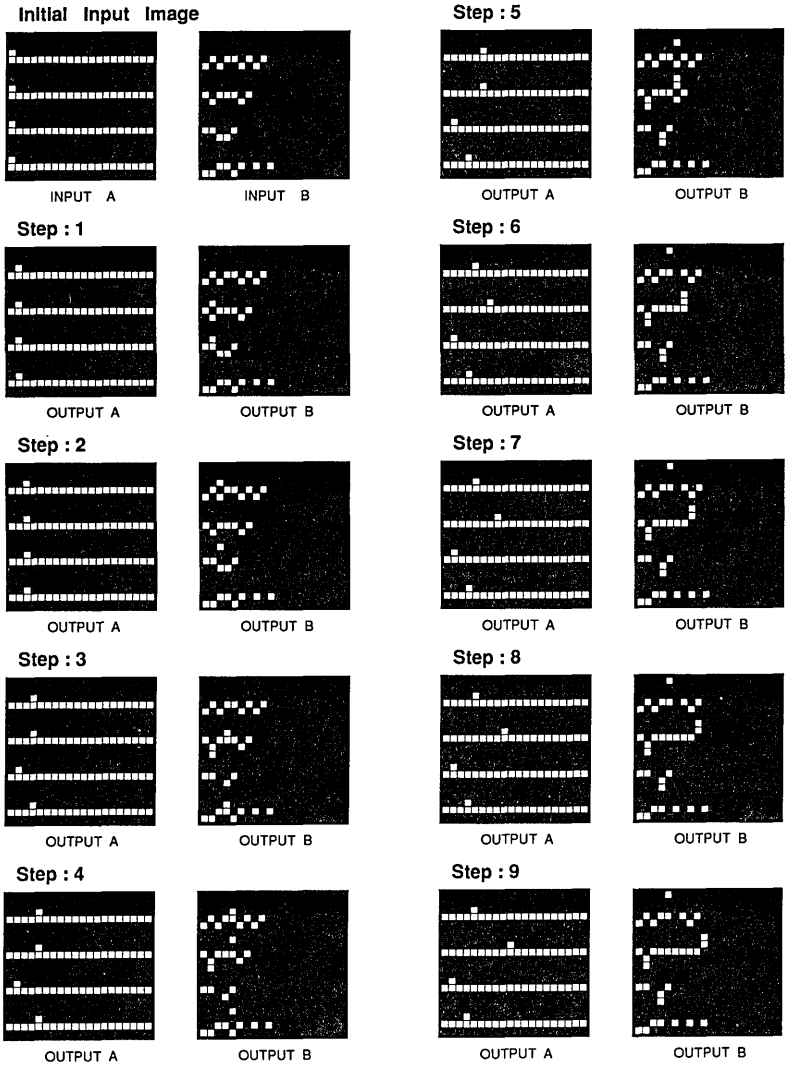


Fig. 15. Experimental results of the Turing machine simulated on the H-OPALS. Four Turing machines are driven in parallel.

Table I. Processing Time Required for One Interation in Executed Programs

Program	Processing time (s)
Maze Solution	9.2
Image Thinning	6.0
Binary Addition	5.5
Turing Machine Simulation	50.0

Table II. Processing Time Required for Each Procedure in OAL

Procedure	Processing time (s)
Encoding	0.054
Displaying Coded Image	1.300
Controlling Galvano Mirror	0.001
Switching LEDs	0.001
Sampling with Filtering	0.117
Summation for Sub-Correlation	0.060
Inverted-OR	0.044

of parallel processing through some kinds of processing on the system. The H-OPALS can execute various kinds of parallel processing with its flexible programmability. However, processing speed of the constructed system is still quite slow. Table I shows the average time required for one iteration in each program. To analyze the bottleneck factor of the system, processing times for individual processing in OAL are measured. Table II tabulates the result, which shows that slow response of the LCD is the main factor to restrict the total processing speed of the system. Therefore, a high speed display device as an interface between optical and electronic systems is strongly desired for the system.

As seen from Table II, most of the procedures executed by electronics require more time than the optical system does, because these procedures are processed sequentially with a PC. Therefore, as we increase the number of pixels, the procedures take longer. However, this problem can be solved by specially designed electronic circuits working on pixel by pixel in parallel. The procedures executed by the circuits do not require complex data flow; namely, all can be achieved by

operations for individual pixels. Such devices can be constructed by designing an elementary circuit for one pixel and duplicating it. Fabrication is not difficult with a current very large scale integration technology. If using such devices, the performance of the H-OPALS would be drastically improved.

Appendix

An easy description of programs is important for developing various programs and archieng them. In OAL, any program can be described with kernel expressions and additional control statements. The authors have developed a simple programming language, which is a modification of conventional programming languages. Here, the notation is briefly explained.

A skeleton program is as follows:

Program statement;
  Variable declarations;
  Execute Statements;
End Statement;

Program statement is followed by a program name and indicates the beginning of a program. Integers, images, and operation kernels are treated as either constants or variables. If they are used as variables, they must be declared with either a **var, image,** or **kernel** statement according to their type.

Integers are allowed to be added, subtracted, multiplied, and divided. Images are allowed to substitute only for an image-type variable. Operation kernels are allowed to be added and multiplied, which are most important in OAL programming, as we explain later. **Exec** and several control statements, such as **for** and **while,** are used for program execution. **Exec (A,B,K)** statement indicates execution of a parallel operation. This statement means that images A and B are assigned as inputs and processed with the operation kernel K. A program is terminated with **end** statement.

Any operation kernel is described by a matrixlike expression. We call an individual term in the expression a matrix. A matrix is composed of symbols specifying kernel units as shown in Table III. Each kernel unit assigns a two-variable binary logic function, $f_{i,j}(a_{ij},b_{ij})$, for the corresponding pixels in inputs A and B. Total operation indicated by a matrix is the result of a logical product of all the symbols in the matrix, which is expressed by

$$\prod_{i,j\in Neighbors} f_{i,j}(a_{i,j},b_{i,j}), \tag{A1}$$

where Π means logical product and *Neighbors* refers to a set of location of neighborhood pixels.

Addition for matrices means that OR operation is required for the results obtained by operation kernels corresponding to the matrices of the addends. Multiplication for matrices is introduced to clarify the meaning of the operation.[10] Using multiplication, we can retrieve an operation kernel from multiple operation kernels; for example, [1.] [.1] can be converted into

Table III. Symbols for Specifying Kernel Units

Function	Symbol	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	a b	11
$\bar{a}\ \bar{b}$	00	0	DD

[11]. Not multiplication itself but a neighborhood operation indicated by the resultant operation kernel is executed in OAL. To create a retrieved operation kernel, calculate a logical product of functions indicated by symbols at the same position in all the matrices of the multiplicands, and simpley convert the resultant function into the symbols shown in Table III.

For convenience of program input, a matrix is expressed with the character | as follows:

```
|..  .1||.1_.0||..  .1|                    (A2)
```

or

```
|.. .1|
|.1_.0|                                    (A3)
|.. .1|.
```

These expressions indicate the same operation kernel as that of the second term shown in Fig. 2(a). As seen from the expression, symbols enclosed by two characters | and | indicate one row of a matrix. The difference between both expressions is whether a new line code is inserted between successive delimiters | |. Therefore, if an interpreter for the OAL program ignores a new line code, we can use a comprehensive expression such as expression (A3). An underbar is a prefix which indicates that the attached symbol corresponds to the origin of the neighborhood area. Using this notation, an OAL program can be put into the system with a standard keyboard.

The following are programs used for processing presented in this paper. They are fed into the system and the system controller executes them sequentially.

A. Maze Solution

```
program MazeSolution;
  var     N, i;
  image   dummy, imageB = MazeData;
  kernel  maze;

  N = 9;
```

```
maze = | .. .1 .. |
       | .1_.0 .1 |
     + | .. .1 |
       | .1_.0 |
       | .. .1 |
     + | .1_.0 .1 |
       | .. .1 .. |
     + | .1 .. |
       |_.0 .1 |
       | .1 .. | + | .1|;
for i = 1 to N do
    imageB = exec(dummy, imageB, maze);
end;
end MazeSolution.
```

B. Image Thinning

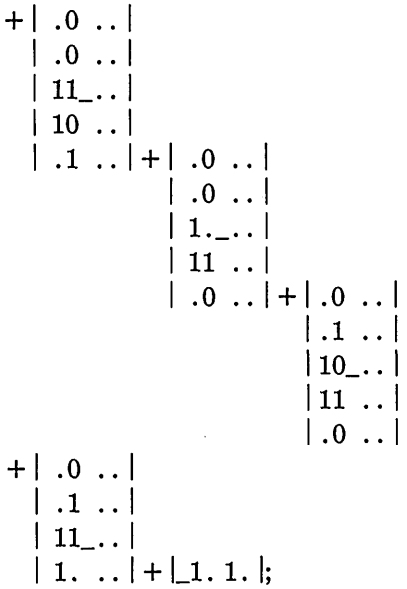
```
program Thinning;
  var     i, N;
  image   dummy, imageB = ImageData;
  kernel  mask1a, mask2a, mask3a, mask4a,
          mask1b, mask2b, mask3b, mask4b, I;

  N = 3;

  I = | .1 |;
mask1a = | .1 .1 .. |
         | .1_.0 .0 |
         | .. .0 .. | + I;
mask1b = | .1 .1 .1 |
         | .._.0 .. |
         | .0 .0 .. | + I;
mask2a = | .. .1 .1 |
         | .0_.0 .1 |
         | .. .0 .. | + I;
mask2b = | .0 .. .1 |
         | .0_.0 .1 |
         | .. .. .1 | + I;
mask3a = | .. .0 .. |
         | .0_.0 .1 |
         | .. .1 .1 | + I;
mask3b = | .. .0 .0 |
         | .._.0 .. |
         | .1 .1 .1 | + I;
mask4a = | .. .0 .. |
         | .1_.0 .0 |
         | .1 .1 .. | + I;
mask4b = | .1 .. .. |
         | .1_.0 .0 |
         | .1 .. .0 | + I;
  for i = 1 to N do
      imageB = exec(dummy, imageB, mask1a);
      imageB = exec(dummy, imageB, mask1b);
      imageB = exec(dummy, imageB, mask2a);
      imageB = exec(dummy, imageB, mask2b);
      imageB = exec(dummy, imageB, mask3a);
      imageB = exec(dummy, imageB, mask3b);
      imageB = exec(dummy, imageB, mask4a);
      imageB = exec(dummy, imageB, mask4b);
  end;
end Thinning.
```

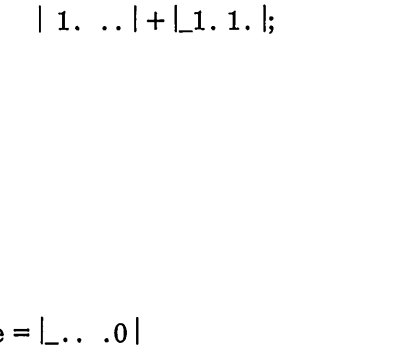
C. Binary Addition

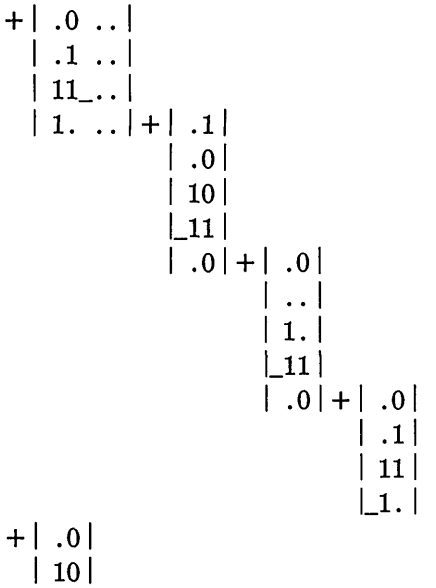
```
program Addition;
  var     i, N;
  image   attr = Attribute, data = Data;
  kernel  add;

  N = 18;

  add = | 1. |
        |_0. | * | .0 |
                |_.1 |
      + | 1. |
        |_0. | * | .1 |
                |_. 0 |
      + |_1. |
        | 0. | * |_...1 |
                | ...1 |;
  for i = 1 to N do
      exec(attr, data, add);
  end;
end Addition.
```

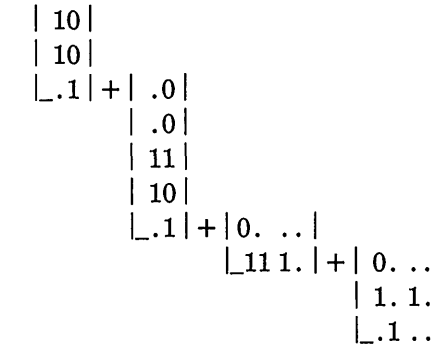
D. Turing Machine

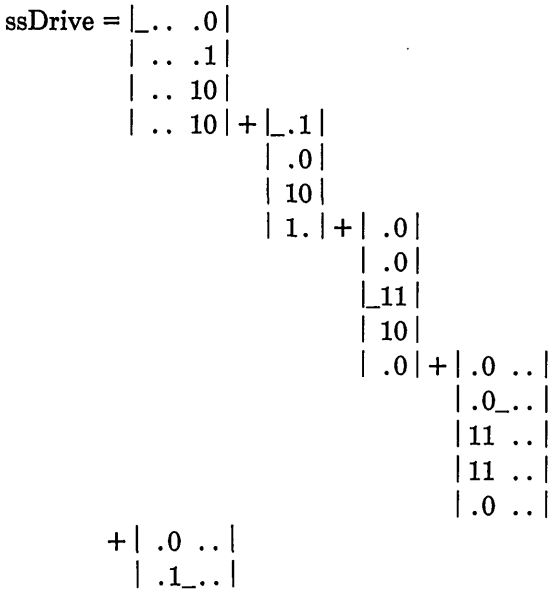
```
program Turing;
  var     i, N;
  image   imageHT = HeadTape,
          imageSS = StateSymbol,
          latchImage;
  kernel  htDrive, ssDrive;

  N = 9;

  htDrive = | .. .0 |
            | .. .1 |
            |_.. 10 |
            | .. 10 | + | .0 |
                        | .0 |
                        |_1. |
                        | 10 |
                        | .0 | + | .1 |
                                  | .0 |
                                  |_10 |
                                  | 1. | + | .0 .. |
                                              | .0 .. |
                                              | 10_.. |
                                              | 10 .. |
```

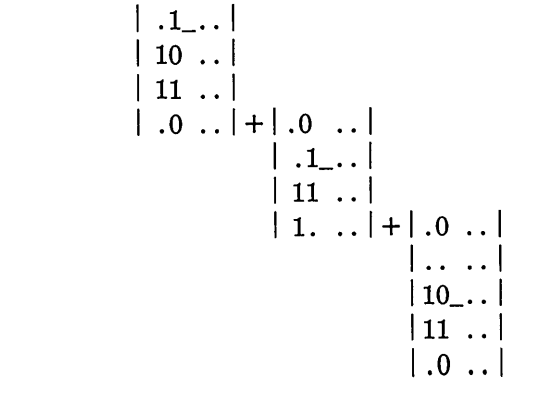












```
  for i = 1 to N do
      latchImage = exec(imageHT, imageSS, htDrive);
      imageSS    = exec(imageHT, imageSS, ssDrive);
      imageHT    = latchImage;
  end;
end Turing.
```

References

1. D. G. Feitelson, *Optical Computing. A Survey for Computer Scientists* (MIT Press, Cambridge, MA, 1988).
2. A. Huang, "Design for an Optical General Purpose Digital Computer," Proc. Soc. Photo-Opt. Instrum.Eng. **232,** 119–127 (1980).
3. B. K. Jenkins, A. A. Sawchuk, T. C. Strand, R. Forchheimer, and B. H. Soffer, "Sequential Optical Logic Implementation," Appl. Opt. **23,** 3455–3464 (1984).
4. B. S. Wherrett, "All-Optical Computation—a Parallel Integrator Based Upon a Single Gate Full Adder," Opt. Commun. **56,** 87–92 (1985).
5. M. J. Murdocca and B. Sugla, "Design for an Optical Random Access Memory," Appl. Opt. **28,** 182–188 (1989).
6. G. Stucke, "Parallel Architecture for a Digital Optical Computer," Appl. Opt. **28,** 363–370 (1989).
7. Y. Ichioka and J. Tanida, "Optical Parallel Logic Gates Using a Shadow-Casting System for Optical Digital Computing," Proc. IEEE **72,** 787–801 (1984).
8. J. Tanida and Y. Ichioka, "Programming of Optical Array Logic. 1: Image Data Processing," Appl. Opt. **27,** 2926–2930 (1988).
9. 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).

10. M. Fukui, J. Tanida, and Y. Ichioka, "Flexible-Structured Computation Based on Optical Array Logic," Appl. Opt. **29,** (1990), in press.
11. J. Tanida and Y. Ichioka, "OPALS: Optical Parallel Array Logic System," Appl. Opt. **25,** 1565–1570 (1986).
12. J. Tanida and Y. Ichioka, "Modular Components for an Optical Array Logic System," Appl. Opt. **26,** 3954–3960 (1987).
13. J. Tanida and J. Nakagawa, and Y. Ichioka, "Birefringent Encoding and Multichannel Reflective Correlator for Optical Array Logic," Appl. Opt. **27,** 3819–3823 (1988).
14. K. Preston, Jr., and M. J. B. Duff, *Modern Cellular Automata. Theory and Applications* (Plenum, New York, 1984).
15. M. Minsky, *Computation: Finite and Infinite Machines* (Prentice-Hall, Englewood Cliffs, NJ, 1967).