Final Technical Report for

Grant AFOSR-84-0181

NONLINEAR REAL-TIME OPTICAL SIGNAL PROCESSING

A. A. Sawchuk, Principal Investigator

Signal and Image Processing Institute
University of Southern California
Mail Code 0272
Los Angeles, California 90089-0272
(213)743-5527

Research Sponsored by the

Air Force Office of Scientific Research
Electronics and Materials Science Division
AFOSR/NE
Bolling AFB, Bldg. 410
Washington, D.C. 20332

Performance Period: 1 July 1984 - 31 January 1990

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1a. REPORT SECURITY CLASSIFICATION
UNCLASSIFIED

1b. RESTRICTIVE MARKINGS

2a. SECURITY CLASSIFICATION AUTHORITY

2b. DECLASSIFICATION / DOWNGRADING SCHEDULE

3. DISTRIBUTION / AVAILABILITY OF REPORT The United States Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright notation here.

4. PERFORMING ORGANIZATION REPORT NUMBER(S)

5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION
Signal and Image Processing Inst.
Univ. of Southern California

6b. OFFICE SYMBOL *(If applicable)*

7a. NAME OF MONITORING ORGANIZATION
Same as 6a

6c. ADDRESS *(City, State, and ZIP Code)*
MC-0272
Los Angeles, CA 90089

7b. ADDRESS *(City, State, and ZIP Code)*
Same as 6c

8a. NAME OF FUNDING / SPONSORING ORGANIZATION
Air Force of Scientific Research

8b. OFFICE SYMBOL *(If applicable)*
NE

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
AFOSR 84-0181

8c. ADDRESS *(City, State, and ZIP Code)*
Bldg. 410, Bolling AFB
Washington, D.C. 20332

10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.
61102F	2305	B1	

11. TITLE *(Include Security Classification)*
Nonlinear Real-Time Optical Signal Processing

12. PERSONAL AUTHOR(S)
A.A. Sawchuk

13a. TYPE OF REPORT
Final Technical

13b. TIME COVERED
FROM 7/1/84 TO 1/31/90

14. DATE OF REPORT *(Year, Month, Day)*
1990 September 1

15. PAGE COUNT
127

16. SUPPLEMENTARY NOTATION

17. COSATI CODES

FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
Optical Computing
Optical Signal Processing

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

During the period 1 July 1984 - 31 January 1990, the research under Grant AFOSR-84-0181 has been concerned with binary parallel optical computing architectures with particular attention to cellular logic and symbolic substitution for pattern recognition and numerical operations. Our approach has been to experimentally implement binary optical cellular logic processors and interconnection arrays; define an instruction set and software suited to optical computing systems; and to study generalizations of optical cellular logic processors such as the cellular hypercube. The results include the experimental implementation of a 54-gate binary optical cellular logic processor with instruction decoders, input/output, memory and test/branch functions; the completion of a binary image algebra (BIA) description of cellular logic, image analysis and symbolic operations; and the development of binary image algebra algorithms for scale and shift invariant pattern recognition. Additional work concerns the relationship of parallel computation paradigms to optical computing and halftone screen techniques for implementing general nonlinear functions.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS

21. ABSTRACT SECURITY CLASSIFICATION
UNCLASSIFIED

22a. NAME OF RESPONSIBLE INDIVIDUAL
[illegible]

22b. TELEPHONE *(Include Area Code)*
202-767-4931

22c. OFFICE SYMBOL
NE

DD Form 1473, JUN 86 Previous editions are obsolete. SECURITY CLASSIFICATION OF THIS PAGE

Contents

1 Summary

During the period 1 July 1984 - 31 January 1990, the research under Grant AFOSR-84-0181 has been concerned with binary parallel optical computing architectures with particular attention to cellular logic and symbolic substitution for pattern recognition and numerical operations. Our approach has been to experimentally implement binary optical cellular logic processors and interconnection arrays; define an instruction set and software suited to optical computing systems; and to study generalizations of optical cellular logic processors such as the cellular hypercube. The results include the experimental implementation of a 54-gate binary optical cellular logic processor with instruction decoders, input/output, memory and test/branch functions; the completion of a binary image algebra (BIA) description of cellular logic, image analysis and symbolic operations; and the development of binary image algebra algorithms for scale and shift invariant pattern recognition. Additional work concerns the relationship of parallel computation paradigms to optical computing and halftone screen techniques for implementing general nonlinear functions.

2 Research Progress

This section summarizes research progress and accomplishments for the period 1 July 1984 - 31 January 1990 on Grant AFOSR-84-0181 for Nonlinear Real-Time Optical Signal Processing. These results are discussed separately in the sections that follow.

2.1 Digital Optical Parallel Computing Systems

We have continued work on an experimental sequential optical binary parallel architecture that is constructed from an array of binary optical switching elements (NOR gates) with interconnections done by a computer-generated hologram. We examined new binary array spatial light modulators (SLM's), high efficiency, high space-bandwidth product (SBWP) interconnection holograms, and compact reflection versions of the general architecture with the intent of building a larger demonstration system with greater capabilities. We have studied improved methods of providing the interconnections in these systems by the use of hybrid digital/analog *(facet)* holograms. A final area of study has been to examine in detail algorithms that are well-suited for implementation on the parallel binary architectures described previously. We have defined several methods for building binary and arithmetic cellular logic processors and have determined some limits due to hologram complexity, gate density, etc.

A reprint describing the general types of system under consideration is included for reference. the paper, by B.K. Jenkins and A.A. Sawchuk, is "Binary Optical Computing Architectures", *Optics News,* Vol. 12, No. 4, pp. 25-26, 1986.

~~imagine two possible paths for this development: an evolution or a revolution.~~

~~For the moment, optical components are used sparingly in predominantly electronic computers. The operating mode of the optical devices is dictated by the existing structure of the computer which in turn has been shaped by the characteristics of semiconductor devices. As optics is increasingly used in computers, it will perhaps become apparent that the traditional advantages of optical computing can be used to enhance the capability of these optical components.~~

~~For instance, if optical memories become commonplace, it will be inevitable that parallel access of these memories will be attempted in order to get to the stored information faster, which will probably create a communication bottleneck in a traditional computing structure. This may bring about the need for more extensive global communication between the memory and the processing elements in the computer, and this, in turn, may require an optical solution for the communication problem that the optical memory created.~~

~~Evolutionary processes of this type are certain to occur. The only question is how far they will go. Will enough optical devices and optical techniques eventually be inserted in an electronic computer to allow us to call this machine an optical computer rather than an electronic computer with some optical components?~~

~~I don't think the answer to this question is important. *What is important is the realization that if optical components are used in computers, then there is a tremendous potential for improvement in the performance through "optical computing" ideas and techniques.*~~

~~The evolutionary path to optical computing, as outlined above, is something that was not planned by anybody. Most of us working on optical computers imagine their development as a revolution; we start with a new technology (optics), and find new ways to solve problems that are suited to this technology, and we find a new set of important problems to solve that present computers have a hard time with. The successful path towards the development of such radically new computers is, needless to say, not certain and not yet universally agreed upon.~~

~~In order to be competitive with the well-entrenched semiconductor technology, it is important that we identify clearly the comparatively advantageous features of optics and try to make the best use possible of them. Global communication and the capacity for dense storage of information are two of the strong suits of optics. We ought to be able to find ways to store large amounts of information optically, and also have ways to retrieve this information very quickly using optical communication.~~

~~If we do this successfully, then we are left with the question of how to process all this information we are retrieving. The possible answers may lie in the area of neural network models.~~

~~A neural network is a very large collection of neurons (perhaps 100 billion of them), with each neuron being connected to thousands of others. The capability to store large amounts of information and the ability to get to this information quickly and efficiently are what give neural networks their computational power. This is accomplished by storing the information in the interconnections among the neurons, which is a most ingenious way to avoid bottlenecks in trying to get to the stored information.~~

~~The problems that neural networks are particularly good at (recognition of images and speech, classification of patterns, and associations) are problems that electronic computers are particularly poor at solving. There is an excellent match between the capabilities and strengths of optics and neural networks. Therefore, if we can obtain some insights from the work of biologists and others who study neural networks, this can prove very helpful in our effort to design optical computers.~~

~~A natural compatibility has emerged between the requirements of problems typically encountered in artificial intelligence, the ways that such problems are solved by a neural network, and the capabilities of optics. The development of optical computers that exploit this compatibility is one of the exciting and *promising prospects for the future of the field.*~~

Binary Optical Computing Architectures

BY B. KEITH JENKINS AND ALEXANDER M. SAWCHUK
SIGNAL AND IMAGE PROCESSING INSTITUTE
UNIVERSITY OF SOUTHERN CALIFORNIA
LOS ANGELES, CALIF.

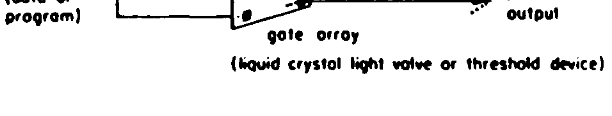
Binary parallel optical computing architectures differ greatly from traditional optical analog and numerical processors. The potential advantages of these architecutres are:

■ They offer flexibility of operations—numerical,

Binary parallel optical computing architectures differ greatly from traditional optical analog and numerical processors: they offer the possibility of high throughput and processing speed with arrays of fast optical switches that are being developed.

symbolic or logical, compared to analog or discrete multilevel processors;

- They have binary digital accuracy and dynamic range;
- They offer computing architectures very different from electronic very large scale integration (VLSI): they permit global interconnections and parallel input-output compared to the local interconnections, clock-skew problems, pin-in/pin-out and bus limitations of very large scale integration (VLSI);
- They utilize the 2-D parallel nature of optical device arrays and low interaction of optical signals for interconnections in 3-D; and
- They offer the possibility of high throughout and processing speed with arrays of fast optical switches that are being developed.


interconnection unit
(computer generated hologram)
parallel inputs
(data or program)
parallel output
gate array
(liquid crystal light valve or threshold device)

An optical digital computing architecture with global interconnections.

The figure shows one concept of an optical sequential logic system with global interconnections. An experimental system based on this concept has been built. The gate array at the bottom is a 2-D array of optical NOR gates formed on the surface of a liquid-crystal light valve or other spatial light modulator having an array of optically activated switching elements. The light valve operates in transmission, so that the inputs are applied on the left side of the device (hidden in the figure), while the outputs are obtained on the right side of the device.

With this arrangement, optical signal inputs and outputs are accessible in parallel. An interconnection unit (which is currently a computer generated hologram) connects gate outputs to gate inputs in a very general way and forms the "wiring." The gate interconnections can be global or local with equal ease, because a third dimension is utilized and optical signals can propagate through each other with minimal effective interaction. Components such as flip-flops, registers, memory, instruction decoders, arithmetic logic units, and central processing units are defined by a fixed wiring pattern. The resulting machine could, in principle, be general-purpose or special-purpose, and could be programmable.

The interconnection wiring can be altered by changing the hologram; thus a binary optical computer with dynamically reconfigurable wiring is a possibility. A wide variety of hologram encoding techniques can be utilized, and the gate array may contain $\approx 10^5$ or 10^6 gates. One important aspect of this system is that it can be configured as a binary parallel computer, which is very different from traditional architectures.

An experimental version of the system in the figure has been implemented. The system is an all-optical 16-gate digital sequential circuit, including clock and flip-flop, implemented using a Hughes liquid-crystal light valve. The system contains a high-resolution computer-generated hologram for gate interconnection, which was made on an electron-beam integrated circuit mask writer. Many different optical systems using different types of computer-generated holograms can be used for the interconnections; current research is concerned with comparing various alternatives and improving hologram resolution and flexibility.

Several computationally demanding practical problems such as parallel digital image processing and image analysis are well-matched to this architecture. In the future, a large ($\approx 10^6$ gates) 2-D array of binary switching (threshold) or bistable devices, preferably all-optical (optical input and output) could be used in the system, which could provide nanosecond switching times. Many alternative technologies exist; they must be compared and evaluated.

2.2 Optical Cellular Logic Processors

We have continued work on optical cellular logic processors (CLP's) and other parallel digital processing architectures that are well-suited for implementation on the sequential optical architecture described in the previous section. Optical CLP's are well matched to this architecture because they are a 2-D, page oriented array of individual processors located at every pixel of a large image. The attached paper by B.K. Jenkins and A.A. Sawchuk, "Optical Cellular Logic Architectures for Image Processing", from *IEEE Computer Society Workshop on Computer Architectures for Pattern Analysis and Image Database Management,* November 1985, summarizes some of these concepts. Work in progress includes studies on the implementation of cellular hypercubes and pyramids, which are not feasible for electronic VLSI, but offer important advantages for improved image processing.

OPTICAL CELLULAR LOGIC ARCHITECTURES FOR IMAGE PROCESSING

B.K. Jenkins and A.A. Sawchuk

Signal and Image Processing Institute
Electrical Engineering - Systems Department
University of Southern California
Los Angeles, California 90089-0272

ABSTRACT

A digital optical processing system consisting of optical gates and optical interconnections is described. Its concept has been demonstrated experimentally. The implementation of an optical cellular logic processor is considered. Optical systems for interconnections are given, and the architectural characteristics of such optical cellular logic machines are discussed and compared with electronic machines.

I. INTRODUCTION

In the optical processing community there has been a substantial amount of research in the area of optical image processing and pattern recognition. Most of the systems to date have been analog, and have the potential of processing large amounts of data in parallel and at high speeds. However, their analog nature limits both the accuracy and the range of operations that can be achieved with a given system. A digital optical system holds the promise of alleviating these restrictions while maintaining some of the advantages of optical systems. A cellular logic processor is one possible use of such a digital optical computer.

The current interest in digital optical computing can be largely attributable to two developments: (1) recent progress in optical materials and devices useful for the implementation of gates, including improvements in size, potential manufacturability, cascadability, and especially switching energy; and (2) the realization that optical systems could have significant advantages over electronic computers in certain application areas. These advantages are due primarily to the optical interconnections, and include the abilities to implement large numbers of interconnecting lines with little or no regard for their length. This stems primarily from the fact that electrons interact at a distance whereas photons do not. These advantages will be discussed in more detail below.

In this paper we will review some of our work in digital optical computing, and discuss the possible optical implementation of a cellular logic processor and some of its architectural characteristics. A general review of digital optical computing is given in Ref. 1.

II. OPTICAL LOGIC SYSTEM

An optical logic system can be built out of optical gates and interconnections. If these interconnections include a provision for feedback, then clocks and memory can constructed in addition to combinatorial logic. These are the minimum hardware requirements to be able to implement, in principle, arbitrary digital processing operations. We have demonstrated an optical logic system that includes these elements; it allows large numbers of interconnections between gates. It is described in this section. We also point out that other digital optical processing systems have been described [1].

In our system a 2-D array of gates is combined with an optical holographic interconnection system to create a general optical sequential logic system. Its inherent 3-D structure provides for a high degree of interconnection flexibility. The idea is to take the array of gate outputs and send it through a holographic system back to the array of gate inputs (Fig. 1). The holographic system connects the output of each gate to inputs of other gates, effectively wiring up a circuit. For ease of manufacture, the holograms can be generated by (electronic) computer and written out using a computer plotting device. We have experimentally demonstrated the concept of this system with a 16-gate circuit. In this section we will discuss the gate array and interconnection system.

2-D arrays of optical gates demonstrated to date have one drawback or another that preclude their use in a practical, competitive optical optical logic system. While current devices can implement 10^5 to 10^6 gates in one array, in most cases the major drawback is the extremely slow speed of the devices. (Typical response times are > 1 ms.) Recent progress in the area of optical bistability, however, provides hope for fast optical logic systems. To date their demonstrations have been primarily on individual (single gate) devices, but in principle they can be used for 2-D arrays as well. Gate switching times on the order of ns have been demonstrated [2], and there is potential for even much faster gates [3,4] (although other considerations such as power may limit the usable response time in a system to $\sim$ ns). Many of these devices are all optical (intrinsic) in that the signal is not converted to electrons and then back to photons again in order to obtain the nonlinearity. This is one of the reasons for their speed advantage. For a review of optical bistability, the reader is referred to [5,6].

We have previously described three different optical interconnection systems for interconnecting the gates [7]. All of them use holograms in conjunction with free-space propagation. Their characteristics differ and this manifests itself in the kinds of circuits and processors that can be implemented most efficiently with each system. Here we discuss one of the systems, which is a hybrid space-variant/space-invariant system. This system has the most general applicability and is the most pertinent to cellular logic processors. A review of all three systems can be found in Ref. 8.

The hybrid interconnection system represents a basis-

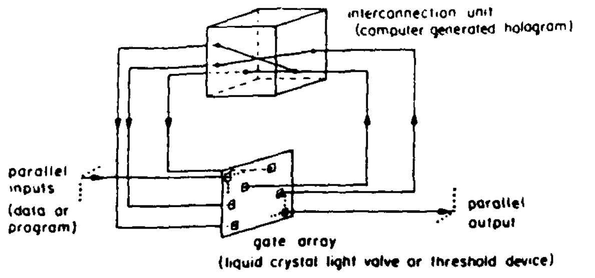

interconnection unit
(computer generated hologram)
parallel inputs
(data or program)
parallel output
gate array
(liquid crystal light valve or threshold device)

Fig. 1. Functional block diagram of sequential optical logic system.

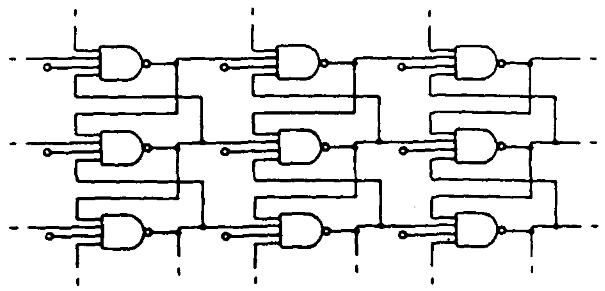


Fig. 2. Example of only one interconnection pattern being used for all gate outputs (M=1).

set approach to interconnections. The optical system consists of two holograms and two or more lenses. The idea is to define a finite number, M, of distinct interconnection patterns, and then assemble the circuit using only these M patterns. A circuit with only one interconnection pattern is shown in Fig. 2 - all gate outputs have exactly the same interconnection pattern. We generally expect that $1 << M << N$, where N is the number of gates. Each different interconnection pattern is essentially stored in only one place on the hologram, and many gate outputs can use this same interconnection in a non-interfering manner. In addition, once the necessary set of interconnection patterns is defined, it may be possible to determine a smaller basis set of new interconnection patterns that still achieves all interconnections [9]. If so, M is reduced even more. The number of gates and interconnection patterns that are implemented determine the complexity of the holograms. The hologram complexity that can be achieved is limited by the capabilities of recording devices (e.g., computer plotting devices). Calculations indicate that with current plotting devices, if there are $M = 50$ interconnection patterns, then $N \sim 10^7$ gates can be interconnected [7]. Increasing M will decrease N, such that MN is constant. (We expect future gate arrays to have $\sim 10^6$, possibly up to 10^7, gates.) Thus with this approach the designer has some minor limitations on the interconnections which can be used, but he has a potentially large number of gates at his disposal.

Of course, with a large enough M, any circuit can be implemented. However, the potential of this system can be exploited more fully by implementing circuits with a high degree of regularity or symmetry. An example is a processor array. Typically interconnections between processing elements (PEs) have a considerable amount of regularity, which can reduce the size of M. Examples include mesh-connected arrays, pyramids, and hypercubes. The interconnections within each PE may be completely arbitrary. The fact that many of the PEs would typically be identical provides a major reduction in M, since each interconnection pattern is stored only once. We also point out that whether the interconnections are global or local has essentially no effect on M or N.

We have experimentally demonstrated the concept of this optical logic system. An array of NOR gates was used with an interconnection system that uses a single hologram. A test circuit consisting of 16 gates was implemented. It comprises a synchronous master-slave flip-flop and a driving clock consisting of an odd number of inverters in a feedback loop. This experiment and its results are described in Ref. 10.

III. COMPARISON OF DESIGN CONSTRAINTS

The design of processors and computers in any technology is constrained by the inherent characteristics of the technology. In this section we compare some of these constraints for electronic and optical systems and discuss how they affect system architectures.

Since the development of electronic LSI and VLSI, the cost of individual gates has constituted only a minor factor in the overall system cost function. The major concern has become internal and external communications [11,12]. The internal wiring network affects the amount of active chip area available for gates; in current systems it is common for more than 70% of the chip area to be devoted to interconnections [13]. Because of the resistance and parasitic capacitance associated with each on-chip wire, the response time of a gate and the propagation delay of a wire both become functions of the length of the wire. Timing and clock skew become problems because of the differing wire lengths [14,15]. (Although the wire resistance can be reduced by process technology, e.g., using thick metal layers, it appears that the wire lengths will still be a limiting factor in the system timing considerations [16].) Another restriction related to interconnections is the limited number of pin-outs which becomes more apparent as the number of gates per chip increases. One result of these restrictions is the need to minimize the number and length of interconnections.

For optical logic systems, the major design considerations are admittedly not so well defined as for VLSI, but it is clear that the cost function is much different. In particular, most of the communication costs affecting VLSI design are not associated with optical systems. An optical system can be made so that all interconnections have the same length to first order. (For example, this is the case in the optical logic system described in Sec. II.) Thus synchronization problems due to clock skew can be eliminated, making large synchronized systems more feasible. Being able to synchronize the circuits eliminates the need for handshaking or other asynchronous techniques which introduce waiting time for individual circuit elements. The other design constraints associated with wire length, namely power consumption and device area utilization, can be avoided with optical systems, for example by using free space propagation in the third dimension for the interconnections.

Pin-outs are not a constraint in optical systems. Optical systems can accept a large number of parallel inputs and can generate a large number of parallel outputs. These are usually in the form of 2-D arrays of data or bits, e.g. bit planes of images. The careful partitioning of large systems is then unnecessary, and limitations on concurrent and pipelined processing due to large I/O requirements is relieved.

Of course, digital optical systems will have constraints of there own. It appears that these will be primarily concerned with the gates. We have seen, for the interconnection system described in Sec. II, that there is a preference for regular or repeated interconnections. The gates might also present some design constraints; this may be in the total number used or in the average repetition rate at which they are switched. In addition, other limitations may of course surface as the technology progresses.

Because of these differences in the cost functions of electronic and optical systems, certain application areas are specifically well-suited to one or the other. VLSI systems are being particularly considered for applications which involve very regular structures and simple data flow that can be handled with only local communications. An example is systolic array architectures [15,17], which are well-suited to many vector-matrix and matrix-matrix operations. On the other hand, algorithms which inherently require global communications cannot be conveniently handled by VLSI, but could, in principle, be implemented with an optical logic system. Examples of such communication-limited operations include some fast Fourier transform (FFT) algorithms which required global communications due to their butterfly structure [18], and some image processing operations which will be discussed in Sec. IV.

IV. CELLULAR LOGIC PROCESSORS

A cellular logic processor uses a PE, or cell, for each pixel or set of pixels of an image. A full array processor has in principle a separate PE for each pixel of an image, so that the number of PEs is equal to or larger than the size of the largest image it will process. The number of PEs in a subarray processor is generally smaller than the image size. The discussion here applies primarily to full array processors, but the possibility of processing images larger than the number of PEs will also be considered.

Possible Optical Implementations

Again we use the optical logic system of Fig. 1. We assume that one gate array can provide $\sim 10^6$ to 10^7 gates. We refer to one gate array plus one interconnection unit as a chip, and note that multiple chips can be connected. We should point out that the number of chips that can actually be used in an overall system will depend on improvements in switching power of the gates and advancements in other areas. Also, while in priciple the system of Fig. 1 can be made small ($\sim$1 cm on a side), we will not consider the effects of physical size in this paper. Now, an interconnection unit or units may be used to connect between a small number of chips. Another method of interconnecting, when the number of chips is moderate, may be to mosaic multiple gate arrays into a larger 2-D array, and to use larger holograms to interconnect both within each gate array and between gate arrays.

The hybrid or basis-set interconnection system described in the previous section could be used to implement a cellular logic processor. Since all PEs are identical (except perhaps for a small number of additional PEs for other purposes), the number of interconnection patterns is relatively small. The gate array will most likely be the limiting factor

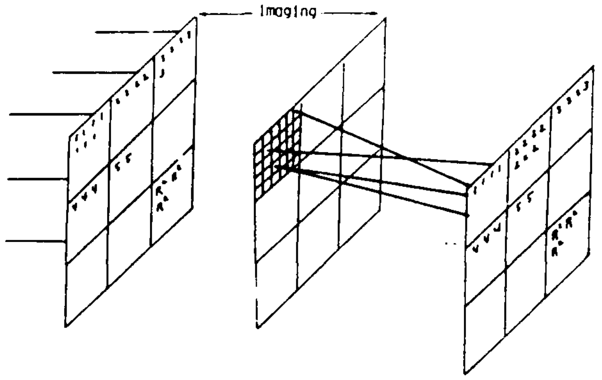


Fig. 3. Optical interconnection system for cellular logic - block mode. Imaging optics are omitted for clarity. Gate outputs enter from the left, and the right plane is sent to the gate inputs. Image pixel inputs are shown, one to each block (PE). Gates numbered 1 are part of PE 1, gates numbered 2 are part of PE 2, etc.

in the number of gates per chip. If the PEs do not all fit on one chip, they may be divided so that $k \times k$ PEs are put on each chip; or, each PE may be distributed over more than one chip so that each processor chip has a portion of every PE on it. This is possible because of the parallel I/O of each chip - conceivably each chip could have 10^6 I/O lines.

A variant of the hybrid interconnection system could be used. In this case volume (thick) holograms are used, which could be optical copies of computer-generated holograms. This provides an increase in optical power efficiency as well as in the achievable hologram complexity. It also provides the possibility of copying multiple computer-generated holograms onto one volume hologram, which might be used to interconnect a mosaic of 2-D gate arrays. Again there are two possible ways of organizing the locations of the PEs.

In one case each PE is physically localized. Topologically neighboring PEs are placed in physical proximity. The hologram or gate array(s) are conceptually divided into blocks, one for each PE (Fig. 3). All gates numbered 1 are part of PE 1, gates numbered 2 are part of PE 2, etc. We refer to this as block mode. Communication within a PE is done by interconnections within each block, and between PEs is done by the same type of interconnections, only they pass into neighboring PEs. In this case, communication within each PE can be arbitrary, between neighboring PEs is easy, but between distant PEs may be more difficult (i.e., it may increase the hologram complexity). The requirements on the hologram complexity in order to implement a cellular logic processor are approximately the same in this case as with the hybrid interconnection system, but since a more complex hologram can be made, more gates can effectively be interconnected.

The other extreme distributes each PE over the gate array(s). In this interleaved mode, corresponding gates, one from each PE, are physically grouped together (Fig. 4). The image is input to one group of gates, which are the input gates of each PE. Within-PE interconnections are then done by connecting an entire group of gates to another entire group of gates. Between-PE interconnections can be done similarly except with a slight misalignment from one group

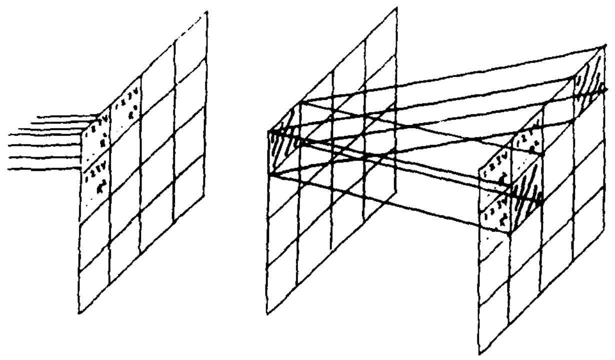


Fig. 4. Optical interconnection system for cellular logic - interleaved mode. Imaging optics are omitted for clarity. Image inputs are shown, one to an input gate of each PE. Gates numbered 1 are part of PE 1, etc.

of gates to the other. Again the hologram complexity dictated by this system is approximately the same as in the hybrid system.

Architectures and Characteristics

One problem with full array cellular logic processors (CLPs) is that there may always be some images to process, say of size m x m, that are larger than the number of PEs, say n^2. When such an image is processed, if it is processed in blocks of n x n pixels, one to a PE, then the borders between blocks can cause problems, especially in iterative calculations. Incorrect data propagates inward from the border by an amount proportional to the number of iterations [19]. One way of avoiding this is by loading blocks of the image with overlapping boundaries into and out of the array during every iteration. This significantly slows the process down in the case of most electronic CLPs; while the calculations for one iteration may be done in O(1) steps, loading data into the PEs takes O(n) steps if n PEs are loaded in parallel. Another way of avoiding this boundary problem is to store $\frac{m^2}{n^2}$ pixels in each PE. This adds to the storage requirements and complexity of the PEs; they must be capable of handling the largest image that will be processed on the machine.

Another problem with electronic full-array CLPs is caused by pin-out limitations of LSI and VLSI chips. If a large array is partitioned into chips with k x k PEs on each chip, then O(k) pins are needed for interconnections between PEs, if the number of I/O lines to each PE is a constant independent of n. If it instead grows with n, as in the case of a hypercube, for example, the number of pins required grows faster than O(k). Finally, to avoid the bottleneck of transferring images into and out of the PEs, as described above, k^2 pins would be needed. We should point out that while these problems appear to be largely inherent in the technology, it does not necessarily prevent future clever solutions from reducing their severity. At times they can be lived with or partially avoided to a substantial degree. A case in point is the MPP [20].

An optical CLP has the potential of bypassing most of these problems. They all amount to communications limitations, either between chips or between processors and memory. An optical full-array CLP could have direct connections between each bit in memory and the PE(s) that correspond to that bit. A gate array that could store on the order of 512 x 512 bits could have 512^2 or 262,144 lines (each with a fanout of 5 for the case of a *4-connected cellular array*), each to the appropriate PEs. Using multiple chips may permit these numbers to be even larger.

In the case of electronic sub-array machines, similar limitations exist. Processing speed is limited primarily by the data rate of the bus between PEs and memory. In addition, *if pixel operations are done by look-up table*, this can put an added load on the bus or on the required storage within the PEs. These points are discussed in [19]. In the optics case, data can again be transferred quickly and in parallel between memory and PEs, so that the data rate of the transfer is not a significant part of the processing time.

Another possible advantage of optical CLPs is in the PE to PE interconnection network topology. Each PE can have a larger number of input and output lines (although additional gates are needed to in each PE to select among the lines). In addition, longer interconnections between PEs are feasible. For example, the hybrid interconnection system of *Sec. II* cannot distinguish between global and local interconnections. In the other interconnection systems of this section, global between-PE interconnections do increase the complexity of the hologram or optics somewhat, but may still be feasible. Such between-PE interconnections can substantially reduce the communication time between PEs. For example, in a simple nearest-neighbor mesh-connected array, it takes $O(n)$ time for data to be transfered between PEs on opposite sides of the array. Going to a hypercube, which connects each PE to PEs at distances of $1,2,4,...,2^k$ in each dimension, lowers this communication time to $O(\log_2 n)$; the number of lines connected to each PE in this case is also $O(\log_2 n)$ [21].

Reference 22 classifies different types of communications between PEs and gives the corresponding communication time on different network topologies. Some classes depend on the diameter of the graph representation of the network. Examples include broadcasting, where one PE sends a message to many PEs, and condensing, where many PEs send messages to one PE, such that the messages can be combined en route to the destination. In both cases a globally connected array such as a hypercube can perform the communication in $O(\log n)$ steps, whereas a conventional mesh requires $O(n)$ steps. Another class of communication operations is one-to-one tasks, or permutations. Here the topology, but not the diameter, determine the time, but again the hypercube requires $O(\log n)$ time, whereas a mesh may require $O(n^2)$ time.

Such between-PE interconnections can substantially reduce the computation time for some algorithms and processing operations used on images in CLPs. Examples of operations that utilize some of the communication tasks listed above are the calculation of transforms, moments, value counting or histogramming, and region property computation. Full array CLPs can do pointwise and local operations in a small number of steps that is independent of image size. Since the above operations require time $O(\log n)$ or greater, and loading or unloading of image data into PEs typically requires time that grows with n in the electronics case, technologies or architectures that reduce these times could have a significant impact on processing times for many image processing algorithms.

CONCLUSIONS

In conclusion, we have discussed possible optical systems for the implementation of digital cellular logic processors. Current optical technology in conjunction with anticipated progress in research may make the construction of such a processor feasible. Looking at the underlying physical characteristics of electronic and optical digital processors reveals that the two are quite different, and this has an effect on the algorithms and architectures that can be implemented with each. In the case of cellular logic processors the communication and interconnection capabilites of optics could provide for substantially reduced computation time for image processing tasks.

ACKNOWLEDGEMENTS

The authors would like to thank T. C. Strand, P. Chavel, R. Forchheimer, and B. H. Soffer for their contributions to this research. This work was supported by the Air Force Office of Scientific Research under grant AFOSR-84-0181.

REFERENCES

1 A.A. Sawchuk and T.C. Strand, "Digital Optical Computing", *Proc. IEEE*, Vol. 72, p. 758, 1984

2. J.L. Jewell, M.C. Rushford, and H.M. Gibbs, "Use of a Single Nonlinear Fabry-Perot Etalon as Optical Logic Gates", *Appl. Phys. Lett.*, Vol. 44, p. 172, 1984; J.L. Jewell, M.C. Rushford, H.M. Gibbs, and N. Peyghambarian, "Single-Etalon Optical Logic Gates", in *Technical Digest Conference on Lasers and Electrooptics*, Optical Society of Amera, Washington, D.C., paper THg2, 1984.

3. R.L. Fork, "Physics of Optical Switching", *Phys. Rev. A*, Vol. 26, p. 2049, 1982.

4 P.W. Smith and W.J. Tomlinson, "Bistable Optical Devices Promise Subpicosecond Switching", *IEEE Spectrum*, Vol. 18, no. 6, pp. 26-33, June 1981.

5. H.M. Gibbs, S.L. McCall, and T.N.C. Venkatesan, "Optical Bistable Devices: The Basic Components of All-Optical Systems?", *Optical Engineering*, Vol. 19, p. 463, 1980.

6 D.A.B. Miller, "Bistable Optical Devices: Physics and Operating Charactristics", *Laser Focus*, Vol. 18, no. 4, p. 79, 1982.

7. B.K. Jenkins et al., "Architectural Implications of a Digital Optical Processor", *Appl. Opt.*, Vol. 23, no. 19, pp. 3465-3474

8 B.K. Jenkins, "Architectural Characteristics of Optical Logic Systems", *Proc. IEEE Computer Conference*, IEEE Cat. No. 85CH2135-2, pp. 336-341, 1985.

9. B.K. Jenkins and T.C. Strand, "Computer Generated Holograms for Space-Variant Interconnections in Optical Logic Systems", *Proc. Int. Conf. on Computer Generated Holograhy*, Sing H. Lee, ed., *Proc. SPIE*, Vol. 437, pp. 110-118, 1983.

10. B.K. Jenkins, et al., Sequential Optical Logic Implementation", *Appl. Opt.*, Vol. 23, no. 19, pp. 3455-3464, 1984.

11. C.A. Mead and L.A. Conway, (eds.), *Introduction to VLSI Systems*, Addison-Wesley, Reading, Ma., Ch. 8, 1980.

12. L.S. Haynes, R.L. Lau, D.P. Siewiorek, and D.W. Mizell, "A Survey of Highly Parallel Computing", *Computer*, Vol. 15, no. 1, pp. 9-24, January 1982.

13. J.W. Tompkins and S. Hollock, "The Impact of Multi-Level Metalisation on Semi-Custon LSI", *Proc. 1982 Custom Integrated Circuits Conf.*, pp. 276-280, 1982.

14. C.L. Seitz, "Systems Timing", Chp. 7, in Ref. 11.

15. H.T. Kung, "Why Systolic Architectures", *Computer*. Vol. 15, no. 1, pp. 37-46, January 1982.

16. C.L. Seitz, "Ensemble Architectures for VLSO - A Survey and Taxonomy", *Proc. 1982 Conf. on Adv. Research in VLSI*, M.I.T., pp. 130-135, 1982.

17. H.T. Kung and C.E. Leiserson, "Algorithms for VLSI Processor Arrays", in Ref. 11.

18. S.Y. Kung, Private Communication.

19. K. Preston, Jr., "Cellular Logic Computers for Pattern Recognition," *Computer*, Vol. 16, no. 1, pp. 36-47, 1983.

20. K.E. Batcher, "Architecture of the MPP," *Proc. Computer Architecture for Pattern Analysis and Image Database Management*, IEEE Cat. No. 83CH1929-9, pp. 170-174, 1983.

21. A. Rosenfeld, "Parallel Image Processing Using Cellular Arrays," *Computer*, Vol. 16, No. 1, pp. 14-20, 1983.

22. T.R. Kushner and A. Rosenfeld, "Interprocessor Communication Requirements for Parallel Image Processing," *Proc. Computer Architecture for Pattern Analysis and Image Database Management*, IEEE Cat No 83CH1929-9, pp. 177-183, 1983.

2.3 Binary Image Algebra

Binary image algebra (BIA) forms the mathematical background for software and hardware systems suitable for optical digital computing. Parallel algorithms for optical cellular logic and symbolic substitution processors can be formalized as compact BIA expressions. BIA also leads to the architectural design of digital optical cellular image processors (DOCIP) which are well-suited to executing the parallel algorithms. The following three papers summarize our recent work in these areas.

The DOCIP is a 2-D, page oriented array of individual processors located at every pixel of a large image. The attached papers by K.S. Huang, B.K. Jenkins and A.A. Sawchuk, "Binary Image Algebra and Optical Cellular Logic Processor Design", from *Computer Vision, Graphics, and Image Processing,* Vol. 45, pp. 295-345, 1989; "Image Algebra Representation of Parallel Optical Binary Arithmetic", from *Applied Optics,* Vol. 28, No. 6, pp. 1263-1278, March 15, 1989; "A Cellular Hypercube Architecture for Image Processing", from *Applications of Digital Image Processing X*, Proc. of SPIE-The International Society for Optical Engineering, Vol. 829, San Diego, California, August, 1987, summarizes these concepts and their algebraic background.

COMPUTER VISION, GRAPHICS, AND IMAGE PROCESSING 45, 295-345 (1989)

Binary Image Algebra and Optical Cellular Logic Processor Design*

K. S. HUANG,[†] B. K. JENKINS, AND A. A. SAWCHUK

Signal and Image Processing Institute, Department of Electrical Engineering, University of Southern California, Los Angeles, California 90089-0272

Received August 14, 1987; accepted August 21, 1988

Techniques for digital optical cellular image processing are presented. A binary image algebra (BIA), built from five elementary images and three fundamental operations, serves as its software and leads to a formal parallel language approach to the design of parallel binary image processing algorithms. Its applications and relationships with other computing theories demonstrate that BIA is a powerful systematic tool for formalizing and analyzing parallel algorithms. Digital optical cellular image processors (DOCIPs), based on cellular automata and cellular logic architectures, serve as its hardware and implement parallel binary image processing tasks efficiently. An algebraic structure provides a link between the algorithms of BIA and architectures of DOCIP. Optical computing suggests an efficient and high-speed implementation of the DOCIP architectures because of its inherent parallelism and 3D global free interconnection capabilities. Finally, the instruction set and the programming of the DOCIPs are illustrated. © 1989 Academic Press, Inc

1. INTRODUCTION

In this paper we combine studies of architectures, algorithms, mathematical structures, and optics to show that: (1) an image algebra extending from mathematical morphology [2]-[5] can lead to a formal parallel language approach to the design of image processing algorithms; (2) cellular automata are appropriate models for parallel image processing machines [6, 7]; (3) an algebraic structure serves as a framework for both algorithms and architectures of parallel image processing; and (4) the parallel processing and global interconnection advantages of optical computing may be useful in efficiently implementing image algebra with cellular logic architectures.

The purpose of the image algebra approach in this paper is for the development of a programming language for a specific parallel architecture, namely a digital optical cellular image processor (DOCIP). The binary image algebra (BIA) described here is based on a set of three specific fundamental operations. These fundamental operations are the key operations in the instruction set of the DOCIP machine. The BIA provides a decomposition of general operations, including low-level image processing operations, into the three fundamental operations of the instruction set. This decomposition is inherently parallel and provides a direct mapping to the machine architecture.

*Supported by the Air Force Office of Scientific Research under Grant AFOSR-84-0181 and by an IBM graduate fellowship. Portions of this paper presented at the 1987 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence [1].

[†]Current Address: Computer Science Department, IBM, T.J. Watson Research Center, Yorktown Heights, New York 10598.

In this section, we first review previous work on image algebra, cellular automata, and cellular logic architectures, then we define the algebraic structure and outline the detailed discussion that follows.

Previous Work on Image Algebra

During the past few years, numerous papers have used an algebraic approach to aid in image processing [2–5, 8–10]. Among them, morphological image algebra has the closest relation to binary image algebra (BIA). Many papers describe either specific theoretical aspects of mathematical morphology or application-specific morphological algorithms [11–18]. The applications of mathematical morphology have been fruitful. In this paper we adapt it to provide the following features:

1. A simplified mathematical structure. Mathematical morphology comprises two branches, integral geometry and geometrical probability, plus a few collateral ancestors (harmonic analysis, stochastic processes, algebraic topology) [2]. The mathematical details and formal proofs in morphology are often intricate and involve advanced set theoretic and topological concepts which are not always necessary for engineering applications.

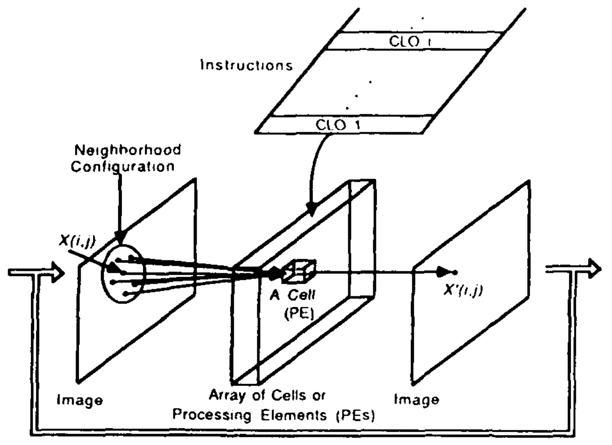
2. A complete algebraic theory. Mathematical morphology defines some algebraic operators and utilizes some algebra. With our adaptation, we would like to answer the following questions:

   - What is the algebraic definition of this mathematical morphology?
   - How powerful is this mathematical morphology?
   - What is the definition of a transformation? Morphological transformations are constrained by four principles [2]; here we introduce a complete definition of image transformations.

3. Clarification of its relationship to other areas. We define its relationship to linear system theory, image processing, and common computing techniques including boolean logic, cellular logic, and algebraic structures.

Here we develop a simple unified complete parallel binary image processing theory based on an algebraic structure—binary image algebra (BIA). In BIA, parallel binary image processing algorithms (including parallel numerical computations) can be written as compact algebraic expressions where an algebraic symbol represents an image (not a pixel) or an image operation (not a pixel-wise operation). A complete algebraic system comprises three fundamental operations and five elementary images which can be combined to generate any image in the three fundamental operations for forming any image transformation. (In fact, one can define four elementary images and two fundamental operations that are sufficient; however, in this paper we will not consider them since they are more difficult to use.)

There are other image algebras, each with its own characteristics [8, 9]. Because of our intended application to a highly parallel computing machine with simple processing elements and a reduced instruction set, we utilize a BIA with only three fundamental operations that can implement any binary image transformation. For example, the counting function, which gives the number of pixels having a certain level, is considered a mapping from a picture type of operand to a number type of



CLO i
Instructions
CLO 1
Neighborhood Configuration
X(i,j)
A Cell (PE)
X'(i,j)
Image
Array of Cells or Processing Elements (PEs)
Image

FIG. 1. A sequential process of cellular logic operations (CLO). The value $X'(i, j)$ is determined by the corresponding $X(i, j)$ in the original image along with the values of its neighbors.

operand in references [8] and [9]; in BIA numbers are also represented as images [19]. BIA suggests several simple but fast parallel image algorithms and a parallel image processing architecture with a very low cell complexity.

Previous Work on Cellular Logic Architectures

To match BIA parallel algorithms by cellular logic architectures in a transparent way, we characterize a cellular automaton by algebraic structure as BIA does. The cellular logic computer was first inspired by the writings of von Neumann [20, 21] on cellular automata. A sequential process of cellular logic operations is described in Fig. 1. Some review of cellular image processors can be found in Refs. [21–25]. Many cellular computers have been constructed previously for implementing cellular logic operations, and some ideas for extending the nearest-neighbor connected cellular logic computers for improving speed and flexibility have been proposed [24]. These architectures include: (1) the cellular string (Fig. 2(a)); (2) the cellular array (Fig. 2(b)); and (3) the cellular hypercube (Fig. 2(c)); and the cellular pyramid (Fig. 2(d)). These three architectures share a common feature in the simplicity and regularity of interconnecting simple processing elements and represent an interconnection topology in 1D, 2D, and 3D, respectively. The 3D case is difficult to implement on a planar VLSI chip [24, 26, 27], but may be realizable by a digital optical system [28, 29]. Two promising architectures based on digital optical cellular image processors (DOCIPs), DOCIP-array and DOCIP-hypercube, are presented below as a means of implementing BIA effectively.

Definition of Algebraic Structure

An algebraic structure (or algebra) [30–32] is a pair (or system) $A = (S, F)$, where

- S is a set, and

- F is a family of operations which are functions,

$$f: S^k \to S,$$

and k is a finite nonnegative integer.

Remark. For any finite nonnegative integer k, we define a k-ary operation on S as an operation which is a function f: $S^k \to S$. Thus, a unary (or 1-ary) operation on S is simply a function on S to S. A binary (or 2-ary) operation on S is a function on S^2 to S. For completeness, we define a nullary (or 0-ary) operation on S to be a particular element of S.

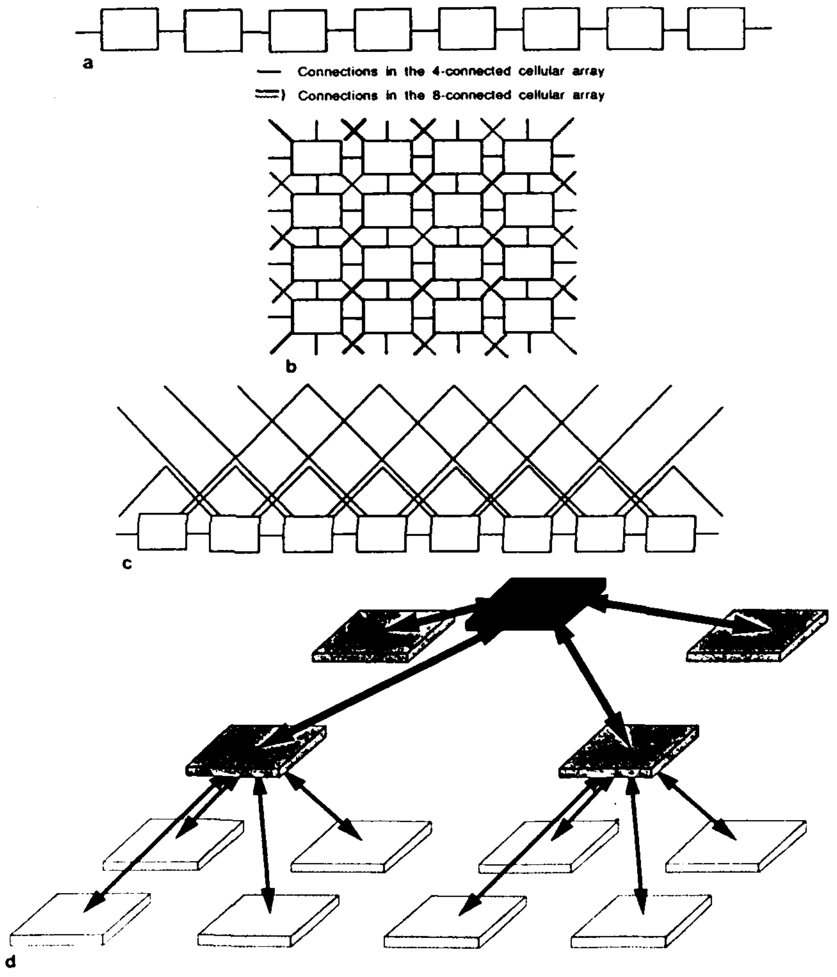

a
Connections in the 4-connected cellular array
Connections in the 8-connected cellular array
b
c
d

FIG. 2. (a) A cellular string. It requires only a 1D interconnection geometry. Each cell only connects with its two nearest cells. (b) A cellular array. It requires a 2D interconnection geometry. Each cell connects with its 4- or 8-nearest cells. (c) A 1-dimensional cellular hypercube [24]. Each cell connects with cells at distances $1, 2, 4, 8, \ldots, 2^k$ from it. Here, only the connections with distances 1, 2, and 4 are shown. (d) A 2-dimensional cellular pyramid. It consists of stages of arrays with connections between two adjacent stages and is most efficiently implemented with a 3D interconnection geometry.

Therefore, the problem to be solved is essentially to find an "appropriate" algebraic structure ((S, F)) for parallel binary image processing, i.e., to search for S and F, and its "efficient" hardware implementation.

Outline

Section 2 contains the framework of BIA: Subsection 2.1 gives the basic definitions; Subsection 2.2 presents two fundamental principles which prove the completeness of BIA.

Section 3 describes some applications of BIA: Subsection 3.1 reviews basic properties of images and image transformations and derives from them some standard image operations; Subsection 3.2 gives some examples of special cases; Subsection 3.3 gives some useful theorems and examples for low level vision operations, including morphological filtering, shape recognition, "salt" and "pepper" noise removal, size, and location verification.

Section 4 discusses the relationship of BIA and other computing theories: Subsection 4.1 describes the relationship with boolean logic; Subsection 4.2 describes the relationship with symbolic substitution and cellular logic; Subsection 4.3 describes the relationship with linear shift invariant system theory, convolution, and correlation; subsection 4.4 describes some standard algebraic structures supported by BIA.

Section 5 presents the implementation of BIA on digital optical cellular image processors (DOCIPs): Subsection 5.1 gives the algebraic description of the DOCIPs which have the same algebraic structure as BIA; Subsection 5.2 gives the general description of the DOCIPs.

Finally, the programming of the DOCIPs is illustrated in Section 6.

2. BINARY IMAGE ALGEBRA (BIA) FUNDAMENTALS

The overall philosophy of BIA is:

- *An image, but not a pixel, is an object.* For parallel languages and machines for image processing, images can be considered as primitive variables for simplifying the design.

- *Complex image processing operations can be reduced to simple instructions.* Although image processing operations appear complex, the fundamental interactions and the elementary components in a system are very simple.

Thus, BIA begins by:

1. Defining the universal image as the working space for images and their image transformations.
2. Defining elementary images which can be combined to generate any image.
3. Defining fundamental operations which can be cascaded to form complex operations.
4. Defining an image processing/analysis algorithm design as the choice of "good" (or "appropriate") reference images and transformations.

A reference image can be any image and is a generalization of structuring elements in mathematical morphology [2]. Reference images contain some predefined image property (or information); image transformations (or operations) are used for measuring the image property from an input image. Image description, image information extraction, or image property measurement is done by using reference images to model or transform the original image to a final state which reveals the desired information or is used to detect the desired properties easily.

Here we give the algebraic structure of BIA first, and then we provide definitions and present two fundamental principles which allow us to generate any reference image and to implement any image transformation. Ideally, BIA may be further generalized to GIA (general image algebra) which deals with grey-level and complex-valued images.

2.1. Definitions

Definition of Binary Image Algebra (BIA). Binary image algebra is an algebra with an image space S, which is the power set of a predefined universal image $P(W)$, and a family F of operations including three fundamental operations $(\oplus, \cup, \bar{\ })$, which are non 0-ary operations, and five elementary images $(I, A, A^{-1}, B, B^{-1})$, which are 0-ary operations. Symbolically,

$$\mathrm{BIA} = \left(P(W); \oplus, \cup, \bar{\ }, I, A, A^{-1}, B, B^{-1}\right),$$

i.e., $S = P(W)$ and $F = (\oplus, \cup, \bar{\ }, I, A, A^{-1}, B, B^{-1})$. The image space S and the family F of operations will be derived in the following.

Basic Definitions

In general, a binary digital image is defined as a function f that maps each spatially sampled grid point (x, y) of the picture on an orthogonal coordinate system onto the set composed of two elements: 1 (i.e., white, foreground point or image point) and 0 (i.e., black or background point). However, it will be more convenient for our algebra, if we use a set of the coordinates of image points (1's) to specify an image. In this paper, an image is treated as the set of coordinates of image points (i.e., foreground points or pixels that have value 1). We begin the description of BIA by defining our artificial universe:

Definition 2.1 (*universal image*). The universal image is the set $W = \{(x, y) | x \in Z_n, y \in Z_n\}$, where $Z_n = \{0, \pm 1, \pm 2, \ldots, \pm n\}$ and n is a positive integer (Fig. 3).

Remark. "$\in$" means "belongs to." Notice that given n, the universal image defines the domain of our images. In fact, for an image with size larger than $(2n + 1) \times (2n + 1)$ (the size of the universal image), we need to increase the size of the universal image or decompose the tested image into subimages whose sizes are smaller than the size of the universal image. For the reason of simple practice, we only consider the square tessellation of images. To deal with nonsquare (e.g. hexagonal) tessellations, we can simply replace the universal image to be the set of grid points corresponding to the new tessellation pattern.

Definition 2.2 (*image space*). The image space is the power set (the set of all subsets) of the universal image, i.e., $S = P(W)$.

Definition 2.3 (*image*). A set X is an image if and only if X is an element of the image space S, i.e., X is a subimage (subset) of the universal image W.

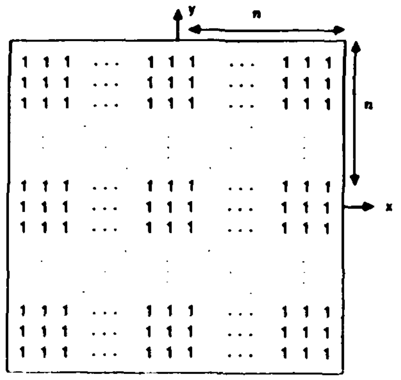


FIG. 3. The universal image W. It has $(2n + 1) \times (2n + 1)$ image points and n is a positive integer.

Symbolically,

$$X \text{ is an image} \leftrightarrow X \in S \leftrightarrow X \subset W.$$

Remark. "$\subset$" means "is included in." There exist $2^{(2n+1)\times(2n+1)}$ different images. Three terms related to images are defined:

1. Size (or area) of an image X, denoted as $\#(X)$, is the cardinality (i.e., the number of elements) of the image X.

2. Foreground of an image X, simply denoted as X, is referred to those pixels with value 1.

3. Background of an image X, denoted as the complement $\overline{X}$ (Definition 2.6), is referred to those pixels with value 0.

Once we know the foreground of an image, the background of this image is well defined (since the universal image is given first). Thus, the foreground is sufficient to specify an image.

DEFINITION 2.4 (*image point* (*foreground point*)). A point (x, y) is an image point of an image X if and only if (x, y) is an element of the set X.

Remark. The largest image is the universal image W and consists of $(2n + 1) \times (2n + 1)$ image points, i.e., $\#(W) = (2n + 1) \times (2n + 1)$; the smallest image is the null image ϕ (defined as the complement $\phi = \overline{W}$) and has no image points, i.e., $\#(\phi) = 0$.

DEFINITION 2.5 (*image transformation*). A transformation T is an image transformation if and only if T is a function mapping from the image space S to the image space S.

Remark. There exist $(2^{(2n+1)\times(2n+1)})^{(2^{(2n+1)\times(2n+1)})}$ image transformations.

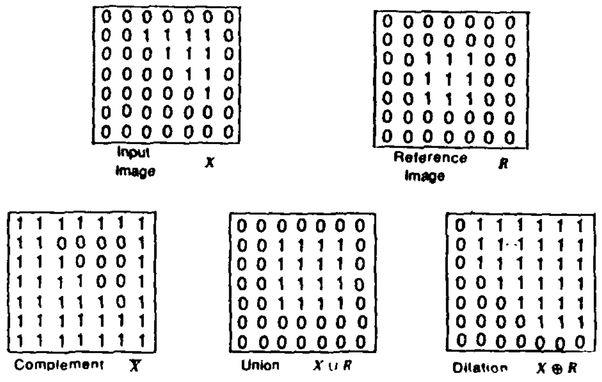


FIG. 4. An example of fundamental operations: complement $^{-}$, union $\cup$, and dilation $\oplus$.

DEFINITION 2.6 (*three fundamental operations*). There are three fundamental operations (Fig. 4):

1. Complement of an image X:

$$\bar{X} = \{(x, y) | (x, y) \in W \wedge (x, y) \notin X\}.$$

2. Union of two images X and R:

$$X \cup R = \{(x, y) | (x, y) \in X \vee (x, y) \in R\}.$$

3. Dilation of two images X and R:

$$X \oplus R = \begin{cases} \{(x_1 + x_2, y_1 + y_2) \in W | (x_1, y_1) \in X, (x_2, y_2) \in R\}, & \\ \qquad (X \neq \varnothing) \wedge (R \neq \varnothing), \\ \varnothing, \quad \text{otherwise}. \end{cases}$$

Remark. "$\wedge$" means "and," and "$\vee$" means "or." Note that X usually represents an input or data image and R is a reference image. The consideration of null image in the dilation operation is missing in mathematical morphology (where the dilation is defined as the union of all translations of X by all image points in R); with this generalization we have a complete theory which is not found in other image algebras because there is less demonstration of their capabilities for implementing any image transformation. We can also define other image operations as fundamental operations instead of these three operations. The reason for choosing these three operations is because of their simplicity, and resulting simple software design and hardware implementation. As shown later, these three operations may be implemented by a 2D optical gate array with 3D interconnections.

DEFINITION 2.7 (*elementary images*). These elementary images are constant images, i.e., 0-ary operations. Each elementary image has only one image point.

There are five elementary images:

1. $I = \{(0,0)\}$—consisting of an image point at the origin
2. $A = \{1,0)\}$—consisting of an image point right of the origin
3. $A^{-1} = \{(-1,0)\}$—consisting of an image point left of the origin
4. $B = \{(0,1)\}$—consisting of an image point above the origin
5. $B^{-1} = \{(0,-1)\}$—consisting of an image point below the origin.

Remark. In fact, these five elementary images could be reduced to four elementary images, because $I = A^0 \equiv A \oplus A^{-1} = B^0 \equiv B \oplus B^{-1}$.

Definition 2.8 (*reflected reference image*). Given a reference image R which is a predefined image for containing some desired image property or image information, its reflected image is defined as

$$\check{R} = \{(-x, -y) | (x, y) \in R\}.$$

Remark. In many useful cases the reference image R is symmetric, then $\check{R} = R$.

2.2. *Two Fundamental Principles*

Two fundamental principles basically define the binary image algebra (BIA). Before stating these two principles, we give some preliminary results.

Lemma 2.1.

$$\overline{(X \oplus \bar{R}) \cup (\bar{X} \oplus R) \cup \bar{I}} = \begin{cases} I & \text{if } X = \check{R} \\ \varnothing & \text{otherwise} \end{cases}$$

$\forall\ X,\ R \in P(W)$, where $I = \{(0,0)\}$ is an elementary image, $\check{R}$ is the reflected reference image of R, and "$\forall$" means "for all."

Proof. Appendix A.

Remark. This lemma says that if the image X matches the image $\check{R}$, then the origin (central pixel) of the above output image has value "1," otherwise it is always "0."

Theorem 2.1. *Any image transformation T: $P(W) \to P(W)$ can be expressed as*

$$T(X) = \bigcup_{i=1}^{k} \left\{ \overline{(\bar{X} \oplus R_i) \cup (X \oplus \bar{R}_i) \cup \bar{I}} \oplus Q_i \right\},$$

where $k \le \#(P(W))$, R_i and Q_i are the reference images used to form any desired image transformation, and

$$\bigcup_{i=1}^{k} R_i \equiv R_1 \cup R_2 \cup \cdots \cup R_k.$$

Proof. Appendix B.

THEOREM 2.2. *Any image can be represented as*

$$X = \bigcup_{(i,j)\in X} A^iB^j,$$

where $A^iB^j \equiv A^i \oplus B^j$,

$$A^i \equiv \underbrace{A \oplus A \oplus \cdots \oplus A}_{i} = \{(i,0)\} \qquad \text{if } i > 0,$$

$$A^i \equiv \underbrace{A^{-1} \oplus A^{-1} \oplus \cdots \oplus A^{-1}}_{-i} = \{(i,0)\} \qquad \text{if } i < 0,$$

and A, B, A^{-1}, B^{-1} *are the elementary images defined in Definition* 2.7.

Proof. Appendix C.

PRINCIPLE 1 (fundamental principle of image transformations). *Any image transformation* T *can be implemented by using appropriate reference images* R *and the three fundamental operations*: (1) *complement* $\bar{X}$ *of an image* X, (2) *union* $\cup$ *of two images*, (3) *dilation* $\oplus$ *of two images*.

Proof. It follows from Theorem 2.1.

In order to use Principle 1 efficiently in practice, we invoke Principle 2 for the generation of reference images.

PRINCIPLE 2 (fundamental principle of reference images). *Any reference image* R *can be generated from elementary images* $(I, A, A^{-1}, B, B^{-1})$ *by using the three fundamental operations.*

Proof. It follows from Theorem 2.2.

Therefore, by the above principles, we can represent BIA as:

$$\text{BIA} = \left(P(W); \oplus, \cup, {}^{-}, I, A, A^{-1}, B, B^{-1}\right).$$

3. DEVELOPMENT OF BINARY IMAGE ALGEBRA (BIA)

BIA can have many applications in character recognition, industrial inspection, medical image processing, and scientific computation. In this section we first review the basic properties of images and image transformations, define 11 standard operations, and give some special cases of dilation [2-5, 33-36]. Then we summarize four theorems and some examples for binary image processing.

This section is primarily a survey of binary image processing algorithms with implementation using BIA fundamental operations. These fundamental operations are so chosen because they form an efficient basis for the instruction set of an optically based cellular image processor. This survey serves as a description of a parallel language for controlling the processor and how it is compiled into low level instructions. The use of BIA for parallel numerical computation is described in [19].

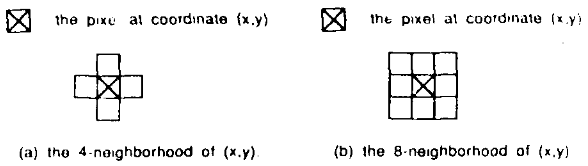

the pixel at coordinate (x,y)
the pixel at coordinate (x,y)
(a) the 4-neighborhood of (x,y)
(b) the 8-neighborhood of (x,y)

FIG. 5. The 4-neighborhood and 8-neighborhood of an image point (x, y).

3.1. *Basic Properties of Images and Image Transformations*

DEFINITION 3.1 (*connectivity in images*). 1. 4-neighbor and 8-neighbor: An image point (x, y) in an image X can have two types of neighbors:

(a) An image point (i, j) is a 4-neighbor of $(x, y) \leftrightarrow (i, j) \in \{(x \pm 1, y), (x, y \pm 1)\}$.
Remark. $\{(x, y), (x \pm 1, y), (x, y \pm 1)\}$ is called the 4-neighborhood of (x, y) and $N_4 \equiv \{(0,0), (0, \pm 1), (\pm 1, 0)\} = I \cup A \cup A^{-1} \cup B \cup B^{-1}$ (Fig 5(a)).

(b) An image point (i, j) is a 8-neighbor of $(x, y) \leftrightarrow (i, j) \in \{(x \pm 1, y), (x, y \pm 1), (x \pm 1, y \pm 1)\}$.
Remark. $\{(x, y), (x \pm 1, y), (x, y \pm 1), (x \pm 1, y \pm 1)\}$ is called the 8-neighborhood of (x, y) and $N_8 \equiv \{(0,0), (0, \pm 1), (\pm 1, 0), (\pm 1, \pm 1)\}$ (Fig. 5(b)).

2. 4-connected and 8-connected:

(a) Two image points (x, y) and (i, j) of an image X are 4-connected $\leftrightarrow$ there exists a sequence of image points $(x, y) = (x_0, y_0), (x_1, y_1), \ldots, (x_m, y_m) = (i, j)$, where (x_k, y_k) is a 4-neighbor of (x_{k-1}, y_{k-1}) and $(x_k, y_k) \in X$, $1 \le k \le m$.

(b) Two image points (x, y) and (i, j) of an image X are 8-connected $\leftrightarrow$ there exists a sequence of image points $(x, y) = (x_0, y_0), (x_1, y_1), \ldots, (x_m, y_m) = (i, j)$, where (x_k, y_k) is an 8-neighbor of (x_{k-1}, y_{k-1}) and $(x_k, y_k) \in X$, $1 \le k \le m$.

Remark 1. "4-connected in X" and "8-connected in X" are equivalence relations (reflexive, symmetric, and transitive).

Remark 2. For any image point (x, y) in a nonnull image X, the set of (i, j) such that (x, y) and (i, j) are 4-connected (or 8-connected) is called a 4-connected (or 8-connected) component of X. A 4-connected (or 8-connected) component of X is just an equivalence class in X under the equivalence relation—"4-connected (or 8-connected) in X." Thus, a collection of 4-connected (or 8-connected) components of X forms a partition of X, i.e., the set of all 4-connected (or 8-connected) components $\{X_i\}_{i \in I}$ (where I is the index set of connected components) is a family of nonnull subimages of X and has the following properties:

(a) $X_i \neq \varnothing$ for all $i \in I$.

(b) $X_i \cap X_j = \varnothing$ for all $i \neq j$, $i, j \in I$. ($X_i \cap X_j = \overline{\overline{X_i} \cup \overline{X_j}}$ as defined in Definition 3.3.)

(c) $X = \bigcup_{i \in I} X_i$.

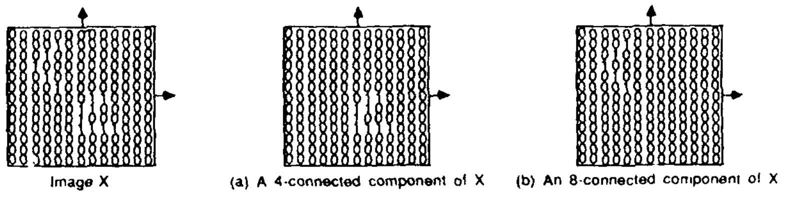


Image X
(a) A 4-connected component of X
(b) An 8-connected component of X

FIG. 6. The 4-connected component and 8-connected component of an image.

Figure 6(a) shows a 4-connected component in an image X and Fig. 6(b) shows an 8-connected component in X.

Remark 3. If an image X has l 4-connected (or 8-connected) components, there are l distinct equivalence classes in X. Each equivalence class X_i can be represented by an image point in X_i. Thus, we may use l distinct image points which belong to l different 4-connected (or 8-connected) components to represent the classes of the image X.

Remark 4. In dealing with connectedness in both X and $\bar{X}$, to avoid the "connectivity paradox" [33], it is preferable to use opposite types of connectedness for X and $\bar{X}$, i.e., if we use "4-connected" for X, then we use "8-connected" for $\bar{X}$ and vice versa.

Remark 5. If any image X is surrounded by a border of 0's, the component of $\bar{X}$ consisting of the points connected to (any one of) these 0's is called the *outside* of X (Fig. 7(a)). If $\bar{X}$ has any other components, they are called *holes* in X (Fig. 7(b)).

For more detailed discussion of geometric properties of images, the reader is referred to [33–35]. For equivalence relations, equivalence classes, and partitions, please refer to [30–32].

DEFINITION 3.2 (*basic properties of image transformations*). The key properties of image transformations are the following ten basic properties:

1. Increasing. An image transformation $T(X)$ is increasing

$$\leftrightarrow (X \subset Y \rightarrow T(X) \subset T(Y)) \qquad \text{for all } X, Y \in P(W).$$

2. Decreasing. An image transformation $T(X)$ is decreasing

$$\leftrightarrow (X \subset Y \rightarrow T(Y) \subset T(X)) \qquad \text{for all } X, Y \in P(W).$$

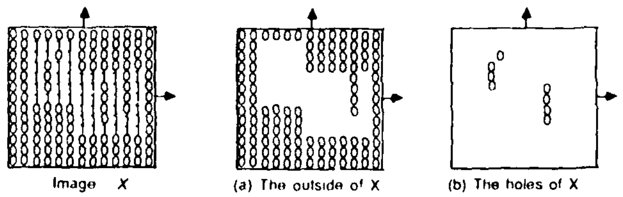


FIG. 7. The outside and holes of an image.

3. Extensive. An image transformation $T(X)$ is extensive

$$\leftrightarrow X \subset T(X) \qquad \text{for all } X \in P(W).$$

4. Antiextensive. An image transformation $T(X)$ is antiextensive

$$\leftrightarrow T(X) \subset X \qquad \text{for all } X \in P(W).$$

5. Idempotent. An image transformation $T(X)$ is idempotent

$$\leftrightarrow T(T(X)) = T(X) \qquad \text{for all } X \in P(W).$$

6. Shift invariant. An image transformation $T(X)$ is shift invariant

$$\leftrightarrow T(X \oplus P) = T(X) \oplus P \text{ for all } X, P \in P(W)$$

and P is a point image which consists of one and only one image point. If an image transformation is not shift invariant, then it is shift variant:

$$T(X \oplus P) \neq T(X) \oplus P \qquad \text{(in general)}.$$

7. Homotopic. An image transformation $T(X)$ is homotopic $\leftrightarrow$ there exists a one-to-one and onto correspondence between the connected components of X and those of $T(X)$, for all $X \in P(W)$. The same is then true for the holes.

8. Commutative. A binary image operation $\cdot$ is commutative

$$\leftrightarrow X \cdot R = R \cdot X \qquad \text{for all } X, R \in P(W).$$

9. Associative. A binary image operation $\cdot$ is associative

$$\leftrightarrow (X \cdot R) \cdot Q = X \cdot (R \cdot Q) \qquad \text{for all } X, R, Q \in P(W).$$

10. Distributive. A binary image operation $\cdot$ is distributive over a binary image operation $+$

$$\leftrightarrow X \cdot (R + Q) = (X \cdot R) + (X \cdot Q) \qquad \text{for all } X, R, Q \in P(W).$$

DEFINITION 3.3 (*standard operations*). Most standard operations can be derived from the three fundamental operations; eleven common ones follow:

1. Difference of X by R (Fig. 8(a)):

$$X/R = \{(x, y) \in X \mid (x, y) \notin R\} = X \cap \bar{R} = \overline{\bar{X} \cup R}.$$

Remark. $\bar{X} = W/X$, where W is the universal image. The difference is an obvious approach to detect defects in the foreground of a tested image.

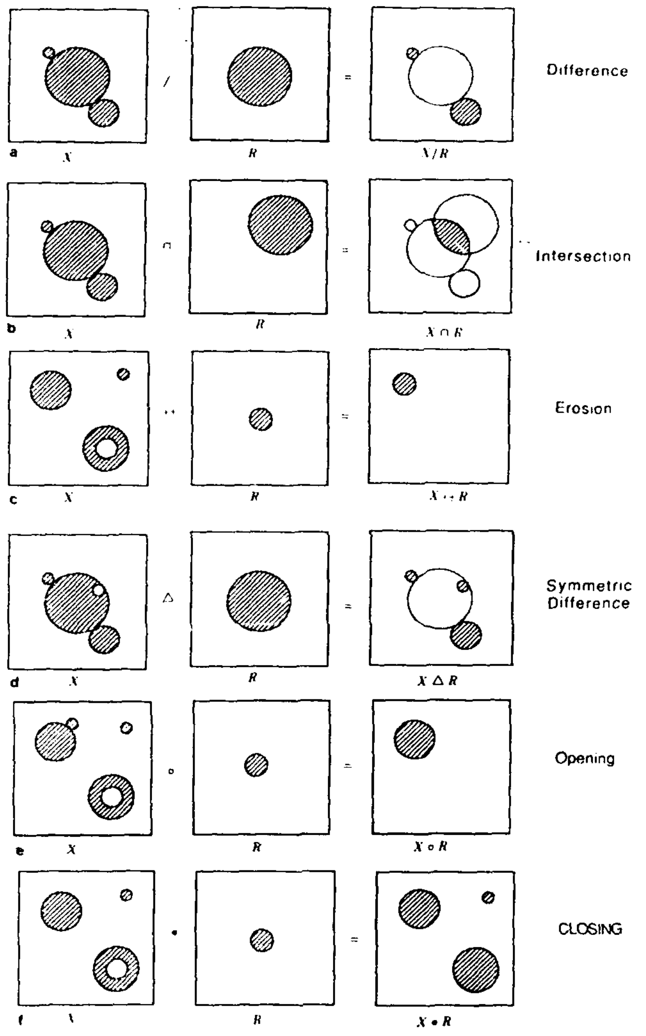


FIG. 8. Eleven standard derived image operations: (a) difference; (b) intersection; (c) erosion; (d) symmetric difference; (e) opening; (f) closing; (g) hit or miss transform (template matching); (h) thinning; (i) thickening; (j) a sequential thinning (used for homotopic skeletonization) [36]; (k) a conditional dilation.

2. Intersection of two images X and R (Fig. 8(b)):

$$X \cap R = \{(x, y) | (x, y) \in X \wedge (x, y) \in R\} = \overline{\overline{X} \cup \overline{R}}.$$

Remark. $X \cup R = \overline{\overline{X} \cap \overline{R}}$. If $X \cap R \neq \varnothing$, then we say that an image x hits (or is joint with) an image R. If $X \cap R = \varnothing$, then we say that an Image X misses (or is disjoint with) an image R.

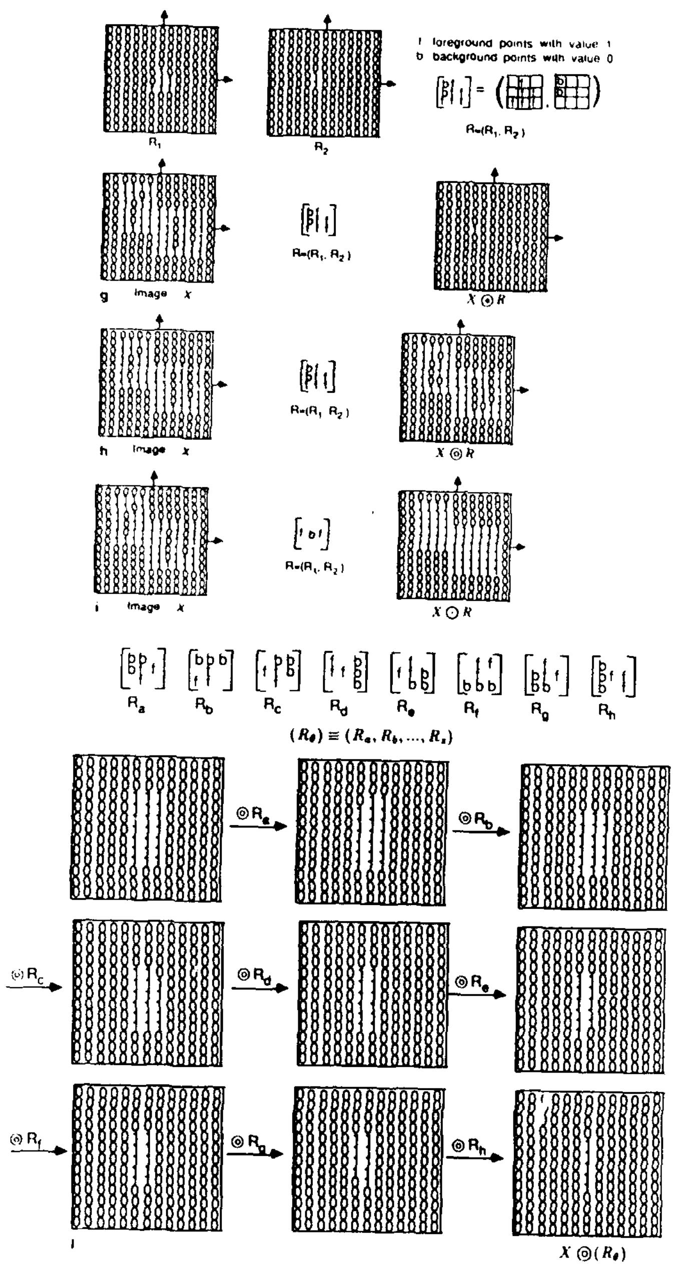


FIGURE 8-*Continued*

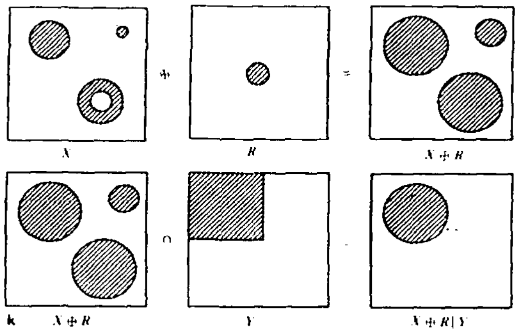


FIGURE 8–*Continued*

3. Erosion of an image X by a reference image R or foreground template matching of X by R (Fig. 8(c)):

$$X \ominus R = \overline{\overline{X} \oplus \check{R}}\,.$$

Remark. $X \oplus R = \overline{\overline{X} \ominus \check{R}}$, and $R = \check{R}$ when R is symmetric. The erosion of an image X by a reference image R can be thought of as the complement of the dilation of the background by the reflection of the reference image R. In general, the erosion of a nonnull image X by a nonnull reference image R can be used to decrease the size of regions, increase the size of holes, eliminate regions, and break bridges in X; on the contrary, the dilation of a nonnull image X by a nonnull reference image R can increase the size of regions, decrease or fill in holes and cavities, and bridge gaps in X. Furthermore, the erosion can be interpreted as a foreground template matching where the foreground points of $X \ominus R$ indicates the occurrences of the foreground template R in X (in this purpose, the size of R usually is much smaller than the size of X).

4. Symmetric difference of two images (mod 2 image addition or subtraction) (Fig. 8(d)):

$$X \cdot R = (X/R) \cup (R/X) = \overline{\overline{X} \cup R} \cup \overline{\overline{R} \cup X}\,.$$

Remark. The symmetric difference is a commutative operation, and its inverse operation can be defined as itself. In Section 4 we show that this operation is the parallel form of boolean EXCLUSIVE-OR. It is an obvious approach to detect defects (including the foreground or background defects) of a tested image.

5. Opening of an image X by a reference image R (Fig. 8(e)):

$$X \circ R = (X \ominus R) \oplus R = \overline{\overline{X} \oplus \check{R}} \oplus R\,.$$

Remark. The opening operation is an erosion followed by a dilaton with the same reference image R. In general, the opening $X \circ R$ with a nonnull reference image R reduces the size of regions and eliminates some image points by removing all features in X which cannot contain the reference image R.

6. Closing of an image X by a reference image R (Fig. 8(f)):

$$X \cdot R = (X \oplus R) \ominus R = \overline{\overline{(X \oplus R)} \oplus \check{R}}\,.$$

Remark. The closing operation is a dilation followed by an erosion with the same reference image R. In general, the closing $X \cdot R$ with a nonnull reference image R increases the size of regions and eliminates some background points by filling in all background areas that cannot contain the reference image R, such as holes and concavities in the image X.

7. Hit or miss transform $\circledast$ of an image X by an image pair $R = (R_1, R_2)$ or template matching of X by R (Fig. 8(g)):

$$X \circledast R = (X \ominus R_1) \cap (\bar{X} \ominus R_2) = \overline{(\bar{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}.$$

Remark. The hit or miss transform of an image X by a reference image pair $R = (R_1, R_2)$ is used to match the shape (or template) defined by the reference image pair R, where R_1 defines the foreground of the shape and R_2 defines the background of the shape. The key conditions are that the foreground X must match R_1 (i.e., $X \ominus R_1$), while simultaneously the background $\bar{X}$ matches R_2 (i.e., $\bar{X} \ominus R_2$). In order to better define the hit or miss transform and its relationship with conventional boolean logic operations, we start from a pixel-wise boolean comparison to derive the hit or miss transform in shape recognition (Theorem 3.2). Note the similarity of the symmetric difference and the hit or miss transform.

8. Thinning $\circledcirc$ an image X by an image pair $R = (R_1, R_2)$ (Fig. 8(h)):

$$X \circledcirc R = X/(X \circledast R) = \overline{\bar{X} \cup \overline{(\bar{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}}.$$

Remark. The thinning operation is antiextensive and decreases the size by removing the central points of the regions which match the reference image pair $R = (R_1, R_2)$.

9. Thickening $\odot$ an image X by an image pair $R = (R_1, R_2)$ (Fig. 8(i)):

$$X \odot R = X \cup (X \circledast R) = X \cup \overline{(\bar{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}.$$

Remark. The thickening operation is extensive and increases the size by filling the image points where the regions match the reference image pair $R = (R_1, R_2)$.

10. Sequential operations (e.g., sequential dilation, sequential erosion, sequential thinning): If an image operation $\cdot$ is successively performed with each reference image (or image pairs) in a sequence $(R_\theta) \equiv (R_a, R_b, \ldots, R_z)$, then we define a sequential image operation

$$X \cdot (R_\theta) = (\cdots((X \cdot R_a) \cdot R_b) \cdots \cdot R_z).$$

Two examples are:

(a) Sequential thinning of an image X by a sequence of image pairs $(R_\theta) \equiv (R_a, R_b, \ldots, R_z)$:

$$X \circledcirc (R_\theta) = (\cdots((X \circledcirc R_a) \circledcirc R_b) \cdots \circledcirc R_z).$$

Remark. The sequential thinning is powerful in many applications, such as constructing a digital homotopic skeleton of an image X. Skeletonization of an image is an operation that transforms the image to a simplified image, called a skeleton, which emphasizes its connectivity. However, a homotopic skeleton cannot be obtained by digitizing an analog skeletonization algorithm; instead, a sequential thinning with a sequence of reference image pairs should be used. Several different algorithms employing different reference image pairs (called masks) have been proposed by several authors [6, 36]. Figure 8(j) shows an example of the skeletonization by a sequential thinning with a sequence of eight reference image pairs proposed by Levialdi *et al.* [36].

(b) Sequential dilation of an image X and a sequence of reference images $(R_\theta) \equiv (R_a, R_b, \ldots, R_z)$:

$$X \oplus (R_\theta) = (\cdots((X \oplus R_a) \oplus R_b) \cdots \oplus R_z).$$

Remark. Since the dilation is commutative and associative, in practice the dilation $X \oplus R$ with a *large* reference image R is usually implemented as a sequential dilation with a sequence of *small* reference images. For example, if $R = E_1 \oplus E_2 \oplus \cdots \oplus E_k$, then

$$X \oplus R = (\cdots((X \oplus E_1) \oplus E_2) \oplus \cdots \oplus E_k);$$

and if $E = E_1 = E_2 = \cdots = E_k$, then

$$R = E^k \equiv \underbrace{E \oplus E \oplus \cdots \oplus E}_{k}.$$

11. Conditional operations (e.g., conditional dilation, conditional erosion, conditional thinning): An image operation $\cdot$ between an image X and a reference image (or image pairs) R performed within a limiting set Y is called a conditional operation and is denoted

$$X \cdot R | Y = (X \cdot R) \cap Y = \overline{\overline{X \cdot R} \cup \overline{Y}}.$$

Remark. Figure 8(k) gives an example of the conditional dilation.

3.2. *Examples of Special Cases: Translation (Shifting), Expansion, Shrinking, and Projection*

Translation (shifting), expansion, shrinking, and projection in a direction can be achieved by the dilation (or erosion) in a direct way.

1. Shifting an image X from coordinate (x, y) to coordinate $(x + i, y + j)$ is done by

$$X \oplus \{(i, j)\} = X \ominus \{(-i, -j)\}.$$

Remark. A point image $\{(i, j)\}$ corresponds to a discrete delta function at $\delta(x - i, y - j)$. Thus, an image function $X(x, y)$ (which corresponds to the image X) convolved with the delta function $\delta(x - i, y - j)$ or correlated with $\delta(x + i, y + j)$ is the same as $X \oplus \{(i, j)\} = X \ominus \{(-i, -j)\}$.

2. Adding a new 8-connected or 4-connected boundary to an image X (i.e., expansion) is done by

$$X \oplus N_4 \qquad \text{or} \qquad X \oplus N_8,$$

where $N_4 \equiv I \cup A \cup A^{-1} \cup B \cup B^{-1}$ and $N_9 \equiv \bigcup_{i,j=-1}^{1} A^iB^j$.

3. Removing the 8-connected or 4-connected boundary of an image X (i.e., shrinking) is done by

$$X \ominus N_4 = \overline{\overline{X} \oplus N_4} \qquad \text{or} \qquad X \ominus N_8 = \overline{\overline{X} \oplus N_8},$$

where $N_4 \equiv I \cup A \cup A^{-1} \cup B \cup B^{-1}$ and $N_8 \equiv \bigcup_{i,j=-1}^{1} A^iB^j$.

4. Projecting an image X to distance k in a direction θ, i.e., producing a shadow of X, where the furthest image point in the shadow in the direction θ is at distance k from the furthest image point in X in the direction θ; this can be achieved by

$$X \oplus \Theta^k,$$

where Θ can be any one of the following:

- East: $E = I \cup A$, $E^k = \bigcup_{i=0}^{k} A^i$
- South: $S = I \cup B^{-1}$, $S^k = \bigcup_{i=0}^{k} B^{-i}$
- West: $W = I \cup A^{-1}$, $W^k = \bigcup_{i=0}^{k} A^{-i}$
- North: $N = I \cup B$, $N^k = \bigcup_{i=0}^{k} B^i$
- Southeast: $S_E = I \cup AB^{-1}$, $S_E^k = \bigcup_{i=0}^{k} A^iB^{-i}$
- Southwest: $S_W = I \cup A^{-1}B^{-1}$, $S_W^k = \bigcup_{i=0}^{k} A^{-i}B^{-i}$
- Northwest: $N_W = I \cup A^{-1}B$, $N_W^k = \bigcup_{i=0}^{k} A^{-i}B^i$
- Northeast: $N_E = I \cup AB$, $N_E^k = \bigcup_{i=0}^{k} A^iB^i$
- Horizontal: $H = \bigcup_{i=-1}^{1} A^i$, $H^k = \bigcup_{i=-k}^{k} A^i$
- Vertical: $V = \bigcup_{i=-1}^{1} B^i$, $V^k = \bigcup_{i=-k}^{k} B^i$
- Left-diagonal: $L_D = \bigcup_{i=-1}^{1} A^{-i}B^i$, $L_D^k = \bigcup_{i=-k}^{k} A^{-i}B^i$
- Right-diagonal: $R_D = \bigcup_{i=-1}^{1} A^iB^i$, $R_D^k = \bigcup_{i=-k}^{k} A^iB^i$.

3.3. Theorems for Low Level Vision

Here we summarize four theorems and some examples for binary image processing applications. Theorem 3.1 gives basic properties of the BIA fundamental

TABLE 1(a)
Basic Properties of Three Fundamental Operations and of Three Derived Operations
(Alternative Fundamental Operations)

Operations / Properties	Complement $\overline{X}$	Union $X \cup R$	Dilation $X \oplus R$	Difference X/R	Intersection $X \cap R$	Erosion $X \ominus R$
Increasing	No	Yes	Yes	Yes	Yes	Yes
Decreasing	Yes	No	No	No	No	No
Extensive	No	Yes	Yes (if $R \supset I$)	No	No	No
Antiextensive	No	No	No	Yes	Yes	Yes (if $R \supset I$)
Idempotent	No	Yes	No	Yes	Yes	No
Shift invariant	No	No	Yes	No	No	Yes
Homotopic	No	No	No	No	No	No
Commutative	No	Yes	Yes	No	Yes	No
Associative	No	Yes	Yes	No	Yes	No
Distributive (with some oper.)	No	Yes (with $\cap$)	Yes (with $\cup$)	No	Yes (with $\cup$, $\triangle$)	No

TABLE 1(b)
Basic Properties of Some Standard Derived Operations

Operations / Properties	Symmetric Difference $X \triangle R$	Opening $X \circ R$	Closing $X \bullet R$	Thinning $X \circledcirc R$	Thickening $X \odot R$	Homotopic skeletonization $X \circledcirc (R_\phi)$
Increasing	No	Yes	Yes	Yes	Yes	No
Decreasing	No	No	No	No	No	No
Extensive	No	No	Yes	No	Yes	No
Antiextensive	No	Yes	No	Yes	No	Yes
Idempotent	No	Yes	Yes	No	No	Yes
Shift invariant	No	Yes	Yes	Yes	Yes	Yes
Homotopic	No	No	No	No	No	Yes
Commutative	Yes	No	No	No	No	No
Associative	Yes	No	No	No	No	No
Distributive (with some oper.)	No	No	No	No	No	No

operations and standard operations. We then describe the implementation of morphological filters, shape recognition algorithms, "salt" and "pepper" noise removal, and size and location verifications. Those more obvious proofs are omitted for brevity.

THEOREM 3.1 (properties of image operations). *The BIA fundamental operations and standard operations have the properties shown in Table* 1(*a*) *and Table* 1(b).

Proof. Appendix D gives some of their mathematical expressions which follow from the definitions.

Examples of morphological filters. Many image transformations are interpreted as morphological filtering [2] or cellular filtering [6]. Some major morphological filters are listed in the following:

1. One kind of morphological low pass filter (Fig. 9(a)): to remove high frequencies in the foreground of an image X can be achieved by opening, i.e.,

$$X \circ R = (X \ominus R) \oplus R = \overline{\overline{X} \oplus \check{R}} \oplus R.$$

2. A second kind of morphological low pass filter (Fig. 9(b)): to remove high frequencies in the background of an image X can be achieved by closing, i.e.,

$$X \cdot R = (X \oplus R) \ominus R = \overline{\overline{(X \oplus R)} \oplus \check{R}}.$$

3. A morphological high pass filter (as shown in Fig. 9(c)) which removes low frequencies in the foreground of an image X can be achieved by the difference of X and its opening, i.e.,

$$X/(X \circ R) = X/((X \ominus R) \oplus R) = X/\left(\overline{\overline{X} \oplus \check{R}} \oplus R\right) = \overline{\overline{X} \cup \left(\overline{\overline{X} \oplus \check{R}} \oplus R\right)}.$$

4. A morphological band pass filter (as shown in Fig. 9(d)) which removes low frequencies and high frequencies in the foreground of an image X can be achieved by the difference of its opening with a smaller reference image R and its opening with a larger reference image Q, where $R \subset Q$, i.e.,

$$\begin{aligned}(X \circ R)/(X \circ Q) &= ((X \ominus R) \oplus R)/((X \ominus Q) \oplus Q)\\ &= \left(\left(\overline{\overline{X} \oplus \check{R}}\right) \oplus R\right)\Big/\left(\left(\overline{\overline{X} \oplus \check{Q}}\right) \oplus Q\right)\\ &= \overline{\left(\overline{\overline{\overline{X} \oplus \check{R}} \oplus R}\right) \cup \left(\overline{\overline{X} \oplus \check{Q}} \oplus Q\right)}.\end{aligned}$$

THEOREM 3.2 (shape recognition (template matching)). 1. *The locations of a shape, that is defined by a nonnull reference image R and a nonnull reference image*

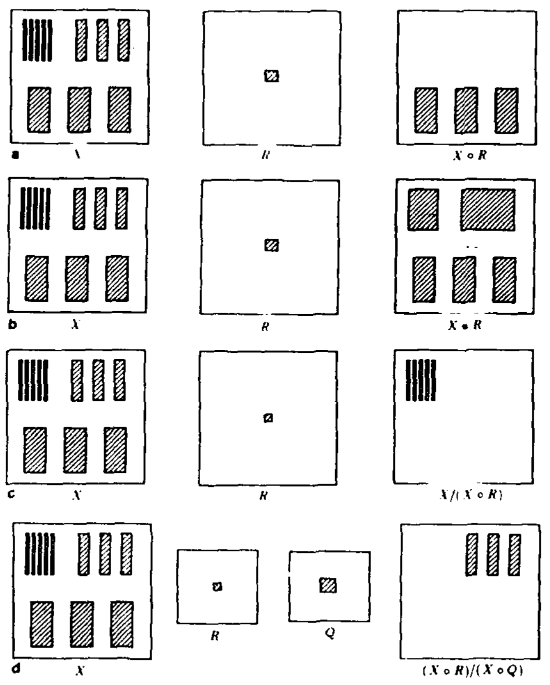


FIG. 9. (a) One kind of morphological low pass filter (opening); (b) a second kind of morphological low pas filter (closing); (c) a morphological high pass filter; (d) a morphological band pass filter.

(called mask) M (Fig. 10(a)), *with $R \subset M \subset W$ (W is the universal image), can be detected by*

$$(X \ominus R) \cap (\bar{X} \ominus (M/R)) = \overline{(\bar{X} \oplus \check{R}) \cup (X \oplus (\check{M}/\check{R}))}$$

$$= \overline{(\bar{X} \oplus \check{R}) \cup \left(X \oplus \overline{\overline{\check{M}} \cup \check{R}}\right)}.$$

Equivalently, setting $R_1 = R$, $R_2 = M/R$, and redefining a nonnull reference image pair $R = (R_1, R_2)$ (Fig. 10(b)) *yields the hit or miss transform of X by R:*

$$X \circledast R = (X \ominus R_1) \cap (\bar{X} \ominus R_2) = \overline{(\bar{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}.$$

2. *The locations of a shape, that is defined by a family of nonnull reference image pairs $\{R(\theta)\}$ with $\theta \in \Theta$ (Θ is the index set of the family of nonnull reference image pairs and $R(\theta) = (R_1(\theta), R_2(\theta))$, can be detected by the union of the hit or miss*

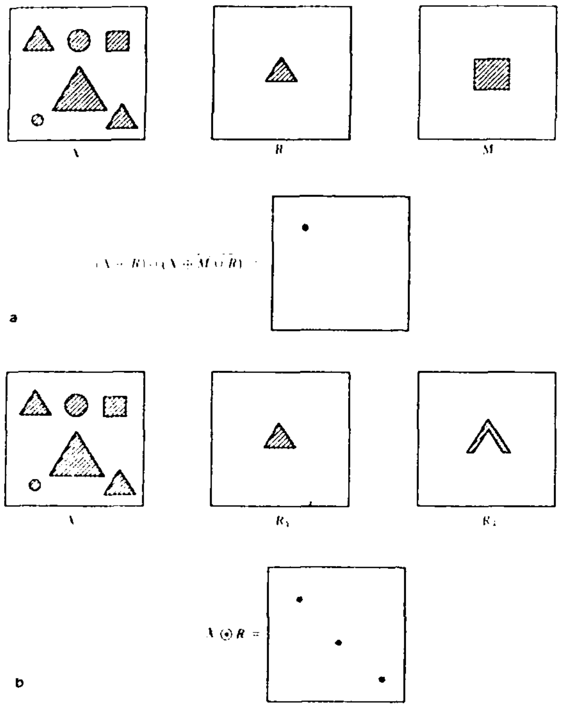


FIG. 10. (a) One kind of shape recognition. R represents the shape to be identified and must lie entirely and exclusively in the mask defined by M. (b) Hit or miss transform which recognizes locations of foreground points given by R_1 in conjunction with background points given by R_2.

transform of X by $R(\theta)$:

$$\bigcup_{\theta\in\Theta} X \circledast R(\theta) = \bigcup_{\theta\in\Theta} (X \ominus R_1(\theta)) \cap (\bar{X} \ominus R_2(\theta))$$
$$= \bigcup_{\theta\in\Theta} \overline{(\bar{X} \oplus \check{R}_1(\theta)) \cup (X \oplus \check{R}_2(\theta))}.$$

Proof. Appendix E.

THEOREM 3.3 ("salt" and "pepper" noise removal). 1. "*Salt*" *noise removal* (*solated image point removal*) (*Fig.* 11(a)): *to remove an image point if its* 4-*connected or* 8-*connected neighbors are background points* (0's) *can be a*

$$X \odot Q_4 = X \cup \overline{\bar{X} \oplus M_4}$$

or

$$X \odot Q_8 = X \cup \overline{\bar{X} \oplus M_8},$$

where $Q_4 = (M_4, I)$, $Q_8 = (M_8, I)$, $M_4 = A \cup A^{-1} \cup B \cup B^{-1} = N_4/I$ *and* $M_8 = N_8/I$.

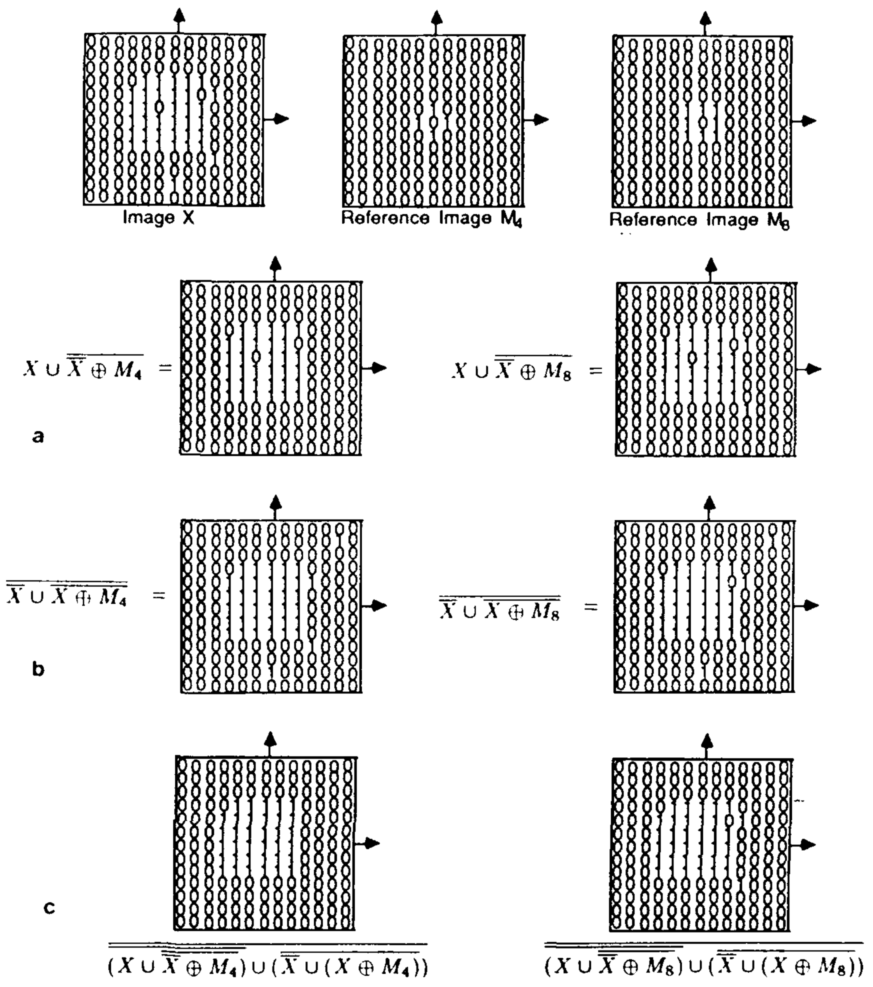


Image X
Reference Image M_4
Reference Image M_8
$X \cup \overline{\overline{X} \oplus M_4}$ =
$X \cup \overline{\overline{X} \oplus M_8}$ =
a
$\overline{\overline{X} \cup \overline{X \oplus M_4}}$ =
$\overline{\overline{X} \cup \overline{X \oplus M_8}}$ =
b
c
$\overline{\overline{(X \cup \overline{\overline{X} \oplus M_4}) \cup (\overline{X} \cup (X \oplus M_4))}}$
$\overline{\overline{(X \cup \overline{\overline{X} \oplus M_8}) \cup (\overline{X} \cup (X \oplus M_8))}}$

FIG. 11. (a) "Salt" noise removal. (b) "Pepper" noise removal. (c) "Salt" and "pepper" noise removal.

2. "*Pepper*" *noise removal* (*interior fill*) (*Fig.* 11(b)): *to create an image point at a coordinate if its* 4-*connected or* 8-*connected neighbors are image points* (1's) *can be achieved by*

$$X \odot R_4 = \overline{\overline{X} \cup \overline{X \oplus M_4}}$$

or

$$X \odot R_8 = \overline{\overline{X} \cup \overline{X \oplus M_8}},$$

where $R_4 = (I, M_4)$, $R_8 = (I, M_8)$.

3. "*Salt and pepper*" *noise removal* (*Fig.* 11(c)): *to remove noise points, that are completely surrounded with* 4-*connected neighbors or* 8-*connected neighbors of the opposite value, can be achieved by*

$$(X \odot Q_4)/(X \circledast R_4) = \overline{\overline{\left(X \cup \overline{\overline{X} \oplus M_4}\right)} \cup \left(\overline{\overline{X} \cup (X \oplus M_4)}\right)}$$

or

$$(X \odot Q_8)/(X \circledast R_8) = \overline{\overline{\left(X \cup \overline{\overline{X} \oplus M_8}\right)} \cup \left(\overline{\overline{X} \cup (X \oplus M_8)}\right)}.$$

Proof. Appendix F.

Remark. This theorem demonstrates the fact that many higher level operations (e.g., involving thinning and thickening) can be efficiently implemented by the three fundamental operations. Using the same design methodology as the "salt and pepper" noise removal, we can design many similar algorithms, such as spur removal, bridge break, and edge detection (perimeter). For example, the detection of the 4-connected or 8-connected edge of an image X (Fig. 12) can be achieved by

$$X/(X \ominus N_8) = \overline{\overline{X} \cup (\overline{X} \oplus N_8)}$$

or

$$X/(X \ominus N_4) = \overline{\overline{X}} \cup \overline{(\overline{X} \oplus N_4)}.$$

THEOREM 3.4 (size and location verification). *The locations in an image X of the regions including the reference image R and included in the reference image Q, where $R \subset Q$, can be detected by*

$$S((X \ominus R)/((X \ominus Q) \oplus Q)) = S\left(\overline{(\overline{X} \oplus \check{R}) \cup \left(\overline{\overline{X} \oplus \check{Q}} \oplus Q\right)}\right),$$

where $S(\cdot)$ means the homotopic skeletonization. (*An example is given in Fig.* 13.)

Proof. Appendix G.

The above theorems serve as the typical rules for morphological image processing. In fact, there are many ways to analyze the shapes and sizes of an image by using only the three fundamental operations. As another example: comparing an image X with its convex hull $C(X)$ [34] is a useful technique to analyze shape. If there is only one object or objects separated by distances greater than their own diameters in the image X, then its convex hull is the intersection of projections (Fig. 14(a)),

$$\bigcap_{i=1}^{4} (X \oplus \Theta_i^k),$$

where Θ_i, $i = 1, 2, 3, 4$, are H, V, R_D, L_D (defined in Definition 3.4), and k should be greater than the longest radius of objects in X. Then the difference of the convex hull and the image $C(X)/X$ indicates how many concavities the image X has and what their individual shapes and sizes are. Figure 14(b) illustrates an example.

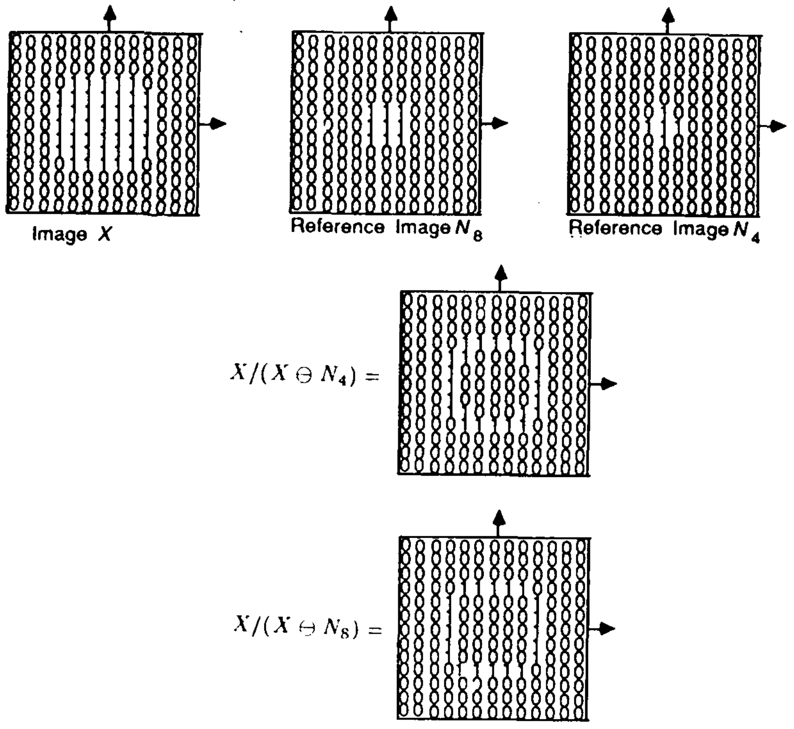


FIG. 12. Edge detection.

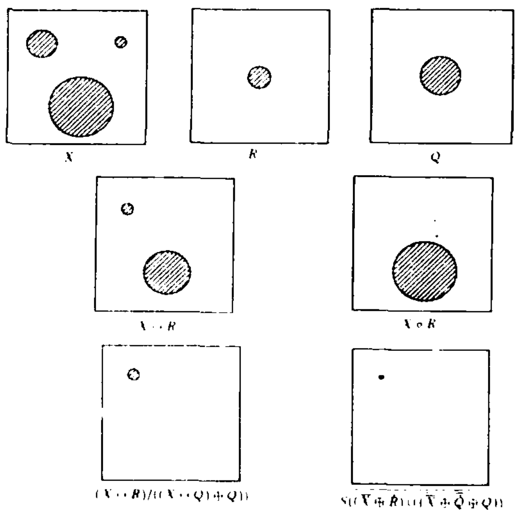


FIG. 13. A size verification (for holes)

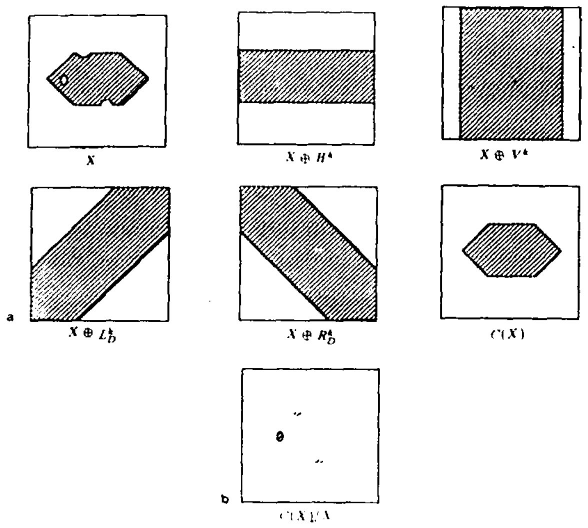


FIG. 14. (a) An example of the convex hull of an image X (implemented by the intersection of projections). (b) The difference of $C(X)$ by X.

4. RELATIONSHIP TO OTHER COMPUTING THEORIES

4.1. Relationship to Boolean Logic

BIA can implement any boolean logic operation on binary images. It is also obvious that BIA fundamental operations can be implemented by a boolean logic gate array with interconnections. The following straightforward correspondence can be drawn between the BIA operations and boolean logic operations:

BIA operations	Boolean logic operations
1. Complement	NOT
2. Union	OR
3. Dilation	Multiple-input OR
4. Intersection	AND
5. Erosion	Multiple-input AND
6. Symmetric difference	EXCLUSIVE-OR

Note that the inputs of OR and AND (corresponding to union and intersection) come from two different images. The multiple inputs of OR and AND (corresponding to dilation and union) come from the same image while the other operand image R only determines the number and location of input pixel values. A complete logical set is able to implement any boolean logic function; it consists of at least one of the following sets: NOT and OR; NOT and AND; NAND; NOR. In BIA, in order to implement any image transformation, we need a complete system of pixelwise logic operations and we also need a translational type of operation (such as, translation, dilation, erosion, convolution, and correlation) to allow the global information extraction in an image or the information exchange between pixels of the same images. In order to have a 2D compact parallel form of image processing algorithms whose variables are whole images, we define the parallel form of those corresponding boolean logic operations as BIA operations. In fact, there are two boolean algebras $(P(W); \cup, \cap, ^{-}, \varnothing, W)$ and $(P(W); \vartriangle, \cap, ^{-}, \varnothing, W)$, supported by BIA also (Subsection 4.4). We can define several possible sets of fundamental operations for implementing any image transformation, such as a parallel form of NOR (or NAND or (NOT and OR) or (NOT and AND)) and a translational-type operation (e.g., translation, dilation, erosion, convolution, and correlation). The reasons that we choose complement, union, and dilation as the three fundamental operations are:

- Nice mathematical properties. The dilation is commutative, associative, and distributive over the union; but the erosion has no such properties.
- Simple hardware implementation. These three operations are easily implemented by the 2D gate array and 3D interconnection technique.
- Simple software design. These three operations are inherently parallel and frequently used operations. Algorithms can be written as compact formulas which easily become very efficient fast parallel algorithms by simply applying the fundamental operations and removing the data dependencies.

Comparing BIA with the conventional boolean expressions for logic functions, the major advantages of BIA are summarized in the following:

- BIA operations are inherently parallel, but boolean logic operations are serial.
- BIA operations include parallel information transferring capabilities which are missing in boolean logic operations.
- Algorithms in BIA are written as compact algebraic formulas whose variables are whole images, while a typical image processing algorithm is very difficult to write in a compact precise boolean logic expression.
- BIA has pictorial physical meaning, while boolean expressions provide little physical feeling for parallel image processing algorithms.

4.2. Relationship to Symbolic Substitution and Cellular Logic

Symbolic substitution is a means of performing parallel digital computations and can be used to implement boolean logic, binary arithmetic, cellular logic, and Turing machines [37, 38]. It involves two steps: (1) recognizing all the locations of a certain spatial pattern within the 2D input data and (2) substituting a new replacement pattern wherever the search pattern was recognized. BIA can be used to realize a symbolic substitution rule,

$$(X \circledast R) \oplus Q = \overline{(\bar{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)} \oplus Q,$$

where X is the 2D input data, $R = (R_1, R_2)$ is the reference image pair corresponding to the search pattern (R_1 and R_2 define the foreground and the background of the search pattern, respectively), $\check{R}$ defines a reflected reference image given by $\check{R} = \{(-x, -y) | (x, y) \in R\}$, and Q is the reference image corresponding to the replacement pattern. Thus, symbolic substitution rules are particular BIA image transformations having the above form; and BIA represents a general complete systematic mathematical tool for formalizing the symbolic substitution algorithms.

Cellular logic architectures have been briefly reviewed in Section 1. A cellular logic operation transforms an array of data into a new array of data where each element in the new array has a value determined only by the corresponding element in the original array along with the values of its neighbors (Fig. 1). In BIA, an image transformation can be written as a polynomial of reference images (Theorem 2.1), where the reference images can have arbitrary large size. In terms of cellular logic, the reference image essentially defines the neighborhood of a cell where the neighborhood can be very large and not just nearest 4- or 8-neighborhood as in conventional cellular logic. Thus, cellular logic operations are also particular cases of image transformations with small local reference images, and BIA also serves as a systematic mathematical tool for formalizing cellular logic.

Because of existing hardware interconnection limitations, it is difficult and costly to implement an image transformation with a large reference image in one clock cycle. In addtion, the conventional nearest-neighbor connected cellular arrays have poor communication capabilities. To improve this, we develop the DOCIP-hypercube architecture in Section 5, which combines features of conventional nearest-neighbor connected cellular logic architectures and conventional hypercube architectures for implementing BIA effectively.

In summary, BIA provides a systematic mathematical formalism for both symbolic substitution and cellular logic. The applications of symbolic substitution and cellular logic can be accomplished by BIA; on the other hand, generalized cellular logic architectures are good candidates for implementing BIA.

4.3. Relationship to Linear Shift Invariant Systems, Convolution, and Correlation

It is well known that the theory of linear shift-invariant (LSI) systems plays a key role in conventional signal (including image) and system analysis [39, 40]. It is very natural that we like to ask what the relation between BIA and LSI system theory is. A system is defined as a transformation or mapping from a set of input functions into a set of output functions, and a 2-dimensional discrete LSI system is defined as a system which obeys two properties:

- Linearity. $T[ax(i, j) + bz(i, j)] = aT[x(i, j)] + bT[z(i, j)]$ for arbitrary constants a and b;
- Shift-invariance. $Y(i, j) = T[x(i, j)] \rightarrow y(i - k, j - l) = T[x(i - k, j - l)]$.

A linear system can be completely characterized by its unit-impulse response $r(i, j; k, l) = T[\delta(i - k, j - l)]$. In an LSI system the unit-impulse response is simply $r(i, j; k, l) = r(i - k, j - l)$, and the output of an LSI system with input $x(i, j)$ and unit-impulse response $r(i, j)$ is the convolution of $x(i, j)$ and $r(i, j)$, denoted by

$$x(i, j) * r(i, j) = \sum_{k, l = -\infty}^{\infty} x(k, l) r(i - k, j - l).$$

Now, let us consider only binary images. In terms of the set notation, an image $X = \{(i, j) | x(i, j) = 1\}$ corresponds to function $x(i, j)$. If we assume $r(i, j) = 1$ at and only at n points which correspond to an image R with n image points, then the convolution of $x(i, j)$ and $r(i, j)$ with a threshold $t = 0$ is

$$\begin{aligned} X * R|_{t=0} &= \left\{(i, j) \middle| \sum_{k,l} x(k, l) r(i-k, j-l) > 0\right\} \\ &= \left\{(i+k, j+l) \middle| \sum_{k,l} x(k, l) r(i, j) > 0\right\} \\ &= \{(i+k, j+l) | x(k, l) r(i, j) > 0\} \\ &= \{(i+k, j+l) | (i, j) \in X, (k, l) \in R\} \\ &= X \oplus R, \end{aligned}$$

where the output of the threshold is defined as 1 if $x(i, j) * r(i, j) > 0$, and is 0 otherwise; and the universal image, as before, contains all image points (i, j), (k, l) and $(i+k, j+l)$. This means that the dilation $X \oplus R$ is the same as adding a threshold $t = 0$ to the convolution sum. The reference image plays a role similar to that of the unit impulse response in the binary image system. Similarly the erosion $X \ominus R$ is the same as the convolution $x(i, j) * r(-i, -j)$ followed by the threshold $t = n - 1$.

Correlators have been used in pattern recognition for a long time [41]. Correlatior is strongly related to convolution: convolution involves folding, shifting, anc summing; correlation involves shifting and summing without folding. Therefore,

$$\begin{aligned} X \oplus R &= X * R|_{t=0} = X \diamond \check{R}|_{t=0} \\ X \ominus R &= X * \check{R}|_{t=n-1} = X \diamond R|_{t=n-1}, \end{aligned}$$

where $*$ means convolution, $\diamond$ means correlation, and $\check{R}$ means the reflecte image of R.

Furthermore, although the three fundamental operations of BIA are nonlineai with appropriate number representations they are able to implement parallel numei ical and linear operations too. Also, BIA can implement both shift invariant an shift variant image transformations.

4.4. Some Standard Algebraic Structures

Some algebraic structures supported by BIA are:

1. $(P(W); \oplus)$ is a semigroup.
2. $(P(W); \oplus, I)$ is a monoid.
3. $(P(W); \Delta, \varnothing, \Delta)$ is an abelian group.
4. $(P(W); \cup, \cap, {}^{-}, \varnothing, W)$ and $(P(W); \Delta, \cap, {}^{-}, \varnothing, W)$ are Boolean algebra
5. $(P(W); \subset)$ is a poset (partially ordered set).
6. $(P(W); \cup, \cap, \subset)$ is a complete lattice.

Proof. (1) A semigroup is a set with an associative binary operation [30–32]. By Theorem 3.1, the dilation $\oplus$ is associative for all images in $P(W)$.

(2) A monoid is a semigroup with an identity [30–32]. By Appendix D, the dilation has an identity $I = \{(0,0)\}$. Note that $(P(W); \ominus)$ is neither a semigroup nor a monoid.

(3) A group is a monoid in which every element has an inverse. An abelian group is a group in which the operation is commutative [30—32]. By the definition of symmetric difference (mod 2 image addition), it can be easily verified that its identity is $\varnothing$ and its inverse operation (mod 2 image subtraction) is itself.

(4) A boolean algebra is a set with operations $\vee$, $\wedge$, $^{-}$, 0, and 1 satisfying: 1. $a \vee b = b \vee a$, $a \wedge b = b \wedge a$ (commutativity); 2. $a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$, $a \wedge (b \vee c) = (a \wedge b) \vee (a \wedge c)$ (associativity); 3. $a \vee 0 = a$ (universal bound); 4. $a \wedge 1 = a$ (universal bound); 5. $a \vee \bar{a} = 1$, $a \wedge \bar{a} = 0$ (complementarity) [30—32]. By Appendix D, $(P(W); \cup, \cap, ^{-}, \varnothing, W)$ and $(P(W); \Delta, \cap, ^{-}, \varnothing, W)$ are Boolean algebras.

(5) A poset is a set with a relation satisfying: 1. the reflexivity; 2. the antisymmetry; and 3. the transitivity [30–32]. The relation $\subset$ satisfies these three conditions: 1. $X \subset X$ for all $X \in P(W)$; 2. if $X \subset R$ and $R \subset X$, then $X = R$; and 3. if $X \subset R$ and $R \subset Q$, then $X \subset Q$.

(6) A complete lattice is a poset $(S; \leq)$ in which every subset of S has a sup (the least upper bound) and an inf (the greatest lower bound) [30–32]. In the algebra $(P(W); \cup, \cap, \subset)$, given any subset of $P(W)$, say $\{X(\theta) | \theta \in \Theta\}$ (Θ is the index set of the elements in this subset of $P(W)$), we have

$$\sup = \bigcup_{\theta \in \Theta} X(\theta)$$

$$\inf = \bigcap_{\theta \in \Theta} X(\theta).$$

Thus, several standard algebraic structures and their properties can be directly implemented and used in BIA.

5. IMPLEMENTATION ON OPTICAL CELLULAR LOGIC PROCESSORS

To map algorithms into architectures, we first use an algebraic approach for describing a cellular image processor. Then we design the digital optical cellular image processors (DOCIPs) and their optical implementation. Figures 15 and 16 show an optical concept for the DOCIP implementation. The optical system can realize an array of cells by a spatially parallel 2D array of optical binary gates and performs interconnections of these gates by an optical hologram. The DOCIPs are:

- The DOCIP-array (Fig. 15), a cellular array processor, which uses optical parallelism to map an inherently 2D parallel data structure to a 2D nearest-neighbor connected cellular computer in a simple and direct way; its performance is primarily limited by its $O(1)$ interconnectivity, and
- The DOCIP-hypercube (Fig. 16), a 2-dimensional cellular hypercube, which uses optical parallelism and 3D global interconnection capabilities to implement a hypercube interconnection.

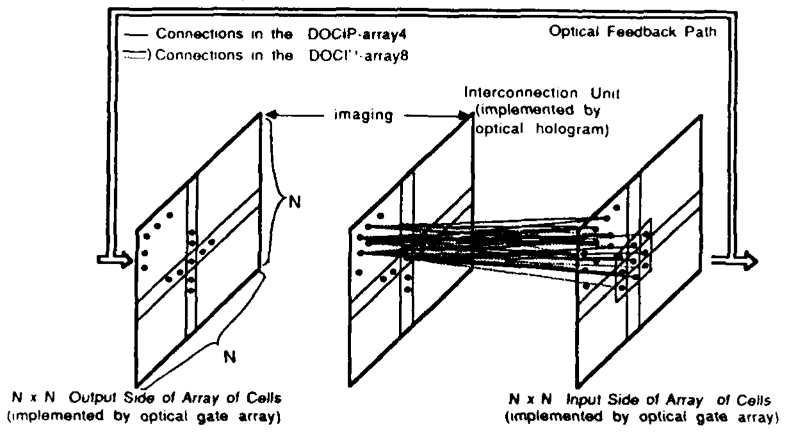

— Connections in the DOCIP-array4
⟹ Connections in the DOCIP-array8
Optical Feedback Path
Interconnection Unit (implemented by optical hologram)
imaging
N
N
N x N Output Side of Array of Cells (implemented by optical gate array)
N x N Input Side of Array of Cells (implemented by optical gate array)

FIG. 15. An optical 4-connected or 8-connected cellular array (DOCIP-array4 or DOCIP-array8). Imaging optics are omitted for clarity. Each cell connects with its four nearest cells and itself by optical 3D free interconnection. The optical hologram provides both intra-cell and inter-cell interconnections. The input and output sides of the optical gate array are interconnected by an optical feedback path and are shown separately for clarity.

Here, the 2-dimensional cellular hypercube is used to match the structure of a 2-dimensional image and further improve the communication ability of a cellular array. Ideally, a conventional hypercube (Fig. 17) increases the interconnectivity to $O(\log N)$ for N computation cells; however, when laid out in 2-dimensional space, its interconnection patterns are not space invariant; such spatial invariance is desirable for image processing and for simple implementation in optical hardware. To include this, we increase the interconnections to make a 2-dimensional cellular hypercube (Fig. 18). The cellular hypercube introduces a symmetrical positive and

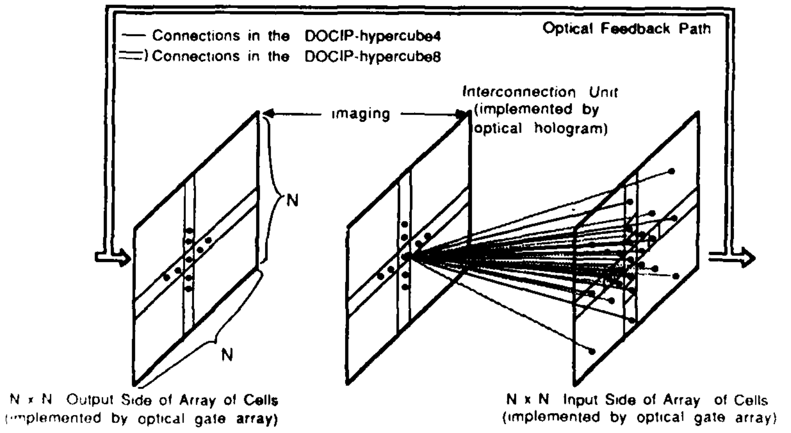

— Connections in the DOCIP-hypercube4
⟹ Connections in the DOCIP-hypercube8
Optical Feedback Path
Interconnection Unit (implemented by optical hologram)
imaging
N
N
N x N Output Side of Array of Cells (implemented by optical gate array)
N x N Input Side of Array of Cells (implemented by optical gate array)

FIG. 16. An optical 4-directed or 8-directed cellular hypercube (DOCIP-hypercube4 or DOCIP-hypercube8). Each cell connects with cells in the 4 directions or 8 directions at distances $1, 2, 4, 8, \ldots, 2^k$ from it by optical 3D free interconnection

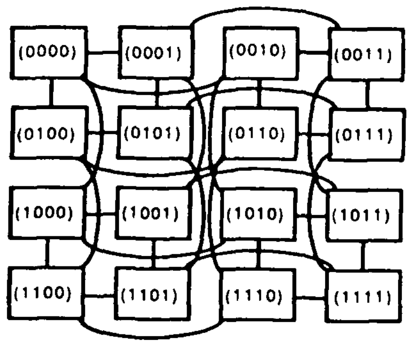


Fig. 17. A conventional hypercube (4-cube) laid out in 2-dimensional space. Its interconnections have no spatial invariance.

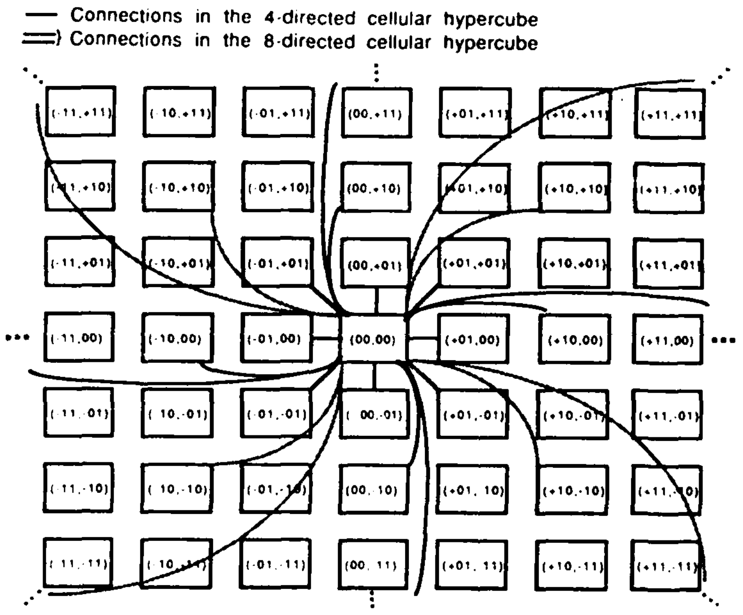


Fig. 18. A 2-dimensional cellular hypercube—DOCIP-hypercube. Each cell is interconnected with other cells having a *relative* one bit difference in coordinate label in positive or negative x and y directions to achieve a spatially symmetric and invariant interconnection pattern. Only connections from the central cell are shown; all cells are connected identically so the resulting interconnections are space invariant.

negative index so that each cell is connected with cells having a *relative* one bit difference in coordinate label in positive or negative x and y directions; the numerical difference of addresses of connected cells is nonzero in at most one bit [42].

5.1. Algebraic Description

Having defined cellular automata and the implementation requirements of BIA, we describe the DOCIP in an algebraic way:

DEFINITION OF CELLULAR AUTOMATA. A cellular automaton is an algebra $A = (S; F, N_c)$, where S is the state space which is a set of states, F is a family of transition functions, and N_c is the neighborhood configuration.

CONSTRAINTS OF IMPLEMENTING BIA.

1. $S \supset P(W)$
2. $F \supset \{\oplus, \cup, {}^{-}\}$
3. $N_c \supset I \cup A \cup A^{-1} \cup B \cup B^{-1}$ (or $N_c \supset A \cup A^{-1} \cup B \cup B^{-1}$), where "$\supset$" means "contains."

Thus, in terms of cellular automata, the DOCIPs have to satisfy the above constraints for realizing BIA. For storing input images and temporary results in a more flexible way, the DOCIPs utilize three memory modules and share the same algebraic structure (except the neighborhood configuration):

$$\text{DOCIP} = (P(W \times W \times W); \oplus, \cup, {}^{-}, N_c),$$

where "$\times$" denotes cross product and N_c can be one of the following four types:

1. DOCIP-array4. Each cell connects with its four nearest neighbors and itself, i.e.,

$$N_{\text{array4}} = I \cup A \cup A^{-1} \cup B \cup B^{-1}.$$

2. DOCIP-array8. Each cell connects with its eight nearest neighbors and itself, i.e.,

$$N_{\text{array8}} = \bigcup_{i,j=-1}^{1} A^iB^j.$$

3. DOCIP-hypercube4. Each cell connects with itself and those cells in the four directions at distances $1, 2, 4, 8, \ldots, 2^k$ from itself, i.e.,

$$N_{\text{hypercube4}} = \bigcup_{i=0,\pm 1,\pm 2,\ldots,\pm 2^k} (A^i \cup B^i),$$

where k is sufficiently large for the connections to traverse the entire array of cells.

4. DOCIP-hypercube8. Each cell connects with itself and those cells in the eight directions at distances $1, 2, 4, 8, \ldots, 2^k$ from itself, i.e.,

$$N_{\text{hypercube8}} = \bigcup_{i=0,\pm 1,\pm 2,\ldots,\pm 2^k} (A^i \cup B^i \cup A^iB^i \cup A^iB^{-i}).$$

5.2. General Description

From the above algebraic description, the DOCIPs have the same algebraic structure and differ only in their neighborhood configurations N_c. Thus, they share the same architecture as shown in Fig. 19, but have different configurations of the reference images E_i, depending on the optical interconnection network which defines the neighborhood. In practical applications, a *larger* reference image R can be generated from a set of *smaller* reference image(s) E_i by a "sequential dilation." If it is possible to decompose R into a sequence $R = E_1 \oplus E_2 \oplus \cdots \oplus E_k$, then

$$X \oplus R = (\cdots((X \oplus E_1) \oplus E_2) \oplus \cdots \oplus E_k).$$

This decomposition may not exist; in which case R can always be decomposed as $R = R_1 \cup R_2 \cup \cdots \cup R_k$, and then

$$X \oplus R = (X \oplus R_1) \cup (X \oplus R_2) \cup \cdots \cup (X \oplus R_k),$$

where each R_j can be composed from the smaller reference images E_i.

Basically, the proposed DOCIP as shown in Fig. 19 is a cellular SIMD machine and consists of an array of cells or processing elements (PEs) under the supervision of a control unit. The control unit includes a clock, a program counter, a test and branch module for feedback control, and an instruction decoder for storing instructions and decoding them to supervise cells. The array of cells includes a $1 \times 3 \times N^2$ bit destination selector, three $N \times N \times 1$ bit memories for storing images, a memory selector, and a dilation unit.

The DOCIP shown in Fig. 19 operates as follows: (1) a binary image ($N \times N$ matrix) is selected by the destination selector and then stored in any memory as the instruction specifies; (2) after storing the images (1 to 3 $N \times N$ matrices), these

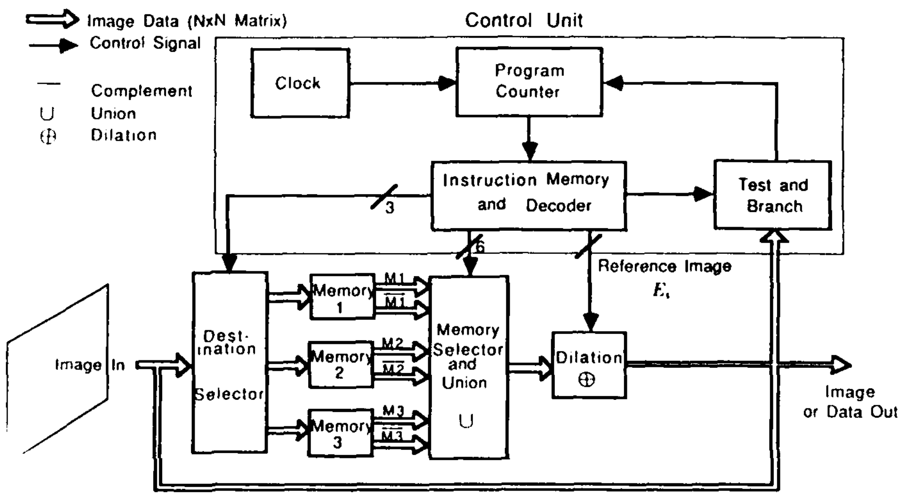


FIG. 19. A digital optical cellular image processor (DOCIP) architecture—one implementation of binary image algebra (BIA). The DOCIP-array requires 9 (or 5) control bits for reference image E_i. The DOCIP-hypercube requires $O(\log N)$ control bits for reference image E_i.

images and their complemented versions are piped into the next stage, which forms the union of any combination of images; (3) the result is sent to a dilation where the reference image specified by the instruction is used to control the type of dilation; (4) finally, the dilated image can be output, tested for program control, or fed back to step (1) by the address field of the instruction.

The entire system can be realized by an optical gate array with optical 3D interconnections [25–28]. It should be noted that current optical technology has implemented only arrays of moderately large numbers of gates (500×500) at very slow ($\sim$ ms) switching speeds, and alternatively, arrays of small numbers of gates (2×2 to 6×6) at fast switching speeds (0.1 μs–50 ps) [43, 44]. Current ongoing research in a number of laboratories looks promising in eventually providing the needed arrays of large numbers of gates with reasonably fast switching speeds. Alternatively, control of the DOCIP can be easily realized by using an electronic host instead of the optical control unit, since control of SIMD systems is primarily a serial process. The trade-off is a possible inefficiency in the interfaces between electronic and optical units. Because of this the all-optical approach may be preferable in the long term. To efficiently utilize optical gates, they can be interconnected with a 2D optical multiplexing technique in which a common controllable mask is used for all cells. The optical multiplexing technique has the following advantages: (1) the DOCIP will no longer require the broadcasting of instructions from the control unit—instead all cells fan their outputs into a common controlling mask pixel; (2) it will reduce the number of gates; and (3) each cell has a simple structure—essentially containing only a 3-bit memory with inverting and noninverting outputs, and a multiple-input OR gate for dilation [45].

To avoid the well-known drawbacks of conventional computers based on von Neumann principles [23, 38], the machine in Fig. 19 has one instruction which implements the three fundamental operations of BIA along with fetch and store. This design uses the parallelism of optics to simultaneously execute instructions involving all N^2 picture elements.

This single instruction has the format

$$(c, d_1, d_2, d_3, s_1, s_2, \ldots, s_6, n_1, n_2, \ldots, n_k, j_1, j_2, a_1, a_2, \ldots, a_l, b_1, b_2, \ldots, b_l),$$

where k is determined by the chosen neighborhood configuration N_c. The DOCIP-array requires $k = 5$ or $k = 9$ bits for controlling reference image R at a clock cycle and the DOCIP-hypercube requires $k = O(\log N)$ for N cells, and l defines the maximum length of a program: 2^l. The functions of these $11 + k + 2l$ instruction codes are:

- c is used to select the image from the input or from the feedback;
- d_1, d_2, and d_3 are used to select the destination memory for storing the image;
- $s_1, s_2, \ldots, s_6$ are used to select the output from the memory elements;
- $n_1, n_2, \ldots, n_k$ are used to control the neighborhood mask, i.e., to supply the reference image;
- j_1 and j_2 are used to flag an absolute jump or conditional jump;
- $a_1, a_2, \ldots, a_l$ are the address for jump; and
- $b_1, b_2, \ldots, b_l$ are the address of the instruction.

TABLE 2
Cellular Image Processor Execution Times for $N \times N$ Image Data

Operation \ Technology		Conventional array (electronics)	DOCIP-array (optics)	DOCIP-hypercube (optics)
Local operations		$O(1)$	$O(1)$	$O(1)$
Global operations		$O(N)$	$O(N)$	$O(\log N)$
	or	$O(N^2)$		
Communication PE ↔ Main Memory		$O(N)$	$O(1)$	$O(1)$
	or	$O(N^2)$		
Input/Output		$O(N)$	$O(1)$	$O(1)$
	or	$O(N^2)$		

Note. Table 2 roughly compares the execution time for the conventional electronic array processor, the DOCIP-array, and the DOCIP-hypercube.

Order of magnitude execution times for image processing on the DOCIP machines and on the conventional-array processors are compared in Table 2. In contrast with the DOCIP-array, the DOCIP-hypercube increases the interconnection complexity to $O(\log N)$, but is able to perform many global operations in $O(\log N)$ time. Comparing with the conventional-array processors having serial or N-parallel input/output, the DOCIP-array will have the same order of performance in local and global operations but will be improved in input/output performance, and in principle could be as low as $O(1)$ in I/O operations. The DOCIP-hypercube will not only be improved in input/output performances but also in global operations. With external memory, it can also be demonstrated to be general purpose in the sense of the ability of simulating any Turing machine. One important feature in the design of the DOCIP-array and DOCIP-hypercube is that optical 3D free interconnection capabilities can be used to reduce the cell hardware requirements as well as solve the global connection and I/O problems which are difficult to solve by planar VLSI technology.

6. A PROGRAMMING EXAMPLE—SIZE VERIFICATION

BIA and DOCIP architectures can have many applications in character recognition, industrial inspection, medical and scientific research. Since BIA is able to implement morphological operations efficiently, the DOCIP machines can efficiently analyze the shape and connectivity of regions as well as measure their size; they also have the potential to accomplish any image transformation. Here we illustrate the programming of the DOCIP machines by a simple size verification algorithm:

- **Problem.** Given an input image X with 31×31 pixels (Fig. 20) which contains some square objects X_t, we want to preserve those square objects X_t which satisfy the condition,

$$\text{size of } R \leq \text{size of } X_t < \text{size of } Q$$

where R and Q are reference images as shown in Fig. 21. Other objects will be eliminated in the output image Y. The expected output image Y is shown in Fig. 22.

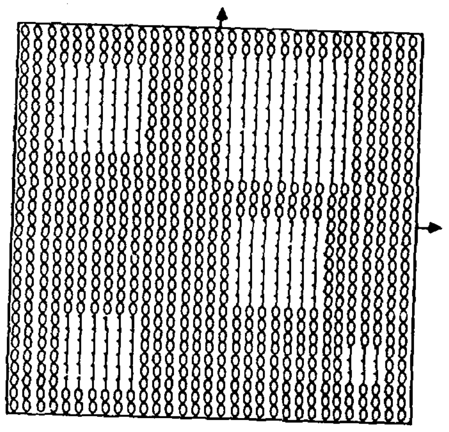
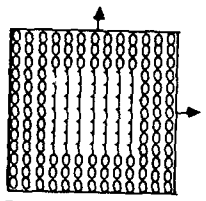


FIG. 20. The input image X.



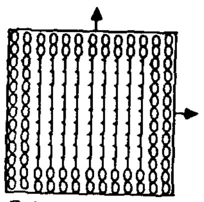


FIG. 21. The reference images R and Q.

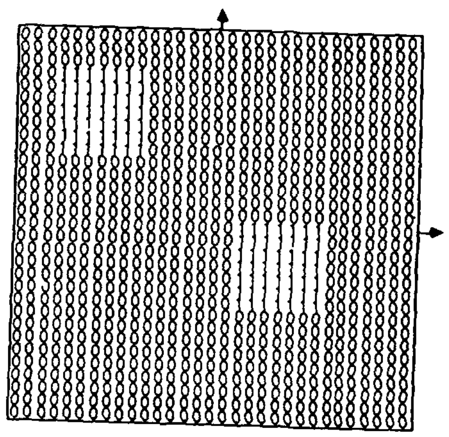


FIG. 22. The expected output image Y.

• **Algebraic expression for the size verification using band pass morphological filtering** (Theorem 3.2),

$$\overline{\left(\overline{\overline{\bar{X} \oplus R} \oplus R}\right) \cup \left(\overline{\bar{X} \oplus Q} \oplus Q\right)},$$

where $R = \check{R}$ and $Q = \check{Q}$ in this special example.

• **Algorithm for the DOCIP-array8.**

$$\overline{\left(\overline{\overline{\bar{X} \oplus E^3} \oplus E^3}\right) \cup \left(\overline{\bar{X} \oplus E^4} \oplus E^4\right)},$$

where E (Fig. 23) is the allowed reference image with the maximum size at a clock cycle in the DOCIP-array8, the reference images $R = E^3 = E \oplus E \oplus E$ and $Q = E^4 = E \oplus E \oplus E \oplus E = R \oplus E$.

The DOCIP-array8 requires 13 steps to complete this algorithm, its program (instructions) is in the following:

```
Assume start with $X \rightarrow M_1$ ( *X stored in Memory*1)
 1. $\overline{M_1} \oplus E \rightarrow M_2 \ (= \bar{X} \oplus E)$
 2. $M_2 \oplus E \rightarrow M_2 \ (= \bar{X} \oplus E^2)$
 3. $M_2 \oplus E \rightarrow M_2 \ (= \bar{X} \oplus E^3)$
 4. $M_2 \oplus E \rightarrow M_3 \ (= \bar{X} \oplus E^4)$
 5. $\overline{M_2} \oplus E \rightarrow M_2 \left(= \overline{\bar{X} \oplus E^3} \oplus E\right)$
 6. $M_2 \oplus E \rightarrow M_2 \left(= \overline{\bar{X} \oplus E^3} \oplus E^2\right)$
 7. $M_2 \oplus E \rightarrow M_2 \left(= \overline{\bar{X} \in E^3} \oplus E^3\right)$
 8. $\overline{M_3} \oplus E \rightarrow M_3 \left(= \overline{\bar{X} \oplus E^4} \oplus E\right)$
 9. $M_3 \oplus E \rightarrow M_3 \left(= \overline{\bar{X} \oplus E^4} \oplus E^2\right)$
10. $M_3 \oplus E \rightarrow M_3 \left(= \overline{\bar{X} \oplus E^4} \oplus E^3\right)$
11. $M_3 \oplus E \rightarrow M_3 \left(= \overline{\bar{X} \oplus E^4} \oplus E^4\right)$
12. $\overline{M_2} \cup M_3 \rightarrow M_3 \left(= \left(\overline{\overline{\bar{X} \oplus E^3} \oplus E^3}\right) \cup \left(\overline{\bar{X} \oplus E^4} \oplus E^4\right)\right)$
13. End with $\overline{M_3} \rightarrow Y \left(= \overline{\left(\overline{\overline{\bar{X} \oplus E^3} \oplus E^3}\right) \cup \left(\overline{\bar{X} \oplus E^4} \oplus E^4\right)}\right)$
```

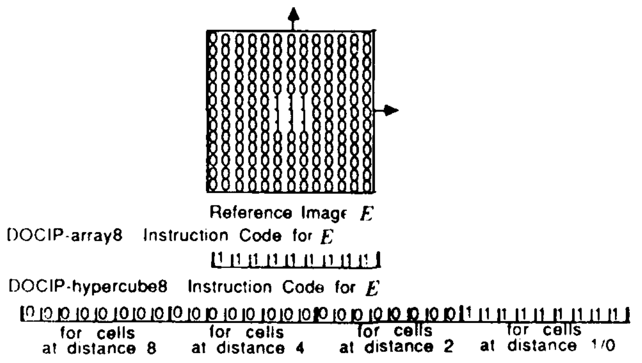


FIG. 23. An allowed reference image E at a clock cycle in the DOCIP-array8 (also allowed in DOCIP-hypercube8) and its corresponding 9 (or 33) bits in instruction ($n_1 n_2 \cdots n_k$) for controlling the neighborhood mask (i.e., the reference image for the dilation).

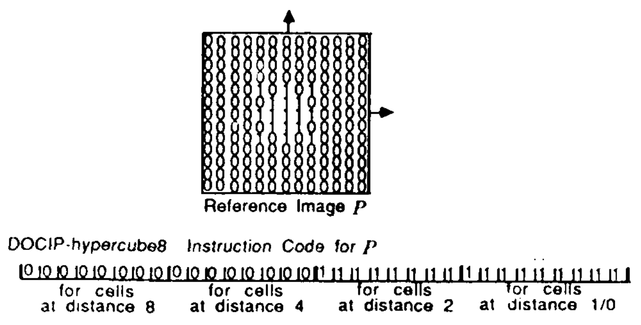


FIG. 24. An allowed reference image P at a clock cycle in the DOCIP-hypercube8 (not allowed in the DOCIP-array8) and its corresponding 33 bits (assume 31×31 cells) in instruction $(n_1 n_2 \cdots n_{33})$ for controlling the neighborhood mask (i.e., the reference image for the dilation).

- **Algorithm for the DOCIP-hypercube8.**

$$\overline{\left(\overline{\overline{\overline{X} \oplus P \oplus E} \oplus P \oplus E}\right) \cup \left(\overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E^2\right)},$$

where P (Fig. 24) and E (Fig. 23) are allowed reference images at a clock cycle in the DOCIP-hypercube8, the reference images $R = E^3 = P \oplus E$ and $Q = E^4 = P \oplus E^2 = R \oplus E$.

The DOCIP-hypercube8 requires 10 steps to complete this algorithm, its program (instructions) is shown in the following:

Assume start with $X \to M_1$ (*X stored in Memory*1)

1. $\overline{M_1} \oplus P \to M_2 \; (= \overline{X} \oplus P)$
2. $M_2 \oplus E \to M_2 \; (= \overline{X} \oplus P \oplus E)$
3. $M_2 \oplus E \to M_3 \; (= \overline{X} \oplus P \oplus E^2)$
4. $\overline{M_2} \oplus P \to M_2 \left(= \overline{\overline{X} \oplus P \oplus E^2} \oplus P\right)$

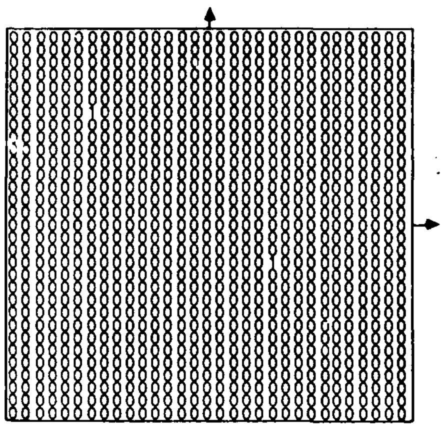


FIG. 25. The locations of the desired objects in the output image Y.

5. $M_2 \oplus E \to M_2 \left(= \overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E \right)$
6. $\overline{M_3} \oplus P \to M_3 \left(= \overline{\overline{X} \oplus P \oplus E^2} \oplus P \right)$
7. $M_3 \oplus E \to M_3 \left(= \overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E \right)$
8. $M_3 \oplus E \to M_3 \left(= \overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E^2 \right)$
9. $\overline{M_2} \cup M_3 \to M_3 \left(= \left(\overline{\overline{\overline{X} \oplus P \oplus E} \oplus P \oplus E} \right) \cup \left(\overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E^2 \right) \right)$
10. End with $\overline{M_3} \to Y \left(= \overline{\left(\overline{\overline{X} \oplus P \oplus E} \oplus E \right) \cup \left(\overline{\overline{X} \oplus P \oplus E^2} \oplus P \oplus E^2 \right)} \right)$

The above programs can be translated into the machine instruction codes directly. If we want to detect the geometric centers (locations) of the desired objects, then we can use a sequential thinning to achieve the homotopic skeleton (Theorem 3.4) (Fig. 25).

7. CONCLUSIONS

We have summarized digital optical cellular image processing, including binary image algebra (BIA) and the DOCIP architectures. BIA suggests a unified theory of parallel binary image processing for developing parallel algorithms/languages and can be generalized to grey-level images. Applications of BIA in binary image processing are illustrated. The DOCIP architectures, especially the DOCIP-hypercube, utilize the parallel communication and global interconnection capabilities of optics for avoiding communication bottlenecks and matching BIA parallel algorithms efficiently. A size verification algorithm is used to demonstrate the programming of these 1-instruction DOCIP machines. Overall, BIA is a simple, precise, and complete algebraic theory of binary images; the DOCIP machines have simple organization, low cell complexity, and potentially fast processing ability.

APPENDIX A

Proof of Lemma 2.1. We start with the case of $X = \check{R}$ and then the case of $X \neq \check{R}$.

Case 1. $X = \check{R}$, i.e., $R = \check{X}$. We want to prove

$$\overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \check{\overline{X}}) \cup \overline{I}} = I \leftrightarrow \overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \check{\overline{X}})} \cap I = I.$$

1. Claim $I \subset \overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \check{\overline{X}})} \cap I$

$\leftrightarrow (0,0) \in \overline{(\overline{X} \oplus \check{X}) \cup (X \oplus \check{\overline{X}})}$

$\leftrightarrow (0,0) \notin (\overline{X} \oplus \check{X}) \cup (X \oplus \check{\overline{X}})$

$\leftrightarrow [(0,0) \notin (\overline{X} \oplus \check{X})] \wedge [(0,0) \notin (X \oplus \check{\overline{X}})]$:

(a) Claim $(0,0) \notin (\overline{X} \oplus \check{X})$. Assume $(0,0) \in (\overline{X} \oplus \check{X})$

$\to (0,0) \in \{(a + (-x), b + (-y)) \in W | (a,b) \in \overline{X}, (-x, -y) \in \check{X}\}$

$\to (0,0) \in \{(a - x, b - y) \in W | (a,b) \notin X, (x,y) \in X\}$

$\to \exists\, (a - x, b - y) = (0,0)$ where $(a,b) \notin X, (x,y) \in X$

$\to \exists\, (x,y) = (a,b)$ where $(a,b) \notin X, (x,y) \in X$

which is impossible, since $(x,y) = (a,b) \notin X$ contradicts with $(x,y) = (a,b) \in X$. Therefore, the assumption is wrong, we have that $(0,0) \notin (\overline{X} \oplus \check{X})$.

(b) Claim $(0,0) \notin (X \oplus \overline{\check{X}})$. Assume $(0,0) \in (X \oplus \overline{\check{X}})$

$\rightarrow$ $(0\ 0) \in \{(x + (-a),\ y + (-b)) \in W \mid (x, y) \in X,\ (-a, -b) \in \overline{\check{X}}\}$

$\rightarrow (0,0) \in \{(x - a, y - b) \in W \mid (x, y) \in X, (-a, -b) \notin \check{X}\}$

$\rightarrow (0,0) \in \{(x - a, y - b) \in W \mid (x, y) \in X, (a, b) \notin X\}$

$\rightarrow \exists\, (x - a, y - b) = (0,0)$ where $(a, b) \notin X, (x, y) \in X$

$\rightarrow \exists\, (x, y) = (a, b)$ where $(a, b) \notin X, (x, y) \in X$

which is impossible, since $(x, y) = (a, b) \notin X$ contradicts with $(x, y) = (a, b) \in X$. Therefore, the assumption is wrong, we have that $(0,0) \notin (X \oplus \overline{\check{X}})$.

By (a) and (b), we have $[(0,0) \notin (\bar{X} \oplus \check{X})] \wedge [(0,0) \notin (X \oplus \overline{\check{X}})]$, i.e.,

$$I \subset \overline{(\bar{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})}.$$

We also know $I \subset I$, then we have

$$I \subset \overline{(\bar{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})} \cap I.$$

2. Claim $\overline{(\bar{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}}) \cup \bar{I}} \subset I$. Since $I \subset I$, it implies

$$\overline{(\bar{X} \oplus \check{X}) \cup (X \ominus \overline{\check{X}}) \cup \bar{I}} = \overline{(\bar{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})} \cap I \subset I.$$

From (1) and (2), we have

$$I \subset \overline{(\bar{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})} \cap I$$

and

$$\overline{(\bar{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}}) \cup \bar{I}} = \overline{(\bar{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})} \cap I \subset I.$$

Thus, by the equivalence of sets, we have $\overline{(\bar{X} \oplus \check{X}) \cup (X \oplus \overline{\check{X}})} \cap I = I$.

Case 2. $X \neq \check{R}$, i.e., $R \neq \check{X}$. We want to prove

$$\overline{(\bar{X} \oplus R) \cup (X \oplus \bar{R}) \cup \bar{I}} = \varnothing \leftrightarrow \overline{(\bar{X} \oplus R) \cup (X \oplus \bar{R})} \cap I = \varnothing .$$

1. Claim $I \not\subset \overline{(\bar{X} \oplus R) \cup (X \oplus \bar{R})}$

$\leftrightarrow (0,0) \notin (\overline{(\bar{X} \oplus R) \cup (X \oplus \bar{R})}$

$\leftrightarrow (0,0) \in (\bar{X} \oplus R) \cup (X \oplus \bar{R})$

$\leftrightarrow (0,0) \in (\bar{X} \oplus R) \vee (0,0) \in (X \oplus \bar{R})$:

Now we assume $(0,0) \notin (\bar{X} \oplus R) \wedge (0,0) \notin (X \oplus \bar{R})$:

(a) If $(0,0) \notin (\bar{X} \oplus R)$

$\rightarrow (0,0) \notin \{(a + k, b + l) \mid (a, b) \in \bar{X}, (k, l) \in R\}$

$\rightarrow (a + k, b + l) \neq (0,0), \forall\, (a, b) \in \bar{X}, \forall\, (k, l) \in R$

$\rightarrow (a, b) \neq (-k, -l), \forall\, (a, b) \notin X, \forall\, (k, l) \in R$

$\rightarrow \forall\, (k, l) \in R, \exists\, (a, b) \in X, (a, b) = (-k, -l)$

$\rightarrow \forall\, (-k, -l) \in \check{R}, \exists\, (a, b) \in X, (a, b) = (-k, -l)$

$\rightarrow \forall\, (i, j) \in \check{R}, \exists\, (a, b) \in X, (c, b) = (i, j)$

$\rightarrow ((i, j) \in \check{R}) \rightarrow ((i, j) \in X)$

$\rightarrow \check{R} \subset X$.

(b) If $(0,0) \notin (X \oplus \bar{R})$, then $X \subset \check{R}$. Since the dilation operation is commutative, by interchanging the variables X and $\check{R}$ and applying the same procedure as (a), we have $X \subset \check{R}$.

2. By the above (a) and (b), we have $X = \check{R}$ which contradicts with $X \neq \check{R}$. Thus, the assumption is wrong, and we get

$$\begin{aligned}
&(0,0) \in (\bar{X} \oplus R) \vee (0,0) \in (X \oplus \bar{R}) \\
\leftrightarrow\ & I \not\subset \overline{(\bar{X} \oplus R) \cup (X \oplus \bar{R})} \\
\leftrightarrow\ & \overline{(\bar{X} \oplus R) \cup (X \oplus \bar{R})} \cap I = \varnothing \\
\leftrightarrow\ & \overline{(\bar{X} \oplus R) \cup (X \oplus \bar{R}) \cup \bar{I}} = \varnothing.
\end{aligned}$$

Hence, by Cases 1 and 2, we have shown that

$$\overline{(\bar{X} \oplus R) \cup (X \oplus \bar{R}) \cup \bar{I}} = \begin{cases} I & \text{if } X = \check{R}, \\ \varnothing & \text{otherwise.} \end{cases}$$

APPENDIX B

Proof of Theorem 2.1. Consider any image transformation (general case),

$$T: \begin{cases} X_1 \to A_1 \\ X_2 \to A_2 \\ \quad\vdots \\ X_l \to A_l, \end{cases}$$

where $X_i \in P(W)$, $A_i \in P(W)$, $i = 1, 2, \ldots, l$.

If we choose $R_i = \check{X}_i$, $Q_i = \check{A}_i$, $i = 1, 2, \ldots, l$ and use Lemma 2.1 and some properties of the dilation (i.e., $I \oplus X = X$ and $\varnothing \oplus X = \varnothing$), then we have

$$T(X) = \bigcup_{i=1}^{l} \left\{ \overline{(\bar{X} \oplus R_i) \cup (X \oplus \bar{R}_i) \cup \bar{I}} \oplus Q_i \right\}.$$

Since some images X_i may map into the null image $\varnothing$ for a given image transformation, by Lemma 2.1 we have that

$$T(X) = \bigcup_{i=1}^{k} \left\{ \overline{(\bar{X} \oplus R_i) \cup (X \oplus \bar{R}_i) \cup \bar{I}} \oplus Q_i \right\},$$

where $k \leq l$, $l = \#(P(W))$ is the cardinality of $P(W)$.

APPENDIX C

Proof of Theorem 2.2. This can be shown in a very straightforward way. Any image is a set of image points and is the union of point images (consisting of one and only one image point). A point image $\{(i, j)\}$ can be written as

$$\{(i, j)\} = A^i B^j.$$

Hence, the union of all point images which are contained in X is the image X. For example, an image $X = \{(2,0), (1,-1), (-1,2)\}$ is denoted by

$$X = A^2 \cup AB^{-1} \cup A^{-1}B^2.$$

APPENDIX D

1. Properties of Complement and Difference

The complement $\bar{\ }$, a unary operation, is decreasing and shift variant (considering the outside of an image). The difference X/R, a binary operation, is increasing (but decreasing with respect to the reference image R), antiextensive with respect to X, and shift variant (the reference image R is fixed once it is given). Note that the difference operation is not commutative, not associative, and not distributive over other operations. Furthermore, the difference operation is more complicated than the complement. Hence, it is preferable to employ the complement as a fundamental operation, but not the difference. The major properties of the complement and the difference are listed in the following:

1. $\overline{X} = W/X$
2. $X/R = X \cap \overline{R}$
3. $\overline{\varnothing} = W$
4. $\overline{W} = \varnothing$
5. $\overline{\overline{X}} = X$ (idempotent for twice complements)
6. $X/\varnothing = X$ (idempotent for a given reference image $R = \varnothing$)
7. $X/X = \varnothing$
8. $X \subset Y \leftrightarrow \overline{Y} \subset \overline{X}$ (decreasing)
9. $X \subset Y \leftrightarrow X/R \subset Y/R$ (increasing)
10. $X/R \subset X$ (antiextensive)
11. $X \subset R \leftrightarrow X/R = \varnothing$
12. $\overline{X \cap R} = \overline{X} \cup \overline{R}$
13. $\overline{X \cup R} = \overline{X} \cap \overline{R}$
14. $X \cap \overline{X} = \varnothing$
15. $X \cup \overline{X} = W$
16. $\overline{X \oplus R} = \overline{X} \ominus \check{R}$, where $\check{R} = \{(-x, -y) | (x, y) \in R\}$
17. $\overline{X \oplus R} = \overline{X} \ominus \check{R}$, where $\check{R} = \{(-x, -y) | (x, y) \in R\}$.

2. Properties of Union and Intersection

The union $\cup$, a binary operation, is increasing, extensive, shift variant, idempotent, commutative, associative, and distributive over intersection. The intersection $\cap$, a binary operation, is increasing, antiextensive, shift variant, idempotent, commutative, associative, and distributive over union. The major properties of the union and the intersection are listed in the following:

1. $X \cup \varnothing = X$
   $X \cap \varnothing = \varnothing$
2. $X \cup X = X$
   $X \cap X = X$
3. $X \cup R = R \cup X$ (commutative)
   $X \cap R = R \cap X$ (commutative)
4. $X \cup (R \cup Q) = (X \cup R) \cup Q$ (associative)
   $X \cap (R \cap Q) = (X \cap R) \cap Q$ (associative)
5. $X \cup W = W$
   $X \cap W = X$ (idempotent for a given reference image $R = W$)
6. $X \cup (R \cap Q) = (X \cup R) \cap (X \cup Q)$ (distributive)
   $X \cap (R \cup Q) = (X \cap R) \cup (X \cap Q)$ (distributive)
7. $X \subset X \cup R$ (extensive)
   $X \cap R \subset X$ (antiextensive)
8. $X \subset Y \leftrightarrow X \cup R \subset Y \cup R$ (increasing)
   $X \subset Y \leftrightarrow X \cap R \subset Y \cap R$ (increasing)
9. $X \subset R \leftrightarrow X \cup R = R$
   $X \subset R \leftrightarrow X \cap R = X$
10. $R \subset X \wedge Q \subset X \rightarrow R \cup Q \subset X$
    $X \subset R \wedge X \subset Q \rightarrow X \subset R \cup Q$
11. $R \subset X \wedge Q \subset Y \rightarrow R \cup Q \subset X \cup Y$
    $R \subset X \wedge Q \subset Y \rightarrow R \cap Q \subset X \cap Y$.

3. Properties of Dilation and Erosion

The dilation $\oplus$, a binary operation, is increasing, extensive for a given reference image R which contains the elementary image I, shift invariant, commutative, associative, distributive over union, and possesses an identity which is I. The erosion $\ominus$, a binary operation, is shift invariant, increasing (but decreasing with respect to the reference image R), and antiextensive for a given reference image R which contains the elementary image I. But, in general, the erosion is not commutative, not associative, not distributive over other operations, and does not possess a left identity. The major properties of the union and the intersection are listed in the following:

1. $X \oplus R = R \oplus X$ (commutative)
   $X \ominus R \neq R \ominus X$ (in general)
2. $(X \oplus R) \oplus Q = X \oplus (R \oplus Q)$ (associative)
   $(X \ominus R) \ominus Q \neq X \ominus (R \ominus Q)$ (in general)
   $(X \ominus R) \ominus Q = (X \ominus Q) \ominus R$
3. $X \oplus (R \cup Q) = (X \oplus R) \cup (X \oplus Q)$ (distributive)
   $X \ominus (R \cup Q) = (X \ominus R) \cap (X \ominus Q)$
   $X \ominus (R \oplus Q) = (X \ominus R) \ominus Q$
4. $X \oplus I = X = I \oplus X$ (identity)
   $X \ominus I = X \neq I \ominus X$ (in general)

5. $X \oplus \varnothing = \varnothing = \varnothing \oplus X$
$X \ominus \varnothing = W \neq \varnothing \ominus X$ (in general)

6. $X \subset X \oplus R$ when $I \subset R$ (extensive)
$X \ominus R \subset X$ when $I \subset R$ (antiextensive)

7. $X \subset Y \leftrightarrow X \oplus R \subset Y \oplus R$ (increasing)
$X \subset Y \leftrightarrow X \ominus R \subset Y \ominus R$ (increasing)

8. $R \subset Q \leftrightarrow X \oplus R \subset X \oplus Q$
$R \subset Q \leftrightarrow X \ominus Q \subset X \ominus R$

9. $X \oplus (R \cap Q) \subset (X \oplus R) \cap (X \oplus Q)$ (distributive inequality)
$X \ominus (R \cap Q) \supset (X \ominus R) \cup (X \ominus Q)$
$(X \cup Y) \ominus R \supset (X \ominus R) \cap (Y \ominus R)$
$(X \ominus R) \oplus Q \subset (X \oplus R) \ominus Q$
Remark. " $\supset$ " means "contains."

4. *Properties of Some Standard Operations*

1. The symmetric difference is shift variant (with a fixed reference image R), commutative, and associative. Symbolically,

(a) $X \triangle R = R \triangle X$

(b) $X \triangle (R \triangle Q) = (X \triangle R) \triangle Q$

(c) $X \triangle \varnothing = X$

(d) $X \triangle X = \varnothing$

(e) $X \triangle \overline{X} = W$

(f) $X \triangle W = \overline{X}$

(g) $X \cap (R \triangle Q) = (X \cap R) \triangle (X \cap Q)$

(h) $X \cup (R \triangle Q) \neq (X \cup R) \triangle (X \cup Q)$ (in general)

(i) $X \triangle R = Y \triangle R \rightarrow X = Y$.

2. The opening $\circ$ is shift invariant, increasing, antiextensive, and idempotent. The closing $\cdot$ is shift invariant, extensive, and idempotent. Symbolically,

(a) $X \circ R \subset X \subset X \cdot R$

(b) $X \subset Y \rightarrow X \circ R \subset Y \circ R$

(c) $X \subset Y \rightarrow X \cdot R \subset Y \cdot R$

(d) $(X \circ R) \circ R = X \circ R$

(e) $(X \cdot R) \cdot R = X \cdot R$.

3. The thinning is shift invariant and antiextensive. The thickening is shift invariant and extensive. The major properties are in the following:

(a) $X \circledcirc R \subset X \subset X \odot R$

(b) $X \subset Y \rightarrow X \circledcirc R \subset Y \circledcirc R$

(c) $X \subset Y \rightarrow X \odot R \subset Y \odot R$.

(d) If $R \subset Q$ (which means $R_1 \subset Q_1$ and $R_2 \subset Q_2$), then we have

$$R \subset Q \rightarrow X \circledcirc R \subset X \circledcirc Q \subset X \subset X \odot Q \subset X \odot R.$$

(e) $\overline{X \odot R} = \overline{X} \circledcirc R^*$, where $R = \{R_1, R_2\}$ and $R^* = \{R_2, R_1\}$.

APPENDIX E

Proof of Theorem 3.2. We can easily see that (2) in Theorem 3.2 is a generalization of (1) in Theorem 3.2. (1) is used for exactly matching shapes (or templates) with shift invariance; (2) is generalized to more general cases. For example, to consider noise and to have rotational invariance, we can choose the family $\{R(\theta)\}$ to incorporate all aspect reference image pairs. In the following, we prove (1) and then (2) will follow from it directly. The proof will demonstrate the mathematical correspondence between boolean logic and BIA. The notations $x(i, j)$ and $r(i, j)$ will be used to represent the binary values (0 or 1) of pixels at coordinate (i, j) of image functions which correspond to the images X and R in BIA notations.

First, let us use the boolean logic XOR (exclusive or) operation, i.e.,

$$x(i, j)\text{XOR}\, r(i, j) = (\bar{x}(i, j) \wedge r(i, j)) \vee (x(i, j) \wedge \bar{r}(i, j)),$$

to achieve the pixelwise comparison, where the output value with "0" means that "$x(i, j)$" matches "$r(i, j)$" and the output value with "1" means that "$x(i, j)$" does not match "$r(i, j)$."

Second, to check the occurrence of the shape (defined by R with M) in the tested image X at coordinate (i, j), we have to shift the origin of the shape to the coordinate (i, j) in X. Then the process of the comparison of the shape and the subimage in X (limited in the mask M) and the indication of "match" (0) and "not match" (1) will be performed by

$$\bigvee_{(k,l)\in M} (\bar{x}(i+k, j+l) \wedge r(k,l)) \vee \bigvee_{(k,l)\in M} (x(i+k, j+l) \wedge \bar{r}(k,l)).$$

If the above equation is considered as a binary operation operating on two images $x(i, j)$ and $r(i, j)$, then this operation is not commutative; in order to achieve the commutativity, we change (k, l) with $(-k, -l)$ and denote $\check{r}(k, l) = r(-k, -l)$:

$$\bigvee_{(-k,-l)\in \check{M}} (\bar{x}(i-k, j-l) \wedge \check{r}(k,l)) \vee \bigvee_{(-k,-l)\in \check{M}} (x(i-k, j-l) \wedge \bar{\check{r}}(k,l)).$$

If the output value of the above equation is "0," then it means that the location (i, j) of the image X has the occurrence of the shape (defined by R and M); if "1," the shape does not occur at (i, j).

Third, let us run over all coordinates (i, j) (i.e., for all $(i, j) \in W$ the universal image) and then the union of those coordinates with value "0" would be the answer. The value "0" at a coordinate (i, j) corresponds to the null image in set notation and the value "1" at a coordinate (i, j) corresponds to the point image $\{(i, j)\}$. For convenience, in the following we mix the notations of boolean logic functions and set notations; if the output of a boolean logic expression is "0," it represents the null image $\varnothing$; if "1," it represents the point image $\{(i, j)\}$. Thus, we have

$$\bigcup_{(i,j)\in W} \left(\bigvee_{(-k,-l)\in \check{M}} (\bar{x}(i-k, j-l) \wedge \check{r}(k,l)) \vee \right.$$
$$\left. \bigvee_{(-k,-l)\in \check{M}} (x(i-k, j-l) \wedge \bar{\check{r}}(k,l)) \right)$$

which is the same as

$$\bigcup_{(i,j)\in W}\left(\bigvee_{(-k,-l)\in \check{M}}\left(\bar{x}(i-k,j-l)\wedge \check{r}(k,l)\right)\right)$$

$$\cup \bigcup_{(i,j)\in W}\left(\bigvee_{(-k,-l)\in \check{M}}\left(x(i-k,j-l)\wedge \bar{\check{r}}(k,l)\right)\right).$$

Since $x(i, j) \neq 0$ only when $(i, j) \in X$ and $\check{r}(k,l) \neq 0$ only when $(k,l) \in \check{R}$, we have

$$\begin{aligned}\bigcup_{(i,j)\in W}\left(\bigvee_{(-k,-l)\in \check{M}}\left(\bar{x}(i-k,j-l)\wedge \check{r}(k,l)\right)\right) &= \left\{(i,j)|(i-k,j-l)\in \bar{X},(k,l)\in \check{R}\right\}\\ &= \left\{(i+k,j+l)|(i,j)\in \bar{X},(k,l)\in \check{R}\right\}\\ &= \bar{X}\oplus \check{R}.\end{aligned}$$

Similarly, we have

$$\bigcup_{(i,j)\in W}\left(\bigvee_{(-k,-l)\in \check{M}}\left(x(i-k,j-l)\wedge \bar{\check{r}}(k,l)\right)\right) = X\oplus(\check{M}/\check{R}).$$

Hence, if we use "0" to indicate "match," we have

$$(\bar{X}\oplus \check{R})\cup\left(X\oplus(\check{M}/\check{R})\right);$$

if we use "1" to indicate "match," then we have

$$\overline{(\bar{X}\oplus \check{R})\cup\left(X\oplus(\check{M}/\check{R})\right)}.$$

Thus, the locations of a shape, which is defined by a nonnull reference image R with a nonnull reference image (called mask) M and $R \subset M \subset W$, are the image points in the following

$$\begin{aligned}\overline{(\bar{X}\oplus \check{R})\cup\left(X\oplus(\check{M}/\check{R})\right)} &= \overline{(\bar{X}\oplus \check{R})\cup\left(X\oplus\overline{\bar{\check{M}}\cup \check{R}}\right)}\\ &= (X\ominus R)\cap\left(\bar{X}\ominus(M/R)\right).\end{aligned}$$

A more intuitive illustration is that the foreground X should match R by $X\ominus R$ (using multiple-input AND gates to examine the locations where the 1's should be), while the background $\bar{X}$ should match M/R by $\bar{X}\ominus(M/R)$. Combining both results by the intersection (AND), we then implement the shape recognition by $(X\ominus R)\cap(\bar{X}\ominus(M/R))$. Replacing R by R_1 and (M/R) by R_2, we obtain the hit or miss transform (template matching) for shape recognition.

APPENDIX F

Proof of Theorem 3.3. (1) The straightforward way for removing the "pepper" noise is the thinning operation $X \circledcirc R_4$ (or $X \circledcirc R_8$). Following this, we have

$$\begin{aligned}
X \circledcirc R_4 &= \overline{\bar{X} \cup \overline{(\bar{X} \oplus I) \cup (X \oplus M_4)}} \\
&= \overline{\bar{X} \cup \overline{\bar{X} \cup (X \oplus M_4)}} \\
&= \overline{\bar{X} \cup \left(X \cap \overline{X \oplus M_4}\right)} \\
&= \overline{(\bar{X} \cup X) \cap \left(\bar{X} \cup \overline{X \oplus M_4}\right)} \\
&= \overline{W \cap \left(\bar{X} \cup \overline{X \oplus M_4}\right)} \\
&= \overline{\bar{X} \cup \overline{X \oplus M_4}}.
\end{aligned}$$

(2) The straightforward way for removing the "pepper" noise is the thickening operation $X \odot Q_4$ (or $X \odot Q_8$). Following this, we have

$$\begin{aligned}
X \odot Q_4 &= X \cup \overline{(\bar{X} \oplus M_4) \cup (X \oplus I)} \\
&= X \cup \overline{(\bar{X} \oplus M_4) \cup X} \\
&= X \cup \left(\overline{\bar{X} \oplus M_4} \cap \bar{X}\right) \\
&= \left(X \cup \overline{\bar{X} \oplus M_4}\right) \cap (X \cup \bar{X}) \\
&= \left(X \cup \overline{\bar{X} \oplus M_4}\right) \cap W \\
&= \left(X \cup \overline{\bar{X} \oplus M_4}\right).
\end{aligned}$$

(3) The straightforward way for removing the "salt and pepper" noise is to take the difference of $X \circledcirc Q_4$ by $X \circledast R_4$ (or the difference of $X \circledcirc Q_8$ by $X \circledast R_8$). By a similar procedure as above we can achieve the desired result.

APPENDIX G

Proof of Theorem 3.4. To extract the region whose sizes are between two reference images R and Q, the straightforward way is to design a morphological band pass filter:

$$(X \circ R)/(X \circ Q) = ((X \ominus R) \oplus R)/((X \ominus Q) \oplus Q).$$

To obtain the locations of those desired regions, we then perform the skelotonization:

$$\begin{aligned}
S(((X \ominus R) \oplus R)/((X \ominus Q) \oplus Q)) &= S((X \ominus R)/((X \ominus Q) \oplus Q)) \\
&= S\left(\overline{(\bar{X} \oplus \check{R}) \cup \left(\overline{\bar{X} \oplus \check{Q}} \oplus Q\right)}\right).
\end{aligned}$$

ACKNOWLEDGMENTS

The authors would like to thaı.k T. C. Strand, P. Chavel, E. K. Blum, and R. Chellappa for their contributions to this research.

REFERENCES

1. K. S. Huang, B. K. Jenkins and A. A. Sawchuk, Optical cellular logic architectures based on binary image algebra, in *Proceedings, IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence, Seattle, October, 1987*, pp. 19–26.
2. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, New York, 1982.
3. R. M. Lougheed, D. L. McCubbrey, and S. R. Sternberg, Cytocomputers: Architectures for parallel image processing, *Proceedings, IEEE Workshop Picture Data Description and Management, CA, 1980*, pp. 281–286.
4. G. Matheron, *Random Sets and Integral Geometry*, Wiley, New York, 1975.
5. R. M. Haralick, S. R. Sternberg, and X. Zhuang, Image analysis using mathematical morphology, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-9**, No. 4, 1987, 532–550.
6. K. Preston and M. J. B. Duff, *Modern Cellular Automata—Theory and Applications*, Plenum, New York, 1984.
7. A. Burks (Ed.), *Essays on Cellular Automata*, Univ. of Illinois Press, Urbana, 1970.
8. G. X. Ritter and P. D. Gader, Image algebra techniques for parallel image processing, *J. Parallel Distrib. Comput.*, **4**, No. 1, 1987, 7–44.
9. C. R. Giardina, The universal imaging algebra, *Pattern Recognit. Lett.* **2**, 1984, 165–172.
10. T. Agui *et al.*, An algebraic approach to the generation and description of binary pictures, *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-4**, No. 6, 1982, 635–641.
11. S. Sternberg, Biomedical image processing, *IEEE Comput.*, Jan. 1982, 22–34.
12. S. Sternberg, An overview of image algebra and related architectures, in *Integrated Technology for Parallel Image Processing* (S. Levialdi, Ed.), pp. 79–100, Academic Press, New York, 1985.
13. J. Klein and J. Serra, The texture analyzer, *J. Microscopy* **95**, No., 2, 1972, 349–356.
14. J. Mandeville, *Novel Method for Automated Optical Inspection of Printed Circuits*, IBM Research Report RC-9900, IBM T. J. Waston Research Center, Yorktown Heights, 1983.
15. Morphological systems software, in *Computer Architecture for Pattern Analysis and Image Database Management, 1985 IEEE Computer Society Workshop*, pp. 429–468.
16. Morphological systems hardware, in *Computer Architecture for Pattern Analysis and Image Database Management, 1985 IEEE Computer Society Workshop*, pp. 469–500.
17. P. W. Verbeek, Implementation of cellular-logic operators using 3 * 3 convolution and table lookup hardware, *Comput. Vision Graphics Image Process.* **27**, 1984, 115–123.
18. S. R. Sternberg and J. Serra (Eds.), Special section on mathematical morphology, *Comput. Vision Graphics Image Process.* **35**, 1986, 273–383.
19. K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, An image algebra representation of parallel optical binary arithmetic, *Appl. Opt.*, in press.
20. J. von Newmann, The general logical theory of automata, in *Cerebral Mechanisms in Behavior—The Hixon Symposium* (L. A. Jeffress, Ed.), Wiley, New York, 1951.
21. J. von Neumann, *Theory of Self-reproducing Automata* (A. W. Burks, Ed.), Univ. of Illinois Press, Urbana, 1966.
22. S. H. Unger, A computer oriented toward spatial problems, *Proc. IRE* **46**, 1958, 1744–1750.
23. K. Preston, Jr. *et al.*, Basics of cellular logic with some applications in medical image processing, *Proc. IEEE* **67**, No. 5, 1979, 826–856.
24. A. Rosenfeld, Parallel image processing using cellular arrays, *IEEE Comput.*, Jan. 1983, 14–20.
25. K. Preston, Jr., Cellular logic computers for pattern recognition, *IEEE Comput.*, Jan 1983, 36–47
26. B. K. Jenkins and A. A. Sawchuk, Optical cellular logic architectures for image processing, in *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, FL, Nov. 1985*, pp. 61–65.
27. A. A. Sawchuk and B. K. Jenkins, Optical cellular logic processors, in *Optical Society of America 198: Annual Meeting, Washington, DC.*
28. A. A. Sawchuk and T. C. Strand, Digital optical computing, *Proc. IEEE* **72**, 1984, 758–779
29. B. K. Jenkins *et al.*, Architectural implications of a digital optical processor, *Appl. Opt.*, **23**, No. 19 1984, 3465–3474.

30. G. Birkhoff and T. C. Bartee, *Modern Applied Algebra*, McGraw-Hill, New York, 1970.
31. G. Birkhoff and S. MacLane, *A Brief Survey of Modern Algebra*, Macmillan Co., New York, 1965.
32. G. Gilbert, *Modern Algebra with Applications*, Wiley, New York, 1976.
33. A. Rosenfeld, Connectivity in digital pictures, *J. Assoc. Comput. Mach.* **17**, 1970, 146-160.
34. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New York/London, 1982.
35. R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*, Wiley, New York, 1973.
36. S. Levialdi, On shrinking of binary patterns, *Commun. ACM* **15**, 1972, 7-10.
37. A. Huang, Parallel algorithms for optical digital computers, in *Technical Digest, IEEE Tenth International Optical Computing Conference, 1983*, pp. 13-17.
38. K.-H. Brenner, A. Huang, and N. Streibl, Digital optical computing with symbolic substitution, *Appl. Opt.* **25**, 1986, 3054-3060.
39. A. V. Oppenheim and R. W. Schafer, *Digital Signal Processing*, Prentice-Hall, Englewood Cliffs, NJ, 1975.
40. W. K. Pratt, *Digital Image Processing*, Wiley, New York, 1978.
41. J. W. Goodman, *Introduction to Fourier Optics*, McGraw-Hill, New York, 1968.
42. K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, A cellular hypercube architecture for image processing, *Proc. Soc. Photo-Opt. Instr. Eng.* **829**, August 1987, pp. 331-338.
43. B. K. Jenkins *et al.*, Sequential optical logic implementation, *App. Opt.* **23**, No. 19, 1984, 3455-3464.
44. T. E. Bell, Optical computing: A field in flux, *IEEE Spectrum* **23**, No. 8, 1986, 34-57.
45. K. S. Huang, A Digital Optical Cellular Image Processor (DOCIP): Theory, Architecture and Implementation, Ph.D. Thesis, University of Southern California, Los Angeles, 1988.

Reprinted from Applied Optics

Image algebra representation of parallel optical binary arithmetic

Kung-Shiuh Huang, B. Keith Jenkins, and Alexander A. Sawchuk

A binary image algebra (BIA) that gives a mathematical description of parallel processing operations is described. Rigorous and concise BIA representations of parallel arithmetic and symbolic substitution operations are given. A sequence of programming steps for implementation of these operations on a parallel architecture is specified by the BIA representation. Examples of arithmetic operations implemented on a digital optical cellular image processor architecture are given.

I. Introduction

Digital optical systems hold the promise of providing more accuracy, flexibility, and programmability than analog optical systems, at the cost of somewhat lower throughput.[1,2] To achieve digital optical computing, there are at least three possible logic systems: residue logic,[3-6] multilevel logic,[7-10] and binary logic.[11,12] Because it is much easier to make reliable two level devices for binary logic and only $\log_2 k$ of them are needed to represent k levels, in this paper we consider only binary parallel optical computing. A digital optical cellular image processor (DOCIP) architecture based on binary image algebra (BIA) has been demonstrated to be very powerful in parallel binary image processing.[13-16] This paper demonstrates that the DOCIP with BIA algebraic techniques can efficiently perform parallel numerical computations also.

Boolean logic equations for binary arithmetic are not well suited to highly parallel operations on planes of data; they do not reflect the location of data except typically by a memory address. Here we first seek a software theory for parallel numerical computation algorithms that simultaneously have binary digital efficiency and the advantages of optical parallel processing. We have developed a binary image algebra (BIA),[16] built from only three fundamental operations and five elementary images, to serve as a complete unified systematic theory for binary pa--llel image processing. Now, we show that BIA can also be considered as a spatial logic which is a generalized parallel form of Boolean logic with an additional parallel information transfer ability. BIA then becomes a formalism and a general technique for developing and comparing parallel numerical computation algorithms for digital optical computers. Previous discussions have relied solely on pictorial descriptions of parallel arithmetic operations. BIA provides a rigorous and concise mathematical description of parallel operations. In this paper we give these rigorous BIA descriptions for parallel addition, subtraction, and multiplication.

Symbolic substitution has been considered as a means for implementing parallel optical arithmetic operations.[17-19] Symbolic substitution rules can be described as particular BIA image transformations (Sec. 5).[20] Three different binary number representations (row-coding, stack-coding, and symbol-coding as originally described in Refs. 17–19) for binary arithmetic in the DOCIP machine are developed. Parallel operations of binary addition, subtraction, and multiplication are derived by BIA and illustrated as examples. Parallelism is achieved by performing arithmetic operations on many pairs of operands simultaneously. The carries for each pair of operands are essentially propagated serially to keep hardware complexity low.[21] Thus speed-ups close to linear, and in some cases equal to linear can be obtained. In this paper we consider only positive numbers. A suitable digital number representation will easily provide for negative numbers also. For example, two's complement arithmetic can be performed with only minor modifications to the algorithms and programs given in this paper, and with the addition of one more bit (the sign bit) to each operand and result.

Section 2 gives a brief review of BIA and the DOCIP architecture. Section 3 presents binary row-coded

When this work was done all authors were with University of Southern California, Department of Electrical Engineering, Signal & Image Processing Institute, Los Angeles, California 90089-0272; K. S. Huang is now with IBM T. J. Watson Research Center, Computer Science Department, P.O. Box 704, Yorktown Heights, New York 10598.

Received 11 February 1988.
0003-6935/89/061263-16$02.00/0.

arithmetic: binary addition and binary multiplication (including a matrix-constant multiplication and an element-element multiplication). Section 4 presents binary stack-coded arithmetic. Section 5 gives a BIA representation of symbolic substitution and discusses binary symbol-coded arithmetic. Section 6 gives a comparison for the above different number representations. Binary subtraction is presented in the Appendices for clarity.

2. Binary Image Algebra (BIA) and DOCIP Architecture

2.1 Review of Binary Image Algebra

Binary image algebra (BIA), extending from mathematical morphology,[22] is a synthesis of Boolean logic, set theory, and image processing. We give here a very brief summary of BIA. Details are contained in Ref. 16.

A binary digital image is usually defined as a function f mapping a spatially sampled set of grid points (x,y) of an orthogonal coordinate system onto the set composed of two elements: 1 and 0. However, it will be more convenient for our image algebra to use only the set of coordinates of pixels that have value 1 to specify an image. In BIA, an image is then treated as a set of coordinates of pixels that have value 1. This paper deals with only binary arithmetic; hence, a pixel represents a binary bit and an image is a finite 2-D bit plane. We list here only those basic definitions and operations which will be referred to later.

Definition of Binary Image Algebra (BIA)

Binary image algebra is an algebra with an image space S and a family F of five elementary images and three fundamental operations. Symbolically,

$$\mathrm{BIA} = [P(W); \oplus, \cup, -, I, A, A^{-1}, B, B^{-1}], \tag{1}$$

where $S = P(W)$ and $F = (\oplus, \cup, -, I, A, A^{-1}, B, B^{-1})$. The image space S, the family F, and all other symbols are defined in the following.

(1) The Universal Image (the bit plane containing all bits with value 1): The universal image is a set $W = \{(x,y)|x \in Z_n, y \in Z_n\}$, where $Z_n = \{0, \pm 1, \pm 2, \ldots, \pm n\}$ and n is a positive integer.

(2) Image Space (the set of all possible bit planes): The image space is the power set (the set of all subsets) of the universal image, i.e., $S = P(W)$.

(3) Image (bit plane): A set X is an image if and only if X is an element of the image space S, i.e., X is a subimage of the universal image W.

(4) Image Point (a bit with value 1): A sampled point (bit) (x,y) is an image point of an image X if and only if (x,y) is an element of the set X.

(5) Image Transformation (a mapping between bit planes): An image transformation T is a function mapping the image space S into the image space S.

(6) Three Fundamental Operations (Fig. 1):

(a) Complement of an image X:

$$\begin{array}{c} \boxed{\begin{matrix} 0&0&0&0&0&0&0\\ 0&0&1&1&1&1&0\\ 0&0&0&1&1&1&0\\ 0&0&0&0&1&1&0\\ 0&0&0&0&0&1&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{matrix}} \\ \text{Tested Image} \quad X \end{array} \qquad \begin{array}{c} \boxed{\begin{matrix} 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0\\ 0&0&1&1&1&0&0\\ 0&0&1&1&1&0&0\\ 0&0&1&1&1&0&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{matrix}} \\ \text{Reference Image} \quad R \end{array}$$

$$\begin{array}{c} \boxed{\begin{matrix} 1&1&1&1&1&1&1\\ 1&1&0&0&0&0&1\\ 1&1&1&0&0&0&1\\ 1&1&1&1&0&0&1\\ 1&1&1&1&1&0&1\\ 1&1&1&1&1&1&1\\ 1&1&1&1&1&1&1 \end{matrix}} \\ \text{Complement} \quad \overline{X} \end{array} \qquad \begin{array}{c} \boxed{\begin{matrix} 0&0&0&0&0&0&0\\ 0&0&1&1&1&1&0\\ 0&0&1&1&1&1&0\\ 0&0&1&1&1&1&0\\ 0&0&1&1&1&1&0\\ 0&0&0&0&0&0&0\\ 0&0&0&0&0&0&0 \end{matrix}} \\ \text{Union} \quad X \cup R \end{array} \qquad \begin{array}{c} \boxed{\begin{matrix} 0&1&1&1&1&1&1\\ 0&1&1&1&1&1&1\\ 0&1&1&1&1&1&1\\ 0&0&1&1&1&1&1\\ 0&0&0&1&1&1&1\\ 0&0&0&0&1&1&1\\ 0&0&0&0&0&0&0 \end{matrix}} \\ \text{Dilation} \quad X \oplus R \end{array}$$

Fig. 1. Example of fundamental operations: complement $-$, union $\cup$, and dilation $\oplus$.

$$\overline{X} = \{(x,y)|(x,y) \in W \wedge (x,y) \notin X\}. \tag{2}$$

(b) Union of two images X and R:

$$X \cup R = \{(x,y)|(x,y) \in X \vee (x,y) \in R\}. \tag{3}$$

(c) Dilation of two images X and R:

$$X \oplus R = \begin{cases} \{(x_1 + x_2, y_1 + y_2) \in W|(x_1,y_1) \in X, (x_2,y_2) \in R\} & (X \neq \varnothing) \wedge (R \neq \varnothing), \\ \varnothing & \text{otherwise.} \end{cases} \tag{4}$$

Remark: $\in$ denotes belongs to, $\wedge$ denotes and, $\vee$ denotes or, and $\varnothing$ is the null image having no image point. Note that X usually represents an input image and R is a reference image containing predefined information. We can define other image operations as fundamental operations instead of these three operations. The reason for choosing these three operations is because of their simplicity, simple software design and simple hardware implementation. Dilation can be interpreted as a parallel mathematical formalism of the pattern substitution step in symbolic substitution (Sec. 5).

(7) Five Elementary Images: There are five elementary images:

(a) $I = \{(0,0)\}$—consisting of an image point at the origin,

(b) $A = \{(1,0)\}$—consisting of an image point right of the origin,

(c) $A^{-1} = \{(-1,0)\}$—consisting of an image point left of the origin,

(d) $B = \{(0,1)\}$—consisting of an image point above the origin,

(e) $B^{-1} = \{(0,-1)\}$—consisting of an image point below the origin.

In fact, these five elementary images could be reduced to four elementary images, because $I = A \oplus A^{-1} = B \oplus B^{-1}$. Any (reference) image can be represented as

$$X = \bigcup_{(i,j) \in X} A^i B^j, \tag{5}$$

where $A^iB^j \equiv A^i \oplus B^j$,

$$A^i = \underbrace{A \oplus A \oplus \ldots \oplus A}_{} = \{(i,0)\} \text{ if } i > 0,$$

$$A^i = \underbrace{A^{-1} \oplus A^{-1} \oplus \ldots \oplus A^{-1}}_{} = \{(i,0)\} \text{ if } i < 0,$$

$$A^0 = A \oplus A^{-1} = I.$$

(8) Reflected Image: Given an image R, its reflected image is defined as

$$\check{R} = \{(-x,-y)|(x,y) \in R\}. \qquad (6)$$

(9) Some Standard Derived Operations:

(a) Difference of X by R [Fig. 2(a)]:

$$X/R = \{(x,y)|(x,y) \in X \wedge (x,y) \notin R\} = X \cap \bar{R} = \overline{\bar{X} \cup R}. \qquad (7)$$

Remark: $\bar{X} = W/X$ where W is the universal image.

(b) Intersection of two images X and R [Fig. 2(b)]:

$$X \cap R = \{(x,y)|(x,y) \in X \wedge (x,y) \in R\} = \overline{\bar{X} \cup \bar{R}}. \qquad (8)$$

Remark: $X \cup R = \overline{\bar{X} \cap \bar{R}}$.

(c) Erosion of an image X by a reference image R [Fig. 2(c)]:

$$X \ominus R = \overline{\bar{X} \oplus \check{R}}, \qquad (9)$$

where $\check{R}$ is defined above. Remark: $X \oplus R = \overline{\bar{X} \ominus \check{R}}$. The erosion of an image X by a reference image R can be thought as the complement of the dilation of the background by the reflection of the reference image R. In general, the erosion of a non-null image X by a non-null reference image R decreases the size of regions, increases the size of holes, eliminates regions, and breaks bridges in X.

(d) Symmetric difference of two images [Fig. 2(d)]:

$$X \triangle R = (X/R) \cup (R/X) = \overline{\bar{X} \cup R} \cup \overline{\bar{R} \cup X}. \qquad (10)$$

Remark: The symmetric difference is a commutative operation, and is its own inverse.

(e) Hit or miss transform $\circledast$ of an image X by an image pair $R = (R_1,R_2)$ [Fig. 2(e)]:

$$X \circledast R = (X \ominus R_1) \cap (\bar{X} \ominus R_2) = \overline{(\bar{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}. \qquad (11)$$

Remark: The hit or miss transform of an image X by a reference image pair $R = (R_1,R_2)$ formally describes the pattern recognition step in symbolic substitution (Sec. 5); and it is used to match the shape (or template) defined by the reference image pair R where R_1 defines the foreground of the shape and R_2 defines the background of the shape. The conditions are that the foreground X must match R_1 (i.e., $X \ominus R_1$), while simultaneously the background $\bar{X}$ matches R_2 (i.e., $\bar{X} \ominus R_2$). Note the similarity of the symmetric difference (parallel bitwise comparison) and the hit or miss transform (parallel shape or symbol recognition).

The important results of BIA are: (1) any image transformation can be implemented by the three fundamental operations with appropriate reference images; (2) any reference image can be generated from the elementary images by using the three fundamental operations; and (3) BIA provides an efficient representation for many parallel image processing algorithms (e.g., shape and size verifications[16]). Here we demonstrate that BIA is also a fundamental tool for parallel numerical computation.

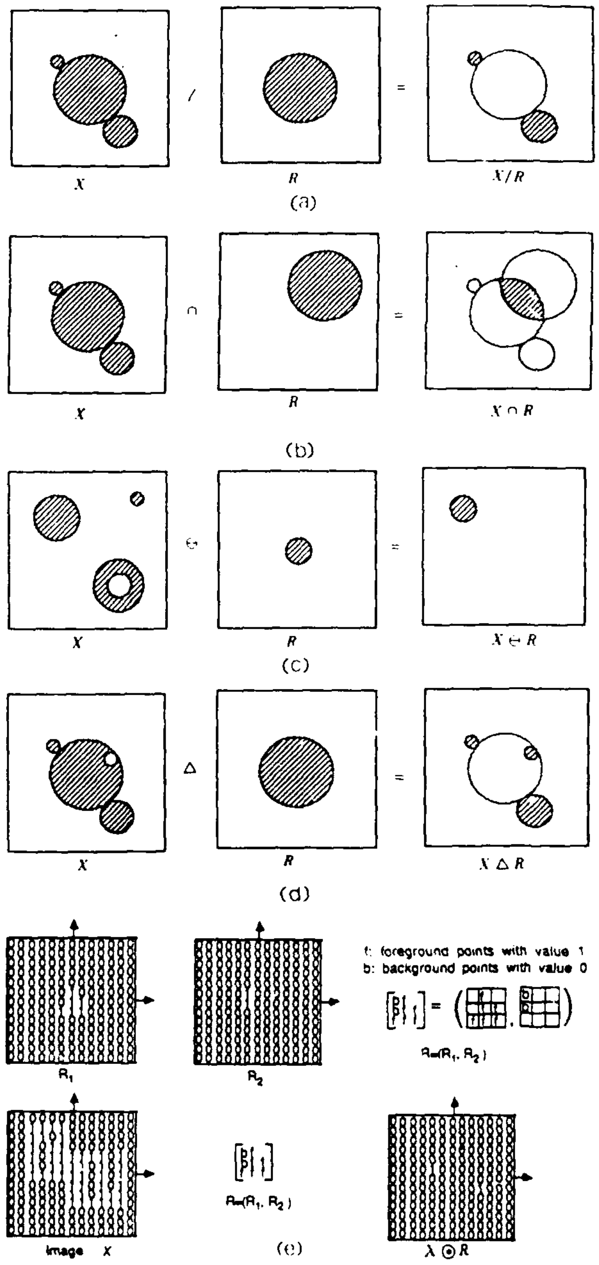


Fig. 2. Some standard derived image operations. The shaded regions in (1)-(d) correspond to pixels with value 1: (a) difference; (b) intersection; (c) erosion; (d) symmetric difference; (e) hit or miss transform (template matching).

2.2. Review of DOCIP Architecture

We have designed a class of the digital optical cellular image processors (DOCIPs) for effectively implementing BIA.[13-15] Here we only summarize their major characteristics. Details are given in Refs. 14 and 15. To map BIA into the DOCIP architecture in a transparent way, we first define the DOCIP algebraically:

Definition of Cellular Automata

A cellular automaton is an algebra $A = (S;F,N_c)$ where S is the state space which is a set of states, F is a family of transition functions, and N_c is the neighborhood configuration.

Constraints on a cellular automaton for Implementing BIA:

(1) $S \supset P(W)$,

(2) $F \supset \{\oplus, \cup, -\}$,

(3) $N_c \supset I \cup A \cup A^{-1} \cup B \cup B^{-1}$ or $N_c \supset A \cup A^{-1} \cup B \cup B^{-1}$,

where $\supset$ means contains.

Thus, in terms of cellular automata, the DOCIPs have to satisfy the above constraints for realizing BIA. For storing input images and temporary results in a more flexible way, the DOCIPs utilize three memory modules and all share the same algebraic structure (except the neighborhood configuration):

$$\mathrm{DOCIP} = [P(W \times W \times W);\oplus,\cup,-,N_c], \tag{12}$$

where $\times$ denotes cross product and N_c can be one of the following four types:

(1) DOCIP-array4: each cell connects with its four nearest neighbors and itself, i.e.,

$$N_{\mathrm{array4}} = I \cup A \cup A^{-1} \cup B \cup B^{-1}. \tag{13}$$

(2) DOCIP-array8: each cell connects with its eight nearest neighbors and itself, i.e.,

$$N_{\mathrm{array8}} = \bigcup_{i,j=-1} A^iB^j. \tag{14}$$

(3) DOCIP-hypercube4: each cell connects with those cells in the 4 directions at distances $1,2,4,8,\ldots,2^k$ from itself, i.e.,

$$N_{\mathrm{hypercube4}} = \bigcup_{i=0,\pm1,\pm2,\ldots,\pm2^k} (A^i \cup B^i), \tag{15}$$

where k is sufficiently large for the connections to traverse the entire array of cells.

(4) DOCIP-hypercube8: each cell connects with those cells in the 8 directions at distances $1,2,4,8,\ldots,2^k$ from itself, i.e.,

$$N_{\mathrm{hypercube8}} = \bigcup_{i=0,\pm1,\pm2,\ldots,\pm2^k} (A^i \cup B^i \cup A^iB^i \cup A^iB^{-i}). \tag{16}$$

From the above algebraic description, the DOCIPs have the same algebraic structure and differ only in their neighborhood configurations N_c. Thus, they share the same architecture shown in Fig. 3, but have different configurations of the reference images E_i depending on the optical interconnection network which defines the neighborhood. In practical applications, a larger reference image R can be generated from a set of smaller reference image(s) E_i by a sequential dilation. If it is possible to decompose R into a sequence $R = E_1 \oplus E_2 \oplus \ldots \oplus E_k$, then

$$X \oplus R = [\ldots[(X \oplus E_1) \oplus E_2] \oplus \ldots \oplus E_k]. \tag{17}$$

This decomposition may not exist, in which case R can always be decomposed as $R = R_1 \cup R_2 \cup \ldots \cup R_k$, and then

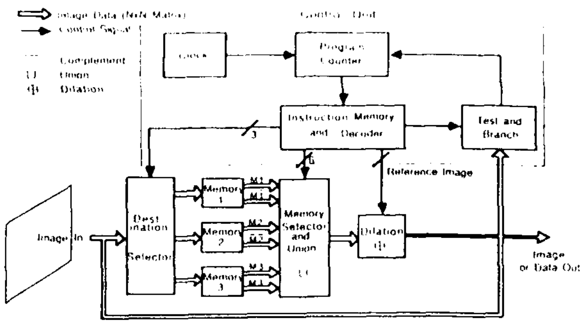


Fig. 3. Digital optical cellular image processor (DOCIP) architecture—one implementation of binary image Algebra (BIA). The DOCIP-array requires 9 (or 5) control bits for reference image E_i. The DOCIP-hypercube requires $O(\log N)$ control bits for reference image E_i.

$$X \oplus R = (X \oplus R_1) \cup (X \oplus R_2) \cup \ldots \cup (X \oplus R_k), \tag{18}$$

where each R_j can be decomposed into smaller reference images E_i.[14,23]

Basically, the proposed DOCIP shown in Fig. 3 is a cellular SIMD machine and consists of an array of cells or processing elements (PEs) under the supervision of a control unit. The control unit includes a clock, a program counter, a test and branch module for feedback control, and an instruction decoder for storing instructions and decoding them to supervise cells. The array of cells includes a 1×3 line destination selector, where each line is N^2 bits wide, three $N \times N \times 1$ bit memories for storing images, a memory selector, and a dilation unit. It operates as follows: (1) a binary image ($N \times N$ matrix) is input into the destination selector and then stored in any memory (or set of memories) as the instruction specifies; (2) after one to three images have been stored, these images and their complements are piped into the next stage, which forms the union of any combination of images (specified by the instruction); (3) the result is sent to a dilation unit where the reference image specified by the instruction is used to control the type of dilation; (4) finally, the dilated image can be output, tested for program control, or fed back to step (1) as the instruction specifies.

The DOCIP machine (Fig. 3) has one instruction; it implements the three fundamental operations of BIA along with fetch and store.[23] This design uses the parallelism of optics to simultaneously execute instructions involving all N^2 picture elements. Each instruction takes one complete cycle to execute. Note that the DOCIP machine can perform a dilation by any reference image R that is a subset of the neighborhood configuration, N_c, in a single clock cycle.

The entire system can be realized by an optical gate array with optical 3-D interconnections.[11,12,24] Figure 4 describes an optical implementation concept for the DOCIP architecture. The DOCIP has very low cell hardware complexity to maximize parallelism, yet enough cell sophistication to permit the machine to

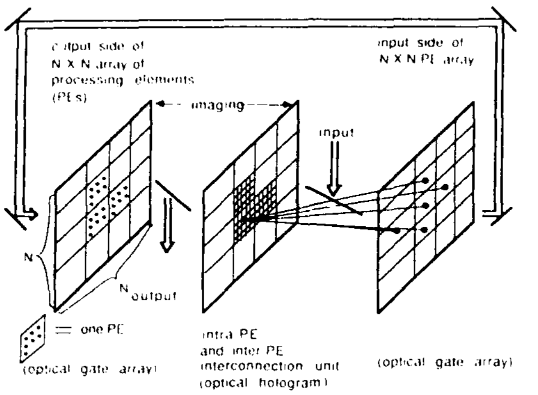


Fig. 4. DOCIP physical concept. Each processing element (PE) or cell connects with its cellular array or cellular hypercube neighbors and itself by optical 3-D interconnections. The optical hologram provides both intracell and intercell interconnections. Intracell interconnections and imaging optics are omtted for clarity. The input and output sides of the optical gate array are interconnected by an optical feedback path and are shown separately for clarity.

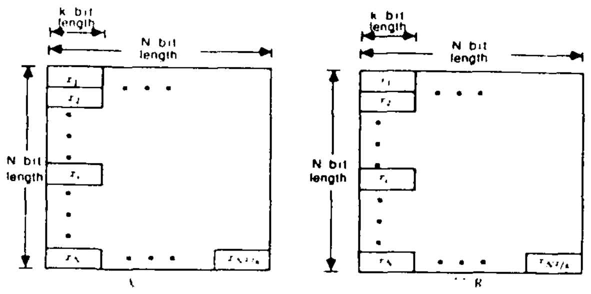


Fig. 5. Binary row-coded numbers.

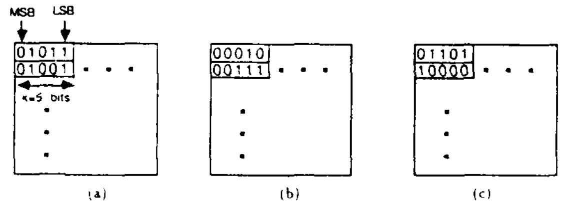


Fig. 6. Parallel addition of binary row-coded numbers (I): (a) image X of operands; (b) image R of other operands; (c) output $X + R$.

execute useful programs. The use of optical interconnections permits a cellular hypercube topology to be implemented without paying a large penalty in chip area (the cellular hypercube interconnections are space invariant which implies relatively low hologram complexity); it also enables images to be input to and output from the machine in parallel.

3. Binary Row-Coded Arithmetic

Binary addition of two k-bit numbers yields at most $k + 1$ bits, and binary multiplication of two k-bit numbers yields at most $2k$ bits. In this paper, we assume that all input numbers are padded with enough zeros to avoid the possibility of overflow. This also guarantees that the different operands in the image will be treated separately. A binary row-coded number is encoded in a part of a row of an image. Although the word lengths of numbers do not need to be equal, we assume in this discussion that an image (bit plane) with $N \times N$ bits contains N^2/k numbers of k-bit length as a simple illustration (Fig. 5). In this section, we describe parallel addition and multiplication by BIA expressions and their programs on the DOCIP machine. Subtraction is discussed in Appendix A.

3.1. Addition of Binary Row-Coded Numbers

Consider an image X [e.g., Fig. 6(a)] composed of N^2/k numbers x_i, $i = 1,2,\ldots,N^2/k$, an image R [e.g., Fig. 6(b)] composed of N^2/k numbers r_i, $i = 1,2,\ldots,N^2/k$, and the output of the addition $S = X + R$ [Fig. 6(c)]. To realize this addition in parallel by means of BIA, we first consider the serial (carry-propagate) addition of 2 binary numbers $s_i = x_i + r_i$. The first step of serial addition is to add the least significant bits, say $x_{i(0)}$ and $r_{i(0)}$. The Boolean logic equations for adding the two least significant bits (half-adder) are

$$\text{sum bit: } s_{i(0)} = x_{i(0)} \text{ XOR } r_{i(0)},$$

$$\text{carry bit: } c_{i(0)} = x_{i(0)} \text{ AND } r_{i(0)}.$$

Now, applying the corresponding parallel operations of XOR and AND, i.e., the symmetrical difference Δ and intersection $\cap$, and shifting the set of carry bits by a dilation $\oplus$, we can implement parallel addition by the following recursive equations:

(1) Define the initial states of images of sum bits and carry bits (called sum-bit image and carry-bit image) at time t_0 as

$$S(t_0) = X, \qquad C(t_0) = R. \tag{19}$$

(2) The recursive relation between the states of the sum-bit image and carry-bit image at two adjacent time intervals is then

$$S(t_{i+1}) = S(t_i)\, \Delta\, C(t_i) = \overline{\overline{S(t_i)} \cup C(t_i)} \cup \overline{S(t_i) \cup \overline{C(t_i)}}, \tag{20}$$

$$C(t_{i+1}) = [S(t_i) \cap C(t_i)] \oplus A^{-1} = \overline{\overline{S(t_i)} \cup \overline{C(t_i)}} \oplus A^{-1}, \tag{21}$$

where $i = 0,1,2,\ldots,k + 1$, and the elementary image A^{-1} is used to shift the carry-bit image one bit to the left for the next iteration.

(3) After a maximum of $k + 1$ iterations, the sum-bit image is the result and the carry-bit image is the null image $\varnothing$:

$$S(t_{k+1}) = X + R, \qquad C(t_{k+1}) = \varnothing. \tag{22}$$

This procedure is illustrated in Fig. 7. The result of parallel addition of binary numbers with a maximum k-bit word size is obtained after $k + 1$ iterations. This algorithm can be implemented in the DOCIP architecture by the program (instructions) given below. M_1, M_2, and M_3 represent the three $N \times N$-bit memories. $X \to M_1$ denotes store X into memory M_1. Each numbered line represents a single DOCIP machine instruction for one value of i. Comments are in brackets.

Assume start with X in $M_1[=S(t_0)]$ and R in $M_2[=C(t_0)]$.

First to kth iterations:

$$(1)\ M_1 \cup M_2 \rightarrow M_3[=S(t_i) \cup C(t_i)],$$

$$(2)\ M_1 \cup M_2 \rightarrow M_1[=S(t_i) \cup C(t_i)],$$

$$(3)\ M_1 \cup M_2 \cup M_3 \rightarrow M_2[=S(t_i) \cup C(t_i)],$$

$$(4)\ M_1 \cup M_2 \rightarrow M_1[=S(t_{i+1})],$$

$$(5)\ M_3 \oplus A^{-1} \rightarrow M_2[=C(t_{i+1})],$$

where $i = 0,1,2,\ldots,k-1$.

$(k+1)$th iteration:

$$(1)\ M_1 \cup M_2 \rightarrow M_3[=S(t_k) \cup C(t_k)],$$

$$(2)\ M_1 \cup M_2 \rightarrow M_1[=S(t_k) \cup C(t_k)],$$

$$(3)\ M_1 \cup M_3 \rightarrow \text{out}[=S(t_{k+1}) = X + R].$$

The total number of clock cycles for the execution of this program on the DOCIP machine is $t(k) \le 5k + 3 = O(k)$, which is independent of the number of words being added.

In fact, BIA can be used to devise a parallel form of a conditional-sum adder or carry-lookahead adder for further extracting additional parallelism, and the execution time of this addition can be reduced to $O(\log_2 k)$. Obviously, a trade-off exists between execution time and hardware complexity. This paper concentrates only on some simple algorithms.

3.2. Multiplication of Binary Row-Coded Numbers

Using the representation illustrated in Fig. 5, we define a parallel (matrix-constant) multiplication of an image set of binary numbers and one single binary number $X \cdot R_r$, and parallel (element–element) multiplication of two image sets of binary numbers $X \times R$.

I. Matrix-Constant Multiplication $X \cdot R_r$

Consider an image X [e.g., Fig. 8(a)] comprising N^2/k numbers x_i, $i = 1,2,\ldots,N^2/k$, and a reference image R_r [e.g., Fig. 8(b)] comprising only one single k-bit binary number $r = [r_{(k-1)}r_{(k-2)}\ldots r_{(0)}]_2$. The output of the parallel multiplication is $X \cdot R_r$ [Fig. 8(c)]. To realize it, we first consider the serial multiplication of two binary numbers that is the sum of the shifted versions of the multiplier or the multiplicand. Then, by applying the corresponding parallel operations and parallel shifting by a dilation $\oplus$, we can implement this parallel multiplication by the equation

$$X \cdot R_r = \sum_{l,\forall r_{(l)}=1} X \oplus A^{-l}, \tag{23}$$

where the sum notation $\sum$ refers to a sequence of parallel additions and the parallel addition $+$ is defined in Subsec. 3.1.

The DOCIP takes $O(k^2)$ clock cycles for implementing this matrix-constant multiplication. Its procedure involves:

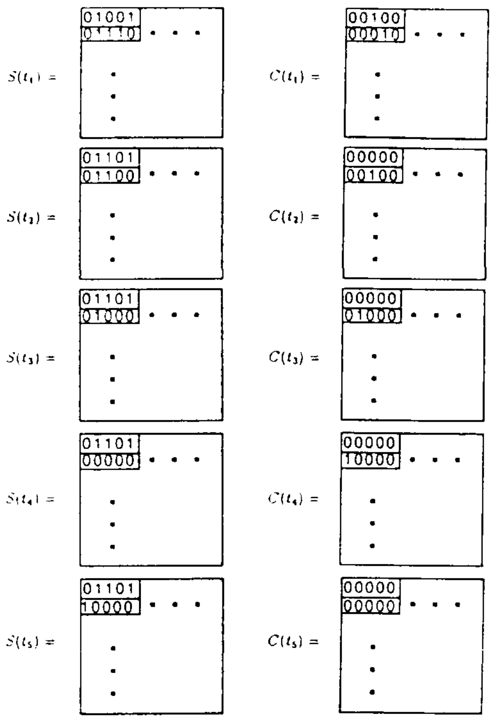


Fig. 7. Parallel addition of binary row-coded numbers (II): The procedure for parallel addition $X + R$ where X and R are shown in Fig. 6, $S(t_5) = S = X + R$ and $C(t_5) = \phi$.

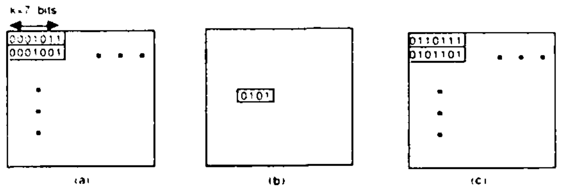


Fig. 8. Parallel (matrix-constant) multiplication of binary row-coded numbers: (a) image X of operands; (b) image R_r containing only a single number; (c) output $X \cdot R_r$.

(1) Generating the term $X \oplus A^{-l}$:

The DOCIP-array requires at most $l \le k - 1 = O(k)$ clock cycles, because

$$A^{-l} = (A^{-1})^l$$

$$= \underbrace{A^{-1} \oplus A^{-1} \oplus \ldots \oplus A^{-1}}_{l}, \tag{24}$$

$$X \oplus A^{-l} = \{\ldots[(X \oplus \underbrace{A^{-1}) \oplus A^{-1}] \oplus \ldots \oplus A^{-1}}\}.$$

The DOCIP-hypercube requires at most $\log_2 l \le \log_2(k-1) = O(\log_2 k)$ clock cycles, because we can rewrite l as a binary number $l = [a_{(\lfloor \log_2 l \rfloor)} \ldots a_{(1)}a_{(0)}]_2$, and we have

$$A^{-l} = \prod_{j=0}^{\lfloor \log_2 l \rfloor} A^{-a_{(j)} \cdot 2^j}$$

$$A^{-a_{(0)}} \oplus A^{-a_{(1)} \cdot 2^1} \oplus \ldots \oplus A^{-a_{(\lfloor \log_2 l \rfloor)} \cdot 2^{\lfloor \log_2 l \rfloor}}. \tag{25}$$

$$X \oplus A^{-l} = (\ldots\{[X \oplus A^{-a_{(0)}}] \oplus A^{-a_{(1)}\cdot 2^1}\}$$

$$\oplus \ldots \oplus A^{-a_{(\lfloor \log_2 l \rfloor)}\cdot 2^{\lfloor \log_2 l \rfloor}})$$

where $\lfloor \log_2 l \rfloor$ is the greatest integer less than or equal to $\log_2 l$, and each dilation with $A^{-a_{(j)}\cdot 2^j}$ can be implemented in the DOCIP-hypercube in one single clock cycle.

The total time delay for generating all required $X \oplus A^{-l}$, $0 \le l \le k - 1$, is bounded by $O(k)$ for both the DOCIP-array and the DOCIP-hypercube. Since

$$X \oplus A^{-l} = [X \oplus A^{-(l-1)}] \oplus A^{-1}, \tag{26}$$

we can generate the new term $X \oplus A^{-l}$ by simply deriving it from the previous term $X \oplus A^{-(l-1)}$ without starting from the original X. The total generating time is then dominated by the number of terms $X \oplus A^{-l}$ which is at most $O(k)$.

(2) Implementing the summation

$$\sum_{l, \forall r_{(l)}=1} X \oplus A^{-l}$$

The DOCIPs require at most $k - 1 = O(k)$ parallel additions to implement this summation, and each parallel addition requires at most $k + 1 = O(k)$ iterations (as shown in Subsec. 3.1). Since it takes $O(k)$ time for generating all the terms $X \oplus A^{-l}$, the total execution time of the DOCIPs for this matrix-constant multiplication of k-bit binary numbers is $O(k) \times O(k) + O(k) = O(k^2)$. From the example shown in Fig. 8, $R_r = I \cup A^{-2}$ contains only a single number $r = (0101)_2 = 5$, and the DOCIP can implement this matrix-constant multiplication $X \cdot R_r$ as follows:

Assume start with X in $M_1(= X \oplus I)$.

$$(1)\ M_1 \oplus A^{-2} \rightarrow M_2(= X \oplus A^{-2}).$$

(2) The instructions of the parallel addition are performed as shown in Subsec. 3.1:

$$M_1 + M_2 \rightarrow \text{out}(= X \cdot R_r).$$

II. Element–Element Multiplication $X \times R$

Consider an image X [e.g., Fig. 9(a)] comprising N^2/k numbers x_i, $i = 1,2,\ldots,N^2/k$, and an image R [e.g., Fig. 9(b)] comprising N^2/k numbers r_i, $i = 1,2,\ldots,N^2/k$. The output of the element–element parallel multiplication is $X \times R$ [Fig. 9(c)]. Because the multiplication of two binary numbers is the sum of the shifted versions of the multiplier or the multiplicand, applying the corresponding parallel operations, we can implement this parallel multiplication by the equation

$$X \times R = \sum_{l=0}^{k-1} (X \oplus A^{-l}) \cap \{[R \cap (M \oplus A^{-l})] \oplus \textstyle\bigcup_{j=0}^{k-l-1} A^{-j}\}$$

$$= \sum_{l=0}^{k-1} \overline{\overline{X \oplus A^{-l}} \cup \overline{\overline{\overline{R} \cup \overline{M \oplus A^{-l}}} \oplus \textstyle\bigcup_{j=0}^{k-l-1} A^{-j}}}, \tag{27}$$

where the mask M [Fig. 9(d)] is used to extract the lth bit [where the 0th bit is least significant and the $(k - 1)$th bit is most significant]. The DOCIPs can implement this element–element multiplication by the procedure

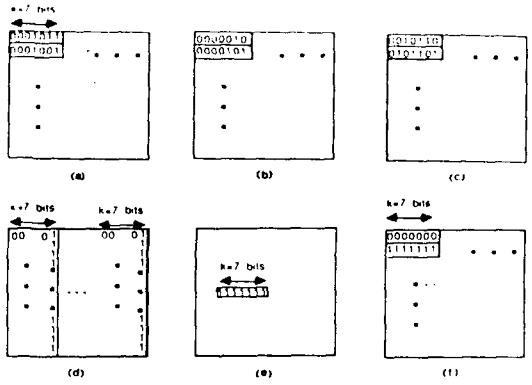


Fig. 9. Parallel (element–element) multiplication of binary row-coded numbers: (a) image X of operands; (b) image R of other operands; (c) output $X \times R$; (d) mask M; (e) image $\cup_{j=0}^{k-1} A^{-j}$; (f) image $(R \cap M) \oplus \cup_{j=0}^{k-1} A^{-j}$.

(1) Generate $X \oplus A^{-l}$ and $\overline{\overline{R} \cup \overline{M \oplus A^{-l}}}$:

Using an argument similar to that in Subsec. I above, the DOCIP-array takes $O(k)$ time and the DOCIP-hypercube takes $O(\log_2 k)$ time.

(2) Generate $\overline{\overline{R} \cup \overline{M \oplus A^{-l}}} \oplus \bigcup_{j=0}^{k-l-1} A^{-j}$:

The DOCIP-array takes $O(k)$ time, because

$$\bigcup_{j=0}^{k-l-1} A^{-j} = \left(\bigcup_{j=0}^{1} A^{-j}\right)^{k-l-1}$$

$$= \underbrace{\left(\bigcup_{j=0}^{1} A^{-j}\right) \oplus \left(\bigcup_{j=0}^{1} A^{-j}\right) \oplus \ldots \oplus \left(\bigcup_{j=0}^{1} A^{-j}\right)}_{k-l-1}, \tag{28}$$

$l \ge 0$, and each dilation by a term in parentheses executes in one clock cycle.

The DOCIP-hypercube takes $O(\log_2 k)$ time, since

$$\bigcup_{j=0}^{k-l-1} A^{-j} = \prod_{n=0}^{\lfloor \log_2 (k-l-1) \rfloor} \left[\bigcup_{j=0}^{n} A^{-a_{(j)}\cdot 2^j}\right], \tag{29}$$

where $k - l - 1 = [a_{\lfloor \log_2 (k-l-1) \rfloor} \ldots a_{(1)} a_{(0)}]_2$, and again each dilation by the term in parentheses executes in one clock cycle.

It takes $O(k)$ time for the DOCIP-array and $O(\log_2 k)$ for the DOCIP-hypercube to generate the term

$$\overline{(X \oplus A^{-l})} \cup \overline{\{[\overline{\overline{R} \cup \overline{(M \oplus A^{-l})}}] \oplus \textstyle\bigcup_{j=0}^{k-l-1} A^{-j}\}}.$$

(3) Implementing the summation

$$\sum_{l=0}^{k-1} \overline{\overline{X \oplus A^{-l}} \cup \overline{\overline{\overline{R} \cup \overline{M \oplus A^{-l}}} \oplus \textstyle\bigcup_{j=0}^{k-l-1} A^{-j}}}.$$

The summation requires at most $(k - 1)$ addition operations, and each addition operation takes $O(k)$ time on the DOCIP system. We also require $O(k)$ time for the DOCIP-array and $O(\log_2 k)$ time for the DOCIP-hypercube to generate each operand of the addition. Thus, for this element–element multiplication of k-bit binary numbers, the total computation time is $O(k^3)$ for the DOCIP-array and $O(k^2 \log_2 k)$ for the DOCIP-hypercube.

Multiplication requires more than three memories. This can be accommodated by either building more memory into the DOCIP machine or by swapping intermediate results into and out of an external memory. In the latter case we assume the external memory can be loaded and unloaded with one image in a single time step. In Sec. 4, binary stack-coded arithmetic also requires more than three memories; we make the same assumptions on the use of an external memory.

For binary column-coded arithmetic, a number is encoded in a part of a column of an image as in Fig. 10. All the algorithms derived in this section can also be applied to binary column-coded numbers except that we replace the elementary image A^{-1} by a different elementary image B for shifting the carry-bit image or borrow-bit image in the vertical direction.

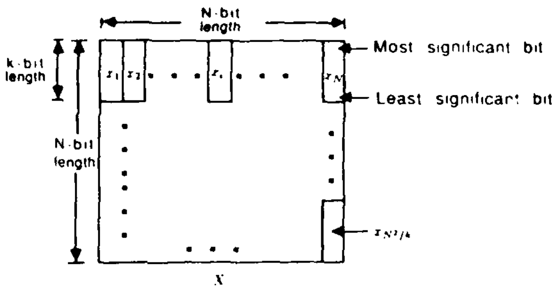


Fig. 10. Binary column-coded numbers.

4. Binary Stack-Coded Arithmetic

In this case, a number is encoded in a stack of k image planes with the least significant bit in the first plane, next least significant bit in the second plane, etc. (Fig. 11). We assume all numbers including the results of arithmetic operations can be represented in k bits, so that k images, each with $N \times N$ bits, contain N^2 binary numbers. Here, we describe parallel addition and multiplication by BIA expressions. Subtraction is discussed in Appendix B.

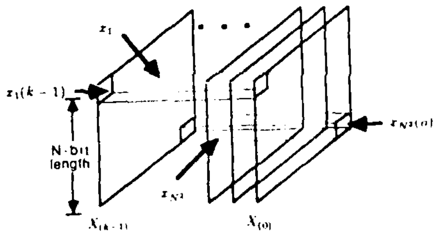


Fig. 11. Binary stack-coded numbers. $x_i(m)$ represents the mth bit of the ith number in the image plane. $X_{(0)}$ represents the image plane of least significant bits and $X_{(k-1)}$ represents the image plane of most significant bits.

4.1. Addition to Binary Stack-Coded Numbers

Using the representation illustrated in Fig. 11, we consider the parallel addition of two sequences of images of binary numbers. Assume a sequence of images $X = [X_{(k-1)},X_{(x-2)}, \ldots ,X_{(0)}]$ [e.g., Fig. 12(a)] storing N^2 binary numbers x_i, $i = 1,2, \ldots ,N^2$, and a sequence of images $R = [R_{(k-1)},R_{(k-2)}, \ldots ,R_{(0)}]$ [e.g., Fig. 12(b)] storing N^2 numbers r_i, $i = 1,2, \ldots ,N^2$. Then the output of the parallel addition is $X + R = S = [S_{(k)},S_{(k-1)}, \ldots ,S_{(0)}]$ as shown in Fig. 12(c). To realize this addition using our three fundamental operations, we implement an array of full adders as described by the equations

(1) The least significant bit planes of sum bits and carry bits are given by

$$S_{(0)} = X_{(0)} \vartriangle R_{(0)} = X_{(0)} \cup R_{(0)} \cup X_{(0)} \cup R_{(0)}, \tag{30}$$

$$C_{(1)} = X_{(0)} \cap R_{(0)} = X_{(0)} \cup R_{(0)}. \tag{31}$$

(2) The recursive relations:

$$\begin{aligned} S_{(i)} &= X_{(i)} \vartriangle R_{(i)} \vartriangle C_{(i)} \\ &= [X_{(i)} \cap R_{(i)} \cap C_{(i)}] \cup [X_{(i)} \cap R_{(i)} \cap C_{(i)}] \cup [X_{(i)} \cap R_{(i)} \cap C_{(i)}] \cup [X_{(i)} \cap R_{(i)} \cap C_{(i)}] \\ &= [X_{(i)} \cup R_{(i)} \cup C_{(i)}] \cup [X_{(i)} \cup R_{(i)} \cup C_{(i)}] \cup [X_{(i)} \cup R_{(i)} \cup C_{(i)}] \cup [\overline{X_{(i)} \cup R_{(i)} \cup C_{(i)}}], \end{aligned} \tag{32}$$

$$\begin{aligned} C_{(i+1)} &= [X_{(i)} \cap R_{(i)}] \cup [X_{(i)} \cap C_{(i)}] \cup [R_{(i)} \cap C_{(i)}] \\ &= [X_{(i)} \cup R_{(i)}] \cup [X_{(i)} \cup C_{(i)}] \cup [R_{(i)} \cup C_{(i)}], \end{aligned} \tag{33}$$

where $i = 0,1,2, \ldots ,k - 1$.

(3) The final solution is

$$X + R = S = [S_{(k)},S_{(k-1)}, \ldots ,S_{(0)}]. \tag{34}$$

where $S_{(k)} = C_{(k)}$ because $X_{(k)} = R_{(k)} = \varnothing$.

This algorithm can be implemented in the DOCIP architecture by the program (DOCIP instructions):

Assume start with $X_{(0)}$ stored in M_1 and $R_{(0)}$ stored in M_2.

Calculate $S_{(0)}$ and $\overline{C_{(1)}}$:

(1) $\overline{M_1} \cup \overline{M_2} \to M_3$ & out$[= \overline{C_{(1)}}]$,

(2) $\overline{M_1} \cup M_2 \to M_1[= \overline{X_{(0)}} \cup R_{(0)}]$,

(3) $\overline{M_1} \cup \overline{M_2} \cup \overline{M_3} \to M_2[= X_{(0)} \cup \overline{R_{(0)}}]$,

(4) $\overline{M_1} \cup \overline{M_2} \to$ out$[= S_{(0)}]$.

Calculate $S_{(1)}$ and $C_{(2)}$:

(1) $X_{(1)} \to M_1$,

(2) $\overline{M_1} \cup \overline{M_3} \to M_2[= \overline{X_{(1)}} \cup C_{(1)}]$,

(3) $M_1 \cup M_3 \to M_1[= X_{(1)} \cup \overline{C_{(1)}}]$,

(4) $M_1 \cup M_2 \to M_1 [= X_{(1)} \Delta C_{(1)}]$,

(5) $R_{(1)} \to M_2$,

(6) $M_1 \cup M_2 \to M_3$,

(7) $M_1 \cup M_2 \to M_2$,

(8) $M_2 \cup M_3 \to \text{out} [= S_{(1)}]$,

(9) $X_{(1)} \to M_1$,

(10) $R_{(1)} \to M_2$,

(11) $M_1 \cup M_2 \to M_3$,

(12) $C_{(1)} \to M_1$,

(13) $M_1 \cup M_2 \to M_2$,

(14) $M_2 \cup M_3 \to M_3$,

(15) $X_{(1)} \to M_2$,

(16) $M_1 \cup M_2 \to M_2$,

(17) $M_2 \cup M_3 \to M_3 \,\&\, \text{out} [= C_{(2)}]$.

Calculate $S_{(2)}$ to $S_{(k-1)}$ and $C_{(3)}$ to $C_{(k)}$:

Use the same instructions for calculating $S_{(1)}$ and $C_{(2)}$ except that $X_{(1)}$ and $R_{(1)}$ [and $S_{(1)}$ and $C_{(2)}$] are replaced by $X_{(i)}$ and $R_{(i)}$ [and $S_{(i)}$ and $C_{(i+1)}$] in each iteration, and in the beginning of an iteration the memory M_3 stores $C_{(i)}$ instead of $\overline{C_{(1)}}$, $i = 2,3, \ldots ,k$.

The complete execution of this operation in the DOCIP requires $t(k) \leq 17(k-1) + 4 = 17k - 13 = O(k)$ clock cycles. Additional parallelism could be extracted to further reduce the execution time by utilizing carry-lookahead techniques or by optimizing the above program.

4.2. Multiplication of Binary Stack-Coded Numbers

Let the result of the parallel multiplication be $X \times R = M = [M_{(2k-1)}, M_{(2k-2)}, \ldots ,M_{(0)}]$ [e.g., Fig. 12(e)]. Since binary multiplication is equivalent to the addition of shifted versions of the multiplicand, applying the corresponding parallel operations, we can implement the parallel multiplication by the equations

$$P^{(0)} = \Big[\underbrace{0,0,\ldots,0,}_{k} X_{(k-1)} \cap R_{(0)}, X_{(k-2)} \cap R_{(0)}, \ldots ,X_{(0)} \cap R_{(0)}\Big] \cdot \qquad (35)$$

$$P^{(i)} = \Big[\underbrace{0,0,\ldots,0,}_{k-i} X_{(k-1)} \cap R_{(i)}, X_{(k-2)} \cap R_{(i)}, \ldots ,X_{(0)} \cap R_{(i)}, \underbrace{0,0,\ldots,0,}_{i}\Big] \cdot \qquad (36)$$

$$X \times R = M = \sum_{i=0}^{k-1} P^{(i)} = P^{(0)} + P^{(1)} + \ldots + P^{(k-1)}, \qquad (37)$$

where $i = 0,1, \ldots ,k-1$, and the addition + is defined in Subsec. 4.1. Since this parallel multiplication requires at most $k-1$ additions, each addition takes $O(k)$ time for the DOCIP, and each $P^{(i)}$ can be generated in $O(k)$ time, the total execution time is $O(k^3)$

5. Symbolic Substitution and Binary Symbol-Coded Arithmetic

Symbolic substitution was first considered as a means of utilizing the parallelism of optics by Huang.[17]

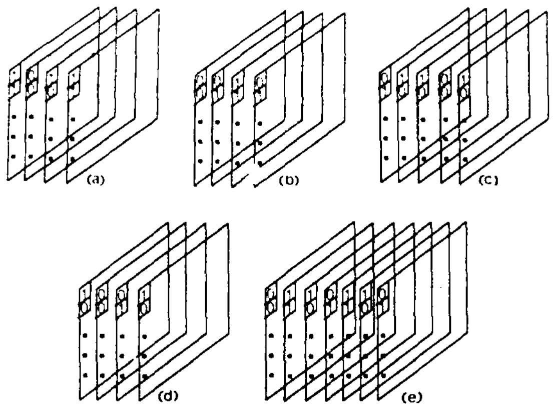


Fig. 12. Parallel arithmetic with binary stack-coded numbers: (a) sequence of images $X = [X_{(3)},X_{(2)},X_{(1)},X_{(0)}]$; (b) sequence of images $R = [R_{(3)},R_{(2)},R_{(1)},R_{(0)}]$; (c) sum $X + R = [S_{(4)},S_{(3)},S_{(2)},S_{(1)},S_{(0)}]$; (d) difference $D = X - R = [D_{(3)},D_{(2)},D_{(1)},D_{(0)}]$; (e) product $M = X \times R = [M_{(7)},M_{(6)}, \ldots ,M_{(0)}]$.

Recently, the use of symbolic substitution as a basis for digital optical computing has been reported in Refs. 17–19 and 25–32. Special symbolic substitution rules can be applied to perform arithmetic operations and simulate a Turing machine.[19] Symbolic substitution demonstrates the ability to solve any computable problem and performs many operations. Here we formalize symbolic substitution by BIA algebraic symbols, demonstrate that symbolic substitution rules are particular BIA image transformations, and give the BIA formal notations of binary symbol-coded (symbolic substitution) arithmetic.

5.1. BIA Representation of Symbolic Substitution

In this subsection we give the BIA equation for symbolic substitution and show how it can be implemented on the DOCIP machine. A symbolic substitution rule involves two steps: (1) recognizing the locations of a certain search-pattern within the 2-D binary input data, and (2) substituting a replacement-pattern wherever the search-pattern is recognized. We derive it by BIA in the following steps (illustrated in Fig. 13):

1. BIA Notations for Symbolic Substitution

2-D binary input data = image (bit plane) X.

Symbol to be recognized (search-pattern) = reference image (or image pairs) R.

Symbol to be replaced (replacement-pattern) = reference image Q.

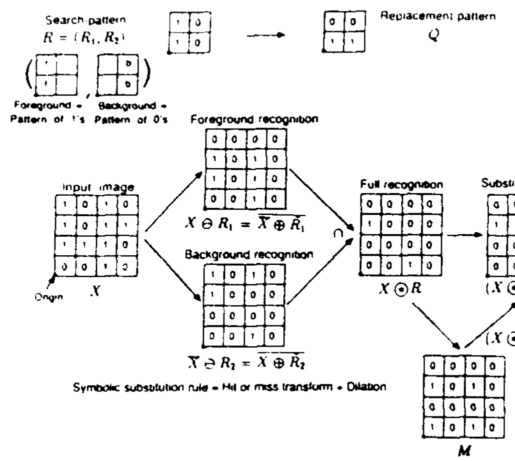

Search-pattern
$R = (R_1, R_2)$
Replacement pattern
Q
Foreground = Pattern of 1's
Background = Pattern of 0's
Foreground recognition
Input image
Origin
X
$X \ominus R_1 = \overline{\overline{X} \oplus \check{R}_1}$
Background recognition
$\overline{X} \ominus R_2 = \overline{X \oplus \check{R}_2}$
Full recognition
$X \odot R$
Substitution output
$(X \odot R) \oplus Q$
$(X \odot R) \cap M$
M
Symbolic substitution rule = Hit or miss transform + Dilation

Fig. 13. BIA representation of symbolic substitution. The optional mask M is for controlling the block seach region.

2. *Symbolic Substitution Rule*

Step 1, recognition of the search-pattern:

(a) Foreground recognizer: the locations of a certain spatial search-pattern R_1 (defined by its foreground) within the foreground of the 2-D input data X can be recognized by the erosion operation of X and R_1:

$$X \ominus R_1 = \overline{\overline{X} \oplus \check{R}_1}. \tag{38}$$

(b) Background recognizer: the locations of a certain spatial search-pattern R_2 within the background of the 2-D input data X can be recognized by the erosion of $\overline{X}$ and R_2:

$$\overline{X} \ominus R_2 = \overline{X \oplus \check{R}_2}. \tag{39}$$

(c) Full recognizer: by combining the two above steps, the locations of a certain spatial search-pattern $R = (R_1,R_2)$ (R_1 defines the foreground, and R_2 defines the background) within the 2-D input data X can be recognized by the hit or miss transform of X and R:

$$X \odot R = (X \ominus R_1) \cap (\overline{X} \ominus R_2) = \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)}. \tag{40}$$

Step 2, substitution of the replacement-pattern:

Substituter: a new replacement-pattern Q can be substituted for R wherever the search-pattern R is recognized by the dilation of $X \odot R$ by Q.

Synthesis:

A complete symbolic substitution rule is implemented by the hit or miss transform of X by R followed by the dilation by Q:

$$\begin{aligned}(X \odot R) \oplus Q &= [(X \ominus R_1) \cap (\overline{X} \ominus R_2)] \oplus Q \\ &= \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)} \oplus Q. \end{aligned} \tag{41}$$

Optional masking:

An optional mask M can be used for controlling the block search region. A symbolic substitution rule can be modified as

$$[(X \odot R) \cap M] \oplus Q. \tag{42}$$

By proper choice of M, the search can be made in overlapping, disjoint or noncontiguous blocks.

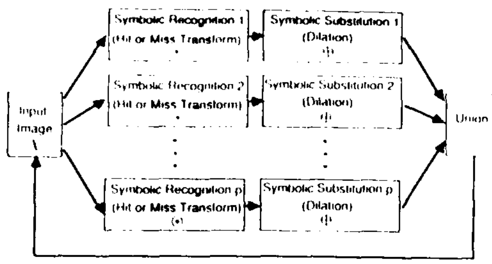

Input Image
Symbolic Recognition 1 (Hit or Miss Transform)
Symbolic Substitution 1 (Dilation)
Symbolic Recognition 2 (Hit or Miss Transform)
Symbolic Substitution 2 (Dilation)
Symbolic Recognition p (Hit or Miss Transform)
Symbolic Substitution p (Dilation)
Union
Out

Fig. 14. Symbolic substitution system with p symbolic substitution rules.

3. *Symbolic Substitution System (Fig. 14)*

To work with more than one rule (say p substitution rules) for practical applications, a symbolic substitution processor produces several copies of the input X, provides p different recognizer–substituter units, and then combines the outputs of various units to form a new output. Thus, a symbolic substitution system is implemented by

$$\bigcup_{i=1}^{p} [X \odot R^{(i)}] \oplus Q^{(i)}, \tag{43}$$

where $R^{(i)}$ and $Q^{(i)}, i = 1,2,\ldots,p$, are the reference image pairs and replacement patterns in the ith symbolic substitution rule. This, then, is the BIA formula for general symbolic substitution.

Hence, a general mathematical formalism of symbolic substitution has been developed. For a local search-pattern and replacement-pattern (i.e., $R_1,R_2,Q \subset N_{\text{array}}$ or $N_{\text{hypercube}}$), the DOCIP-array or DOCIP-hypercube can implement a symbolic substitution rule in four (or five with the optional mask) steps:

Assume start with X in M_1.

(1) $\overline{M}_1 \oplus \check{R}_1 \to M_2$,

(2) $M_1 \oplus \check{R}_2 \to M_3$,

(3) $M_2 \cup M_3 \to M_3$,

(4) $\overline{M}_3 \oplus Q \to \text{out}[= (X \odot R) \oplus Q]$.

Let the pixels used in the substitution rule(s) of a symbolic substitution processor be the neighborhood, N_{ss} of the processor. We see from the above steps that the DOCIP can simulate the symbolic substitution processor in constant time if the two machines have the same neighborhood. If N_{ss} is not a subset of the DOCIP neighborhood, the simulation will take longer. In either case, it is not presently known how many steps it takes the symbolic substitution processor to simulate the DOCIP.

5.2. Binary Symbol-Coded (Symbolic Substitution) Arithmetic

A bit in a binary number is encoded symbolically as pixels of an image (Fig. 15). In this subsection, we primarily concentrate on single-pixel coding: a logic value (0 or 1) is represented by a single pixel (dark or bright) [Fig. 15(a)], as in the binary row and stack-coded number representations, but the operands of

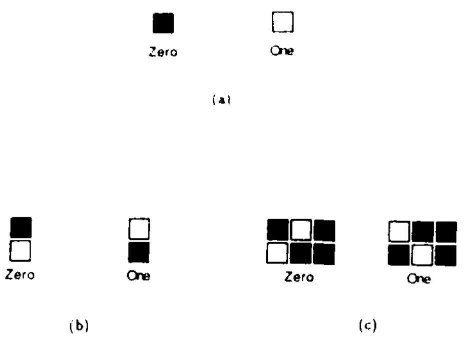


Fig. 15. Bit encoded as a symbol: (a) single-pixel coding of zero and one (a bit is a pixel); (b) two-pixel coding of zero and one (a bit is encoded as two pixels) (adapted from Refs. 18 and 19); (c) six-pixel coding of zero and one (a bit with value zero or one is encoded as six pixels) (adapted from Ref. 31).

binary numbers x_i and r_i are stored in the same input image X as shown in Fig. 16(a). The expected output images of symbolic substitution for binary addition and binary subtraction are shown in Figs. 16(b) and (c). To achieve these desired operations, the symbols associated with the operands are recognized and then replaced by new symbols associated with the results of the operation. Systems for implementing binary addition and subtraction are formalized and illustrated as examples of binary symbol-coded arithmetic below.

5.2.1. Addition of Binary Symbol-Coded Numbers

This parallel binary addition (Fig. 17) can be implemented with four symbolic substitution rules [Fig. 17(a)].[17,18] In the case of single-pixel coding, as we will show, Rule 1 is not necessary. The symbolic substitution system for single-pixel coding can be realized as

$$Y(t_0) = X, \tag{44}$$

$$Y(t_{j+1}) = \bigcup_{i=1}^{4} \{[Y(t_j) \circledast R^{(i)}] \cap M\} \oplus Q^{(i)}, \tag{45}$$

where $Y(t_{k+1})$ is the result, $j = 0,1,2, \ldots ,k + 1$, k is word size (i.e., the number of bits in an operand); $R^{(i)} = [R_1^{(i)},R_2^{(i)}]$ and $Q^{(i)}$ are shown in Fig. 17(b) and represented as

(1) $R_1^{(1)} = \varnothing, \quad R_2^{(1)} = \bigcup_{i=0}^{1} B^i, \quad Q^{(1)} = \varnothing,$

(2) $R_1^{(2)} = I, \quad R_2^{(2)} = B, \quad Q^{(2)} = I,$

(3) $R_1^{(3)} = B, \quad R_2^{(3)} = I, \quad Q^{(3)} = I,$

(4) $R_1^{(4)} = \bigcup_{i=0}^{1} B^i, \quad R_2^{(4)} = \varnothing, \quad Q^{(4)} = A^{-1}B.$

Here the null image $\varnothing$ and the elementary images are as defined in Subsec. 2.1; the mask M [Fig. 17(c)], used for controlling the block search region, is the image corresponding to the coordinates of the origins (lower-left pixels) of the input symbols in the input image X. An example is given in Fig. 17(d). Note that $Q^{(1)} = \varnothing$ implies

$$\{[Y(t_j) \circledast R^{(1)}] \cap M\} \oplus Q^{(1)} = \varnothing, \tag{46}$$

so that

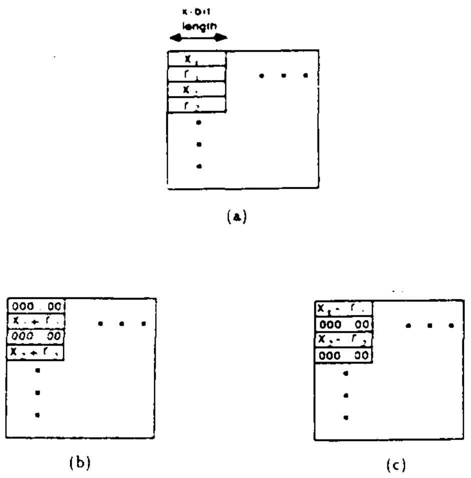


Fig. 16. Binary symbol-coding (symbolic substitution) binary arithmetic): (a) input image X contains the operands x_i and r_i; (b) output of parallell addition; (c) output of parallel subtraction.

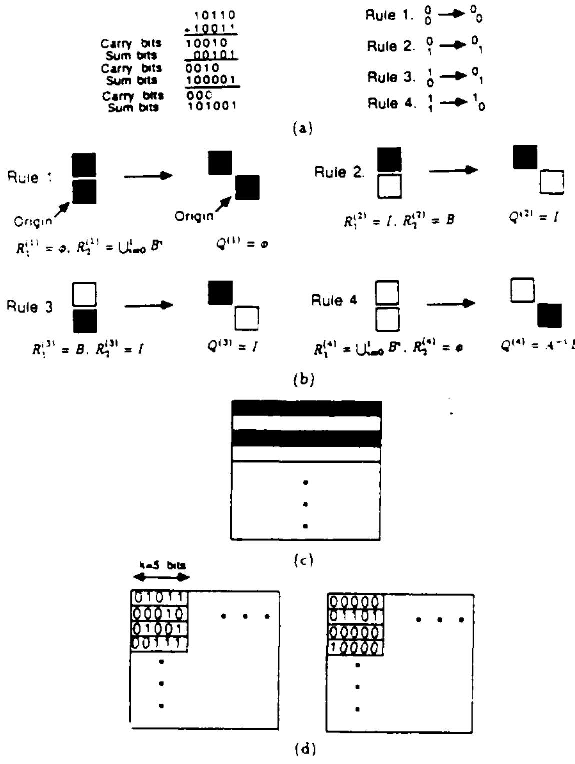

10110
+10011
Carry bits 10010
Sum bits 00101
Carry bits 0010
Sum bits 100001
Carry bits 000
Sum bits 101001
Rule 1. $\genfrac{}{}{0pt}{}{0}{0} \rightarrow {}^{0}{}_{0}$
Rule 2. $\genfrac{}{}{0pt}{}{0}{1} \rightarrow {}^{0}{}_{1}$
Rule 3. $\genfrac{}{}{0pt}{}{1}{0} \rightarrow {}^{0}{}_{1}$
Rule 4. $\genfrac{}{}{0pt}{}{1}{1} \rightarrow {}^{1}{}_{0}$
(a)
Rule 1
Origin
Origin
$R_1^{(1)} = \varnothing,\ R_2^{(1)} = \bigcup_{i=0}^{1} B^i$
$Q^{(1)} = \varnothing$
Rule 2
$R_1^{(2)} = I,\ R_2^{(2)} = B$
$Q^{(2)} = I$
Rule 3
$R_1^{(3)} = B,\ R_2^{(3)} = I$
$Q^{(3)} = I$
Rule 4
$R_1^{(4)} = \bigcup_{i=0}^{1} B^i,\ R_2^{(4)} = \varnothing$
$Q^{(4)} = A^{-1}B$
(b)
(c)
k=5 bits
01011
00010
01001
00111
00000
01101
00000
10000
(d)

Fig. 17. Parallel addition of binary symbol-coded numbers: (a) four symbolic substitution rules for addition; (b) reference image pairs $R^{(i)}$ and reference images $Q^{(i)}$, $i = 1,2,3,4$, used for addition; $Q^{(1)}$ is a null image. Rule 1 is not needed for this single-pixel coding. (c) mask M; (d) example of parallel addition of binary symbol-coded numbers.

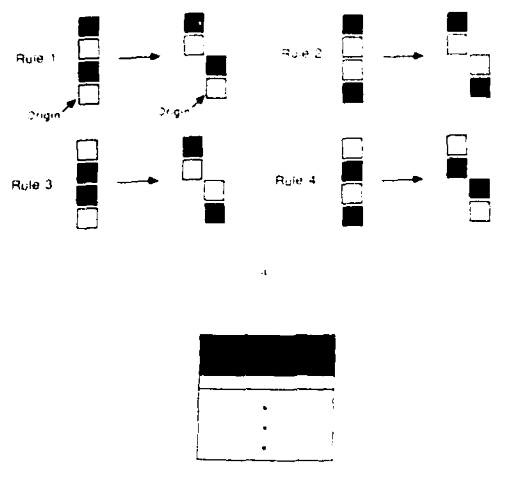


Fig. 18. Symbolic substitution binary addition with two-pixel coding: (a) reference image pairs $R^{(i)}$ and reference images $Q^{(i)}, i = 1,2,3,4$, used for addition (with two-pixel coding) (adapted from Refs. 18 and 19); (b) mask M.

$$Y(t_{j+1}) = \cup_{i=1}^{4} \{[Y(t_j) \odot R^{(i)}] \cap M\} \oplus Q^{(i)}$$
$$= \cup_{i=2}^{4} \{[Y(t_j) \odot R^{(i)}] \cap M\} \oplus Q^{(i)}$$
$$= \cup_{i=2}^{4} \{[Y(t_j) \odot \check{R}_1^{(i)}] \cup [Y(t_j) \odot \check{R}_2^{(i)}] \cap M\} \oplus Q^{(i)}. \quad (47)$$

Thus, for single-pixel coding of symbolic substitution, we can reduce the four rules of binary addition to only three rules. However, this reduction of complexity cannot be applied to two-pixel (i.e., dual-rail) or six-pixel coding.

When implemented on the DOCIP, this addition requires at most $k + 1$ iterations, each iteration requiring two union operations of three results of symbolic substitution rules, and each rule is realized within five steps as shown in Subsec. 5.1. Thus, the total execution time in the DOCIP is

$$t(k) \leq (3 \times 5 + 2)(k + 1) = 17(k + 1) = O(k).$$

When using two or six pixels to represent a logic value [Figs. 15(b) and (c)], we can formalize symbolic substitution addition as follows.

With two-pixel coding [Fig. 15(b)],[19,20] we can implement a full recognition with only a background recognizer (or foreground recognizer):

$$Y(t_{j+1}) = \cup_{i=1}^{4} \{[Y(t_j) \odot R^{(i)}] \cap M\} \oplus Q^{(i)}$$
$$= \cup_{i=1}^{4} \{[Y(t_j) \odot \check{R}_1^{(i)}] \cup [Y(t_j) \odot \check{R}_2^{(i)}] \cap M\} \oplus Q^{(i)}$$
$$= \cup_{i=1}^{4} [Y(t_j) \odot \check{R}_2^{(i)} \cap M] \oplus Q^{(i)}, \quad (48)$$

where $j = 0,1,2,\ldots,k$; $R^{(i)} = [R_1^{(i)},R_2^{(i)}]$ and $Q^{(i)}$ are shown in Fig. 18(a) and represented by elementary images as

(1) $R_1^{(1)} = I \cup B^2, \quad R_2^{(1)} = B \cup B^3, \quad Q^{(1)} = I \cup A^{-1}B^2,$

(2) $R_1^{(2)} = \cup_{i=1}^{2} B^i, \quad R_2^{(2)} = I \cup B^3, \quad Q^{(2)} = B \cup A^{-1}B^2,$

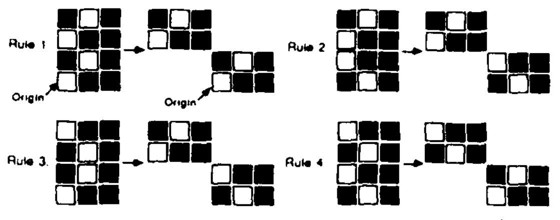


Fig. 19. Symbolic substitution binary addition with encoding a bit as six pixels (adapted from Ref. 31).

(3) $R_1^{(3)} = I \cup B^3, \quad R_2^{(3)} = \cup_{i=1}^{2} B^i, \quad Q^{(3)} = B \cup A^{-1}B^2,$

(4) $R_1^{(4)} = B \cup B^3, \quad R_2^{(4)} = I \cup B^2, \quad Q^4 = I \cup A^{-1}B^3;$

and the mask M is shown in Fig. 18(b). Since

$$[Y(t_j) \oplus \check{R}_1^{(i)}] \cup [Y(t_j) \oplus \check{R}_2^{(i)}] = [Y(t_j) \oplus \check{R}_1^{(i)}]$$
$$= [Y(t_j) \oplus \check{R}_2^{(i)}], \quad (49)$$

for the two-pixel coding, $R^{(i)}$ can be represented by only its foreground $R_1^{(i)}$ or background $R_2^{(i)}$. For implementation on the DOCIP, this algorithm requires four rule~, and each rule involves two dilations and one union or intersection. Because they may be not included in N_{array} or $N_{\text{hypercube}}$, each dilation of $R_2^{(i)}$ or $Q^{(i)}$ is implemented by 2–4 steps for the DOCIP-array8 and 1–2 steps for the DOCIP-hypercube8. The total execution time is bounded by $28(k + 1)$ for the DOCIP-array8 and $18(k + 1)$ for the DOCIP-hypercube8. Moreover, it requires more difficult two-pixel coding and doubles the device area.

With six-pixel coding [Fig. 15(c)],[31] the mask M is not needed and

$$Y(t_{j+1}) = \bigcup_{i=1}^{4} [Y(t_j) \odot R^{(i)}] \oplus Q^{(i)}, \quad (50)$$

where $j = 0,1,2,\ldots,k,k$ is the word size; $R^{(i)} = [R_1^{(i)},R_2^{(i)}]$ and $Q^{(i)}$ are shown in Fig. 19 and are represented as

(1) $R_1^{(1)} = I \cup AB \cup B^2 \cup AB^3,$

$R_2^{(1)} = B \cup B^3 \cup A \cup AB^2 \cup (\cup_{i=0}^{3} A^2B^i),$

$Q^{(1)} = A^{-3}B^2 \cup A^{-2}B^3 \cup I \cup AB,$

(2) $R_1^{(2)} = (\cup_{i=1}^{2} B^i) \cup A \cup A^3,$

$R_2^{(2)} = I \cup B^3 \cup (\cup_{i=1}^{2} AB^i) \cup (\cup_{i=0}^{3} A^2B^i),$

$Q^{(2)} = A^{-3}B^2 \cup A^{-2}B^3 \cup B \cup A,$

(3) $R_1^{(3)} = I \cup B^3 \cup (\cup_{i=1}^{2} AB^i),$

$R_2^{(3)} = (\cup_{i=1}^{2} B^i) \cup A \cup A^3 \cup (\cup_{i=0}^{3} A^2B^i),$

$Q^{(3)} = A^{-3}B^2 \cup A^{-2}B^3 \cup B \cup A,$

(4) $R_1^{(4)} = B \cup B^3 \cup A \cup AB^2,$

$R_2^{(4)} = I \cup B^2 \cup AB \cup AB^3 \cup (\cup_{i=0}^{3} A^2B^i),$

$Q^{(4)} = A^{-3}B^3 \cup A^{-2}B^2 \cup I \cup AB.$

The six-pixel coding removes the need for the mask M, but requires more difficult encoding, more difficult implementation of the hit or miss transform by $R^{(i)}$ and dilation by $Q^{(i)}$, and six times the hardware area. Addition on the DOCIP-array or DOCIP-hypercube using six-pixel coding takes much more time (a factor of more than 10 times) than the time required for single-pixel coding or two-pixel coding.

6. Complexity of Parallel Optical Binary Arithmetic

We have shown that BIA offers a general tool for mapping serial binary arithmetic into different forms of parallel binary arithmetic (including binary row-coding, binary stack-coding, and three coding techniques for symbolic substitution arithmetic) in a precise and compact way. The complexity of parallel addition and subtraction of two $N \times N$ arrays of binary numbers (each number with k-bit length) for these different number representations are compared in Tables I and II. Binary row-coded arithmetic requires the smallest number O of fundamental operations. Binary stack-coded arithmetic requires the lowest number of processing elements (or cells) P and the smallest overall $O \times P$ complexity (assume each parallel fundamental operation corresponds to P processing elements executing in parallel). For the normal case in which the word size is larger than one and much smaller than the image size ($1 < k \ll N$), binary row-coded arithmetic can be implemented in the DOCIP with the fastest computation speed (assume the DOCIP can input all operands in an image at a time). For implementation on the DOCIPs, the complexity of binary symbol-coded (symbolic substitution) arithmetic is in all cases higher than that of binary row-coded and binary stack-coded arithmetic. For implementing symbolic substitution algorithms on the DOCIPs, the single-pixel coding is superior to the other symbol coding techniques.

Table I. Complexity of Parallel Optical Binary Addition of Two $N \times N$ Arrays of k-Bit Binary Numbers; Each Parallel Fundamental Operation Corresponds to P Processing Elements Executing in Parallel

Number Representation	Binary Row-coding	Binary Stack-coding	Symbolic Substitution (single-pixel coding)	Symbolic Substitution (two-pixel coding)
No. of Dilations (or Erosions)	k	0	9(k+1)	8(k+1)
No. of Unions (or Intersections)	4k+3	16k-12	15(k+1)	16(k+1)

No. of Complements	7k+4
	*2k+2
		20k-13
		*7k-5
			12(k+1)
			*3(k+1)
				16(k+1)
				*4(k+1)

Total No. of Parallel Fundamental Operations O	12k+7
	*7k+5
		36k-25
		*23k-17
			36(k+1)
			*27(k+1)
				40(k+1)
				*28(k+1)

No. of Processing Elements P	kN^2	N^2	$2kN^2$	$4kN^2$
Total No. of Computations OxP	$(12k+7)kN^2$ $*(7k+5)kN^2$	$(36k-25)N^2$ $*(22k-16)N^2$	$76k(k+1)N^2$ $*54k(k+1)N^2$	$160k(k+1)N^2$ $*112k(k+1)N^2$
DOCIP Execution Time T	5k+3	17k-13	17(k+1)	18(k+1) or 28(k+1)
PxT	$(5k+3)kN^2$	$(17k-13)N^2$	$34k(k+1)N^2$	$72k(k+1)N^2$ or $112k(k+1)N^2$

* indicates the number of operations when erosion and intersection are also allowed

Table II. Complexity of Parallel Optical Binary Subtraction of Two $N \times N$ Arrays of k-Bit Binary Numbers

Number Representation	Binary Row coding	Binary Stack coding	Symbolic Substitution (single pixel coding)	Symbolic Substitution (two pixel coding)
No. of Dilations (or Erosions)	k	0	6(k+1)	8(k+1)
No. of Unions (or Intersections)	4k+3	16k-12	10(k+1)	16(k+1)

No. of Complements	6k+4
	*3k+2
		22k-18
		*11k-8
			8(k+1)
			*2(K+1)
				16(k+1)
				*4(k+1)

Total No. of Parallel Fundamental Operations O	11k+7
	*8k+5
		43k-33
		*33k-26
			24(k+1)
			*18(k+1)
				40(k+1)
				*28(k+1)

No. of Processing Elements P	kN^2	N^2	$2kN^2$	$4kN^2$
Total No. of Computations OxP	$(11k+7)kN^2$ $*(8k+5)kN^2$	$(43k-33)N^2$ $*(33k-26)N^2$	$48k(k+1)N^2$ $*36k(k+1)N^2$	$160k(k+1)N^2$ $*112k(k+1)N^2$
DOCIP Execution Time T	4k+3	20k-17	11(k+1)	18(k+1) or 28(k+1)
PxT	$(4k+3)kN^2$	$(20k-17)N^2$	$22k(k+1)N^2$	$72k(k+1)N^2$ or $112k(k+1)N^2$

* indicates the number of operations when erosion and intersection are also allowed

7. Conclusion

Optical computers can operate on 2-D planes of data in parallel. Boolean logic equations do not provide a complete description of such parallel operations for binary arithmetic. An optical system that operates on planes of data should employ an inherently parallel mathematical description for its arithmetic. In this paper we use binary image algebra (BIA) to develop parallel numerical computation algorithms, and to describe the execution of these algorithms on a digital optical cellular image processor (DOCIP) architecture.

BIA is demonstrated to be a general technique for developing and formulating parallel numerical and non-numerical computation algorithms for digital optical computers. The DOCIP is a simple optical architecture for effectively implementing BIA. Symbolic substitution is a subset of BIA and can be formalized in compact BIA expressions. Three different techniques for parallel optical binary arithmetic, based on binary row-coding, binary stack-coding, and binary symbol-coding (symbolic substitution), are illustrated for implementation on the DOCIP. Binary row-coding arithmetic has fast DOCIP execution and binary stack-coding arithmetic requires the lowest number of computations $O \times P$. In summary, BIA and the DOCIP represent a simple yet powerful parallel digital optical algorithmic and architectural technique for both numerical and non-numerical applications.

The authors would like to thank P. Chavel and E. K. Blum for their contributions to this research.

This work was supported by the Air Force Office of Scientific Research under grant AFOSR-84-0181, by the Office of Naval Research under contract N00014-86-0602, and by an IBM graduate fellowship. Portions of this paper were presented at the 1987 Annual Meeting of the Optical Society of America, Rochester, NY, Oct. 1987.[20]

Appendix A: Subtraction of Binary Row-Coded Numbers

Let the output of the parallel subtraction be $D = X - R$ [e.g., Figs. 20(a)–(c)]. To realize it, we first consider the serial binary subtraction of 2 binary numbers $d_i = x_i - r_i$. The procedure in the least significant bits $x_{i(0)}$ and $r_{i(0)}$ of binary subtraction generates a difference bit $d_{i(0)}$ and a borrow bit $b_{i(0)}$. The Boolean logic equations for subtracting the two least significant bits (half-subtractor) are

$$\text{difference bit:}\quad s_{i(0)} = x_{i(0)}\ \text{XOR}\ r_{i(0)},$$

$$\text{borrow bit:}\quad c_{i(0)} = x_{i(0)}\ \text{AND}\ r_{i(0)}.$$

Now, applying the corresponding parallel operations and shifting the set of borrow bits by a dilation ⊕, we can implement the parallel subtraction as follows:

(1) Define the initial states of images of difference bits and borrow bits (called difference-bit image and borrow-bit image) at time t_0 as

$$D(t_0) = X,\qquad B(t_0) = R. \tag{51}$$

(2) The recursive relation between the states of the difference-bit image and borrow-bit image at two adjacent time intervals is

$$D(t_{i+1}) = D(t_i)\ \Delta\ B(t_i) = \overline{D(t_i) \cup B(t_i)} \cup \overline{\overline{D(t_i)} \cup \overline{B(t_i)}}, \tag{52}$$

$$B(t_{i+1}) = [\overline{D(t_i)} \cap B(t_i)] \oplus A^{-1} = \overline{D(t_i) \cup \overline{B(t_i)}} \oplus A^{-1}, \tag{53}$$

where $i = 0,1,2,\ldots,k+1$, and the elementary image A^{-1} is used to shift the borrow-bit image one bit to the left for the next iteration.

(3) After a maximum of $k+1$ iterations, the difference-bit image is the result and the borrow-bit image becomes the null image $\varnothing$:

$$D(t_{k+1}) = X - R,\qquad B(t_{k+1}) = \varnothing. \tag{54}$$

This procedure is illustrated in Fig. 20(d). The result of parallel subtraction of binary numbers with a maximum k-bit word size is obtained after $k+1$ iterations. The DOCIP architecture can realize this by the following program (instructions):

Assume start with X in $M_1[= D(t_0)]$ and R in $M_2[= B(t_0)]$.

First to kth iterations:

$$(1)\ \ \overline{M_1} \cup M_2 \to M_3[= \overline{\overline{D(t_i)} \cup B(t_i)}],$$

$$(2)\ \ M_1 \cup \overline{M_2} \to M_1[= \overline{D(t_i) \cup \overline{B(t_i)}}],$$

$$(3)\ \ M_1 \cup M_3 \to M_1[= D(t_{i+1})],$$

$$(4)\ \ M_3 \oplus A^{-1} \to M_2[= B(t_{i+1})],$$

where $i = 0,1,2,\ldots,k-1$.

$(k+1)$th iteration:

$$(1)\ \ \overline{M_1} \cup M_2 \to M_3[= \overline{\overline{D(t_k)} \cup B(t_k)}],$$

$$(2)\ \ M_1 \cup \overline{M_2} \to M_1[= \overline{D(t_k) \cup \overline{B(t_k)}}],$$

$$(3)\ \ M_1 \cup M_3 \to M_1[= D(t_{k+1}) = X - R].$$

The total number of clock cycles in the DOCIP to complete this subtraction process is $t(k) \le 4k + 3 = O(k)$.

Appendix B: Subtraction of Binary Stack-Coded Numbers

Let the result of the parallel subtraction be $X - R = D = [D_{(k-1)}, D_{(k-2)}, \ldots, D_{(0)}]$ [e.g., Fig. 12(d)]. To realize it using the three fundamental operations, we consider a serial full-subtractor. Applying the corresponding parallel operations, we can implement this parallel subtraction by the equations

(1) The least significant bit planes of difference bits and borrow bits:

$$D_{(0)} = X_{(0)}\ \Delta\ R_{(0)} = \overline{\overline{X_{(0)}} \cup R_{(0)}} \cup \overline{\overline{R_{(0)}} \cup X_{(0)}}, \tag{55}$$

$$B_{(1)} = \overline{X_{(0)}} \cap R_{(0)} = \overline{X_{(0)} \cup \overline{R_{(0)}}}. \tag{56}$$

(2) The recursive relations:

$$\begin{aligned} D_{(i)} &= [\overline{X_{(i)}} \cap \overline{R_{(i)}} \cap B_{(i)}] \cup [\overline{X_{(i)}} \cap R_{(i)} \cap \overline{B_{(i)}}] \\ &\quad \cup [X_{(i)} \cap \overline{R_{(i)}} \cap \overline{B_{(i)}}] \cup [X_{(i)} \cap R_{(i)} \cap B_{(i)}] \\ &= \overline{[X_{(i)} \cup R_{(i)} \cup \overline{B_{(i)}}]} \cup \overline{[X_{(i)} \cup \overline{R_{(i)}} \cup B_{(i)}]} \\ &\quad \cup \overline{[\overline{X_{(i)}} \cup R_{(i)} \cup B_{(i)}]} \cup \overline{[\overline{X_{(i)}} \cup \overline{R_{(i)}} \cup \overline{B_{(i)}}]}, \end{aligned} \tag{57}$$

$$\begin{aligned} B_{(i+1)} &= [\overline{X_{(i)}} \cap R_{(i)} \cap B_{(i)}] \cup [\overline{X_{(i)}} \cap R_{(i)} \cap \overline{B_{(i)}}] \\ &\quad \cup [\overline{X_{(i)}} \cap \overline{R_{(i)}} \cap B_{(i)}] \cup [X_{(i)} \cap R_{(i)} \cap B_{(i)}] \\ &= \overline{[X_{(i)} \cup \overline{R_{(i)}} \cup \overline{B_{(i)}}]} \cup \overline{[X_{(i)} \cup \overline{R_{(i)}} \cup B_{(i)}]} \\ &\quad \cup \overline{[X_{(i)} \cup R_{(i)} \cup \overline{B_{(i)}}]} \cup \overline{[\overline{X_{(i)}} \cup \overline{R_{(i)}} \cup \overline{B_{(i)}}]}, \end{aligned} \tag{58}$$

where $i = 0,1,2,\ldots,k-1$.

(3) The final solution:

$$X - R = D = [D_{(k-1)}, D_{(k-2)}, \ldots, D_{(0)}]. \tag{59}$$

This algorithm can be implemented in the DOCIP architecture by the program (instructions):

Assume start with $X_{(0)}$ in M_1 and $R_{(0)}$ in M_2.

Calculate $D_{(0)}$ and $B_{(1)}$:

$$(1)\ \ M_1 \cup \overline{M_2} \to M_3\ \&\ \text{out}\ [= \overline{B_{(1)}}],$$

$$(2)\ \ \overline{M_1} \cup M_2 \to M_1\ [= \overline{X_{(0)}} \cup R_{(0)}],$$

$$(3)\ \ \overline{M_2} \cup \overline{M_3} \to \text{out}\ [= D_{(0)}].$$

Calculate $D_{(1)}$ and $B_{(2)}$:

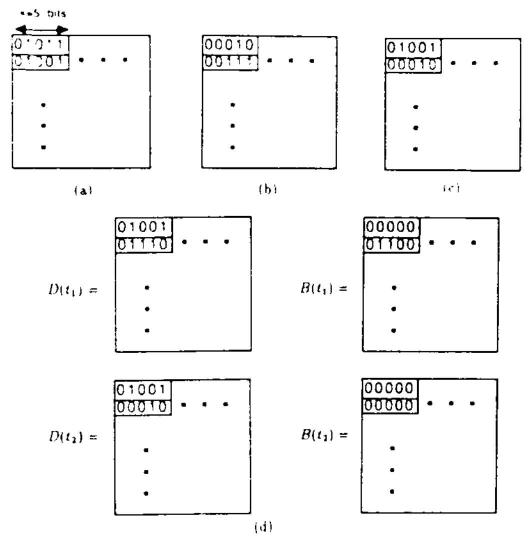


Fig. 20. Parallel subtraction of binary row-coded numbers: (a) image X of operands; (b) image R of other operands; (c) ouput $X - R$; (d) procedure for parallel subtraction $X - R$.

(1) $X_{(1)} \rightarrow M_1$,

(2) $M_1 \cup M_3 \rightarrow M_2$,

(3) $M_1 \cup M_3 \rightarrow M_1$,

(4) $R_{(1)} \rightarrow M_3$,

(5) $M_2 \cup M_3 \rightarrow M_2$,

(6) $M_1 \cup M_3 \rightarrow M_3$,

(7) $M_2 \cup M_3 \rightarrow M_2$,

(8) $R_{(1)} \rightarrow M_3$,

(9) $M_1 \cup M_3 \rightarrow M_1$,

(10) $M_1 \cup M_2 \rightarrow M_2$,

(11) $X_{(1)} \rightarrow M_2$,

(12) $M_1 \cup M_3 \rightarrow M_3$,

(13) $B_{(1)} \rightarrow M_1$,

(14) $M_1 \cup M_3 \rightarrow M_3$,

(15) $M_2 \cup M_3 \rightarrow$ out $[= D_{(1)}]$,

(16) $X_{(1)} \rightarrow M_3$,

(17) $M_1 \cup M_4 \rightarrow M_1$,

(18) $R_{(1)} \rightarrow M_3$,

(19) $M_1 \cup \bar{M}_3 \rightarrow M_1$,

(20) $\bar{M}_1 \cup M_2 \rightarrow M_3$ & out $[= B_{(2)}]$.

Calculate $D_{(2)}$ to $D_{(k-1)}$ and $B_{(3)}$ to $B_{(k)}$:

Use the same instructions for calculating $D_{(1)}$ and $B_{(2)}$ except that $X_{(1)}$ and $R_{(1)}$ [and $D_{(1)}$ and $B_{(2)}$] are replaced by $X_{(i)}$ and $R_{(i)}$ [and $D_{(i)}$ and $B_{(i+1)}$] in each iteration, and in the beginning of an iteration the memory M_3 stores $B_{(i)}$ instead of $\overline{B_{(1)}}$, $i = 2,3,\ldots,k$.

Therefore, the total execution time in the DOCIP to complete this parallel subtraction is $t(k) \le 20(k-1) + 3 = 20k - 17 = O(k)$.

Appendix C. Subtraction of Binary Symbol-Coded Numbers

Similar to addition, we gradually use 4 symbolic substitution rules [Fig. 21(a)], but Rules 1 and 4 are not necessary for single-pixel coding. The symbolic substitution system using single-pixel coding for binary subtraction can be realized as

$$Y(t_0) = X, \quad (60)$$

$$Y(t_{j+1}) = \cup_{i=1}^{4} \{[Y(t_j) \odot R^{(i)}] \cap M\} \oplus Q^{(i)}$$
$$\cup_{i=1}^{4} \{[Y(t_j) \oplus \check{R}_1^{(i)}] \cup [Y(t_j) \oplus \check{R}_2^{(i)}] \cap M\} \oplus Q^{(i)}$$
$$\cup_{i=2}^{3} \{[Y(t_j) \oplus \check{R}_1^{(i)}] \cup [Y(t_j) \oplus \check{R}_2^{(i)}] \cap M\} \oplus Q^{(i)}, \quad (61)$$

where $Y(t_{k+1})$ is the result of the subtraction, $j = 0,1,2,\ldots,k$, k is word size (i.e., the number of bits in an operand); $R^{(i)} = [R_1^{(i)},R_2^{(i)}]$ and $Q^{(i)}$ are shown in Fig. 21(b) and represented as

$$(1)\ R_1^{(1)} = \phi,\quad R_2^{(1)} = \cup_{i=0}^{1} B^{-i},\quad Q^{(1)} = \phi,$$

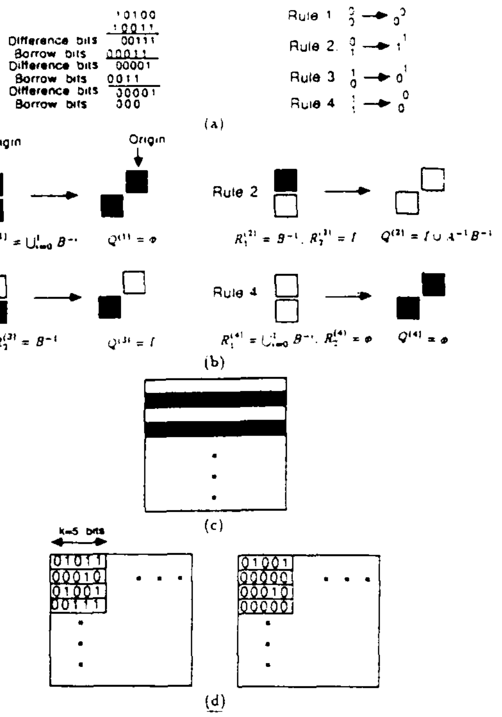


Fig. 21. Parallel subtraction of binary symbol-coded numbers: (a) four symbolic substitution rules for subtraction; (b) reference image pairs $R^{(i)}$ and reference images $Q^{(i)}$,$i = 1,2,3,4$, used for subtraction; because $Q^{(1)}$ and $Q^{(4)}$ are null images, Rules 1 and 4 are not needed for single-pixel coding; (c) mask M; (d) example of parallel subtraction of binary symbol-coded numbers.

$$(2)\ R_1^{(2)} = B^{-1},\quad R_2^{(2)} = I,\quad Q^{(2)} = I \cup A^{-1}B^{-1}$$

$$(3)\ R_1^{(3)} = I,\quad R_2^{(3)} = B^{-1},\quad Q^{(3)} = I,$$

$$(4)\ R_1^{(4)} = \cup_{i=0}^{1} B^{-i},\quad R_2^{(4)} = \phi,\quad Q^{(4)} = \phi.$$

where the null image ϕ and the elementary images are as defined in Subsec. 2.1; and the mask M [Fig. 21(c)] is a shifting of the mask for binary addition. Because $Q^{(1)}$ and $Q^{(4)}$ are null images, and the dilation of a null image is a null image, Rules 1 and 4 are not needed for simple intensity coding. Figure 21(d) gives an example. The execution time for the DOCIP is $t(k) \le 11(k+1) = O(k)$.

Similar to binary addition, we can develop symbolic substitution binary subtraction algorithms with BIA representations for coding a symbol with two or six pixels. However, four symbolic substitution rules are still required because $Q^{(1)}$ and $Q^{(4)}$ will not be equal to the null image. The DOCIPs take approximately the same execution time for binary subtraction using two-pixel or six-pixel coding as for binary addition.

References

1. A. A. Sawchuk and T. C. Strand, "Digital Optical Computing," Proc. IEEE **72**, 758 (1984).
2. Special Issue on Optical Computing, Proc. IEEE 72, No. 7 (1984).
3. A. Huang, Y. Tsunoda, J. W. Goodman, and S. Ishihara, "Optical

Computation Using Residue Arithmetic," Appl. Opt. 18, 149 (1979).
4. D. Psaltis and D. Casasent, "Optical Residue Arithmetic: A Correlation Approach," Appl. Opt. 18, 163 (1979).
5. F. A. Horrigan and W. W. Stoner, "Residue-Based Optical Processor," Proc. Soc. Photo-Opt. Instrum. Eng. 185, 19 (1979).
6. A. M. Tai, I. Cindrich, J. R. Fienup, and C. C. Aleksoff, "Optical Residue Arithmetic Computer with Programmable Computation Modules," Appl. Opt. 18, 2812 (1979).
7. G. Abraham, "Multiple-Values Logic for Optoelectronics," Opt. Eng. 25, 3 (1986).
8. T. T. Tao and D. M. Campbell, "Multiple-Valued Logic: An Implementation," Opt. Eng. 25, 14 (1986).
9. R. Arrathoon and S. Kozaitis, "Shadow Casting for Multiple-Valued Associative Logic," Opt. Eng. 25, 29 (1986).
10. S. L. Hurst, "Multiple-Valued Threshold Logic: Its Status and Its Realization," Opt. Eng. 25, 44 (1986).
11. B. K. Jenkins, A. A. Sawchuk, T. C. Strand, R. Forchheimer, and B. H. Soffer, "Sequential Optical Logic Implementation," Appl. Opt. 23, 3455 (1984).
12. B. K. Jenkins, P. Chavel, R. Forchheimer, A. A. Sawchuk, and T. C. Strand, "Architectural Implications of a Digital Optical Processor," Appl. Opt. 23, 3465 (1984).
13. K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, "Binary Image Algebra and Digital Optical Cellular Image Processors," in *Technical Digest of Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1987), pp. 20–23.
14. K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, "A Cellular Hypercube Architecture for Image Processing," Proc. Soc. Photo-Opt. Instrum. Eng. 829, 331 (1987).
15. K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, "Optical Cellular Logic Architectures Based on Binary Image Algebra," in *Proceedings, IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, Seattle (Oct. 1987), pp. 19–26.
16. K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, "Binary Image Algebra and Optical Cellular Logic Processor Design," Comput. Vision Graphics Image Process. Feb. 1989.
17. A. Huang, "Parallel Algorithms for Optical Digital Computers," in *Technical Digest, IEEE Tenth Internationrl Optical Computing Conference* (1983), pp. 13–17.
18. K. Brenner and A. Huang, "An Optical Processor Based on Symbolic Substitution," in *Technical Digest of Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1985), paper WA4.
19. K.-H. Brenner, A. Huang, and N. Streibl, "Digital Optical Computing with Symbolic Substitution," Appl. Opt. 25, 3054 (1986).
20. K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, "Binary Image Algebra Representations of Optical Cellular Logic and Symbolic Substitution," in *Technical Digest of 1987 Annual Meeting* (Optical Society of America, Washington, DC, 1987).
21. D. Psaltis and R. A. Athale, "High Accuracy Computation with Linear Analog Optical Systems: A Critical Study," Appl. Opt. 25, 3071 (1986).
22. J. Serra, *Image Analysis and Mathematical Morphology* (Academic, New York, 1982).
23. K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, "Programming a Digital Optical Cellular Image Processor," in *Technical Digest of 1987 Annual Meeting* (Optical Society of America, Washington, DC, 1987).
24. B. K. Jenkins and A. A. Sawchuk, "Optical Cellular Logic Architectures for Image Processing," in *Proceedings, IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management*, Florida (Nov. 1985), pp. 61–65.
25. K.-H. Brenner, "New Implementation of Symbolic Substitution Logic," Appl. Opt. 25, 3061 (1986).
26. K.-H. Brenner and G. Stucke, "Programmable Optical Processor Based on Symbolic Substitution, in *Technical Digest of Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1987), pp. 6–8.
27. J. N. Mait and K.-H. Brenner, "Optical Systems for Symbolic Substitution," in *Technical Digest of Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1987), pp. 12–15.
28. T. J. Cloonan, "Strengths and Weaknesses of Optical Architectures Based on Symbolic Substitution," in *Technical Digest of Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1987), pp. 12–15.
29. C. D. Capps, R. A. Falk, and T. L. Houk, "Optical Arithmetic/Logic Unit Based on Residue Number Theory and Symbolic Substitution," in *Technical Digest of Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1987), pp. 62–65.
30. P. A. Ramamoorthy and S. Antony, "Optical MSD Adder Using Polarization Coded Symbolic Substitution," in *Technical Digest of Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1987), pp. 111–114.
31. Ho-In Jeon, "Digital Optical Processor Based on Symbolic Substitution Using Matched Filtering," in *Technical Digest of Topical Meeting on Optical Computing* (Optical Society of America, Washington, DC, 1987), pp. 115–118.
32. M. T. Taso, *et al.*, "Symbolic Substitution Using ZnS Interference Filters," Opt. Eng. 26, 41 (1987).

A Cellular Hypercube Architecture for Image Processing [†]

K. S. Huang, B. K. Jenkins, A. A. Sawchuk
Signal and Image Processing Institute
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-0272

Abstract

In this paper we present a two-dimensional cellular hypercube architecture for image processing that combines features of the conventional hypercube and cellular logic architectures for 2-D computation cells. A unified theory of parallel binary image processing, binary image algebra (BIA), serves as a software tool for designing parallel image processing algorithms. To match the hardware to the software, we characterize the cellular processors using the same algebraic structure as BIA. The two-dimensional cellular hypercube image processor is a cellular SIMD machine with N^2 cells and has a simple overall organization, low cell complexity and fast processing ability. An optical cellular hypercube implementation of BIA is proposed which offers parallel input/output and global interconnection capabilities which are difficult to do in planar VLSI technology.

1. Introduction

Image processing and image analysis tasks have large data processing requirements and inherent parallelism. Parallel cellular logic architectures are generally considered appropriate models for parallel image processing. The cellular logic computer was first inspired by the writings of von Neumann [1][2] on cellular automata. The first highly parallel cellular image processor was suggested by Unger [3][4]. Unger proposed and, later, simulated a two-dimensional array of modules (or processing elements or cells) as a natural spatial computer architecture for image processing and recognition. In this approach, each computational cell is responsible for one pixel (or one element of an image) with its neighboring pixels. A cellular logic (or neighboring logic) operation is then referred to as a transfo:m of an array of data $X(i,j)$ into a new array of data $X'(i,j)$ where each element in the new array has a value determined only by the corresponding element in the original array along with the values of its neighbors. Fig. 1 shows a typical conventional nearest-neighbor connected cellular logic architecture. Some review of cellular image processors can be found in Ref. [5]-[8].

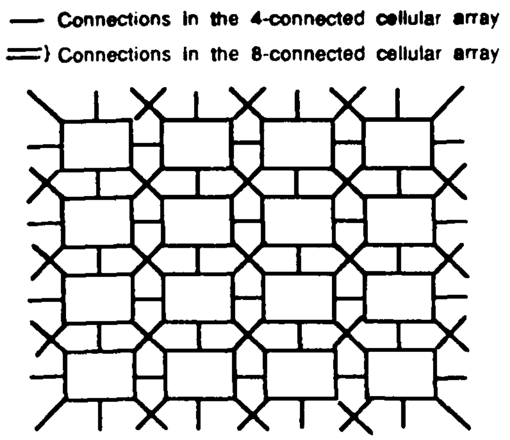

Connections in the 4-connected cellular array
Connections in the 8-connected cellular array

Figure 1: A nearest-neighbor connected cellular logic architecture. Each retangular box represents a computation cell.

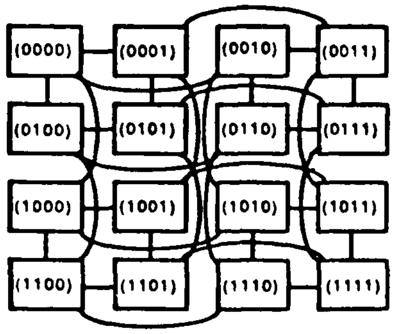


Figure 2: A conventional hypercube (4-cube) laid out in two dimensional space. Its interconnections have no spatial invariance.

One important problem in cellular image processing is that the nearest-neighbor connected cellular image processors have poor communication capabilities and no unified systematic theory for both parallel image processing algorithms and architectures. Section 2 suggests a cellular hypercube architecture to improve the communication capabilities of cellular logic computers. Section 3 summarizes a binary image algebra (BIA) to serve as a unified binary image processing theory for both algorithms and architectures. Section 4 discusses a digital optical cellular hypercube image processor, DOCIP-hypercube, for efficiently implementing BIA.

[0†]Supported by the Air Force Office of Scientific Research under grant AFOSR-84-0181 and by an IBM graduate fellowship.

*From *Applications of Digital Image Processing X*, Proc. of SPIE-The International Society for Optical Engineering Vol. 829, San Diego, California, August, 1987.

2. Cellular Hypercube Architecture

Conventional nearest-neighbor connected cellular arrays have poor communication capabilities; their performance is primarily limited by their $O(1)$ interconnectivity. To improve this while preserving a reasonable number of interconnections, ideally a conventional hypercube increases the interconnectivity to $O(log_2 M)$ for M processing elements (PEs). (We refer to a PE in a cellular computer as a cell which usually has no address and index registers.) A conventional SIMD hypercube computer is comprised of $M = 2^l$ PEs, where l is a non-negative integer. All the PEs are synchronized and operated under the control of a single instruction stream. They are indexed 0 through $M-1$ and the p^{th} PE is referred : $PE(p)$ for $p \in [0, M-1]$. A hypercube is denoted as a l-cube where $l = log_2 M$ represents the number of directly connected PEs. Let $p_{l-1}p_{l-2}...p_0$ be the binary representation of p, and let $p^{(b)}$ be the number whose binary representation is $p_{l-1}...p_{b+1}\overline{p_b}p_{b-1}...p_0$, where $\overline{p_b}$ is the complement of p_b and $0 \leq b < l$. In the hypercube model, $PE(p)$ is connected to those $PE(p^{(b)})$ for $0 \leq b < l$ (i.e. a direct connection exists only between processors whose binary indices differ by one bit position), and data can be transmitted from one PE to another in one step only via this interconnection pattern [9]. The worst case for an inter-PE communication requires $log_2 M$ routes.

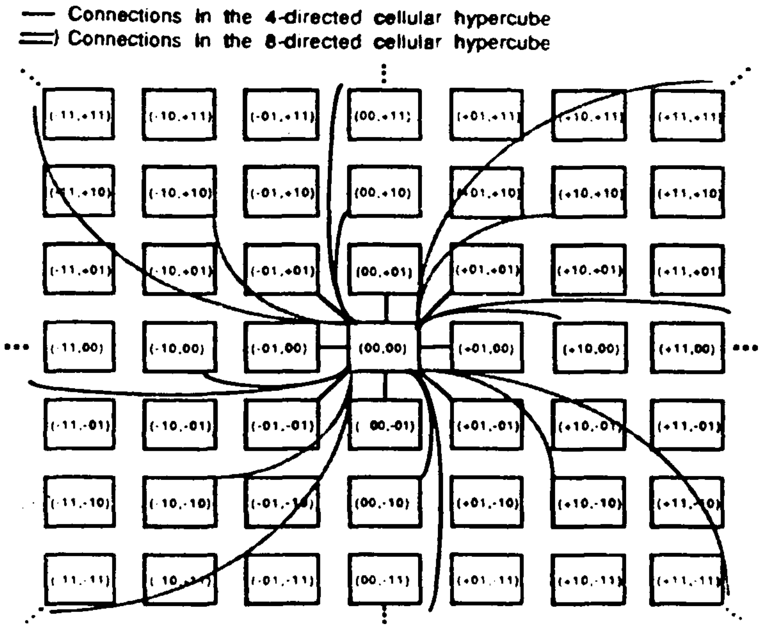


Figure 3: A two-dimensional cellular hypercube — DOCIP-hypercube. Each cell is interconnected with other cells having a *relative* one bit difference in coordinate label in positive or negative x and y directions to achieve a spatially symmetric and invariant interconnection pattern. Only connections from the central cell are shown; all cells are connected identically so the resulting interconnections are space invariant.

However, when a conventional hypercube is laid out in two-dimensional space (e.g. Fig. 2 gives a 4-cube), its interconnection patterns are not space invariant; such spatial invariance is desirable for image processing and for simple hardware implementation. To include this, we increase the interconnections to make a two dimensional cellular hypercube (Fig. 3). The cellular hypercube introduces a symmetrical positive and negative index so that each cell is connected with cells having a *relative* one bit difference in coordinate label in positive or negative x and y directions; the numerical difference of addresses of connected cells is nonzero in at most 1 bit. A two-dimensional SIMD cellular hypercube computer consists of $M = N^2 = (2n-1)^2$ cells and $n = 2^k$, k is a non-negative integer. They are indexed $(-n+1, -n+1)$ through $(n-1, n-1)$ and the $(q,r)^{th}$ cell is refered as $CELL(q,r)$ for $q, r \in [-n+1, n-1]$. In the 4-directed cellular hypercube (cellular hypercube4) model, $CELL(q,r)$ is connected to those $CELL(q \pm 2^d, r)$ and $CELL(q, r \pm 2^d)$ for $0 \leq d < k$; and in the 8-directed cellular hypercube (cellular hypercube8) model, $CELL(q,r)$ is connected to those $CELL(q \pm 2^d, r)$, $CELL(q, r \pm 2^d)$ and $CELL(q \pm 2^d, r \pm 2^d)$ for $0 \leq d < k$. Data can be transmitted from one cell to another in one step only via this interconnection pattern, although it occurs in parallel for each pixel. For $N^2 = (2n-1)^2$ cells, the worst case for an inter-cell communication requires $2log_2 n$ or $4log_2 n$ (they are $O(log_2 N)$) routes for the 8-directed or 4-directed cellular hypercube repectively.

This cellular hypercube architecture requires a 3-D global interconnection mechanism which is difficult to implement on a planar VLSI chip [7][10][11]. However, in principle, the 3-D interconnection mechanism is realizable by digital optical systems, because the general architectural structure of a digital optical computer is inherently 3-dimensional [12][13]. Thus, a digital optical cellular image processor based on the cellular hypercube architecture

(DOCIP-hypercube) is a possible implementation.

To develop a two-dimensional image processor, we face the problem that image processing has no standard unified theory, and so many image processing algorithms and architectures exist in a state of chaos. Thus, we first discuss a simple unified consistent theory of image processing (covering both algorithms and architectures) in section 3, and then consider its optical implementation on a digital optical cellular hypercube processor, DOCIP-hypercube, in section 4.

3. Binary Image Algebra

An algebraic structure provides a theoretical framework of image processing because algebra is a foundation of mathematics, computer language and automata theories. During the past few years, numerous papers have used an algebraic approach to aid in image processing [14]-[19]. Here, a binary image algebra (BIA) is summarized to serve as the software theory of cellular image processors.

3.1 Basic Definitions

In general, a binary digital image is defined as a function f mapping each grid point (x,y) of the picture on an orthogonal coordinate system onto the set composed of 1 (white, i.e. image point) and 0 (black, i.e. background point). However, to have a better compact parallel representing form of a binary image, we can use the coordinates of image points ('1's) to specify an image. In this paper, an image is treated as the set of coordinates of image points (set of pixels that have value 1). We begin the description of BIA by defining our artificial universe:

Definition 3.1 The Universal Image: The universal image is a set $W = \{(x,y) \mid x \in Z_n, y \in Z_n\}$, where $Z_n = \{0, \pm1, \pm2, ..., \pm n\}$ and n is a positive integer. Thus, all images are defined in a $(2n+1) \times (2n+1)$ array of points.

Definition 3.2 Image Space: The image space is the power set (the set of all subsets) of the universal image, i.e. $S = P(W)$.

Definition 3.3 Image: A set X is an image if and only if X is an element of the image space S, i.e. X is a subimage of the universal image W.

Definition 3.4 Image Point: A point (x,y) is an image point of an image X if and only if (x,y) is an element of the set X.

Definition 3.5 Image Transformation: A transformation T is an image transformation if and only if T is a function mapping from the image space S to the image space S.

Definition 3.6 Three Fundamental Operations

There are three fundamental operations:

1. Complement of an image X: $\overline{X} = \{(x,y) \mid (x,y) \in W \wedge (x,y) \notin X\}$
2. Union of two images X and R: $X \cup R = \{(x,y) \mid (x,y) \in X \vee (x,y) \in R\}$
3. Dilation of two images X and R:

$$X \oplus R = \begin{cases} \{(x_1+x_2, y_1+y_2) \in W \mid (x_1,y_1) \in X, (x_2,y_2) \in R\} & (X \neq \phi) \wedge (R \neq \phi) \\ \phi & otherwise \end{cases}$$

Remark: "$\in$" means "belongs to", "$\wedge$" means "and", "$\vee$" means "or", and "ϕ" is the null image having no image point. Note that X usually represents an input or data image and R is a reference image. We can define other image operations as fundamental operations instead of these three operations. The reason for choosing these three operations is because of their simplicity, simple software design and simple hardware implementation. Figure 4 gives an example of these fundamental operations.

$$\underset{\text{Input image}\quad X}{\boxed{\begin{matrix}0&0&0&0&0&0&0\\0&0&1&1&1&1&0\\0&0&0&1&1&1&0\\0&0&0&0&1&1&0\\0&0&0&0&0&1&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{matrix}}}
\quad
\underset{\text{Reference image}\quad R}{\boxed{\begin{matrix}0&0&0&0&0&0&0\\0&0&0&0&0&0&0\\0&0&1&1&1&0&0\\0&0&1&1&1&0&0\\0&0&1&1&1&0&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{matrix}}}
\quad
\underset{\text{Complement}\quad \overline{X}}{\boxed{\begin{matrix}1&1&1&1&1&1&1\\1&1&0&0&0&0&1\\1&1&1&0&0&0&1\\1&1&1&1&0&0&1\\1&1&1&1&1&0&1\\1&1&1&1&1&1&1\\1&1&1&1&1&1&1\end{matrix}}}
\quad
\underset{\text{Union}\quad X \cup R}{\boxed{\begin{matrix}0&0&0&0&0&0&0\\0&0&1&1&1&1&0\\0&0&1&1&1&1&0\\0&0&1&1&1&1&0\\0&0&1&1&1&1&0\\0&0&0&0&0&0&0\\0&0&0&0&0&0&0\end{matrix}}}
\quad
\underset{\text{Dilation}\quad X \oplus R}{\boxed{\begin{matrix}0&1&1&1&1&1&1\\0&1&1&1&1&1&1\\0&1&1&1&1&1&1\\0&0&1&1&1&1&1\\0&0&0&1&1&1&1\\0&0&0&0&1&1&1\\0&0&0&0&0&0&0\end{matrix}}}$$

Figure 4: An example of fundamental operations: complement $^-$, union $\cup$, and dilation $\oplus$.

Definition 3.7 Elementary Images: There are 5 elementary images:

1. $I = \{(0,0)\}$ — consisting of an image point at the origin

2. $A = \{(1,0)\}$ — consisting of an image point right of the origin
3. $A^{-1} = \{(-1,0)\}$ — consisting of an image point left of the origin
4. $B = \{(0,1)\}$ — consisting of an image point above the origin
5. $B^{-1} = \{(0,-1)\}$ consisting of an image point below the origin

In fact, these 5 elementary images could be reduced to 4 elementary images, because $I = A^0 \equiv A \oplus A^{-1} = B^0 \equiv B \oplus B^{-1}$.

3.2 Two Fundamental Principles

Two fundamental principles basically define the binary image algebra (BIA). Before stating these two principles, we give some preliminary results. The proofs are omitted here for brevity [19].

Lemma 1. $\overline{(X \oplus \overline{R}) \cup (\overline{X} \oplus R) \cup \overline{I}} = \begin{cases} I & if X = \check{R} \\ \phi & otherwise \end{cases} \quad \forall X, R \in P(W)$.

Remark: $I = \{(0,0)\}$ is an elementary image, $\check{R} = \{(-x,-y) \mid (x,y) \in R\}$ is a *reflected* reference image, and "$\forall$" means "for all".

Theorem 1. Any image transformation $T : P(W) \rightarrow P(W)$ can be expressed as

$$T(X) = \bigcup_{i=1}^{k} \{ \overline{(\overline{X} \oplus R_i) \cup (X \oplus \overline{R_i}) \cup \overline{I}} \oplus Q_i \}$$

where $k \leq l$, l is the cardinality (i.e. the number of elements) of $P(W)$, and R_i and Q_i are the reference images used to form any desired image transformation.

Remark: $\bigcup_{i=1}^{k} R_i = R_1 \cup R_2 \cup ... \cup R_k$.

*Theorem 2.*Any image can be represented as

$$X = \bigcup_{(i,j) \in X} A^i B^j$$

where $A^i B^j \equiv A^i \oplus B^j$, $A^i \equiv \underbrace{A \oplus A \oplus ... \oplus A}_{i} = \{(i,0)\}$ (if $i = -k$, i is a negative integer, then $A^{-k} \equiv \underbrace{A^{-1} \oplus A^{-1} \oplus ... \oplus A^{-1}}_{k} = \{(-k,0)\}$) and A, B, A^{-1}, B^{-1} are the elementary images defined in Definition 3.7.

Principle 1. Fundamental Principle of Image Transformations
Any image transformation T can be implemented by using appropriate reference images R and the three fundamental operations: 1. Complement $\overline{X}$ of an image X, 2. Union $\cup$ of two images, 3. Dilation $\oplus$ of two images.

Proof. It follows from Theorem 1.

Principle 1 solves almost any problem in binary image processing/analysis, especially in shape inspection, size verification, and pattern recognition with shift, scaling and rotational invariance [19] [20]. However, in reality, how can we build a computer that offers arbitrary and programmable reference images for dilations? Do we really need a large memory to store many kinds of reference images? The answer is "no". The second fundamental principle suggests an economical way to accomplish this.

Principle 2. Fundamental Principle of Reference Images
Any reference image R can be generated from elementary images $(I, A, A^{-1}, B, B^{-1})$ by using the three fundamental operations.

Proof. It follows from Theorem 2.

Therefore, by the above principles, the algebraic structure of BIA can be defined as:

Definition of Binary Image Algebra (BIA)
Binary image algebra is an algebra with an image space S and a family F of finitary operations including 5 elementary images, which are 0-ary operations, and 3 fundamental operations, which are non 0-ary operations. Symbolically,

$$\text{BIA} = (P(W); \oplus, \cup, ^{-}, I, A, A^{-1}, B, B^{-1})$$

i.e. $S = P(W)$ and $F = (\oplus, \cup, ^{-}, I, A, A^{-1}, B, B^{-1})$.

Remark: For any integer k, a k-ary operation on S is defined to be a function $f : S^k \rightarrow S$. Thus, a unary (or 1-ary) operation on S is simply a function on S to S. A binary (or 2-ary) operation on S is a function on S^2 to S. For completeness, we define a nullary (or 0-ary) operation on S to be a particular element of S.

4. Implementation: DOCIP-hypercube

To map algorithms into architectures in a transparent way, we use an algebraic approach for describing a cellular image processor first. Then we design the digital optical cellular image processors (DOCIPs) and their optical implementation. The DOCIP-hypercube, a two-dimensional cellular hypercube, uses optical parallel global interconnection capabilities and offers further improvements in speed and flexibility.

4.1 Algebraic Description

Having defined cellular automata and the implementation requirements of BIA, we describe the DOCIP in an algebraic way:

Definition of Cellular Automata

A cellular automaton is an algebra $A = (S; F, N_c)$ where S is the state space which is a set of states, F is a family of transition functions, and N_c is the neighborhood configuration.

Constraints of Implementing BIA:

1. $S \supset P(W)$
2. $F \supset \{\oplus, \cup, ^-\}$
3. $N_c \supset I \cup A \cup A{-}1 \cup B \cup B^{-1}$ (or $N_c \supset A \cup A^{-1} \cup B \cup B^{-1}$)

where "$\supset$" means "contains".

Thus, in terms of cellular automata, the DOCIPs have to satisfy the above constraints for realizing BIA. For storing input images and temporary results in a more flexible way, the DOCIPs utilize three memory modules and share the same algebraic structure (except the neighborhood configuration):

$$\text{DOCIP} = (P(W^3); \oplus, \cup, ^-, N_c)$$

where N_c can be one of the following 4 types:

1. DOCIP-array4: each cell connects with its four nearest neighbors and itself, i.e. $N_{array4} = I \cup A \cup A^{-1} \cup B \cup B^{-1}$.
2. DOCIP-array8: each cell connects with its eight nearest neighbors and itself, i.e. $N_{array8} = \bigcup_{i,j=0,\pm 1} A^i B^j$.
3. DOCIP-hypercube4: each cell connects with those cells in the 4 directions at distances $1, 2, 4, 8, ..., 2^k$ from it, i.e. $N_{hypercube4} = (\bigcup_{i=o,\pm 1,\pm 2,\ldots,\pm 2^k} A^i) \cup (\bigcup_{i=\pm 1,\pm 2,\ldots,\pm 2^k} B^i)$.
4. DOCIP-hypercube8: each cell connects with those cells in the 8 directions at distances $1, 2, 4, 8, ..., 2^k$ from it, i.e. $N_{hypercube8} = \bigcup_{i,j=0,\pm 1,\pm 2,\ldots,\pm 2^k} A^i B^j$.

Among of these, DOCIP-hypercube4 is most preferred because its hardware requirement is simpler than DOCIP-hypercube8 while they have the same order of performance. The DOCIP-array architectures are nearest-neighbor connected but have poor global communication capabilities.

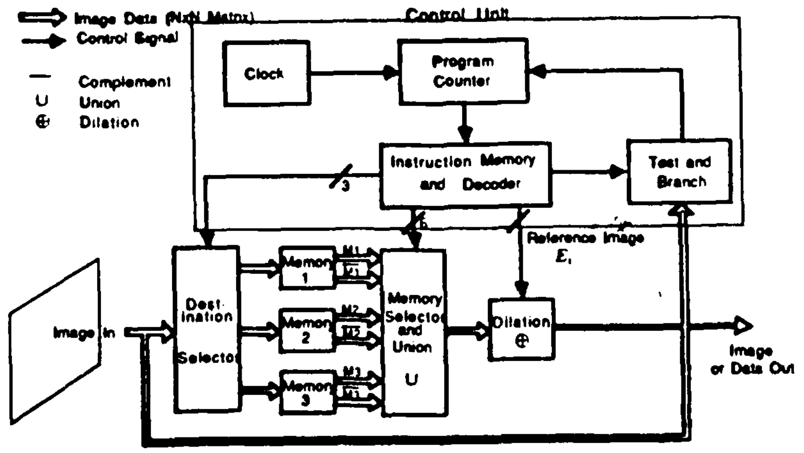


Figure 5: A digital optical cellular image processor (DOCIP) architecture — one implementation of binary image algebra (BIA). The DOCIP-array requires 9 (or 5) control bits for reference image E_i. The DOCIP-hypercube requires $O(logN)$ control bits for reference image E_i.

4.2 General Description

From the above algebraic description, the DOCIPs have the same algebraic structure and differ only in their neighborhood configurations N_c. Thus, they share the same architecture as shown in Fig. 5, but have different configurations of the reference images E_i depending on the optical interconnection network which defines the neighborhood. In practical applications, a *larger* reference image R can be generated from a set of *smaller* reference image(s) E_i by a "sequential dilation". If it is possible to decompose R into a sequence $R = E_1 \oplus E_2 \oplus ... \oplus E_k$, then

$$X \oplus R = (...((X \oplus E_1) \oplus E_2) \oplus ... \oplus E_k).$$

Figure 6 gives an example: a dilation with a polygon (filled) reference image R is implemented by a sequential dilation with four line reference images E_i; it requires $O(L)$ time for the DOCIP-array and $O(log_2 L)$ time for the DOCIP-hypercube where L is the maximun length of the four line reference images. This decomposition may not always exist, in which case R can always be decomposed as $R = R_1 \cup R_2 \cup ... \cup R_k$, and then

$$X \oplus R = (X \oplus R_1) \cup (X \oplus R_2) \cup ... \cup (X \oplus R_k)$$

where each R_j can be composed from the smaller reference images E_i.

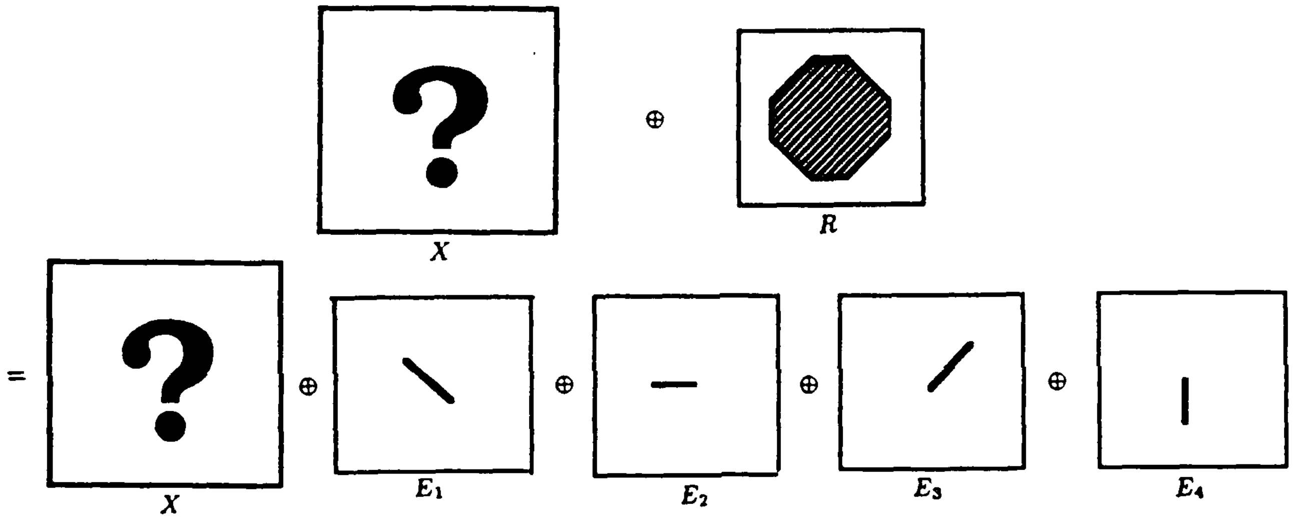


Figure 6: An example of decomposing a dilation with a *larger* reference image R into a sequential dilation with some *smaller* reference image E_i. It requires $O(N)$ and $O(logN)$ time for DOCIP-array and DOCIP-hypercube respectively.

The proposed DOCIP as shown in Fig. 5 is a cellular SIMD machine and consists of an array of cells or processing elements (PEs) under the supervision of a control unit. The control unit includes a clock, a program counter, a test and branch module for feedback control, and an instruction decoder for storing instructions and decoding them to supervise cells. The array of cells includes a $1 \times 3 \times N^2$ bit destination selector, three $N \times N \times 1$ bit memories for storing images, a memory selector, and a dilation unit.

The DOCIP shown in Fig. 5 operates as follows: (1) a binary image ($N \times N$ matrix) is selected by the destination selector and then stored in any memory as the instruction specifies; (2) after storing the images (1 to 3 $N \times N$ matrices), these images and their complemented versions are piped into the next stage, which forms the union of any combination of images; (3) the result is sent to a dilation where the reference image specified by the instruction is used to control the type of dilation; (4) finally, the dilated image can be output, tested for program control, or fed back to step (1) by the address field of the instruction.

4.3 Optical Implementation

The entire system can be realized by an optical gate array with optical 3-D interconnections [10][20]. Figure 7 shows an optical concept for the DOCIP-hypercube implementation. It embeds an array of cells in an array of optical binary gates and performs interconnections of these gates by an optical hologram. It should be noted that current optical technology has implemented only arrays of moderately large numbers of gates (500 x 500) at very slow (~ms) switching speeds, and alternatively, arrays of small numbers of gates (2 x 2 to 6 x 6) at fast switching speeds (0.1μs - 50ps) [21][22]. Current ongoing research in a number of laboratories looks promising in eventually providing the needed arrays of large numbers of gates with reasonably fast switching speeds. Alternatively, control of the DOCIP can be easily realized by using an electronic host instead of the optical control unit, since control of SIMD systems is primarily a serial process. The tradeoff is a possible inefficiency in the interfaces between electronic and optical units. Because of this, the all-optical approach may be preferable in the long term. To efficiently utilize optical gates, they can be interconnected with a 2-D optical multiplexing technique in which a common controllable mask is used for all cells. The optical multiplexing technique has following advantages: 1) the DOCIP will no longer require the broadcasting of instructions from the control unit — instead all cells fan their outputs into a common controlling mask pixel; 2) it will reduce the number of gates; and 3) each cell has a simple structure — essentially containing only a 3-bit memory with inverting and non-inverting outputs, and a multiple-input OR gate for dilation.

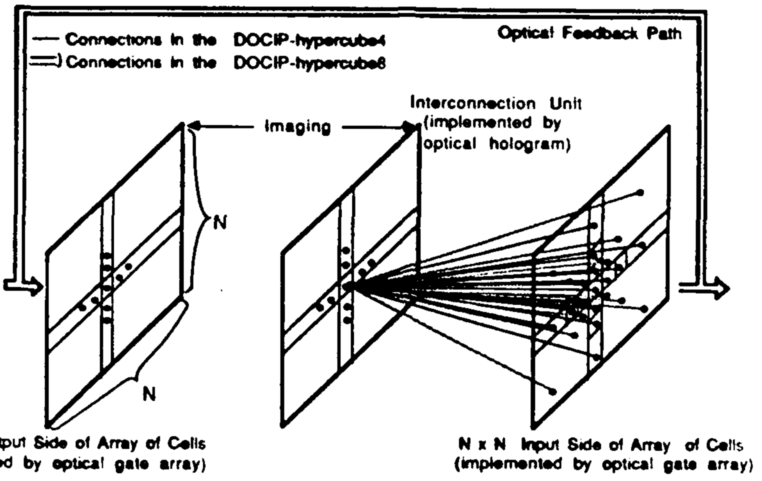

— Connections in the DOCIP-hypercube4
⟹ Connections in the DOCIP-hypercube8
Optical Feedback Path
Interconnection Unit (implemented by optical hologram)
Imaging
N
N
N x N Output Side of Array of Cells (implemented by optical gate array)
N x N Input Side of Array of Cells (implemented by optical gate array)

Figure 7: An optical cellular hypercube (DOCIP-hypercube4 or DOCIP-hypercube8). Imaging optics are omitted for clarity. Each cell connects with cells in the 4 directions or 8 directions at distances $1, 2, 4, 8, ..., 2^k$ from it by optical 3-D free interconnection. The input and output sides of the optical gate array are interconnected by an optical feedback path and are shown separately for clarity.

To avoid the well-known drawbacks of conventional computers based on von Neumann principles [12] [13], the machine in Fig. 5 has one instruction which implements the three fundamental operations of BIA along with fetch and store. This design uses the parallelism of optics to simultaneously execute instructions involving all N^2 picture elements.

This single instruction has the following format:

$$(s_1, s_2, ..., s_6, n_1, n_2, ..., n_k, d_1, d_2, d_3, j_1, j_2, a_1, a_2, ..., a_l, b_1, b_2, ..., b_l)$$

where k is determined by the chosen neighborhood configuration N_c, the DOCIP-array requires $k = 5$ or $k = 9$ bits for controlling the reference image R at a clock cycle, the DOCIP-hypercube requires $k = O(log_2 N)$ for N^2 cells, and l defines the maximum length of a program: 2^l. The functions of these $11 + k + 2l$ instruction codes are:

- $s_1, s_2, ..., s_6$ select the output from the memory elements;
- $n_1, n_2, ..., n_k$ control the neighborhood mask, i.e. to specify the reference image;
- d_1, d_2, and d_3 select the destination memory for storing the image;
- j_1 and j_2 flag an absolute jump or conditional jump;
- $a_1, a_2, ..., a_l$ are the address for a jump; and
- $b_1, b_2, ..., b_l$ are the address of the instruction.

In contrast with the DOCIP-array, the DOCIP-hypercube increases the interconnection complexity to $O(log_2 N)$, but is able to perform many global operations in $O(log_2 N)$ time. Compared with conventional-array processors having serial or N-parallel input/output, the DOCIP-array will have the same order of performance in local and global operations but will be improved in input/output performance, and in principle could be as low as $O(1)$ in I/O operations. The DOCIP-hypercube will not only be improved in input/output performance but also in global operations. With external memory, it can be demonstrated to be general purpose in the sense that it simulates any Turing machine. One important feature in the design of the DOCIP-array and DOCIP-hypercube is that optical 3-D free interconnection capabilities can be used to reduce the cell hardware requirements as well as solve the global connection and I/O problems which are difficult to solve by planar VLSI technology.

5. Conclusions

A two-dimensional cellular hypercube architecture has been proposed to have the best features of both two-dimensional hypercube and cellular logic architectures for image processing. BIA suggests an unified theory of parallel binary image processing for developing parallel algorithms/languages and can be generalized to grey-level images. The DOCIP-hypercube utilizes the parallel communication and global interconnection capabilities of optics for avoiding communication bottlenecks and matching BIA parallel algorithms with the two-dimensional cellular hypercube architecture. The current design of DOCIP-hypercube has an extremely simple cell organization with only a 3-bit flip-flop memory and a multiple-input OR gate for emphasizing the binary image processing and reducing the hardware cost. BIA and DOCIP-hypercube can have many applications in character recognition, industrial inspection, medical and scientific research, especially morphological image processing. The future work is its optical experimental demonstration and its analysis of different cell structures with larger grain sizes for developing fast sophisticated vision algorithms.

Ackowledgements

The authors would like to thank T. C. Strand, P. Chavel, E. K. Blum, and R. Chellappa for their contributions to this research.

References

[1] J. von Neumann, "The General Logical Theory of Automata," in *Cerebral Mechanisms in Behavior-The Hixon Symposium* , L. A. Jeffress, Ed. New York: Wiley, 1951.

[2] J. von Neumann,*Theory of Self-reproduction Automata,* edited by A. W. Burks, Univ. of Illinois Press, 1966.

[3] S. H. Unger, "A Computer Oriented Toward Spatial Problems," *Proc. IRE,* Vol.46,1958, pp. 1744-1750.

[4] S. H. Unger, "Pattern Detection and Recognition," *Proc. IRE,* Vol. 47, 1959, pp.1737-1752.

[5] K. Preston and M. J. B. Duff, *Modern Cellular Automata — Theory and Applications,* Plenum Press, New York, 1984.

[6] K. Preston Jr. et al., "Basics of Cellular Logic with Some Application's in Medical Image Processing," *Proc. of IEEE,* Vol. 67, No. 5, 1979, pp. 826-856.

[7] A. Rosenfeld, "Parallel Image Processing Using Cellular Arrays," *IEEE Computer,* Jan. 1983, pp. 14-20.

[8] K. Preston Jr., "Cellular Logic Computers for Pattern Recognition," *IEEE Computer,* Jan. 1983, pp.36-47.

[9] K. Hwang and F. A. Briggs, *Computer Architecture and Parallel Processing,* McGraw-Hill, New York, 1984.

[10] B. K. Jenkins and A. A. Sawchuk, "Optical Cellular Logic Architectures for Image Processing," *IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management,* Florida, , pp. 61-65, Nov. 1985.

[11] A. A. Sawchuk and B. K. Jenkins, "Optical Cellular Logic Processors," *Optical Society of America 1985 Annual Meeting,* Washington, D.C..

[12] A. A. Sawchuk and T. C. Strand, "Digital Optical Computing," *Proc. IEEE,* Vol. 72, pp. 758-779,1984.

[13] B. K. Jenkins, et al., "Architectural Implications of A Digital Optical Processor," *Applied Optics,* Vol. 23, No. 19, pp. 3465-3474, 1984.

[14] J. Serra, *Image Analysis and Mathematical Morphology,* Academic Press, Inc., New York, 1982.

[15] R. M. Lougheed, D. L. McCubbrey, and S. R. Sternberg, " Cytocomputers: Architectures for Parallel Image Processing," *Proc. IEEE Workshop Picture Data Description and Management,* CA, pp. 281-286, 1980.

[16] G. X. Ritter and P. D. Gader, "Image Algebra Techniques for Parallel Image Processing",*Journal of Parallel and Distributed Computing,* Vol. 4, No. 1, pp. 7-44, 1987.

[17] C. R. Giardina, "The Universal Imaging Algebra", *Pattern Recognition Letters,* Vol. 2, pp. 165-172, 1984.

[18] T. Agui, et al., "An Algebraic Approach to the Generation and Description of Binary Pictures", *IEEE Transaction on Pattern Analysis and Machine Intelligence,* Vol. PAMI-4, No. 6, pp. 635-641, 1982.

[19] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Binary Image Algebra", *Paper in Preparation.*

[20] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Novel Parallel Pattern Recognition Algorithms Based on Binary Image Algebra", *Paper in Preparation.*

[21] B. K. Jenkins, et al., "Sequential Optical Logic Implementation," *Applied Optics,* Vol. 23, No. 19, pp. 3455-3464, 1984.

[22] T. E. Bell, "Optical Computing: A Field in Flux," *IEEE Spectrum,* Vol. 23, No. 8, pp. 34-57, 1986.

2.4 Nonlinear Optical Processing with Halftones: Degradation and Compensation Models

This paper is concerned with the halftone process used in coherent optical spatial filtering systems to provide general nonmontonic nonlinear functions.

Nonlinear optical processing with halftones: accurate predictions for degradation and compensation

Ahmad Armand, Alexander A. Sawchuk, and Timothy C. Strand

A general analysis of the halftone process for nonlinear transformations in optical signal processing is presented. The analysis considers the effects of the nonideal characteristics of the recording medium. The results predict output errors due to different parts of the recording medium characteristic curve for any nonlinear transformation. A synthesis method for a discrete halftone screen density profile is also described. This produces an optimum halftone screen density profile for any form of recording medium characteristic curve and any type of nonlinearity in the sense that it minimizes the mean-square difference between desired and degraded outputs. The results of a computer simulation for logarithmic and level slice functions are given.

I. Introduction

In halftone nonlinear optical signal processing the continuous tone input picture is transformed into a binary picture by contact printing the continuous input data through a halftone screen onto a high-contrast recording medium. The product of the input and halftone screen transmittances is clipped in the process, giving an array of binary dots whose size is a function of clip level, the input transmittance, and the halftone transmittance profile. The periodic nature of the halftone screen causes each subregion of the binary image corresponding to a constant input intensity to become locally periodic. When placed in the usual coherent optical filtering system, multiple diffraction orders appear in the Fourier transform plane because of the sampled input. The procedure for producing nonlinearities involves use of one diffraction order combined with specially made halftone screens. A filter is placed in the Fourier plane that transmits the light around one diffraction order and blocks everything else. This in effect demodulates the image.[1] After the filtered diffraction order in the Fourier plane is inverse transformed, the continuous nonlinearity transformed output appears.

When this work was done all the authors were with University of Southern California, Signal & Image Processing Institute, Los Angeles, California 90089. A. Armand is now with Wilkes College, Wilkes-Barre, Pennsylvania 18700, and T. C. Strand is now with IBM Almaden Research Center, San Jose, California 95120.

Received 26 March 1986.
0003-6935/87/061007-08$02.00/0.
© 1987 Optical Society of America.

The above process has been formulated with the assumption that a binary recording medium is used in the halftoning step.[2] With this assumption, once the output intensity is expressed as a function of input intensity and halftone screen density profile (analysis), we can easily invert the problem and get the halftone screen density profile given the relationship between the output and input intensities (synthesis). Unfortunately, almost all recording media deviate from the binary assumption. This deviation, which is quite small for high-contrast photographic films, is quite noticeable for any real-time spatial light modulators presently available.[3,4] Consequently, to utilize accurately the halftone technique, we remove the assumption of a binary recording medium from the mathematical formulation of these processes. Dashiell and Sawchuk modified the results on halftone processing for an ideal recording medium by including the effects of finite slope and saturation density.[5] They developed a numerical procedure, valid for monotonic halftone cells, to compensate in advance for some recording medium effects (precompensation). A closed form solution to this problem has been obtained by Batten and Everett[6] for some limited nonlinear transformations. The formulation of Dashiell and Sawchuk does not predict the effects of general nonlinear characteristics of the recording medium on the overall nonlinear transformation. Moreover, their formulation is restricted to monotonic halftone cell shapes.

In this paper we present a formulation of the halftone process which considers a recording medium with a characteristic curve of general shape and which predicts the final degradation of the output for any halftone screen cell shape. This formulation is examined for the case of a binary recording medium, and its result is compared to previous derivations. To obtain a general solution to the precompensation problem, an approximate method which considers a discrete halftone screen density profile is described. This gives the halftone screen density profile for any form of recording medium characteristic curve and any type of nonlinearity by minimizing in the mean-square sense the difference between desired and degraded outputs. The results of computer simulation for logarithmic and level slice functions are shown.

II. Degradation

When a general recording medium is used in the halftone process, the amplitude transmittance of the halftoned picture consists of pulses such as shown in Fig. 1 that are not binary. The amplitude, width, and shape of these pulses depend on the input picture density levels, halftone screen density profile, and shape of the characteristic curve of the recording medium. Each group of these pulses corresponds to a constant intensity subregion in the input picture. The period L (Fig. 1) of the halftone screen is chosen to be small compared with the period of the highest spatial frequency component in the input picture. Thus any local region of the amplitude transmittance of the halftoned transparency $t_h(x)$ is approximately a periodic sequence of pulses which can be expanded in a complex Fourier series,

$$t_h(x) = \sum_{k=-\infty}^{+\infty} \mathbf{B}_k \exp\left(\frac{-j2\pi kx}{L}\right), \tag{1}$$

where

$$\mathbf{B}_k = \frac{1}{L}\int_0^L t_h(x) \exp\left(\frac{j2\pi kx}{L}\right) dx. \tag{2}$$

In the sum of Eq. (1) each term (denoted by k) represents a grating diffraction order. When we produce the Fourier transform of the halftoned picture in the coherent optical processor, these orders appear in the Fourier plane as isolated spectral islands. The spatial filter in this plane selects a single order. Hence the resulting intensity distribution I_0 at the processor output is

$$\begin{aligned} I_0(I_{in},k) &= |\mathbf{B}_k|^2 \\ &= \left|\frac{1}{L}\int_0^L t_h(x) \exp\left(\frac{j2\pi kx}{L}\right) dx\right|^2, \end{aligned} \tag{3}$$

which relates the intensity at any point of the output picture to the amplitude transmittance of the halftoned picture and the selected order k. Now $t_h(x)$ can be related to the input intensity I_{in} as follows. Let the local input picture intensity that produced the above train of pulses on the halftoned picture be denoted by I_{in}. If the density variation of one period of the halftone screen is represented by $f(x)$, the intensity transmitted by the halftone screen is $I_{in} \times 10^{-f(x)}$.[1] If we let the amplitude transmittance vs log exposure ($\log E$) curve of the recording medium be described by $g(\log E)$, we can write

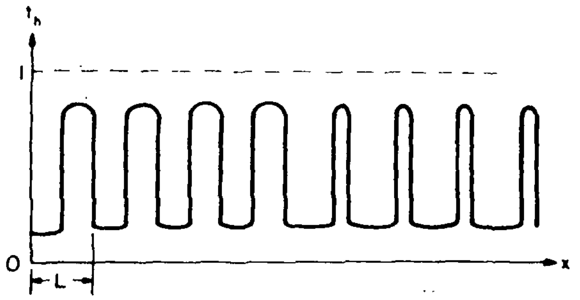


Fig. 1. Amplitude transmittance of a halftoned transparency made with a general nonbinary recording medium.

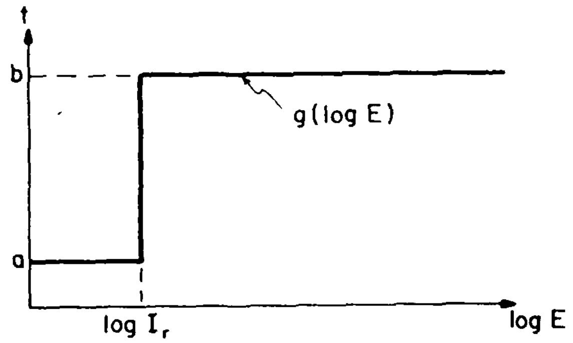


Fig. 2. Characteristic curve of a binary recording medium.

$$t_h(x) = g\{\log[I_{in} \times 10^{-f(x)}]\} = g[\log I_{in} - f(x)]. \tag{4}$$

Replacing this in Eq. (3), we have

$$I_0(I_{in},k) = \left|\frac{1}{L}\int_0^L g[\log I_{in} - f(x)] \exp\left(\frac{j2\pi kx}{L}\right) dx\right|^2, \tag{5}$$

which relates the intensity at any point of the output picture to the intensity of the corresponding point in the input picture through a nonlinear integral relationship. When the specific forms of $g(\log E)$ and $f(x)$ are substituted in this relationship and the integral is solved, the overall relation between the output intensity and the input intensity is nonlinear. This nonlinearity depends on $g(\log E)$, $f(x)$, and the value of the order selected.

A. Binary Recording Medium

The characteristic curve of a binary recording medium having a threshold at $\log I_r$ is shown in Fig. 2. Ideally, $a = 0$ and $b = 1$. Note that this form of characteristic curve is applicable to a positive transparency. We could also consider the more familiar negative transparency, although the basic results remain the same. We will choose the positive transparency curve because it is more similar to the characteristic curves for real-time devices. We now simplify the general relationship of Eq. (5) using the characteristic curve of Fig. 2.

B. Zero Order

For the zero-order case $k = 0$, and Eq. (5) becomes

$$I_0(I_{in},0) = \left|\frac{1}{L}\int_0^L g[\log I_{in} - f(x)]dx\right|^2 . \quad (6)$$

Given $g(\log E)$ as the binary function shown in Fig. 2 and assuming $f(x)$ to be a monotonically increasing function, we have

$$\text{if } \log I_{in} - f(x) < \log I_r \text{ or } x > f^{-1}\left(\log \frac{I_{in}}{I_r}\right), \text{ then } g(\log E) = a.$$

Also,

$$\text{if } \log I_{in} - f(x) \geq \log I_r \text{ or } x \leq f^{-1}\left(\log \frac{I_{in}}{I_r}\right), \text{ then } g(\log E) = b.$$

Substituting these relations into Eq. (6), we have

$$I_0(I_{in},0) = \left[a + \frac{(b-a)}{L}\cdot f^{-1}\left(\log \frac{I_{in}}{I_r}\right)\right]^2 , \quad (7)$$

where $f^{-1}(\cdot)$ is the inverse function of $f(\cdot)$. This result is the same as that obtained previously for the case $a = 0, b = 1$ (Ref. 2) except for the fact that it is obtained in a more straightforward manner and can easily be generalized.

C. Nonzero Order

For $k \neq 0$, the above simplifications for the characteristic curve can be used in Eq. (5) to obtain

$$I_0(I_{in},k) = \left|\frac{1}{L}\left[\int_0^{f^{-1}\left(\log\frac{I_{in}}{I_r}\right)} b \exp\left(\frac{j2\pi kx}{L}\right)dx + \int_{f^{-1}\left(\log\frac{I_{in}}{I_r}\right)}^{L} a \exp\left(\frac{j2\pi kx}{L}\right)dx\right]\right|^2 . \quad (8)$$

If we let

$$x_1 = f^{-1}\left(\log \frac{I_{in}}{I_r}\right), \quad (9)$$

then

$$I_0(I_{in},k) = \frac{(b-a)^2}{\pi^2k^2}\sin^2\left(\frac{\pi k x_1}{L}\right). \quad (10)$$

After substituting the expression for x_1 we have

$$I_0(I_{in},k) = \frac{(b-a)^2}{\pi^2k^2}\sin^2\left[\frac{\pi k}{L} f^{-1}\left(\log \frac{I_{in}}{I_r}\right)\right]. \quad (11)$$

This also agrees with the previously obtained results.[2] This new analysis provides not only a straightforward derivation of the above results, it also leads directly to the new precompensation techniques described in the next section.

III. Precompensation

Several methods are available in practice for generating halftone screens with a desired density profile. Some methods are purely optical and involve the photographic recording of geometrical shadows or diffraction patterns from ruled gratings. Although this technique produces continuous halftone screens, it does not have the flexibility to produce precisely arbitrary screens needed for nonlinear processing.

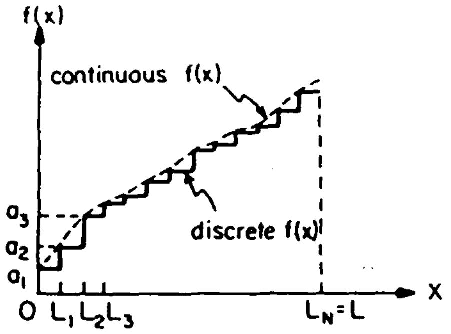


Fig. 3. Step approximation to halftone screen density profile.

Another type of halftone screen density profile that can be generated in practice is a step function approximation to the desired continuous density. These halftone screens are generated by digital image recorders, plotting microdensitometers, or step-and-repeat cameras. Hence the theoretical accuracy available in designing the halftone screen density profile is limited by the practical limitations in making the screen. This motivates the following analysis which considers the halftone screen density profile as a step function approximation of the ideal profile. As will be shown in this section, this assumption helps to simplify the formulas and allows us to obtain some conclusions which cannot be drawn from the continuous density analysis. To do so, we utilize a discrete density halftone screen and derive optimization formulas for the zero and nonzero orders.

A. Zero Order

Equation (6) is the general formula relating the output intensity in the zero order to the input intensity. Let $f(x)$ be approximated as shown in Fig. 3, then Eq. (6) can be written as

$$I_0(I_{in},0) = \left|\frac{1}{L}\left[\int_0^{L_1} g(\log I_{in} - a_1)dx + \int_{L_1}^{L_2} g(\log I_{in} - a_2)dx + \ldots + \int_{L_{N-1}}^{L} g(\log I_{in} - a_N)dx\right]\right|^2 . \quad (12)$$

Assuming that the x-axis intervals are all equal, i.e.,

$$L_1 - 0 = L_2 - L_1 = \ldots = L_N - L_{N-1} = \frac{L}{N}, \quad (13)$$

Eq. (12) reduces to

$$I_0(I_{in},0) = \left|\frac{1}{N}\sum_{i=1}^{N} g(\log I_{in} - a_i)\right|^2 . \quad (14)$$

The above formula gives the output intensity in the zero order as a function of the discrete grey levels on the halftone screen and the characteristic curve of the recording medium.

To design the proper halftone screen, we want $a_i, 1, \ldots, i, \ldots, N$ in Eq. (14) where $g(\cdot)$ and the desired functional relationship of $I_0(I_{in},0)$ in terms of I_{in} are given. Although Eq. (14) is an approximate representation of I_{in}, we can require it to be exact for a discrete set of values of $I_{in} = I_m$, where $1,\ldots,m,\ldots,M$. Thus in discrete form Eq. (14) can be written as

$$I_0(I_1,0) = \left|\frac{1}{N}\sum_{i=1}^{N} g(\log I_1 - a_i)\right|^2,$$

$$I_0(I_m,0) = \left|\frac{1}{N}\sum_{i=1}^{N} g(\log I_m - a_i)\right|^2, \quad (15)$$

$$I_0(I_M,0) = \left|\frac{1}{N}\sum_{i=1}^{N} g(\log I_M - a_i)\right|^2.$$

One procedure to find the optimum a_i terms is to minimize the mean square error expression

$$\min_{a_1,a_2,\ldots,a_N} E = \sum_{m=1}^{M}\left|I_0(I_m,0) - \left[\frac{1}{N}\sum_{i=1}^{N} g(\log I_m - a_i)\right]^2\right|^2. \quad (16)$$

This should produce values of a_i that bring the output intensity in the zero order as close as possible to the desired output intensity in the mean square sense.

Now, because the a_i terms are the different density values on the halftone screen, we constrain their values to lie within certain limits, as expressed by

$$c \le a_i \le d, \quad (17)$$

where c and d are given non-negative constants. This makes the optimization problem one of constrained minimization. We want a transformation from $[-\infty,\infty]$ to $[c,d]$. The functions $\sin^2$ and $\cos^2$ are examples of functions that transform $[-\infty,\infty]$ to $[0,1]$. We arbitrarily choose the $\sin^2$ function. If we let[7]

$$a_i = (d-c)\sin^2 y_i + c, \quad (18)$$

this limits the values of the a_i to the range $[c,d]$. With this change of variable, Eq. (16) now becomes

$$\min_{y_1,y_2,\ldots,y_N} E = \sum_{m=1}^{M}\left|I_0(I_m,0) - \left\{\frac{1}{N}\sum_{i=1}^{N} g[\log I_m - (d-c)\sin^2 y_i - c]\right\}^2\right|^2, \quad (19)$$

which gives the values of y_i terms. The corresponding a_i terms can then be found from Eq. (18).

In going from Eq. (16) to Eq. (19) we should make sure that the minimum of E with respect to the a_i terms is in fact the same as the minimum of E with respect to the y_i terms. To check this, note that when we minimize E with respect to y_i, we set

$$\frac{\partial E}{\partial y_i} = 0, \quad 1,\ldots,i,\ldots,N, \quad (20)$$

which is equivalent to

$$\frac{\partial E}{\partial a_i}\frac{\partial a_i}{\partial y_i} = 0, \quad 1,\ldots,i,\ldots,N, \quad (21)$$

from which we have

$$\frac{\partial E}{\partial a_i} = 0, \quad 1,\ldots,i,\ldots,N \quad (22)$$

or

$$\frac{\partial a_i}{\partial y_i} = 0, \quad 1,\ldots,i,\ldots,N. \quad (23)$$

Hence to have Eq. (22) true when Eq. (20) is true, we should avoid satisfying Eq. (23). To check for this condition note that

$$\frac{\partial a_i}{\partial y_i} = (d-c)\sin 2y_i, \quad (24)$$

which when set to zero gives

$$y_i = 0 \rightarrow a_i = c \quad 1,\ldots,i,\ldots,N \quad (25)$$

or

$$y_2 = \frac{\pi}{2} \rightarrow a_i = d \quad 1,\ldots,i,\ldots,N. \quad (26)$$

It can be seen that the only values of a_i in the range $[c,d]$ that make $\partial a_i/\partial y_i = 0$ are the boundaries of this interval. This can be prevented from causing a problem by choosing c and d so that the interval $[c,d]$ contains the limit values for the a_i terms.

As an example of a function possible with the zero order consider the design of a logarithmic screen. In a logarithmic process we want the relationship

$$I_0(I_{in},0) = p\cdot\log(I_{in}/I_r) \quad (27)$$

between the output and input intensities where p is a constant and I_r is shown in Fig. 2. Hence, to find the optimum screen giving the above logarithmic relationship with a recording medium characteristic curve $g(\log E)$, we must perform the minimization

$$\min_{y_1,y_2,\ldots,y_N} E = \sum_{m=1}^{M}\left(p\cdot\log(I_m/I_r) - \left\{\frac{1}{N}\sum_{i=1}^{N} g[\log I_m - (d-c)\sin^2 y_i - c]\right\}^2\right)^2. \quad (28)$$

For the initial values of the above minimization we will use the halftone screen density profile values obtained for a binary recording medium. This helps the minimization converge more rapidly, particularly when the recording medium response is close to binary. In practice, we do not expect to perform halftone nonlinear processing with a very low gamma recording medium.

To obtain the halftone screen density profile for the logarithmic process with a binary recording medium as shown in Fig. 2, we equate Eqs. (7) and (27) to get

$$p\log\left(\frac{I_{in}}{I_r}\right) = \left[a + \frac{(b-a)}{L}f^{-1}\left(\log\frac{I_{in}}{I_r}\right)\right]^2 \quad (29)$$

and substitute $u = \log(I_{in}/I_r)$ in the above equation to obtain

$$p\cdot u = \left[a + \frac{(b-a)}{L}f^{-1}(u)\right]^2. \quad (30)$$

Because log is a monotonic function, the corresponding

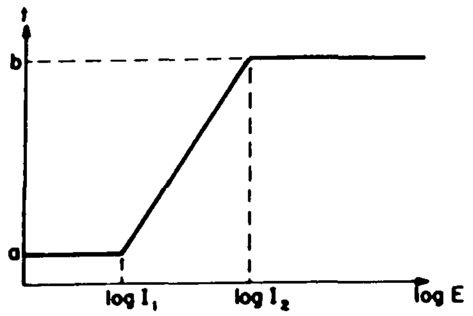


Fig. 4. Characteristic curve of a piecewise linear recording medium.

halftone screen density profile will also be a monotonic function. Hence f is a monotonic function so we can write $f[f^{-1}(u)] = u$. Using this in Eq. (30) we have

$$p \cdot f[f^{-1}(u)] = \left[a + \frac{(b-a)}{L} f^{-1}(u)\right]^2 . \tag{31}$$

Now let $x = f^{-1}(u)$ in the above equation to obtain

$$f(x) = \frac{1}{p}\left[a + \left(\frac{b-a}{L}\right)x\right]^2 , \quad 0 \le x \le L. \tag{32}$$

Hence the initial values for the a_i terms are

$$a_i = \frac{1}{p}\left[a + \left(\frac{b-a}{L}\right)x_i\right]^2 , \quad 0 \le x \le L, \quad 1,\ldots,i,\ldots,N. \tag{33}$$

From Eq. (18) the corresponding y_i terms are

$$y_i = \sin^{-1}\left[\left(\frac{1}{p(d-c)}\left[a + \left(\frac{b-a}{L}\right)x_i\right]^2 - c\right)^{1/2}\right] . \tag{34}$$

The above y_i terms will now initialize the minimization procedure of Eq. (28) to obtain the corresponding y_i terms for a nonbinary recording medium.

A computer algorithm was written which performs the above minimization. It uses the ZXMIN subroutine which is taken from the IMSL library.[8] This subroutine is based on a quasi-Newton algorithm for finding the minimum of a function of N variables.[9] In the quasi-Newton method we do not directly solve for the minimum of E by solving Eq. (20) directly. Rather we use an iterative procedure which starts from the initial point and uses Eq. (20) to get as close as possible to the minimum. Thus we do not expect to produce the undesired solution of Eq. (23) as the minimum.

The results for recording media of the type shown in Fig. 4 with $a = 0$, $b = 1$, and different slopes in the linear part are shown in Figs. 5–8. The slopes are called gamma in the figure but should not be confused with the usual photographic gamma. It is assumed that there are thirty discrete points in the halftone screen density profile, and the density values are between 0 and 2 in these figures. It is also assumed that the values of $\log I_r$ (shown in Fig. 2) lie between the middle of the values of $\log I_1$ and $\log I_2$ (shown in Fig. 4). Note that the plots of the input–output curves are semilogarithmic, and hence the result is a straight line. In Figs. 5 and 7 the graphs labeled ideal represent the desired relationship between input and output intensities. This can be obtained with the halftone screen density values of Eq. (32) and binary recording medium characteristic curve of Fig. 2 in Eq. (5). The degraded graph is obtained by using the density values of Eq. (32) and the recording medium of Fig. 4 in Eq. (5). The optimized graph is generated by using the density values obtained from Eq. (28) and the characteristic cure of the recording medium for which those density values were produced in Eq. (5).

In Figs. 6 and 8, the ideal graph represents the density profile of a halftone screen for a logarithmic process using a binary recording medium. The optimized graph is the density values obtained from Eq. (28) for a recording medium with the characteristics shown in Fig. 4. It is seen in Fig. 5 that for gamma of 3.0, there is a significant amount of degradation, and the optimized screen has been successful in removing this degradation. For a gamma of 10, shown in Fig. 10, as one might expect, there is less degradation, and the optimized screen is even more successful in producing the ideal result.

B. Nonzero Order

The general formula relating the output intensity in the nonzero orders to the input intensity is

$$I_0(I_{in},k) = \left|\frac{1}{L}\int_0^L g[\log I_{in} - f(x)] \exp\left(\frac{j2\pi kx}{L}\right) dx\right|^2 \tag{35}$$

from Eq. (5). Using a quantized approximation to $f(x)$, as in the zero-order case, we can write (from Fig. 3)

$$I_0(I_{in},k) = \left|\frac{1}{L}\left[\int_0^{L_1} g(\log I_{in} - a_1) \exp\left(\frac{j2\pi kx}{L}\right) dx + \int_{L_1}^{L_2} g(\log I_{in} - a_2) \exp\left(\frac{j2\pi kx}{L}\right) dx + \ldots + \int_{L_{N-1}}^{L} g(\log I_{in} - a_N) \exp\left(\frac{j2\pi kx}{L}\right) dx\right]\right|^2 . \tag{36}$$

For simplicity we let

$$g(\log I_{in} - a_i) = g_i, \; 1,\ldots,i,\ldots,N. \tag{37}$$

Then

$$I_0(I_{in},k) = \frac{1}{4\pi^2k^2}\left|g_1\left\{\exp\left[\frac{j2\pi k(L_1 - 0)}{L}\right] - 1\right\} + \ldots + g_N \times \exp\left(\frac{j2\pi kL_{N-1}}{L}\right)\left\{\exp\left[\frac{j2\pi k(L - L_{N-1})}{L}\right] - 1\right\}\right|^2 . \tag{38}$$

Now from Eq. (13)

$$L_1 - 0 = L_2 - L_1 = \ldots = L_N - L_{N-1} = L/N, \tag{39}$$

so when Eq. (39) is used in Eq. (38) we have

$$I_0(I_{in},k) = \frac{1}{4\pi^2k^2}\left|g_1 + g_2\exp\left(\frac{j2\pi kL_1}{L}\right) + \ldots + g_N \times \exp\left(\frac{j2\pi kL_{N-1}}{L}\right)\right|^2 \left|\exp\left(\frac{j2\pi k}{N}\right) - 1\right|^2 . \tag{40}$$

Note also that

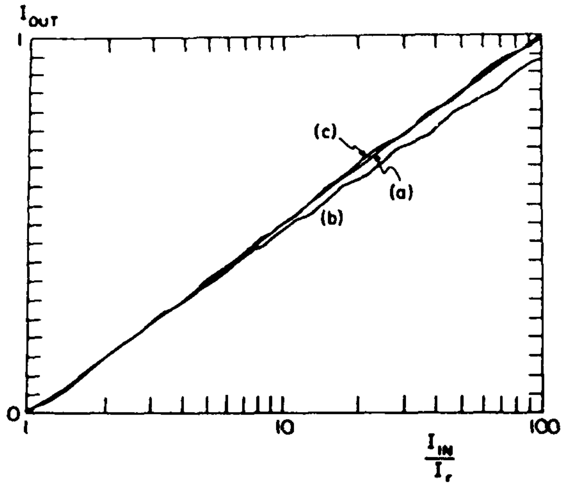


Fig. 5. Logarithmic transfer function for a piecewise linear recording medium with gamma = 3.0: (*a*) ideal; (*b*) degraded; (*c*) optimized.

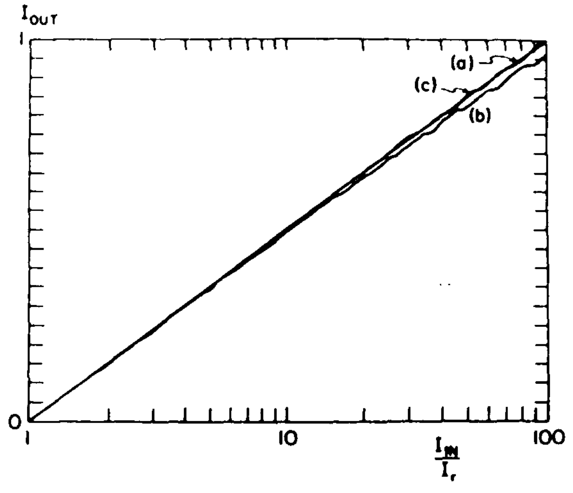


Fig. 7. Logarithmic transfer function for a piecewise linear recording medium with gamma = 10.0: (*a*) ideal; (*b*) degraded; (*c*) optimized.

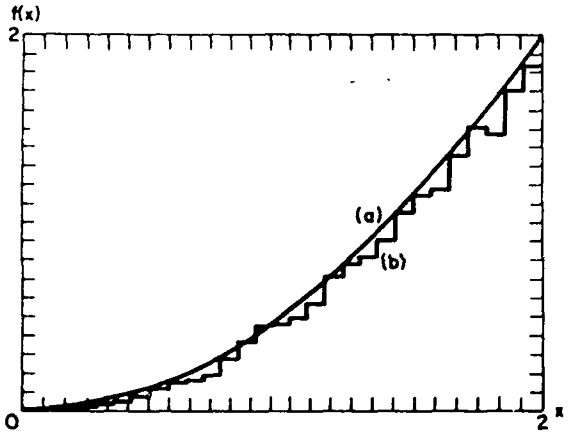


Fig. 6. Halftone cell shape corresponding to Fig. 5: (*a*) ideal; (*b*) optimized.

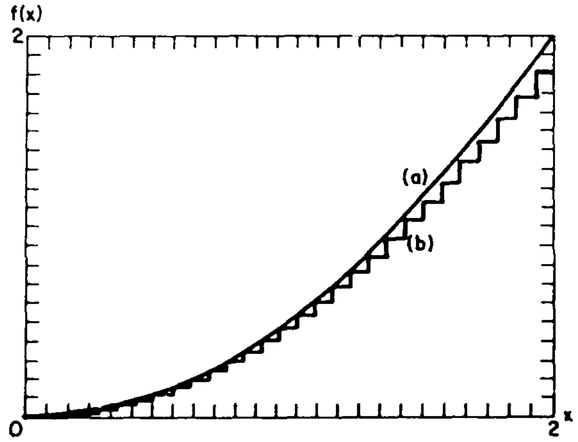


Fig. 8. Halftone cell shape corresponding to Fig. 7: (*a*) ideal; (*b*) optimized.

$$L_1 = \frac{L}{N}, \quad L_2 = 2\frac{L}{N}, \ldots, \quad L_i = i\frac{L}{N}, \quad 1,\ldots,i,\ldots,N. \quad (41)$$

Consequently Eq. (40) can be written as

$$I_0(I_{in},k) = \frac{1}{4\pi^2k^2}\left|\exp\left(\frac{j2\pi k}{N}\right) - 1\right|^2 \cdot \left|\sum_{i=1}^{N} g(\log I_{in} - a_i)\exp\left[\frac{j2\pi k(i-1)}{N}\right]\right|^2. \quad (42)$$

Equation (42) gives the output intensity in any nonzero order k in terms of the input intensity, the characteristic function of the recording medium, and the discrete grey levels on the halftone screen. To design the proper halftone screen, as in the case of the zero order, we should determine suitable a_i values from Eq. (42). The procedure that we take is the same as the one for the zero order. Namely, we minimize the expression

$$\min_{a_1,a_2,\ldots,a_N} E = \sum_{m=1}^{M}\left\{I_0(I_m,k) - \frac{1}{4\pi^2k^2}\left|\exp\left(\frac{j2\pi k}{N}\right) - 1\right|^2 \cdot \left|\sum_{i=1}^{N} g(\log I_m - a_i)\exp\left[\frac{j2\pi k(i-1)}{N}\right]\right|^2\right\}^2. \quad (43)$$

To limit the resulting density values as in Eq. (17) we transform the problem to the minimization

$$\min_{y_1,\ldots,y_N} E = \sum_{m=1}^{M}\left\{I_0(I_m,k) - \frac{1}{4\pi^2k^2}\left|\exp\left(\frac{j2\pi k}{N}\right) - 1\right|^2 \cdot \left|\sum_{i=1}^{N} g[\log I_m - (d-c)\sin^2 y_i - c] \times \exp\left[\frac{j2\pi k(i-1)}{N}\right]\right|^2\right\}^2 \quad (44)$$

The y_i terms are related to the a_i terms through Eq.

(18). To initialize the values of the y_i terms for this minimization procedure, we use the density values obtained for the screen when the recording medium is binary.

As an example of a function possible in the nonzero order we consider the design of a level slice screen. To obtain a level slice transformation the first order is a suitable choice. For this function we want

$$I_0(I_{\text{in}},1) = \begin{cases} 0, & \text{for } I_{\text{in}} < I_a, \\ \dfrac{1}{\pi^2}, & \text{for } I_a \le I_{\text{in}} < I_b, \\ 0, & \text{for } I_{\text{in}} \ge I_b. \end{cases} \tag{45}$$

This function is shown in Fig. 9. The density profile given a relationship for the binary recording medium shown in Fig. 2 for $a = 0$ and $b = 1$ is

$$f(x) = \begin{cases} \log(I_a/I_r), & \text{for } 0 \le x < \dfrac{L}{2}, \\ \log(I_b/I_r), & \text{for } \dfrac{L}{2} \le x \le L, \end{cases} \tag{46}$$

which is shown in Fig. 10. The corresponding density levels in the discrete screen are

$$a_i = \begin{cases} \log(I_a/I_r), & \text{for } i < \dfrac{N}{2}, \\ \log(I_b/I_r), & \text{for } i \ge \dfrac{N}{2}. \end{cases} \tag{47}$$

Using Eq. (47) in Eq. (18), we have

$$y_i = \begin{cases} \sin^1 \left\{ \left[\dfrac{\log(I_a/I_r) - c}{d - c} \right]^{1/2} \right\} & \text{for } i < \dfrac{N}{2}, \\ \sin^1 \left\{ \left[\dfrac{\log(I_b/I_r) - c}{d - c} \right]^{1/2} \right\} & \text{for } i \ge \dfrac{N}{2}, \end{cases} \tag{48}$$

With the initial values for y_i terms as above we then want to minimize

$$\min_{y_1,\ldots,y_N} E = \sum_{m=1}^{M} \left\{ I_0(I_m,1) - \frac{1}{4\pi^2} \left| \exp\left(\frac{j2\pi}{N}\right) - 1 \right|^2 \cdot \left| \sum_{i=1}^{N} g[\log I_m - (d-c)\sin^2 y_i - c] \times \exp\left[\frac{j2\pi(i-1)}{N}\right] \right|^2 \right\}^2. \tag{49}$$

A computer routine similar to the one for the logarithmic process has been written to perform the above minimization. With the same assumptions about the recording medium and the halftone screen as with the logarithmic process, the result is shown in Figs. 11–14. Figure 11 shows the ideal level slice function and the results that would be obtained for a recording material with an effective gamma of 3.0. Also shown is the optimized level slice obtained by precompensating the halftone screen profile. It is seen that although the mean square error between the ideal and the actual response curves can be significantly reduced, the finite slope of the leading and trailing edges of the level slice is still limited by the gamma of the recording medium. Figure 12 shows the halftone profile for an ideal recording material and the precompensated halftone profile for a recording material with a gamma of 3. Figures 13 and 14 are the corresponding results assuming a recording material with a gamma of 10, which is much closer to the ideal.

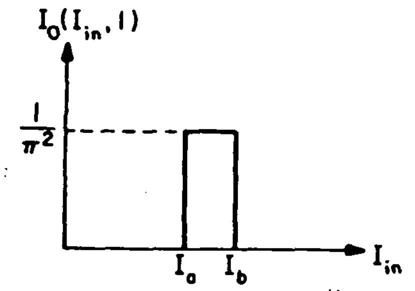


Fig. 9. Level slice function.

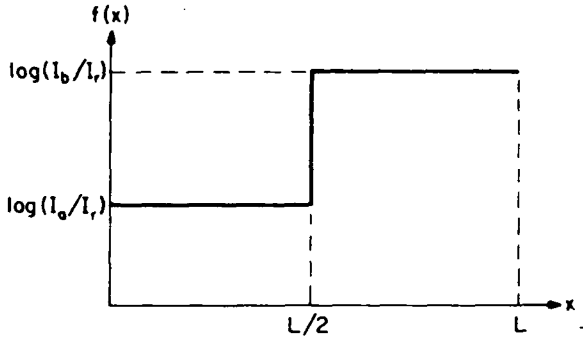


Fig. 10. Halftone screen density for the level slice function of Fig. 9.

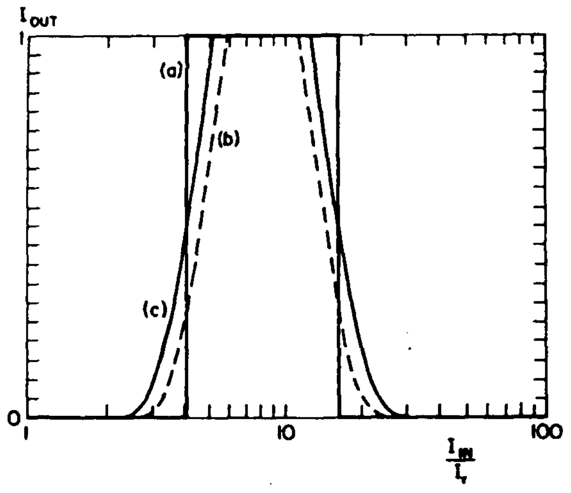


Fig. 11. Level slice transfer function for a piecewise linear recording medium with gamma = 3.0: (a) ideal; (b) degraded; (c) optimized.

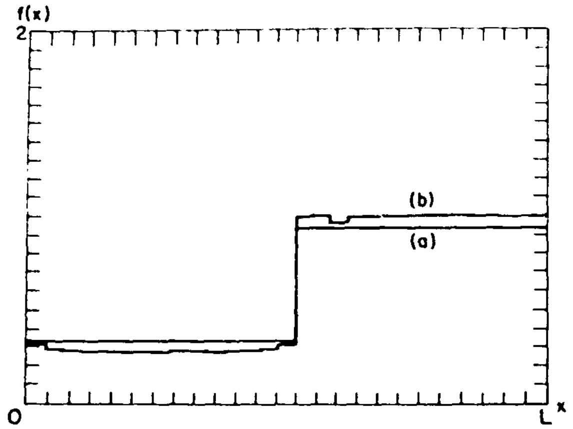


Fig. 12. Halftone cell shape corresponding to Fig. 12: (*a*) ideal; (*b*) optimized.

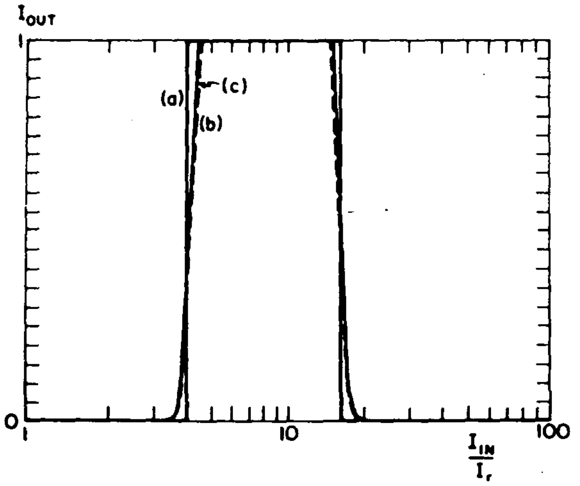


Fig. 13. Level slice transfer function for a piecewise linear recording medium with gamma = 10.0: (*a*) ideal; (*b*) degraded; (*c*) optimized.

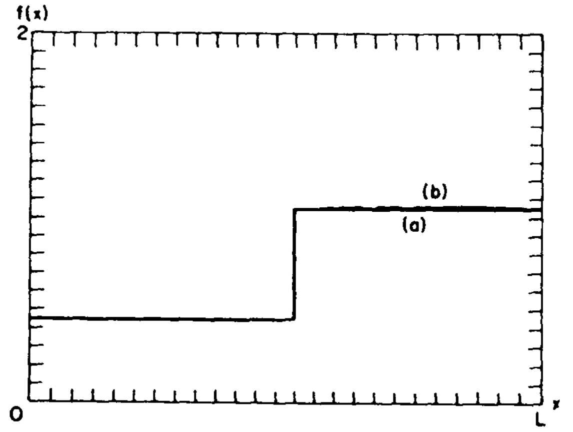


Fig. 14. Halftone cell shape corresponding to Fig. 13: (*a*) ideal; (*b*) optimized.

IV. Conclusions

A new formulation of the halftone nonlinear processing technique has been presented. The formulation is general and works for any recording medium characteristic curve shape and any halftone screen cell shape. Thus one can easily predict the amount of degradation of the output due to a nonbinary characteristic curve of the recording medium. This is particularly useful for real-time realization of the halftone processing. In this case, a real-time image transducer is used as the recording medium. Because the presently developed devices do not possess the desired sharp threshold characteristic, halftone screen precompensation is necessary.

The problem of the design of the halftone screen density profile with a nonideal recording medium has been solved by an approximate method which obtains the halftone screen density profile by minimizing the difference in a mean square sense between desired and degraded outputs. Results of computer simulations for logarithmic and level-slice transformations have been given. The results show that for smooth nonlinearities like the logarithmic function, it is possible to compensate for the nonideal characteristic of the recording medium. For nonlinearities with sharp jumps like the level-slice function, the compensation is less successful.

References

1. J. W. Goodman, "Operations Achievable with Coherent Optical Information Processing Systems," Proc. IEEE 65, 29 (1977).
2. S. R. Dashiell and A. A. Sawchuk, "Nonlinear Optical Processing: Analysis and Synthesis," Appl. Opt. 16, 1009 (1977).
3. S. Iwasa and J. Feinleib, "The PROM Device in Optical Processing Systems," Opt. Eng. 13, 235 (1974).
4. J. Grinberg *et al.*, "A New Real-Time Non-Coherent to Coherent Light Image Converter," Opt. Eng. 14, 217 (1975).
5. S. R. Dashiell and A. A. Sawchuk, "Nonlinear Optical Processing: Effects of Input Medium and Precompensation," Appl. Opt. 16, 2279 (1977).
6. G. W. Batten, Jr., and R. L. Everett, "Control of Film Characteristics by Modulating Intensity and Space," J. Opt. Soc. Am. 68, 1118 (1978).
7. R. W. Hamming, *Numerical Methods for Scientists and Engineers* (McGraw-Hill, New York, 1973), p. 673.
8. This subroutine was taken from the IMSL library 2 which is available from IMSL, 7500 Bellaire Blvd., Houston, TX 77036.
9. R. Fletcher, "Fortran Subroutines for Minimization by Quasi-Newton Methods," Report R7125 AERE, Harwell, England.

2.5 Optical Symbolic Substitution and Pattern Recognition

The attached paper "Optical Symbolic Substitution and Pattern Recognition Algorithms Based on Binary Image Algebra" by K.S. Huang, B.K. Jenkins and A.A. Sawchuk from *ICO Topical Meeting on Optical Computing,* Toulon, France, August 29 - September 2, 1988 describes the application of binary image algebra to pattern recognition systems for cellular processors.

Optical Symbolic Substitution and Pattern Recognition Algorithms Based on Binary Image Algebra

K. S. Huang, B. K. Jenkins, A. A. Sawchuk

Signal and Image Processing Institute
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-0272, USA

February 27, 1988

Abstract

Pattern recognition algorithms and algebraic properties of binary image algebra (BIA) are used to improve the speed, flexibility and complexity of symbolic substitution.

Submitted to *ICO Topical Meeting On Optical Computing*, Toulon, France, August 29 - September 2, 1988.

Optical Symbolic Substitution and Pattern Recognition Algorithms Based on Binary Image Algebra

K. S. Huang, B. K. Jenkins, A. A. Sawchuk
Signal and Image Processing Institute, Department of Electrical Engineering,
University of Southern California, Los Angeles, CA 90089-0272, USA

Summary

Binary image algebra (BIA), a unified systematic complete theory of parallel binary image processing [1], also provides a unified spatial logic of digital optical computing for describing symbolic substitution, cellular logic and Boolean logic in parallel [2]. Symbolic substitution has been used to implement logic, arithmetic, communication and simulating a Turing machine [3]; but its implementation of some operations (e.g. parallel binary arithmetic) is relatively complicated to other BIA implementations [2]. In this paper we further suggest some BIA algebraic techniques and pattern recognition algorithms, including a shift, scale and rotation invariant algorithm, to improve the speed, flexibility and complexity of symbolic substitution.

A symbolic substitution rule involves two steps: 1) recognizing the locations of a certain spatial search-pattern within the 2-D input data, and 2) substituting a new replacement-pattern wherever the search-pattern is recognized. As illustrated in Fig. 1, BIA can be used to realize a symbolic substitution rule defined by:

$$(X \odot R) \oplus Q = ((X \ominus R_1) \cap (\overline{X} \ominus R_2)) \oplus Q = \overline{(\overline{X} \oplus \check{R}_1) \cup (X \oplus \check{R}_2)} \oplus Q \qquad (1)$$

where X is the 2-D input data, $R = (R_1, R_2)$ is the reference image pair corresponding to the search-pattern (R_1 and R_2 define the foreground and the background of the search-pattern respectively), $\check{R}$ defines a reflected reference image given by $\check{R} = \{(-x, -y) \mid (x, y) \in R\}$, Q is the reference image corresponding to the replacement-pattern, "$\odot$" denotes the hit or miss transform which is the pattern recognizer, "$\ominus$" denotes the erosion operation, and "$\oplus$" denotes the dilation operation which is the pattern replacement operator. To work with more than one rule (say p substitution rules) for practical applications, a symbolic substitution system (Fig. 2) produces several copies of the input X, provides p different recognizer-substituter units, and then combines the outputs of various units to form a new output. Thus, a symbolic substitution system is implemented by

$$\bigcup_{i=1}^{p} (X \odot R^{(i)}) \oplus Q^{(i)} \qquad (2)$$

where $R^{(i)}$ and $Q^{(i)}$, $i = 1, 2, ..., p$, are the reference image pairs and replacement patterns in the i^{th} symbolic substitution rule. This, then, is the BIA formula for general symbolic substitution.

However, in many cases the above form is inefficient and can be reduced to a relatively simpler form or implemented in a more efficient way by using some BIA algebraic techniques. Here are some examples: 1) the full recognition can be implemented by only the background or foreground recognition under certain conditions; 2) if $Q^{(i)} = \phi$, the i^{th} symbolic substitution rule in Eq. (2) is not needed (e.g. the four rules of binary subtraction in simple intensity coding of arithmetic data can be reduced to only two rules [2]); and 3) if $Q^{(i)} = Q$ for all $1 \leq i \leq p$ (this happens in those cases that a class of search-patterns is defined by a set of reference image pairs $R^{(i)}$, $i = 1, 2, ..., p$), we should combine the results of the hit or miss transforms first and then replace them by the same replacement-pattern Q instead of implementing p substitution units for realizing the same substitution step, i.e.

$$(\bigcup_{i=1}^{p} X \odot R^{(i)}) \oplus Q. \qquad (3)$$

The practical difficulty with the implementation in Eqs. (2) and (3) is that the hit or miss transform is only efficient for the shift invariant recognition and would require a large number of intricate reference image pairs to perform the recognition step in the presence of changes in scale, rotation or both. Thus, it might be too costly to implement scale and rotation invariant recognition of intricate patterns for symbolic substitution based on the above formula. For example, if we want to substitute all "square patterns" in an input image by the same character "S", it would be very inefficient to use the above symbolic substitution implementation techniques.

To solve this kind of scale and rotation invariant problem, here we recognize all the desired patterns by reversing the growing procedure of a family of patterns. This family defines all patterns in the presence of changes in scale, rotation or both, and transforms all the desired patterns into their original seeds, which are isolated single image points. We have developed a description of this procedure in terms of BIA. For brevity, here we describe only the case of shift and scale invariant recognition. Suppose we want to recognize all square patterns with different scales and locations in the input image X (e.g. Fig. 3(a)) and to produce the output image Y (e.g. Fig. 3(b)). The procedure is: 1) determine a growing sequence of the desired patterns T_i (e.g. Fig. 3(c)), where $0 \leq i \leq m$ and the largest size of the desired patterns is $m \times m$; 2) find a small set of *good* reference image pairs $\{R(\theta)\}$ (e.g. Fig. 3(d) has only 5 small reference image pairs for recognizing all square objects with different scales) satisfying some criteria, where each reference image pair in $\{R(\theta)\}$ corresponds to a possible neighborhood of a given *foreground* image point in a pattern T_i, $1 \leq i \leq m$, whose previous state in the pattern T_{i-1} is a *background* point; 3) transform the desired patterns T_i, $i = 1, 2, ..., m$, in the 2-D input image $X = X(t_0)$ into their original seeds (i.e. T_0 which contains one and only one foreground image point) by the recursive relation $X(t_{k+1}) = X(t_k) / \bigcup_{\theta \in \Theta} X(t_k) \odot R(\theta)$, where $0 \leq k \leq m$; and 4) pick up the original seeds by $Y = X(t_m) \odot Q$, where Q (Fig. 3(e)) is a reference image pair with one and only one foreground image point at the center and Y is the final recognition output. By selecting *good* reference image pairs associated the growing sequences of rotation patterns, we can extend shift and scale invariance to include rotation invariance in a similar way. This algorithm can efficiently reduce the computation complexity for a certain class of pattern recognition and symbolic substitution problems; their computation times depend only on the diameter of the largest desired pattern, but not on the number of patterns nor the size of the whole image.

A digital optical cellular image processor (DOCIP) [1] [2] implements all the above algorithms of symbolic substitution and pattern recognition in a flexible and efficient way compared to a symbolic substitution processor (Fig. 2) with p fixed recognizer-substituter units. The DOCIP programming for these algorithms will be illustrated.

References

[1] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Binary Image Algebra and Optical Cellular Logic Processor Design," submitted to *Computer Vision, Graphics, and Image Processing.*

[2] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "An Image Algebra Representation of Parallel Optical Binary Arithmetic", submitted to *Applied Optics.*

[3] K.-H. Brenner, A. Huang, and N. Streibl, "Digital Optical Computing with Symbolic Substitution," *Applied Optics*, Vol. 25, pp. 3054-3060, 1986.

Acknowledgements

This research was supported by the Air Force Office of Scientific Research under grant AFOSR-84-0181, by the Office of Naval Research under contract N00014-86-K-0602, and by an IBM graduate fellowship.

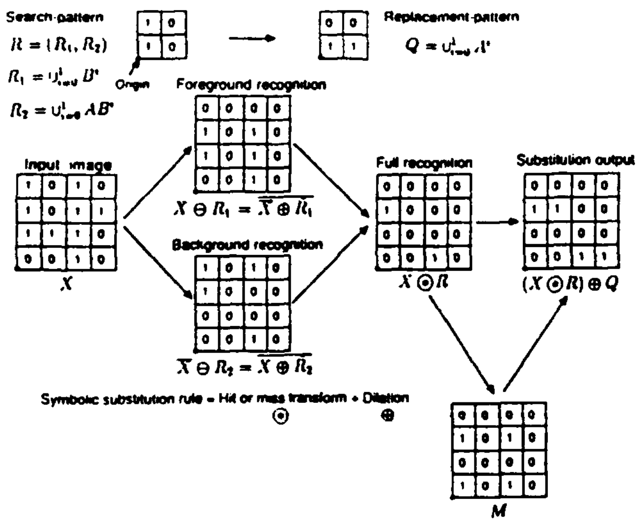
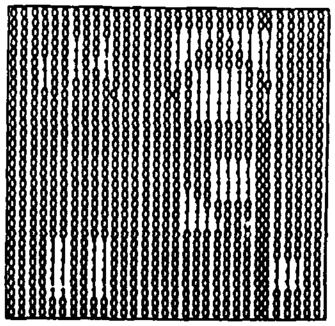
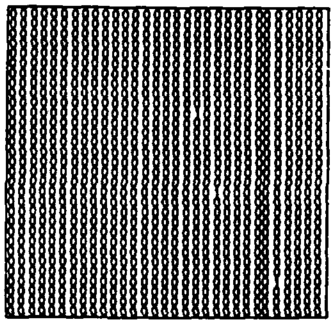

Search pattern
$R = (R_1, R_2)$
$R_1 = \cup^1_{i=0} B^i$
$R_2 = \cup^1_{i=0} AB^i$
Origin
Replacement pattern
$Q = \cup^1_{i=0} A^i$
Foreground recognition
Input image
X
$X \ominus R_1 = \overline{\bar{X} \oplus \check{R}_1}$
Background recognition
$\bar{X} \ominus R_2 = \overline{X \oplus \check{R}_2}$
Full recognition
$X \circledcirc R$
Substitution output
$(X \circledcirc R) \oplus Q$
Symbolic substitution rule = Hit or miss transform + Dilation
M

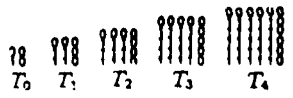
Figure 1. BIA representation of symbolic substitution. The optional mask M is for controlling the block search region.



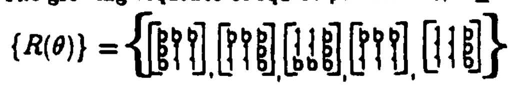
(a) An input image X.



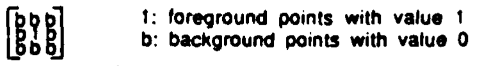
(b) The output image Y.



(c) The growing sequence of square patterns T_i, $o \leq i \leq 4$.



(d) A set of *good* reference image pairs $\{R(\theta)\}$ for square patterns with different scales.


1: foreground points with value 1
b: background points with value 0

(e) The reference image pair Q.

Figure 3. A shift and scale invariant pattern recognition of square patterns.

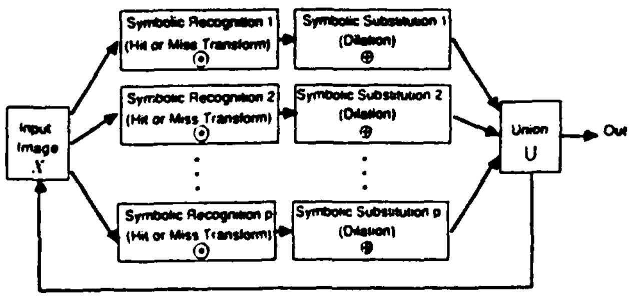

Input Image X
Symbolic Recognition 1 (Hit or Miss Transform)
Symbolic Substitution 1 (Dilation)
Symbolic Recognition 2 (Hit or Miss Transform)
Symbolic Substitution 2 (Dilation)
Symbolic Recognition p (Hit or Miss Transform)
Symbolic Substitution p (Dilation)
Union U
Out

Figure 2. A symbolic substitution system with p symbolic substitution rules.

2.2 Digital Optical Cellular Architectures

The papers reprinted in this section discuss details of optical cellular architectures and their instruction set.

The DOCIP is a 2-D, page oriented array of individual processors located at every pixel of a large image. The attached paper by K.S. Huang, B.K. Jenkins and A.A. Sawchuk, "Binary Image Algebra and Optical Cellular Logic Processor Design", submitted to *Computer Vision, Graphics and Image Processing,* summarizes some of these concepts and their algebraic background. Following this paper is "Optical Symbolic Substitution and Pattern Recognition Algorithms Based on Binary Image Algebra", by K.S. Huang, B.K. Jenkins and A.A. Sawchuk, from the *ICO Topical Meeting on Optical Computing,* Toulon, France, 1988, which contains additional information.

This paper is concerned with the hardware implementation of one cell of a prototype digital optical cellular image processor (DOCIP).

Implementation of A Prototype Digital Optical Cellular Image Processor (DOCIP)

K. S. Huang, A. A. Sawchuk, B. K. Jenkins, P. Chavel*, J. M. Wang*, A. G. Weber, C. H. Wang, I. Glaser

Signal and Image Processing Institute
Department of Electrical Engineering
University of Southern California
Los Angeles, CA 90089-0272, USA

*Institut d'Optique
Laboratoire Associé au CNRS
Université de Paris Sud
BP43, 91406 Orsay cedex, FRANCE

February 27, 1988

Abstract

A processing element of a prototype digital optical cellular image processor (DOCIP) is implemented to demonstrate a particular parallel computing and interconnection architecture.

Submitted to *ICO Topical Meeting On Optical Computing*, Toulon, France, August 29 - September 2, 1988.

Implementation of A Prototype Digital Optical Cellular Image Processor (DOCIP)

K. S. Huang, A. A. Sawchuk, B. K. Jenkins, P. Chavel*, J. M. Wang*, A. G. Weber, C. H. Wang, I. Glaser

Signal and Image Processing Institute, Department of Electrical Engineering,
University of Southern California, Los Angeles, CA 90089-0272, USA

*Institut d'Optique, Laboratoire Associé au CNRS,
Université de Paris Sud, BP43, 91406 Orsay cedex, FRANCE

Summary

Digital optical cellular image processor (DOCIP) architectures, DOCIP-array and DOCIP-hypercube, can perform the tasks of parallel binary image processing and parallel binary arithmetic [1]. The use of optical interconnections permits a cellular hypercube topology to be implemented without paying a large penalty in chip area (the cellular hypercube interconnections are space-invariant which implies relatively low hologram complexity); it also enables images to be input to and output from the machine in parallel. Table 1 gives a comparison of three different interconnection networks: cellular array (DOCIP-array interconnection network), conventional hypercube, and cellular hypercube (DOCIP-hypercube interconnection network). In this paper we experimentally demonstrate the concept of the DOCIP architecture by implementing one processing element of a prototype optical computer including a 49-gate processor, an instruction decoder, and electronic input/output interfaces.

A multiple-exposure multi-facet interconnection hologram provides the fixed interconnections between the outputs and the inputs of an array of 7×7 optical gates. The input data and the instructions are supplied from an LED array. The outputs of optical gates are detected by a video camera and compared with the results of a software simulation. A diagram of the main components of this experimental system is shown in Fig. 1.

A space-variant interconnection system [2] for within-processor interconnection is used in this experimental demonstration. A computer controlled system is used to make an array of 49 interconnection subholograms. An optical point source S, whose position is controlled by the mirror M2 with two rotational stages (Fig. 1), is used to provide an object beam for determining an interconnection of a subhologram in the multi-facet hologram. A mask with a circular aperture, controlled by two translational stages, is used to determine the sizes and positions of subholograms in a holographic plate. The interconnection hologram for this 49-gate optical processing element comprises 49 subholograms, which are laid out in a 7×7 array. Each subhologram covers a circular area with a diameter of 1.5 mm. The spacing between the centers of two subholograms is 3.0 mm. Note that the path of the object beam and the mask for subholograms are only used for making the interconnection hologram; they are blocked or moved when we reconstruct the hologram to implement the interconnections of the optical gates. We use a volume phase hologram with a dichromated gelatin medium for obtaining high diffraction efficiencies.

The array of 7×7 optical gates is implemented by a Hughes liquid-crystal light valve (LCLV) with liquid-crystal molecules in a 45° twisted nematic configuration [2]. The LCLV is read out between crossed polarizers and is biased to implement a NOR operation. The gate size in this experiment has a diameter of 0.3 mm and the spacing between the centers of two gates is 0.6 mm.

The circuit diagram of the processing element, as shown in Fig. 2, consists of 49 NOR gates with maximum fan-in of 3 and fan-out of 4. The processing element includes a 3-bit destination selector, a 3-bit master-slave flip-flop memory, a 6-bit memory selector with a union module, and a 5-bit neighborhood selector (for DOCIP-array4 [1]) with a dilation module. This experimental DOCIP system has one instruction, supplied from an LED array and decoded by the optical hardware. This instruction has the format: $(c, d_1, d_2, d_3, s_1, s_2, ..., s_6, n_1, n_2, ..., n_5)$ where c selects the image from the input or from the feedback; d_1, d_2, and d_3 select the destination memory for storing the image; $s_1, s_2, ..., s_6$ select the output from the memory elements; and $n_1, n_2, ..., n_5$ control the neighborhood mask, i.e. supply the reference image. We will experimentally demonstrate the DOCIP architecture concept with this system.

References

[1] K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "Binary Image Algebra and Optical Cellular Logic Processor Design," submitted to *Computer Vision, Graphics, and Image Processing*; K. S. Huang, B. K. Jenkins, A. A. Sawchuk, "An Image Algebra Representation of Parallel Optical Binary Arithmetic", submitted to *Applied Optics*.

[2] A. A. Sawchuk and T. C. Strand, "Digital Optical Computing," *Proc. IEEE*, Vol.72, pp. 758-779,1984; B. K. Jenkins, et al, "Sequential Optical Logic Implementation," *Applied Optics*, Vol. 23, No. 19, pp. 3455-3464, 1984; B. K. Jenkins, et al, "Architectural Implications of A Digital Optical Processor," *Applied Optics*, Vol. 23, No. 19, pp. 3465-3474, 1984.

Acknowledgements

The authors thank W. Kressen, K. Prager and T. Mayeda for their contributions to this work. This research was supported by the Air Force Office of Scientific Research under grant AFOSR-84-0181 and by an IBM graduate fellowship.

Interconnections	Cellular Array	Conventional Hypercube	Cellular Hypercube
Connectivity (Interconnections per PE)	O(1)	O(logN)	O(logN)
2-D Spatial Invariance	Yes	No	Yes
Inter-PE Communication Time Complexity	O(N)	O(logN)	O(logN)
VLSI Chip Area	$O(N^2)$	$O(N^4)$	$O(N^4)$
Hologram Space-bandwidth Product	$O(N^2)$	$O(N^2 \log N)$	$O(N^2)$

Table 1. A comparison between three different interconnection networks of NxN processeng elements (PEs): cellular array, conventional hypercube and cellular hypercube. When laid out on a VLSI chip, both the conventional hypercube and cellular hypercube pay a large penalty in chip area while the cellular hypercube has a relatively low hologram complexity.

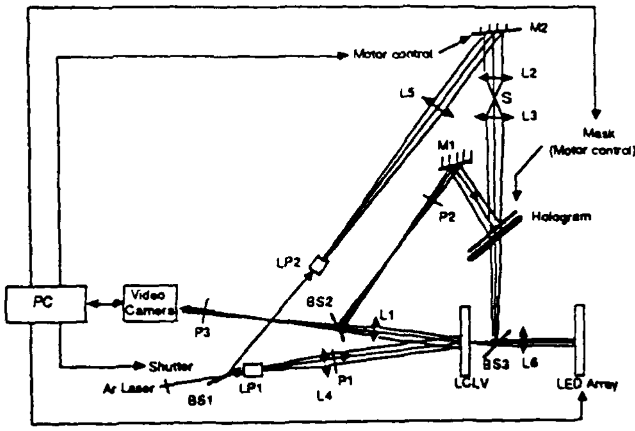


Figure 1. Experimental DOCIP system. Lens L1 images from the LCLV gate output plane to the hologram plane. Beam Splitter BS3 combines the external input signals from LED array and the feedback signals from interconnection hologram. LP1 and LP2 are lens-pinhole assemblies. P1 and P2 are crossed polarizers. The hologram comprises an array of subholograms. Mirror M2 controls the position of point source S during hologram exposure. After the hologram is made, the mask and all components in the path from BS1 to the hologram are not needed.

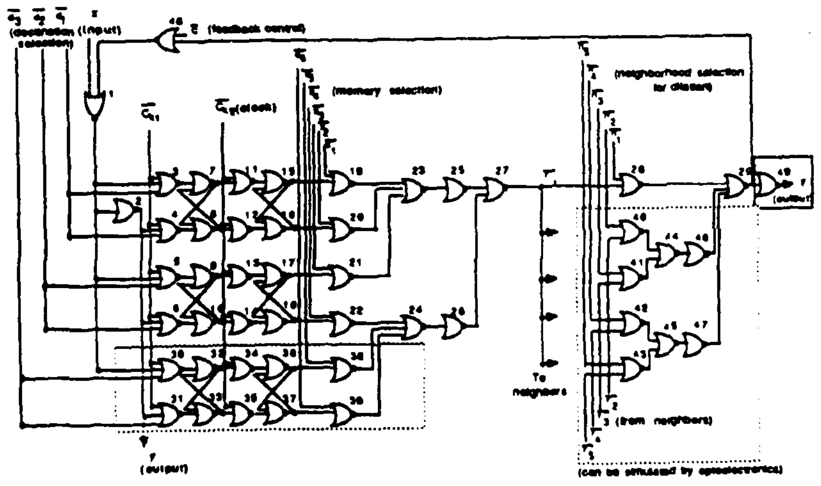


Figure 2. The circuit diagram of a 49-gate processing element of the DOCIP-array4.

2.6 Parallel Processing and Optical Computing

A third area of research on this grant has been the general investigation of the impact of optical computing technology on parallel *computing architectures*, including consideration of SIMD, MIMD and data flow structures. We have studied the relationship of these architectures at low to high levels of processor graininess. the following paper "Parallel Processing Paradigms and Optical Computing" by B.K. Jenkins and A.A. Sawchuk, which appeared in the *Proc. Optical Computing Symposium,* SPIE Vol. 625, Los Angeles, January 1986, discusses shared memory and graph/network models for parallel computing in the context of the physical constraints and technology of optical computing.

Parallel Processing Paradigms and Optical Computing

B. Keith Jenkins

Signal and Image Processing Institute MC-0272, University of Southern California, Los Angeles, California 90089-0272

and

C. Lee Giles

Air Force Office of Scientific Research/NE, Bolling AFB, D.C 20032-6448

ABSTRACT

Parallel processing models as computational paradigms are discussed and related to optical computing. Two classes of parallel computing models are discussed - shared memory models and graph/network models. These models are used to analyze some of the possible effects of optical technology on parallel computing. It is found that the use of optics potentially provides certain fundamental advantages. In addition, some factors that limit the communication capabilities of optical systems in the case of network models are found.

INTRODUCTION

In this paper we look at paradigms and models for parallel processing as an attempt to increase our understanding of the role optical computing. Most of the parallel architectures discussed in the parallel processing community are heavily influenced by the constraints of electronic systems. The purpose of our approach in this paper is to abstract the notion of parallel computing from the limitations of any given technology. This abstract model can then be used as a starting point for the design of parallel optical computing architectures. In the process, some of the consequences of inherent differences between optical and electronic systems start to become apparent.

Computing paradigms are important for understanding the level and class of problems that the computer scientist is addressing. Consider the following structural paradigmatic classification: physical, functional, computational. A representation and example of each of these paradigms is illustrated in Table 1. Here we are only concerned with the computational paradigm and the optical implications.

Table 1. Processing paradigm levels

PARADIGMS	REPRESENTATION	EXAMPLE
Physical	Hardware/Technology	IC, Board
Functional	Architecture	PE, Memory, Interconnection Topology
Computational	Algorithms/Metrics	Turing Machine, Automata, Random Access Machine

Before discussing computational models, both sequential and parallel, we define computational order or complexity as it is used in this paper. The interest here is in establishing a quantitative measure of the computational power or cost of a problem, task or algorithm of size n. The parameter n provides a measure of the difficulty of the problem in the sense that the time required to solve the problem or the storage space required, or both, will increase as it grows. The measure or cost of running or executing an algorithm on a problem of size n is defined as the complexity function f. Thus, f is a measure of the time or space required for the execution of the algorithm. For a time measure, $f(n)$ is called the time complexity function; for a space or storage measure, $f(n)$ the space complexity function. Unless otherwise denoted the complexity function used in the paper is the time complexity function.

Our principal concern is with the performance of algorithms for *large values* of n, i.e. the asymptotic behavior of complexity function. If the value of n is sufficiently small, then even inefficient algorithms will cost the same to run. We assume the choice of an algorithm for small problems is not usually critical. The asymptotic behavior of is defined as $O(f)$, the order of f. We will not give a formal definition of $O(f)$ but illustrate its properties in Table 2. For a formal definition and more extensive discussion of these concepts see

Stanat and McAllister (1977). Table 3 illustrates the growth of certain complexity functions as a function of the size of the problem (after Stanat and McAllister). As one can see, a problem can get out of control rather quickly for certain orders of complexity.

Table 2. Examples of computation complexity and order.

Complexity f	Order O(f)
63	1
$50n$	n
$n^2+n+100$	n^2
$n \log_a n$	$n \log n$

Table 3. Growth of some common complexity functions. The entries are proportional to the time required to solve a problem of size n.

Problem size n	$\log n$	n	$n \log n$	n^2	2^n	$n!$
5	3	5	12	25	32	120
10	4	10	33	10^2	1024	3×10^6
10^2	7	10^2	664	10^4	1.3×10^{30}	$>10^{100}$
10^3	10	10^3	9965	10^6	$>10^{100}$	$>10^{100}$
10^4	14	10^4	1.4×10^5	10^8	$>10^{100}$	$>10^{100}$

Computational models are important because they measure the performance of general classes of both sequential and parallel algorithms on an idealized abstract machine. However, the performance of these models is highly dependent on the class of algorithms. If the generic class of algorithms is known for a specific problem (e.g. the communication algorithms of broadcasting, reporting, sorting, etc.), then the computational model which efficiently runs these algorithms would be a starting point for the design of a computer architecture that would do the same. The basic assumption is that algorithms which run well on a computational model should run well on the model-derived architecture. Our intention is to show that optics has a greater potential than electronics for physically realizing some of these computational models.

SEQUENTIAL COMPUTATIONAL MODELS

Since parallel computational models are for the most part extensions of sequential models, we briefly discuss these sequential machines. The most primitive and basic cf the sequential machines is a Turing machine (TM) of which there exist many forms: universal, non-deterministic, multi-tape, multi-head, 2-D tape, finite state automata, etc. (A finite state automation (FSA) can be described by a TM in which the tape moves in only one direction). The universal TM has the capability of computing any algorithm that is computable (a rather circular thesis since a universal TM defines what is computable). A principal application of the TM is in determining lower bounds on the space or time necessary to solve algorithmic problems. Since the TM is a well-known computational model, we highly recommend for further interest the very informative text by Minsky (1967).

The Random Access Machine (RAM) is a less primitive computational model which can be stylized as a primitive computer. The RAM model is a one-accumulator computer in which the instructions are not allowed to modify themselves. Figure 1 illustrates a RAM which consists only of a read-only input tape, a write-only output tape, a program and a memory (Aho, Hopcroft, and Ullman, 1974). Notice the close similarity to a TM. In fact time on the RAM is bounded above by a polynomial function of time on the TM. In particular, for a TM of time complexity $T(n) \geq n$, a RAM can simulate the TM in $O(T(n))$ or $O(T(n) \log n)$ time, depending on the cost function used for the RAM. For the converse, using a TM to simulate a RAM, the bounds on time required by the TM are higher and are highly dependent on the RAM cost function used (Aho, Hopcroft, and Ullman, 1974). The program of a RAM is not stored in memory and is unmodifiable. A sample RAM instruction set is shown in Table 4. A common RAM model is the uniform cost one, which assumes that each RAM instruction requires one unit of time and each register one unit of space. It is from attempts to parallelize the RAM computational model that many parallel computational models emerged.

Table 4. Sample RAM instruction set.
JGTZ is jump if greater than zero.
JZERO is jump if equal to zero.

Operation Code	Address
1. LOAD	operand
2. STORE	operand
3. ADD	operand
4. SUB	operand
5. MULT	operand
6. DIV	operand
7. READ	operand
8. WRITE	operand
9. JUMP	label
10. JGTZ	label
11. JZERO	label
12. HALT	

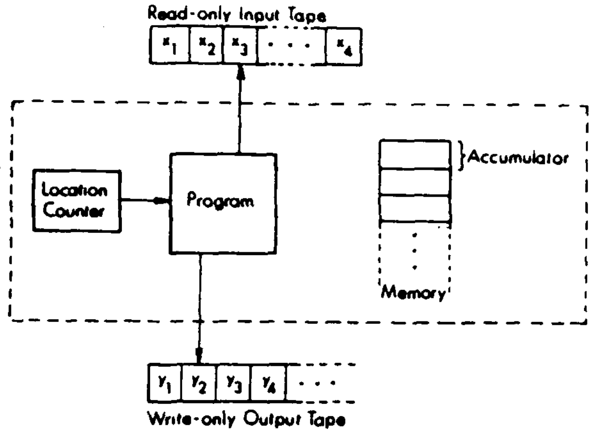


Fig. 1. Random access machine (RAM)

SHARED MEMORY MODELS

We will discuss only two classes of parallel computational models; shared-memory models and graph/network models. As might be inferred from the shared memory term, these models are based on global memories and are differentiated by their accessibility to memory. In Fig. 2 we see a typical shared memory model where individual processing elements (PE's) have variable simultaneous access to an individual memory cell. (A processing element is a physically isolated computational unit consisting of some local memory and computational power. A PE can be construed as a computational primitive from which more sophisticated architectures can be constructed (Hwang and Briggs, 1984)). Each PE can access any cell of the global memory in unit time. In addition, many PE's can access many different cells of the global memory simultaneously. In the models we discuss, each PE is a slightly modified RAM without the input and output tapes, and with a modified instruction set to permit access to the global memory. A separate input for the machine is provided. A given processor can generally not access the local memory of other processors.

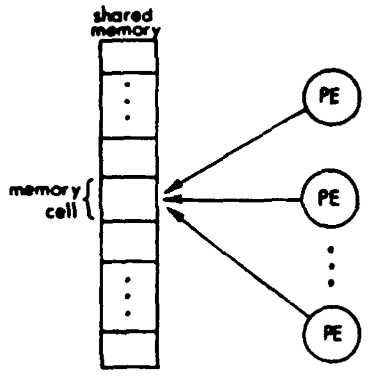


Fig. 2. Conceptual diagram of shared memory models.

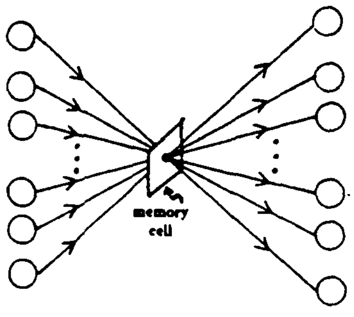


Fig. 3. One memory cell (or pixel) of an array, showing multiple optical beams providing contention-free read access.

The models differ primarily in whether they allow simultaneous reads and/or writes to the *same* memory cell. The PRAC, parallel random access computer (Lev, Pippenger and Valiant, 1981) does not allow simultaneous reading or writing to an individual memory cell. The PRAM, parallel random access machine, (Fortune and Wyllie, 1978) permits simultaneous reads but not simultaneous writes to an individual memory cell. The WRAM, parallel write random access machine, denotes a variety of models that permit simultaneous reads and certain writes, but differ in how the write conflicts are resolved. For example, a model by Shiloach and Vishkin (1981) allows a simultaneous write only if all processors are trying to write the same value. The paracomputer (Schwartz, 1980) has simultaneous writes but only "some" of all the information written to the cell is recorded. The models represent a hierarchy of time complexity given by

$$T^{\text{PRAC}} \geq T^{\text{PRAM}} \geq T^{\text{WRAM}}$$

where T is the minimum number of parallel time steps required to execute an algorithm on each model. More detailed comparisons are dependent on the algorithm (Borodin and Hopcroft, 1985).

Implications of optics

In general, none of these shared memory are physically realizable because of actual fan-in limitations. Optical interconnections permit greater fan-in than electronic systems. In addition, the non-interacting property of photons in a linear medium (versus the mutual interaction of electrons) may permit simultaneous memory reads much more easily. As an electronic example, the ultracomputer (Schwartz, 1980) is an architectural manifestation of the paracomputer that uses a hardwired Omega network between the PE's and memories; it simulates the paracomputer within a time penalty of $O(\log n)$.

Optical systems could in principle be used to implement this parallel memory read capability. As a simple example, a single 1-bit memory cell can be represented by one pixel of an array; the bit could be represented by the state (opaque or transparent) of the memory cell. Many optical beams could simultaneously read the contents of this memory cell without contention (Fig. 3). In addition to this an interconnection network is needed between the PE's and the memory, that can allow any PE to communicate with any memory cell, preferably in one step, and with no contention. A crossbar is *not* sufficient for this because fan-in to a given memory cell must be allowed. Optical systems can potentially implement crossbars that also allow this fan-in. For example, some of the optical crossbar designs discussed in Sawchuk and Jenkins (1986) can include fan-in capability.

GRAPH/NETWORK MODELS

Graph/network models are characterized by a collection of usually identical PE's that are interconnected with a fixed network. They can be represented by graphs, with a node of the graph for each PE and an arc or link of the graph for each PE to PE interconnection. The models differ from one another in the length of time required for a message to traverse one arc of the graph, and on the assumptions placed on the PE's such as their ability to respond to multiple messages. The feasibility of implementation of these models depends on the connectivity of the graph; if the connectivity is not too high, the model is much more readily implemented than the shared memory models.

Network models can be compared to shared memory models. Any of the shared memory models can efficiently simulate (in $O(1)$ time) a network model. This is done by dedicating a different cell of the global memory for each link of the network. One PE sends a message to another by writing the message to a memory cell which the other PE then reads. Conversely, suppose the network model is capable of (partial) routing in $r(n)$ time. Then it can simulate *one* step of the PRAC, PRAM, or WRAM in $O(r(n))$ time (Borodin and Hopcroft, 1985).

In a highly parallel machine communications are exceedingly important and for many tasks can dominate the execution time of the algorithm. We therefore concentrate on communications in our analysis of these models. The effectiveness of different PE network topologies can be evaluated by comparing *metrics*, essentially measures of the topological characteristics, or by comparing the number of time steps required to complete various fundamental communication tasks or algorithms. Examples of metrics include *diameter*, the shortest distance between the two most separated nodes where distance is measured in terms of number of links, and *bandwidth*, the maximum number of messages that can be simultaneously sent over the network in one time step. Levitan (1985) compared different architectures based on network models using both metrics and communication tasks, and concluded that communication tasks are a better predictor of actual run time performance on a given topology than are metrics. In our analysis we will use communication tasks.

PE Complexity and Communications

Since the performance of network models depends on the assumptions on the individual processing elements, we need to consider these assumptions and their relationships to communication tasks. We will show that in general the communications between PE's (or the network topology) cannot be completely decoupled from the hardware complexity of the PE's themselves. After giving a relationship between PE space complexity and interconnection capability, we will be able to identify what reasonable assumptions on the PE complexity are for the optics and electronics cases. These assumptions will be used in assessing the performance of different communication tasks on network models. In this paper the term PE complexity refers to the *space* complexity of *each* PE. We will not discuss time complexity of individual PE's.

For simplicity, we will assume the bandwidth of each I/O line to a PE is fixed and is given. Thus we are *a priori* not considering one of the potential advantages of an optical system over an electronic one. We will, however, consider the effect of the number of I/O lines to a PE. In the case of input lines, the signals coming in may be immediately combined, or may be kept separate and stored into separate registers. Consider the former case. Examples include forming the sum of all simultaneous inputs or just forming the logical AND over all of them. In the simplest case of a logical operation over all inputs, the PE must accommodate the required fan-in. To do this with gates of a fixed size (fixed fan-in per gate) requires $O(l_i)$ gates for l_i input lines. Thus the PE complexity grows $O(l_i)$ merely to accommodate the input lines. If the input gates are allowed to have a fan-in that increases with l_i, then the PE complexity still grows with l_i because the complexity of the input gates (instead of the number of input gates) grows $O(l_i)$. In the case of the PE keeping the input signals separate, $O(l_i)$ gates are needed for the input, and if stored into memory then l_i memory cells are required. Thus the PE complexity must be at least $O(l_i)$; if the PE can arbitrarily rearrange the signals in a small number of time steps then the PE complexity grows even faster (e.g., $O(l_i^2)$ if a crossbar is used). Similar arguments can be applied to the case of PE *output* lines. Thus a PE with l_i input lines and l_o output lines has complexity that grows $O(l_i+l_o)$ in the simpler cases; if too many demands are placed on its ability to process or move these signals around, then its complexity grows faster.

Implications of this lie in the communication ability of PE networks, particularly in the optics case. With electronic technology, the number of I/O lines to a PE is generally quite limited and this limits the ability of the PE's to communicate. This is due to limited pinout, cost of interconnections, etc. The PE complexity is in practice not an issue for communications. In the optics case, however, there are no pinout restrictions and many parallel interconnection lines are feasible. However, there *are* limitations on the total number of interconnections in an optical PE network; these are due to the PE complexity itself. In other words, the PE's have to be able to accommodate all of the I/O lines. The optics case apparently allows a balance between the interconnections and the PE complexity; in the electronics case the interconnections are further limited by technology factors.

Consider a fully connected array, that is one in which every PE has a hardwired line to every other PE. For a network of N PE's, the complexity of each PE grows $O(N)$ because it has N I/O lines; therefore the total complexity of all the PE's is $O(N^2)$. In this case the total number of interconnection lines is also $O(N^2)$. The total complexity of the PE network is $O(N^2)$.

We consider three specific examples of PE complexity in the case of a fully connected array.

Example 1. The PE's are made up of binary gates, either optical or electronic. This example was included in the discussion above; each PE has $O(N)$ gates and has complexity that grows $O(N)$ in the simplest cases.

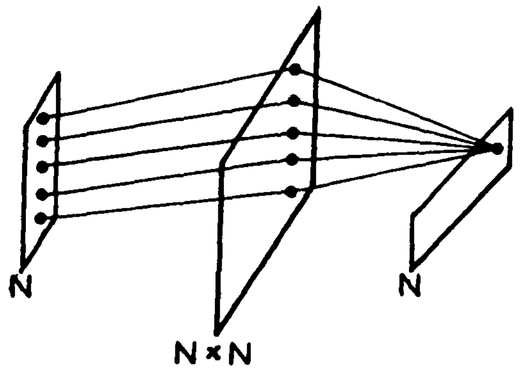


Fig. 4. Optical inner product matrix-vector multiplier as an example of a fully connected array.

Example 2. Consider the optical matrix-vector multiplier of Fig. 4. This is also a fully connected array, because in general every input or source is connected to every output or detector. Thus we expect each "PE" to have complexity at least $O(N)$. Each PE can be viewed as a detector, any thresholding, A/D, or processing electronics, and a source (which is part of the same PE if feedback is included). The 2-D SLM or mask is considered part of the interconnections in this case. Even for the simple case of binary sources and mask transmittances, each detector must distinguish $O(N)$ levels. In addition, there are accuracy requirements on the source intensities for these levels to be distinguishable. We conjecture that the PE complexity must increase at least $O(N)$. This is clear if we assume that the ability to distinguish an analog signal to within 1 part in N implies a complexity of $O(N)$. Thus the total complexity of all PE's in the network must grow at least $O(N^2)$.

Example 3. Consider again the optical matrix-vector multiplier, but now define each PE to include the multiplies also. Each PE is physically distributed, and includes N mask pixels (shown), 1 detector (shown), and 1 source. There are N PE's; each PE performs N multiplies and N adds, and has N spatially separate input lines (at the 2-D mask). The network is still fully connected, as each source (PE output) fans out to all PE's. In this case, the accuracy requirements on the sources, detectors and ensuing electronics are the same as in the previous example. In addition to this the accuracy of each multiply (the pixel transmittance) must be taken into account; one PE contains N of these multiplies and the accuracy (and therefore the complexity) of *each* multiply must increase with N. Thus the complexity of each PE must increase faster than $O(N)$; the complexity of the network grows *faster than* $O(N^2)$. This is reflected in the hardware requirements in making a large fully connected (optical) system.

Communication Tasks on Network Models

In this section we will give the time required to execute different communication tasks on different network topologies. We are concerned with fine-grained systems, that is systems with a large or very large number of relatively simple PE's. As a minimum, we assume each PE can store its own address so that it knows where it is located. Many algorithms can become quite difficult without this feature. This implies that the PE complexity must be allowed to grow $O(\log N)$.

In an electronic system, the number and length of interconnections is important and ideally should be minimized. The number of connections to each PE or node of the graph is limited to small values due to I/O constraints. This limits the connectivity of graphs that can be efficiently implemented. The *degree* of a graph is the number lines connected to each node. Electronic systems limit the degree of the graph to a relatively small value; for large enough N the degree must be a constant, independent of N.

Optical systems have no I/O restrictions on the PE's *per se*, but as discussed above the degree of the graph will be limited by the complexity of the PE's. Since the PE complexity must be at least $O(\log N)$ anyway, in the optics case the degree of the graph can easily be $O(\log N)$. Larger degrees, e.g. $O(N^{1/p})$, where $p \geq 2$ may also be feasible.

The time required for different interprocessor communication tasks performed on different fixed networks has been studied by Kushner and Rosenfeld (1983) and Levitan (1985). They show substantially reduced computation time for many communication tasks on networks of larger degree (e.g. hypercubes), as compared to simpler networks such as arrays. Augmented trees (Uhr, 1983) and bushy trees may also permit reduced time for these tasks. Examples of bushy trees are trees of degree $1+m^{1/p}$, which have diameter $2p$, where m is the number of leaf nodes and is a power of p.

In order to calculate communication times on a network model, certain assumptions need to be specified. We assume that all messages are the same size and are routed to their destinations over the fixed connection network by passing over links and through PE's. One time step is defined as the time for a PE to send a message, the message to travel over one link, be received by the PE at the end of the link, and for the PE to perform any computation on the message (such as altering its tag or combining messages that arrive simultaneously). The processors operate synchronously. Finally, the number of messages that can simultaneously be accepted or output by each PE must be considered. In the electronics case, the number of messages that can be simultaneously accepted by a PE is relatively small (because of the degree limitation), and will probably need to be a constant independent of N (Kushner and Rosenfeld, 1983). For simplicity this can be taken to be 1. A PE can output identical copies of the same message, but not multiple messages. For the optics case, we assume only a limit on the PE complexity; this then dictates how flexible the inputs and outputs of the PE can be. We limit the PE complexity to the degree of the network or $\log N$, whichever is greater. Each PE can accept d simultaneous messages, where d is the degree of the network, and may increase with N. Each PE can output d identical messages simultaneously; outputting different messages simultaneously (in conjunction with inputing several messages simultaneously) can involve an increase in PE complexity, depending on what the PE is required to do.

Kushner and Rosenfeld (1983) classify communication tasks as one-to-many, many-to-one, and one-to-one. One to many tasks include broadcasting, in which one PE (the root if there is a node so distinguished) sends the same message to many other PE's, in the worst case to all other PE's. In the more general one-to-one case the messages may be altered as they travel, e.g. each message could have a value that is incremented by one each time it passes through a PE; thus it keeps track of the distance it has traveled. Broadcasting must take time at least as long as the distance to the farthest node to be reached.

Many to one tasks must be divided into two classes. In both classes many PE's all send messages to the same PE (root). In one case, *condensing*, the messages can be combined (e.g. added) in route to the destination. An example of this would be for computing the area of a region - each PE in the region sends a 1, and the sum of all messages is equal to the area. This is essentially the inverse of broadcasting, and the time is again limited by the farthest distance to be traveled. In the case of *funneling*, the messages must be kept separate. This in general takes much longer. If all N PE's send messages to be funneled, then the time is bounded below by N/d, where d is the network degree, because of the "bottleneck" at the destination node. Whether this lower bound is achieved depends on the network topology.

One to one tasks are permutations, in which each PE sends a message to one other PE. In the worst case, half the PE's send messages to the other half, each message with a different destination PE. This of course must take time at least equal to the farthest distance to be traveled (the diameter of the network for the worst case). In general, bottlenecks will cause the time to be larger than the network diameter; actual time again depends on the topology.

Table 5. Order of magnitude time for communication tasks on a fixed interconnection PE network with N PE's.

Network Topology	Degree	Complexity of each PE	Broadcasting and Condensing	Permuting	Funneling
Array (nearest neighbor)	4	log$N$	$\sqrt{N}$ (1)	N? (1)	N (1)
Tree	3	log$N$	log$N$ (1)	N (1)	N (1)
Hypercube	log N	log$N$	log$N$ (1)	log$N$ (1)	$\left(\frac{N}{\log N}\right)$ (4)
Tree(2)	$1+b \approx N^{1/p}$	$N^{1/p}$	$\log_b N$	$N^{1-1/p}$ (3)	$N^{1-1/p}$ (4)
Fully connected	N	N	1 (1)	1 (1)	1 (4)

(1) From Kushner and Rosenfeld (1983).
(2) b = branching factor of tree = $m^{1/p}$, where m = no. of leaf nodes. p = radius = no. of levels - 1. $p \geq 1$.
(3) Allowing the root node to have complexity $O(N^{2/p})$.
(4) Could be higher depending on algorithm requirements

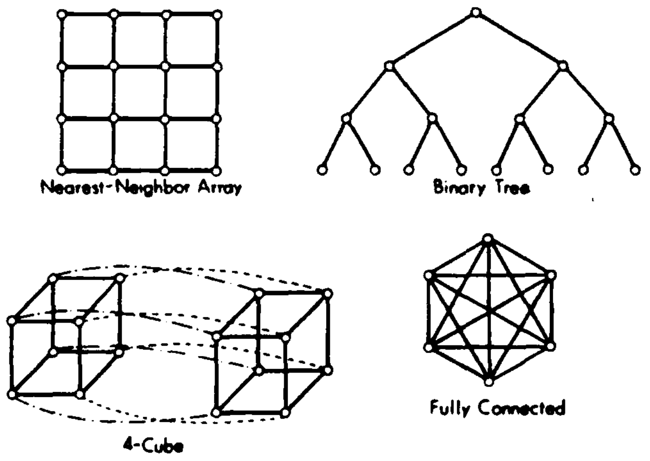


Fig. 5. Examples of network topologies.

The worst case order of magnitude communication time for several networks of different topologies and degrees is given in Table 5. The array and binary tree take the same time under our optics and electronics assumptions. The optics assumptions allow degrees that are a function of N, and further reduce the time for funnelling (from time $=N$) in some cases. Examples of nearest neighbor array, tree, hypercube, and fully connected networks are shown in Fig. 5.

CONCLUSIONS

We have studied abstract models of parallel machines at the computational paradigm level. By attempting to abstract out the limitations of electronic systems, we have found some potential advantages of optical computing systems in contention-free parallel read access to global memories, associated reconfigurable interconnection networks, and in implementing PE networks of increased degree over electronics. We have also pointed out that the connectivity of even optical systems is not unlimited; it is limited by the complexity of the components that are being connected.

REFERENCES

Aho, A.V., J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms,* Reading, Mass. Addison-Wesley, 1974.

Borodin, A. and J.E. Hopcroft, "Routing, Merging, and Sorting on Parallel Models of Computation." *Journal of Computer and System Sciences,* Vol. 30, pp. 130-145 1985.

Fortune, S., and J. Wyllie, "Parallelism in Random Access Machines," *Proc 10th Annual ACM STOC,* San Diego, California, pp. 114-118, 1978.

Hopcroft, J.E. and JK.D. Ullman, *Introduction to Automata Theory, Languages, and Computation,* Reading, Mass., Addison-Wesley, 1979.

Hwang, K. and F. A. Briggs, *Computer Architecture and Parallel Processing,* New York, McGraw-Hill, 1984.

Kushner, T. R. and A. Rosenfeld, "Interprocessor Communication Requirements for Parallel Image Processing," *Proc. IEEE 1983 Computer Architecture for Pattern Analysis and Image Database Management,* IEEE Cat. No. 83CH1929-9, pp. 177-183, 1983.

Lev, G., N. Pippenger, and L.G. Valiant, "A Fast Parallel Algorithm for Routing in Permutation Networks" *IEEE Trans. on Computers,* Vol. C-30, No. 2, pp. 33-100, Feb 1981.

Minsky, M. L., *Computation: Finite and Infinite Machines,* Engelwood Cliffs, N. J., Prentice-Hall, 1967.

Sawchuk, A. A. and B. K. Jenkins, "Dynamic Optical Interconnections for Parallel Processors," *Proc. SPIE,* Vol. 625, this issue, 1986.

Schwartz, J.T., "Ultracomputers," *A.C.M. Trans on Prog. Lang. and Sys.,* Vol. 2, No. 4, pp. 484-521, October 1980.

Shiloach, Y., and Vishkin, U., "Finding the Maximum, Merging and Sorting in a Parallel Computation Model," *J. Algorithms,* pp. 88-102, March 1981.

Stanat, D.F. and D.F. McAllister, *Discrete Mathematics in Computer Science,* Englewood Cliffs, N.J., Prentice-Hall, 1977.

Uhr, L. "Augmenting Pyramids and Arrays by Compounding them with Networks," *Proc. IEEE 1983 Computer Architecture for Pattern Analysis and Image Database Management,* IEEE Cat. No. 83CH1929-9, pp. 162-168, 1983.

2.7 Acousto-optic Signal Processing

A final area of study has been in high speed acousto-optic systems for matrix-matrix multiplication. The attached paper "Acousto-Optic Matrix-Matrix Multiplier" by D.S. Kalivas, G. Albanese and A.A. Sawchuk in *Optics Letters*, Vol. 13, pp. 291-293, April 1988 summarizes these results.

Reprinted from Optics Letters, Vol. *13*, page 291, April 1988.
Copyright © 1988 by the Optical Society of America and reprinted by permission of the copyright owner.

Acousto-optic matrix–matrix multiplier

D. S. Kalivas, G. Albanese, and A. A. Sawchuk

Signal and Image Processing Institute, University of Southern California, Los Angeles, California 90089

Received November 9, 1986; accepted January 4, 1988

A new architecture for an optical matrix–matrix multiplier is presented. It is based on the beam-modulation and beam-deflection properties of Bragg acousto-optic cells. Its parallel structure makes it very fast. Some physical limitations are discussed.

There exist many architectures for optical matrix algebraic processors.[1] Recently, several architectures that use acousto-optic cells have been proposed.[2,3] An interesting frequency-multiplexed pipelined matrix-vector processor was presented by Casasent *et al.*[4] This system is a systolic processor because the inputs enter in a clocked time-sequential pipeline manner. It is a beam-deflector-based processor, which can also be used for matrix–matrix multiplication. In this Letter we present a matrix–matrix multiplier that makes use of both the beam-deflection and the beam-modulation properties of acousto-optic (AO) cells. We describe the architecture of the processor and explain how it works and consider some basic features and limitations from a quantitative point of view.

Bragg cells have two basic properties that can be utilized in the design of an AO algebraic processor.[5,6] The first is the modulation of the intensity of a light beam, which is obtained by modulation of the amplitude of the acoustic wave. The second property is the deflection of light beams in different directions caused by frequency modulation of the acoustic wave. Our processor exploits both properties.

The architecture is shown schematically in Fig. 1, and a top view is shown in Fig. 2. It multiplies two matrices **A** and **B** of dimension $N \times N$. Various stops and unwanted diffraction orders are omitted for clarity. At the left is AO cell array A, composed of N independent AO cells. The direction of propagation of the acoustic waves in each AO cell is oriented vertically. The N cells are arranged side by side horizontally. The illumination on AO cell array A shown by the arrows at the left of Fig. 1 is a plane monochromatic wave of constant complex amplitude. The plane wave front is parallel to the left of the AO cell array A, and its aperture is large enough to illuminate array A completely. Each row of the matrix **A** drives in parallel each cell of AO cell array A (plane A). The single long AO cell B (plane B) in Fig. 1 is oriented vertically. It is shown artificially divided into N levels in Fig. 1 for the purpose of explaining the operation of the processor. The AO cell B is driven by a vector **b** generated by row scanning[7] the matrix **B**. Let us call b_{ij} ($i = 1, 2, \ldots, N; j = 1, 2, \ldots, N$) the elements of the matrix **B**. Then **b** is equal to the vector $(b_{11}, b_{12}, \ldots, b_{1N}, b_{21}, \ldots, b_{2N}, \ldots, b_{N1}, \ldots, b_{NN})^t$, where t denotes the matrix transpose. In the illustrations of Figs. 1 and 2, both lenses have the same focal length f_L, and all five components of the system are spaced at the same distance f_L. The lens located between planes A and B brings the light from plane A to a line focus in plane B (Fig. 2), and the AO cells in plane A provide a vertical deflection and amplitude modulation. At the right is an instantaneous detector array C having $N \times N$ elements, which gives the output matrix $\mathbf{C} = \mathbf{A} \times \mathbf{B}$.

In each AO cell of the array A a transducer launches

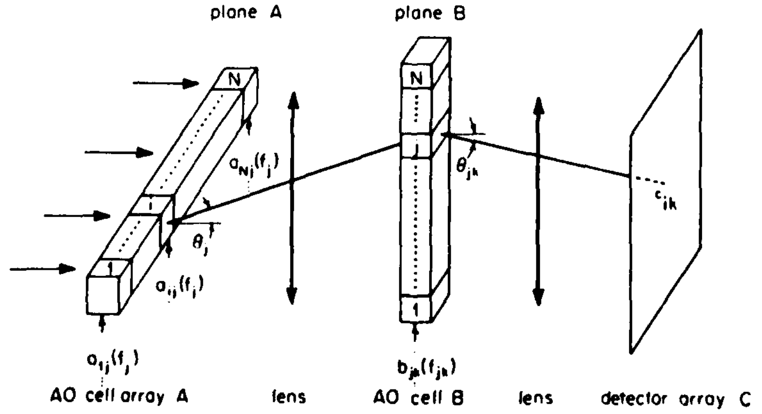


Fig. 1. Side view of the processor.

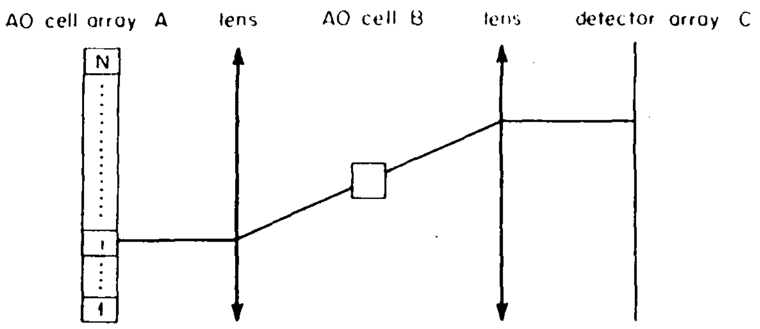


Fig. 2. Top view of the processor.

a multifrequency acoustic wave. Let us consider the ith AO cell, which is fed with the ith row of the matrix **A**. Its input has N frequency components. The jth component has amplitude a_{ij} and frequency f_j. Thus the beam incident upon this cell is split into N beams. These beams have intensities proportional to the matrix elements a_{ij} ($j = 1, 2, \ldots, N$) and are deflected at different angles θ_j, which are proportional to the frequencies f_j. The frequency f_j is such that the deflected beam is directed to the jth level of the AO cell B.

At this jth level of AO cell B there are N incident beams from each of the N AO cells of array A. Synchronized with these beams, AO cell B contains a propagating multifrequency acoustic wave with amplitudes b_{jk} ($k = 1, 2, \ldots, N$) and frequencies f_{jk} ($k = 1, 2, \ldots, N$). Each one of these incident beams is split into N beams having intensities proportional to $a_{ij}b_{jk}$ ($k = 1, 2, \ldots, N$) and is deflected at different angles θ_{jk} determined by the frequencies f_{jk} ($k = 1, 2, \ldots, N$). The frequency f_{jk} is such that the corresponding deflected beam is directed vertically from the jth level of AO cell B toward the kth row of the detector array. The horizontal angular offset of this beam is converted by the second lens to a spatial location in the ith column. Thus the combination of these two deflections directs the beam to the (i, k)th element of the detector array. At the same element (i, k) all the beams having intensities proportional to $a_{ij}b_{jk}$ ($j = 1, 2, \ldots, N$) are summed and detected. The result is proportional to the element c_{ik} of the matrix **C**.

We denote by T the time it takes for the elements of matrix **B** to enter AO cell B. After matrix **B** enters the system, the matrix–matrix multiplication is done instantaneously; thus the total processing time is equal to T. The equivalent systolic matrix–matrix processor, presented by Casasent *et al.*,[3] has a total processing time equal to $2T$.

The processing time can be drastically reduced if an array of AO cells as shown in Fig. 3 is substituted for the single long AO cell B. AO cell array B is composed of N independent AO cells whose direction of propagation is oriented vertically, and these N cells are stacked end to end vertically. Each AO cell of array B *is driven in parallel by the corresponding row of the* matrix **B,** thus eliminating the row scanning needed to input the entire matrix **B** as before. For example, the jth AO cell is driven by the jth row of the **B** matrix ($b_{j1}, \ldots, b_{jk}, \ldots, b_{jN}$). The new total processing time is equal to T/N. This time reduction is achieved at the cost of increased architectural complexity.

The AO cells described above serve as light modulators and beam deflectors. The equations that determine these two operations are[6]

$$\eta = \sin^2\left(\pi L \frac{\sqrt{MI_\alpha}}{\sqrt{2}\lambda \cos\theta_B}\right), \tag{1}$$

where η is the diffraction efficiency, L is the length of interaction between light and sound (i.e., the thickness of the AO cell along the direction of light propagation), λ is the wavelength of light in free space, I_α is the acoustic intensity, M is a constant having dimensions of square meters per watt defined by the AO cell material, and θ_B is the Bragg angle given by

$$\theta_B = \sin^{-1}\left(\frac{\lambda f_0}{2nu}\right). \tag{2}$$

Here f_0 is the acoustic frequency, u is the phase velocity of the sound, and n is the optical index of refraction of the AO cell material. Assuming small acoustic power and a small Bragg angle, we can further simplify the above equations to

$$\eta \approx \frac{\pi^2 L^2 M I_\alpha}{2\lambda^2 \cos^2\theta_B}, \tag{3}$$

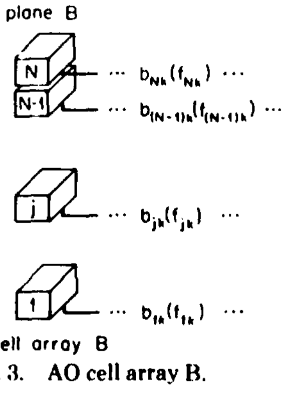


Fig. 3. AO cell array B.

$$\theta_B \approx \frac{\lambda f_0}{2nu} . \quad (4)$$

From relation (3) we see that the diffraction efficiency is approximately linearly proportional to the acoustic intensity I_a. Thus modulation of the acoustic intensity results in modulation of the diffracted light intensity. Also, relation (4) shows that the deflection angle $2\theta_B$ is approximately linearly proportional to the acoustic frequency.

The most important parameters in this processor are the number of resolvable beam spots, or, in other words, the dimension of the matrices that can be multiplied and the computation time T. In addition, the processor must operate with acceptable efficiency.

The number of resolvable beam spots is given by[6]

$$N = \frac{\pi D \Delta f}{4u \cos\theta_B} , \quad (5)$$

where Δf is the bandwidth of the acoustic wave around the center frequency f_0 and D is the diameter of the illuminating beam measured along the direction of acoustic-wave propagation. Thus the basic parameter that determines the dimension N is Δf, and it must satisfy the two conditions[6]

$$\Delta f \leq f_0, \quad (6)$$

$$\Delta f \leq \frac{2nu^2}{\lambda L f_0} . \quad (7)$$

The condition of relation (6) ensures modulation without nonlinear distortion, while relation (7) arises from the need to maintain the proper Bragg angle between incident light and acoustic waves in the AO cell (a phase-mismatch condition). These conditions also affect the thickness L of the AO cell. To obtain a large N a small L is desirable, but this results in low efficiency [Eq. (1)]. Thus there is a trade-off in assigning a value for L. This problem can be overcome by using a beam-steering technique.[8]

We now consider a practical example and evaluate the dimension N, the computation time T, and the diffraction efficiency η for the system shown in Fig. 1. Let the medium of the AO cells be $PbMoO_4$, which has the parameters $n = 2.4$, $\mu = 3.75$ km/sec, and $M = 7.3 \times 10^{-14}$ m^2/W. Choosing $\lambda = 0.5145$ μm, $f_0 = 100$ MHz, $\Delta f = 100$ MHz, $I_a = 250$ mW/cm^2, $L = 1$ cm, and $D = 3$ mm, we obtain $N = 60$, $T = 60$ μsec, and $\eta = 0.34$. If we use instead the AO cell array B shown in Fig. 3 in the system of Fig. 1, the computation time is 1 μsec. These results show that the processor is very fast and operates with acceptable efficiency.

A final consideration concerns the phase mismatch of the AO cells. We have assumed that the beams incident upon the cells arrive at the correct angle for Bragg diffraction. While this is true for AO cell array A (Fig. 1), it is not generally true for AO cell B. In fact, the beams incident upon cell B necessarily arrive at different angles because they are the output beams of the AO cell array A and cannot arrive at the correct Bragg angle. This phase mismatch results in a decrease in the number of resolvable spots, although it may be possible to reduce the effects of the phase mismatch by the use of correcting lenses or lens arrays.

In this Letter we have presented a matrix–matrix multiplier. Its operation is based on the modulation and deflection properties of AO cells. It is fast because of its parallel architecture. The dimension of the matrices that it can multiply is sufficiently large. This processor can be used for implementations of algorithms that require matrix–matrix multiplications such as LU decomposition, QR decomposition, direct solution of linear equations, and Kalman filtering.[3]

References

1. R. Athale, in *Proceedings of the 10th International Optical Computing Conference* (Institute of Electrical and Electronics Engineers, New York, 1983), pp. 24–31.
2. W. T. Rhodes and P. S. Guilfoyle, Proc. IEEE **72,** 820 (1984).
3. D. Casasent, Proc. IEEE **72,** 831 (1984).
4. D. Casasent, J. Jackson, and C. Newman, Appl. Opt. **22,** 115 (1983).
5. A. Yariv, *Optical Electronics* (Holt, Rinehart and Winston, New York, 1985).
6. A. Yariv and P. Yeh, *Optical Waves in Crystals* (Wiley, New York, 1984), pp. 366–404.
7. W. K. Pratt, *Digital Image Processing* (Wiley-Interscience, New York, 1978).
8. A. Korpel, R. Adler, P. Desmares, and W. Watson, Proc. IEEE **54,** 1429 (1966).

3 Written Publications

During the period 1 July 1984 through 31 January 1990, a number of papers based on this research have been published or submitted. A list of these follows:

1. A.A. Sawchuk, "Numerical Optical Computing Techniques", *Proc. 13th Congress of International Commission for Optics, ICO-13,* Sapporo, Japan, August 20-24, 1984.

2. A.A. Sawchuk and T.C. Strand, "Digital Optical Computing", *Proc. IEEE,* Vol. 72, pp. 758-779, 1984.

3. B.K. Jenkins, *et al,* "Sequential Optical Logic Implementation", *Applied Optics,* Vol. 23, pp. 3455-3464, 1984.

4. B.K. Jenkins, *et al,* "Architectural Implications of a Digital Optical Processor", *Applied Optics,* Vol. 23, pp. 3465-3474, 1984.

5. B.K. Jenkins and A.A. Sawchuk, "Characteristics of Digital Optical Processors", *Proc. IEEE Global Telecommunications Conf.,* Atlanta, GA, November 26-29, 1984.

6. B.K. Jenkins, "Architectural Characteristics of Optical Logic Systems", *Proc. IEEE Computer Conf.,* San Francisco, CA, February 26-28, 1985.

7. B.K. Jenkins and A.A. Sawchuk, "Computer Generated Hologram Considerations for Sequential Optical Logic Interconnections", presented at Optical Society of America 1984 Annual Meeting, San Diego, CA, October 29 - November 2, 1984.

8. A.A. Sawchuk and B.K. Jenkins, "Algorithms for Digital Optical Processors, *Tech. Digest for Topical Meeting on Optical Computing,* Optical Society of America, March 18-20, 1985, Incline Village, NV, pp. TuA2-1 - TuA2-4.

9. B.K. Jenkins and A.A. Sawchuk, "Optical Cellular Logic Architectures for Image Processing", *Proc. IEEE Computer Society Workshop on Computer Architectures for Pattern Analysis and Image Processing,* Miami Beach, FL, November 1985.

10. A.A. Sawchuk and B.K. Jenkins, "Optical Cellular Logic Processors", 1985 Annual Meeting, Optical Society of America, Washington, D.C., October 1985; *Journal of the Optical Society of America-A,* Vol. 2, p. P26, December 1985.

11. B.K. Jenkins, "Recent Developments in Digital Optical Computing", 1985 Annual Meeting, Optical Society of America, Washington, D.C., October 1985; *Journal of the Optical Society of America-A,* Vol. 2, p. P22, December 1985.

12. A.A. Sawchuk, "Prospects for Optical Computing and Interconnections", International Conference on Lasers '85, Las Vegas, December 2-6, 1985, (invited paper).

13. B.K. Jenkins and A.A. Sawchuk, "Binary Optical Computing Architectures", *Optics News,* Vol. 12, No. 4, pp. 25-26, April 1986.

14. K.S. Huang, B.K. Jenkins and A.A. Sawchuk, "Binary Image Algebra and Digital Optical Cellular Image Processors", *Topical Meeting on Optical Computing* Technical Digest Series 1987, Vol. 11, Optical Society of America, Paper MB5, pp. 20-23, Washington, D.C., 1987.

15. A. Armand, A.A. Sawchuk and T.C. Strand, "Nonlinear Optical Processing with Halftones: Accurate Predictions for Degradation and Compensation," *Applied Optics,* Vol. 26, No. 6, pp. 1007-1014, 1987.

16. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Binary Image Algebra Representations of Optical Cellular Logic and Symbolic Substitution," *OSA Annual Meeting,* Rochester, NY, October 18-23, 1987.

17. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Programming A Digital Optical Cellular Image Processor," *OSA Annual Meeting,* Rochester, NY, October 18-23, 1987.

18. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Optical Cellular Logic Architectures Based on Binary Image Algebra," *Proc. IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence,* Seattle, WA, October 1987.

19. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "A Cellular Hypercube Architecture for Image Processing," *Applications of Digital Image Processing X,* Proc. of SPIE-The International Society for Optical Engineering, Vol 829, San Diego, CA, August 1987.

20. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "An Image Algebra Representation of Parallel Optical Binary Arithmetic," *Applied Optics,* Vol. 28, pp. 1263-1278, March 15, 1989.

21. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Binary Image Algebra and Optical Cellular Logic Processor Design," *Computer Vision, Graphics, and Image Processing,* Vol. 45, pp. 295-345, (1989).

22. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Optical Symbolic Substitution and Pattern Recognition Algorithms Based on Binary Image Algebra," *ICO Topical Meeting on Optical Computing,* Toulon, France, August 29 - September 2, 1988.

23. K.S. Huang, A.A. Sawchuk, B.K. Jenkins, P. Chavel, J.M. Wang, A.G. Weber, C.H. Wang, and I. Glaser, "Implementation of A Prototype Digital Optical Cellular Image Processor (DOCIP)," *ICO Topical Meeting On Optical Computing,* Toulon, France, August 29 - September 2, 1988.

4 Professional Personnel and Advanced Degrees

The following individuals contributed to the research effort supported by this grant:

Dr. Alexander A. Sawchuk, Professor of Electrical Engineering, Director, Signal and Image Processing Institute; Principal Investigator.

Dr. B.K. Jenkins, Assistant Professor of Electrical Engineering; Senior Investigator.

Herb Barad, Research Assistant, Ph.D. Candidate, Department of Electrical Engineering.

Kung-Shiuh Huang, Research Assistant, Ph.D. Candidate, Department of Electrical Engineering.

Dr. Isaia Glaser, Visiting Associate Professor of Electrical Engineering.

Dr. Pierre Chavel, Visiting Scientist, Signal and Image Processing Institute.

The following received advanced degrees through the research effort sponsored by this grant:

Kung-Shiuh Huang, Ph.D., Department of Electrical Engineering, January 1989.

Herb Barad, Ph.D., Department of Electrical Engineering, January 1988.

5 Interactions (Coupling Activities)

During the period 1 July 1984 through 31 January 1990, oral presentations have been made at meetings and conferences based on this work. A list of these follows:

1. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Binary Image Algebra Representations of Optical Cellular Logic and Symbolic Substitution," *OSA Annual Meeting,* Rochester, NY, October 18-23, 1987.

2. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Programming A Digital Optical Cellular Image Processor," *OSA Annual Meeting,* Rochester, NY, October 18-23, 1987.

3. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Optical Cellular Logic Architectures Based on Binary Image Algebra," *Proc. IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence,* Seattle, WA, October 1987.

4. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "A Cellular Hypercube Architecture for Image Processing," *Applications of Digital Image Processing X,* Proc. of SPIE-The International Society for Optical Engineering, Vol 829, San Diego, CA, August 1987.

5. K.S. Huang, B.K. Jenkins, and A.A. Sawchuk, "Optical Symbolic Substitution and Pattern Recognition Algorithms Based on Binary Image Algebra," submitted to, *ICO Topical Meeting on Optical Computing,* Toulon, France, August 29 - September 2, 1988.

6. K.S. Huang, A.A. Sawchuk, B.K. Jenkins, P. Chavel, J.M. Wang, A.G. Weber, C.H. Wang, and I. Glaser, "Implementation of A Prototype Digital Optical Cellular Image Processor (DOCIP)," submitted to, *ICO Topical Meeting On Optical Computing,* Toulon, France, August 29 - September 2, 1988.

7. B.K. Jenkins and A.A. Sawchuk, "Optical Cellular Logic Architectures for Image Processing", *Proc. IEEE Computer Society Workshop on Computer Architectures for Pattern Analysis and Image Processing,* Miami Beach, FL, November 1985.

8. A.A. Sawchuk and B.K. Jenkins, "Optical Cellular Logic Processors", 1985 Annual Meeting, Optical Society of America, Washington, D.C., October 1985; *Journal of the Optical Society of America-A,* Vol. 2, p. P26, December 1985.

9. B.K. Jenkins, "Recent Developments in Digital Optical Computing", 1985 Annual Meeting, Optical Society of America, Washington, D.C., October 1985; *Journal of the Optical Society of America-a,* Vol. 2, p. P22, December 1985.

10. A.A. Sawchuk, "Prospects for Optical Computing and Interconnections", International Conference on Lasers '85, Las Vegas, December 2-6, 1985, (invited paper).

11. B.K. Jenkins and A.A. Sawchuk, "Binary Optical Computing Architectures", *Optics News*, Vol. 12, No. 4, pp. 25-26, April 1986.

12. B.K. Jenkins and A.A. Sawchuk, "Development and Future of Optical Computing", Panel Members, Optical Computing Symposium, Society of Photo-Optical Instrumentation Engineers '86 Optoelectronics and Laser Applications in Science and Engineering, Los Angeles, CA, January 1986.

13. A.A. Sawchuk, "Numerical Optical Computing Techniques", *Proc. 13th Congress of International Commission for Optics, ICO-13*, Sapporo, Japan, August 20-24, 1984.

14. B.K. Jenkins and A.A. Sawchuk, "Characteristics of Digital Optical Processors", *Proc. IEEE Global Telecommunications Conf.*, Atlanta, GA, November 26-29, 1984.

15. B.K. Jenkins, "Architectural Characteristics of Optical Logic Systems", *Proc. IEEE Computer Conf.,* San Francisco, CA, February 26-28, 1985.

16. B.K. Jenkins and A.A. Sawchuk, "Computer Generated Hologram Considerations for Sequential Optical Logic Interconnections", presented at Optical Society of America 1984 Annual Meeting, San Diego, CA, October 29 - November 2, 1984.

17. A.A. Sawchuk and B.K. Jenkins, "Algorithms for Digital Optical Processors, *Tech. Digest for Topical Meeting on Optical Computing,* Optical Society of America, March 18-20, 1985, Incline Village, NV, pp. TuA2-1 - TuA2-4.

18. K.S. Huang, B.K. Jenkins and A.A. Sawchuk, "Binary Image Algebra and Digital Optical Cellular Image Processors", *Topical Meeting on Optical Computing* Technical Digest Series 1987, Vol. 11, Optical Society of America, Paper MB5, pp. 20-23, Washington, D.C., 1987.