Extended coding for optical array logic

Jun Tanida, Masaya Iwata, and Yoshiki Ichioka

We present extended coding for optical array logic (OAL) to avoid the marginal effect. The marginal effect is defined as an effect caused by the finite size of the image region, and it is a problem in massively parallel processing by OAL. OAL is a paradigm of optical computing suitable for optical implementation utilizing image coding and discrete correlation. To avoid the marginal effect in the context of OAL, we propose a new coding rule and consider possible operations with this coding. With extended coding, binary data can be identified from background with the same number of pixels as that used in the original OAL. Simulation results of the operations verify the correctness of the proposed technique.

Key words: Digital optics, optical computing, coding, optical-array logic, parallel processing.

1. Introduction

A variety of methods, techniques, and system architectures that use the desirable features of light such as parallelism, high speed, and connectivity have been presented for overcoming problems of electron-based information processing.[1] Novel schemes for optical computing, such as optical neural networks, are interesting as a future goal, but setting a shorter term goal is also important to clarify problems on the way to developing a practical system and to promoting more study. An optical digital computing scheme is considered to be a promising candidate for the near future owing to its compatibility with current technologies in information processing, the availability of various techniques and devices, and the ease of investigation with computer simulation. Of course, specific techniques and considerations are necessary in optical digital computing to utilize the features associated with the difference in dimensionality as compared with current one-dimensional-based information processing.

There is an essential principle behind achieving efficient massively parallel processing: the processing of as much data as possible at a time. Following this principle, one should send a lot of data to a processing register and apply appropriate operations to them. However, as the amount of data increases, a problem arises: how do we control and handle a large amount of data without loss of processing efficiency? A single-instruction multiple-data (SIMD) stream is an effective solution, and the suitability of SIMD processing is an important motivation for optical digital computing. We have proposed a paradigm for optical digital computing called optical array logic[2] (OAL) and an effective processing scheme called pattern logic.[3,4] Utilizing both techniques, we can process multiple data collectively with SIMD operations.

When considering implementation of pattern logic or other optical digital computing schemes, one will encounter a troublesome problem. Namely, in some situations the margin of the objective image produces undesired processing results. For example, image processing based on convolution with a spatially extended kernel produces an invalid data region around the margin of the objective image.[5] With the Laplacian edge detector the valid image region is decreased in the same way. Another instance of the problem is found in the communication between neighborhood pixels of an image by a parallel shift operation. The shift operation transfers the information of the pixels to other locations in the image, but at the same time it brings undefined information from outside the image region. These problems are caused by the finite size of the image region and are found in parallel processing. In the following, we call this undesired effect the marginal effect. To overcome the problems associated with the marginal effect, we need a sophisticated implementation, and it should be reflected in the design of optical devices for optical digital computing.

In this paper we present extended coding to avoid the marginal effect in the context of the pattern logic

When this study was performed, the authors were with the Department of Applied Physics, Osaka University, 2-1 Yamadaoka, Suita 565, Japan. M. Iwata is now with Electrotechnical Laboratory, 1-1-4 Umezono, Tsukuba 305, Japan.

Received 15 July 1993; revised manuscript received 12 November 1993.

0003-6935/94/173663-07$06.00/0.

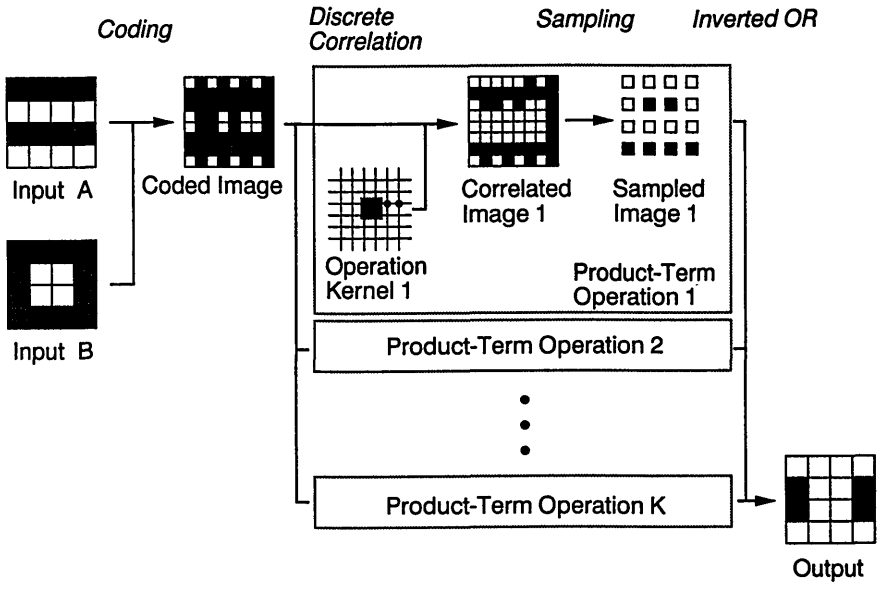

Fig. 1. Schematic diagram of OAL.

implemented by OAL. In Section 2, OAL and pattern logic are explained. In Section 3 the marginal effect is defined, and three methods are shown to avoid the effect. In Section 4 extended coding for OAL is presented and some useful operations are provided. Finally, application and processing efficiency are discussed.

2. Optical Array Logic and Pattern Logic

OAL is a paradigm for optical digital computing based on image encoding and discrete correlation, as shown in Fig. 1. By OAL, arbitrary logic operations on neighboring pixels can be executed in parallel for all pixels belonging to two input binary images, which are fundamental operations in the two-dimensional data and image processing. In the procedure of OAL an individual pair of pixels at the corresponding location on the input images is converted into a coded pattern according to the coding rule shown in Fig. 2. There are four cases for the combination of the pixel values; thus four patterns are prepared, and one of them is selected in the coding. After the coding process, the coded image composed of the coded patterns is correlated with a discrete kernel consisting of several delta functions located at specific positions. These kernels are called operation kernels. This operation is called discrete correlation and is considered a fundamental operation in optical digital computing. The correlated image is then sampled at every other structuring cell to obtain an intermediate product-term image. Inverting and logic summing for several intermediate images, we complete a logic neighborhood operation specified by the combination of operation kernels used in discrete correlation.

To explore the capability of OAL, we have developed various programming techniques and algorithms.[6–9] Pattern logic is a conceptual idea but is useful in the context that a large amount of data must be processed efficiently. Figure 3 shows the conceptual diagram of the pattern logic. In this technique, information primitives to be processed are converted into a set of pixels accompanied by another set of pixels that have the attributes of the information primitives. Each of the pixel patterns is set at the corresponding location in two input images for OAL. With the appropriate design of neighborhood operations in OAL the pixel patterns on the images are processed collectively without care for individual location of the pixel patterns. After processing, the processed pixel patterns are recovered into the original information form.

In pattern logic an image in itself is nothing but a container for pixel patterns. Ideally, such an image should be transparent and not affect the processing on the pixel patterns. However, in actual implementation an image has a finite region and brings an

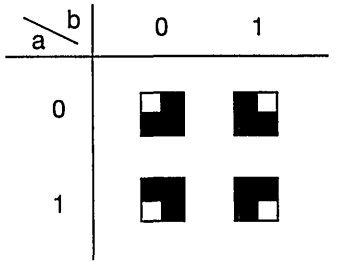

Fig. 2. Coding rule used in OAL.

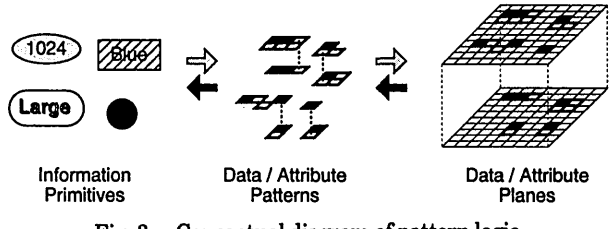

Fig. 3. Conceptual diagram of pattern logic.

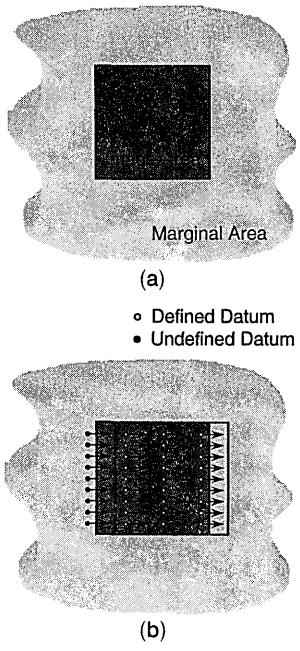

Fig. 4. Image region and marginal effect.

undesired effect owing to its margin when the neighborhood operations are applied. A specific programming technique based on template matching with attribute pixel paterns enables us to avoid this problem, but in this case we must pay a penalty in processing efficiency and pixel usage. Consequently, to execute the pattern logic efficiently, we need a sophisticated technique.

3. Marginal Effect

Before proceeding in our discussion, we must define and estimate the marginal effect. In Fig. 4(a), an image is defined as an area enclosed by four edges. Usually, valid information is located in the image as a collection of discrete elements, called pixels. When the image is shifted horizontally and/or vertically, undefined information in the marginal area is brought into the image area, as shown in Fig. 4(b). We define the effect caused by intrusion of the marginal information as the marginal effect.

Since the physical size of an actual device is limited, the marginal effect cannot be avoided. Table 1 shows a calculation of the ratio of the pixels unaffected by the marginal effect to the total number of pixels as the pixel number per dimension and the neighborhood number vary. For pixel number L the neighborhood area is determined as $(2L + 1)(2L + 1)$. As seen from the table, the rate of the unaffected pixels becomes quite low for a large neighborhood number. Since optical interconnection has an advantage over electronic interconnection for a long distance and complicated connection (i.e., for a large neighborhood number), the marginal effect is recognized as a serious problem.

Table 1. Ratio of the Pixels Unaffected by the Marginal Effect to the Pixel Number

	Neighborhood Number[a]				
Pixel Number[b]	1	4	16	64	256
2	0.00				
8	0.56	0.00			
32	0.88	0.56	0.00		
128	0.97	0.88	0.56	0.00	
512	0.99	0.97	0.88	0.56	0.00

[a]Neighborhood number L corresponds to $(2L + 1)(2L + 1)$ neighborhood area.

[b]The pixel number is expressed by a number per dimension.

As solutions for the above problem, three methods are considered, as shown in Fig. 5: the wraparound plane method, the marginal mode method, and the extended coding method. In the wraparound plane method [Fig. 5(a)] an ideal image plane, whose edges are connected, is assumed to eliminate margins from the processing image. Although this method seems elegant, its implementation is difficult and requires a complicated setup. In the marginal mode method [Fig. 5(b)], values of the marginal pixels are implicitly assigned to either 0 (called normal mode) or 1 (called inverse mode). The essential problem caused by the marginal effect is that undefined values intrude the image area. Thus assigning implicit values to the marginal pixels can avoid the problem. Although this technique was used in our parallel processing based on OAL, we note that an OAL programmer needs to pay attention to the marginal mode when writing a program. In the extended coding method [Fig. 5(c)], special code patterns are employed to identify the existence of data. In general this method requires an extension of the coding rule to represent multiple status, or more than two states. In section 4 we explain the details of the extended coding for

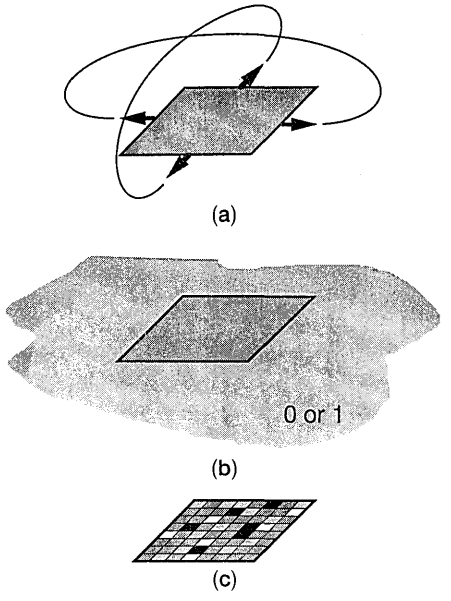

Fig. 5. Three solutions to manage the marginal effect: (a) wraparound plane method, (b) marginal mode method, (c) extended coding method.

OAL to avoid the marginal effect and to implement pattern logic efficiently.

4. Extended Coding for Optical Array Logic

A. Coding Rule

As described in Section 2, in pattern logic, information primitives are coded into pixel patterns and set in images. However, the OAL scheme is based on binary logic, in which only two values, 0 and 1, are employed to represent the data. As a result, there is no identification between the information primitives and the background images. This suggests that multivalued coding for more than two states is required.

Figure 6 shows the coding rule used in the proposed extended coding. In the extended coding, a new state, called none, is introduced to represent nonexistence, or invalidity, of data. For two input variables there are nine cases, including the none state, as shown in Fig. 6. For example, the state in which a is 0 and b is none means that only datum a is valid, and its value is 0.

To increase the possible number of states represented by coding, coded patterns with multiple pixels can be also used.[4,10,11] Although this method is simple and is implemented within the original OAL scheme, it wastes pixels and reduces the total number of data located on the processing images. On the other hand, the proposed coding requires the same number of pixels as used in the original OAL, thus no penalty is paid for the pixel number required by coding.

B. Logic Operations

The extended coding is an upper set of the original OAL coding. Thus operations in the original OAL are effective for the four types of coded patterns used in the original coding. In addition, logical operations in the original OAL offer interesting responses for the other coded patterns. Figure 7 shows output patterns for nine possible coding cases obtained by discrete correlation with operation kernels used in the original OAL for 16 logic operations. The numbers under each output pattern indicate the optical intensity at the center part of the pattern by the unit of illuminated intensity and the corresponding logic value. Since intensity varies between 0 and 4, we assign intensity 0 to logic 1 and anything else to logic 0 according to the procedure in OAL.

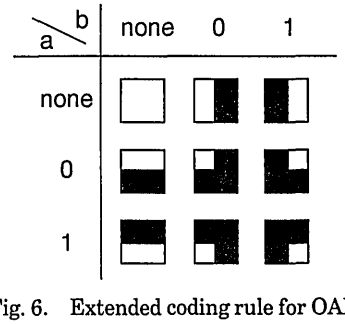

Fig. 6. Extended coding rule for OAL.

Figure 8 depicts the logic outputs for three operations, AND, OR, and NOT A, in truth-table form. Looking into the tables carefully, one can find that each operation provides reasonable outputs for the nonexisting input cases. For example, in AND operation, if two data are valid, i.e., not in the none state, and their values are 1, then logic 1 is output. In OR operation, if at least one datum is valid and its value is 1, then 1 is output. In NOT A operation, regardless of b, if a is valid and its value is 0, then 1 is output.

Figure 9 shows results from a computer simulation for the AND, OR, and NOT A operations for two images. Assuming a situation appeared in the pattern logic, we see that several isolated pixels coded from information primitives are set in the image planes [Figs. 9(a) and 9(b)]. The extended coding image is shown in Fig. 9(c). The effective image area is set to be surrounded by the coded patterns corresponding to the state in which both a and b are invalid. In practical implementation this setting is achieved by a transparent area surrounding the image area. Figures 9(d)–9(f) are correlated images of the coded image with operation kernels for AND, OR, and NOT A operations. Although the background intensity varies for each case, logic 1 is provided by the common intensity 0. Therefore logic outputs are obtained as shown in Figs. 9(g)–9(i). As seen from the result, it is verified that correct results are obtained.

C. Data Validity Test

In addition to logic operations a useful operation can be derived by use of the analog nature of the output signals. Referring to the rightmost column of Fig. 7, one can find that the intensity reflects the number of valid data. Figure 10 shows the correspondence in table form. Therefore correlation with the operation kernel consisting of 2×2 delta functions, as shown in Fig. 11(a), can be used to test the data validity. We call the operation the data validity test (DVT). If the output intensity is equal to 1 in this configuration, it is verified that both inputs a and b have valid data, i.e., either 0 or 1.

The DVT operation can be extended for testing pixel patterns arranged in a neighborhood area. With the above kernel, locations (i, j) in the input images, in which both data $a_{i,j}$ and $b_{i,j}$ are valid, are detected. In OAL an operation kernel larger than 2×2 provides neighborhood operation. In the same manner the target of the DVT operation can be enlarged. For example, to execute a neighborhood operation between the pixel at the origin of the neighborhood area ($a_{i,j}$ and/or $b_{i,j}$) and its under pixel ($a_{i+1,j}$ and/or $b_{i+1,j}$), we need an operation kernel of size 2×4. In this configuration, if delta functions are set at all possible positions in the kernel, as shown in Fig. 11(b), locations (i, j) at which the data $a_{i,j}$, $a_{i+1,j}$ $b_{i,j}$, and $b_{i+1,j}$ are valid provide the output intensity equal to the number of locations involved in the neighborhood area (i.e., 2). Therefore the output intensity equal to the number of locations involved in the neighborhood area indicates the location

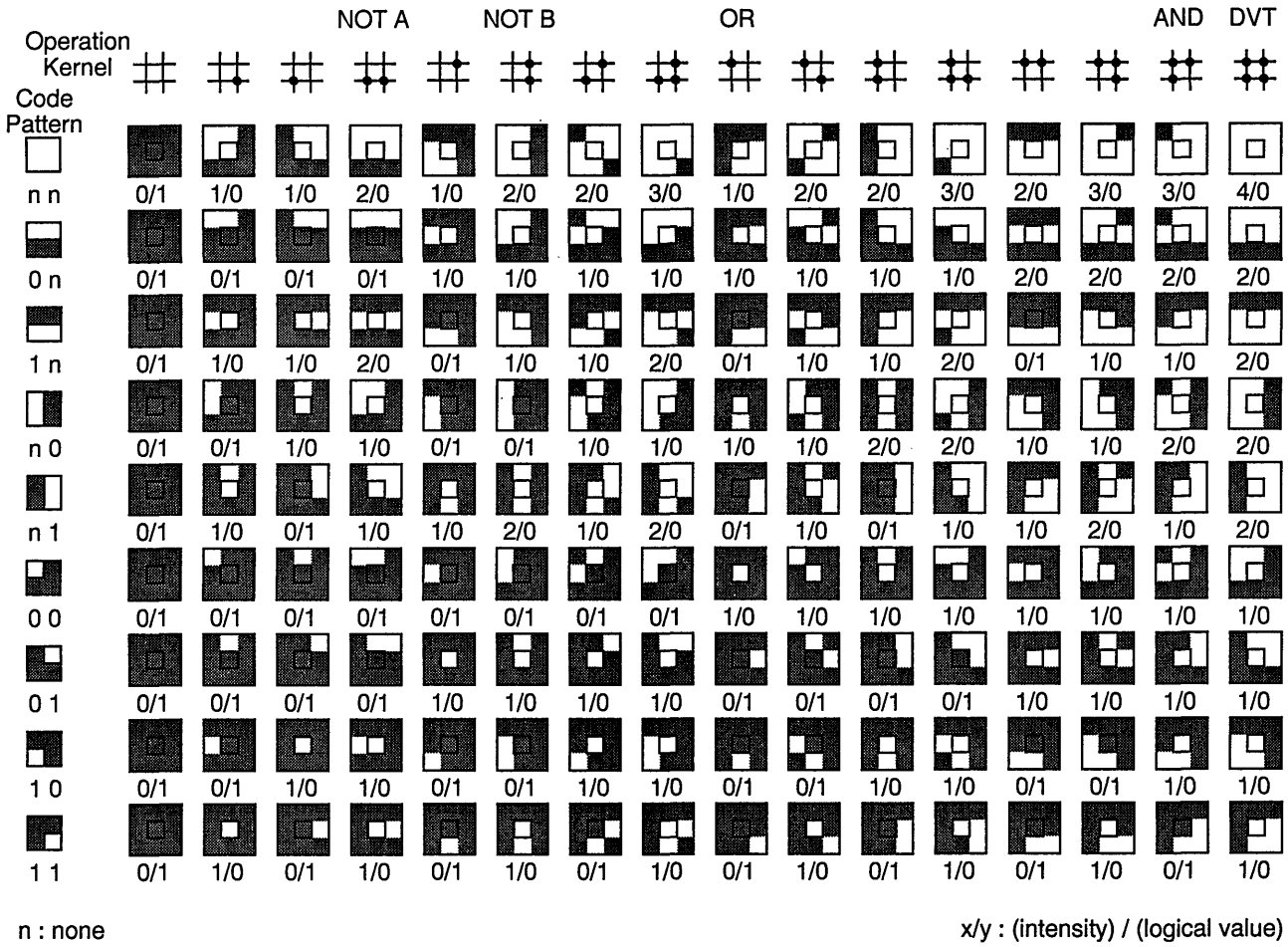

Fig. 7. Output patterns for extended coding by logic operations in OAL: n means none state; numbers under each pattern represent intensity (left) and logic value (right).

at which all of the neighborhood pixels have valid data. Of course, the shape of the neighborhood area can be specified by the operation kernel used.

Figure 12 shows computer-simulation results of the DVT operation for single and double locations. The input images contain pixel patterns with different shapes [Figs. 12(a) and 12(b)]. The extended coding image is shown in Fig. 12(c). Figures 12(d) and 12(e) are correlated images with operation kernels in Fig. 11(a) and 11(b), respectively. To detect locations in which all data in the specified neighborhood area are valid, we select a specific intensity and assign it to logic 1; for the single location case the intensity is 1, for the double location case the intensity is 2. In Figs. 12(f) and 12(g), locations (i, j) satisfying $\{(i, j) | a_{i,j} \neq \text{none}, b_{i,j} \neq \text{none}\}$ and $\{(i, j) | a_{i,j} \neq \text{none}, a_{i+1,j} \neq \text{none}, b_{i,j} \neq \text{none}, b_{i+1,j} \neq \text{none}\}$ are detected, respectively.

To implement the DVT operation, we need a specialized circuit to compare the signal detected at the correlated plane and the signal corresponding to the number of locations involved in the neighborhood area. Although it might be executed time sequentially by a program sequence, effective hardware can be constructed as shown in Fig. 13. This DVT circuit consists of two optical circuits based on a

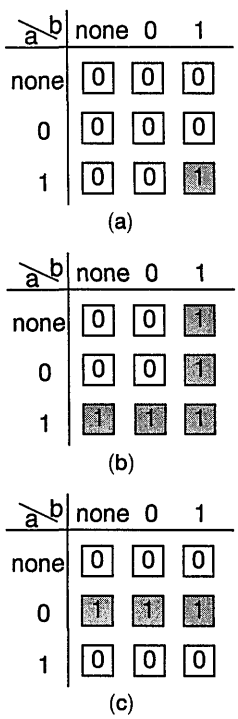

Fig. 8. Truth tables of (a) AND, (b) OR, and (c) NOT A operations for extended coding.

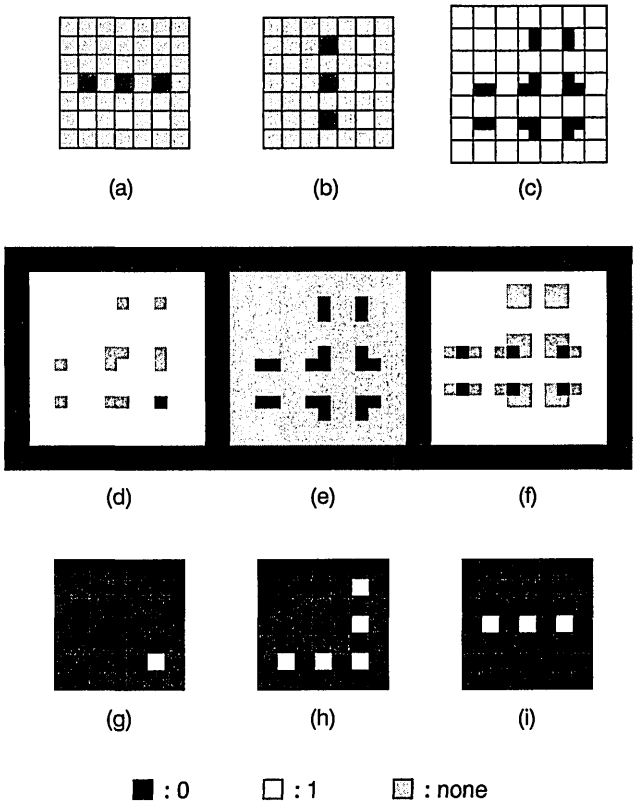

Fig. 9. Simulation results of AND, OR, and NOT A operations: (a), (b) input images, (c) extended coding image, (d)–(f) correlated images, (g)–(i) output images; (d) and (g) are for AND, (e) and (h) are for OR, and (f) and (i) are for NOT A.

shadowgram utilizing spatial multiplexing. Between the normal sampling positions in OAL, extra detectors are set to detect the signals for the DVT operation. With this configuration the DVT operation can be executed independently from the logic operation. For the individual operations two sets of operation kernels, one for logic operation and the other for DVT operation, are specified independently.

D. Application

Extended coding is expected to be useful in massively parallel processing based on pattern logic. With extended coding and DVT operation, pixel patterns converted from information primitives can be identified and processed effectively. However, if the results obtained are used in the following operation, they must be encoded by the extended coding rule with attention to data validity. This process might be an overhead of the processing. Regardless of the overhead, flexibility and capability obtained by the extended coding are attractive for massively parallel processing.

a \ b	none	0	1
none	4	2	2
0	2	1	1
1	2	1	1

Fig. 10. Output intensity for extended coding by the DVT operation.

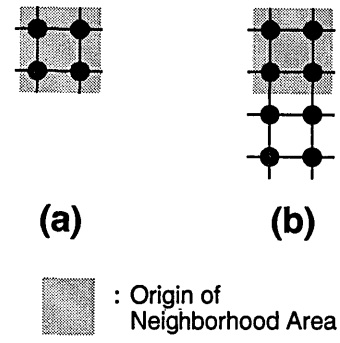

Fig. 11. Operation kernels for the DVT operations. The target neighborhood area is (a) only the origin and (b) the origin and the next lowest location.

The DVT operation is used to detect the ready state in data flow computing.[12] In the data flow computing architecture, processors, all of which are ready to process input data, start their operations asynchronously. However, when the number of processors is

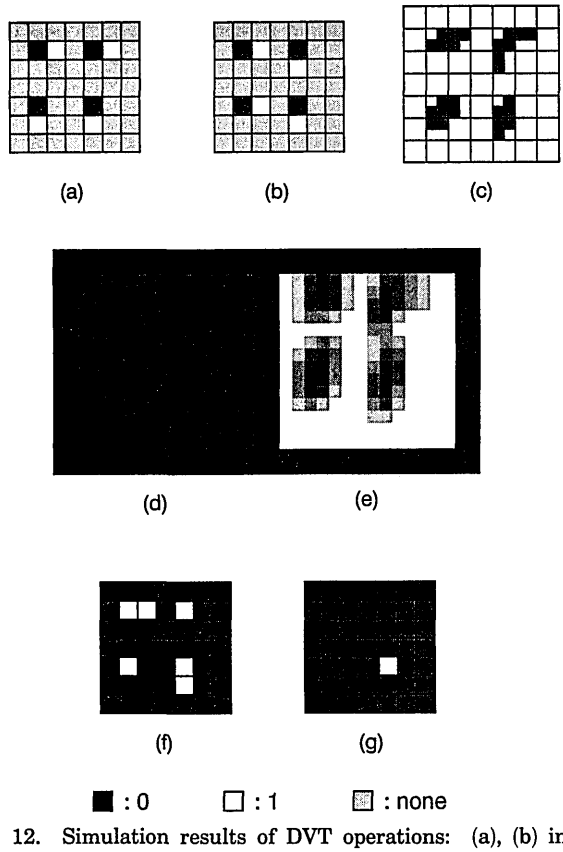

Fig. 12. Simulation results of DVT operations: (a), (b) input images, (c) extended coding image, (d), (e) correlated image, (f), (g) output images; (d) and (f) are for single location and (e) and (g) are for double-location cases.

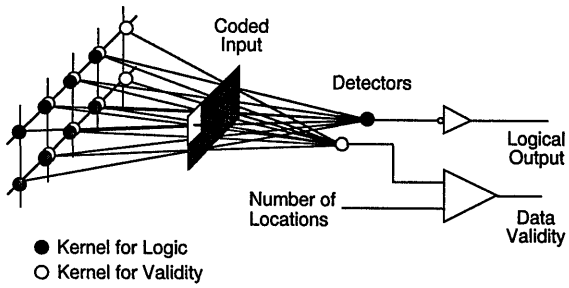

Fig. 13. Example of the optical setup for concurrent DVT operation.

large, it is troublesome to detect which processor is ready to go. Hence the extended coding and DVT operation can be used for this purpose. The flexible-structure computing technique presented in Ref. 6 will be a powerful computing method with the extended coding scheme.

5. Conclusion

In this paper we have presented extended coding to manage the marginal effect in pattern logic implemented by OAL. First, OAL and pattern logic is explained. Then, the marginal effect is defined and three solutions are presented. As an effective solution, extended coding is presented and its useful operations are shown as well as its promising applications.

References and Notes

1. Feature on optical computing, Appl. Opt. **29,** 1999–2200 (1990); feature on optical computing, Appl. Opt. **31,** 5423–5728 (1992).
2. J. Tanida and Y. Ichioka, "Programming of optical array logic. 1. Image data processing," Appl. Opt. **27,** 2926–2930 (1988).
3. 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).
4. J. Tanida and Y. Ichioka, "A paradigm for digital optical computing based on coded pattern processing," Intl. J. Opt. Comput. **1,** 113–128 (1990).
5. W. K. Pratt, *Digital Image Processing* (Wiley, New York, 1978), Chap. 9, pp. 214–231.
6. M. Fukui, J. Tanida, and Y. Ichioka, "Flexible-structured computation based on optical array logic," Appl. Opt. **29,** 1604–1609 (1990).
7. S. Kakizaki, J. Tanida, and Y. Ichioka, "Gray-image processing using optical array logic," Appl. Opt. **31,** 1093–1102 (1992).
8. M. Iwata, J. Tanida, and Y. Ichioka, "Inference engine for expert system using optical array logic," Appl. Opt. **31,** 5604–5613 (1992).
9. M. Iwata, J. Tanida, and Y. Ichioka, "Database management using optical array logic," Appl. Opt. **32,** 1987–1995 (1993).
10. K.-H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," Appl. Opt. **25,** 3054–3060 (1986).
11. M. J. Murdocca, "Digital optical computing with one-rule cellular automata," Appl. Opt. **26,** 682–688 (1987).
12. K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing* (McGraw-Hill, New York, 1985), Chap. 10, pp. 732–768.