An Optical Content-Addressable Parallel Processor for Fast Searching and Retrieving *

Ahmed Louri
Department of Electrical and Computer Engineering
The University of Arizona, Tucson, Arizona 85721

Associative processing based on content-addressable memories has been argued to be the natural solution for non-numerical information processing applications. Unfortunately, the implementation requirements of these architectures using conventional electronic technology have been very cost prohibitive, and therefore associative processors have not been realized. Instead, software methods that emulate the behavior of associative processing have been promoted and mapped onto conventional location-addressable systems. This however, does not bring about the natural parallelism of associative processing, namely the ability to access many data words simultaneously.

The inherently parallel nature and high speed of optics, combined with the recent technological advancements in optical logic, storage and interconnect devices are raising hopes for practical realization of highly parallel optical computing systems. This paper presents the principles of designing an optical content-addressable parallel processor, called OCAPP, for the efficient support of high speed symbolic computing. The architecture is designed to fully exploit the parallelism an high speed of optics. Several parallel algorithms are mapped onto OCAPP in bit-parallel as well as word-parallel fashion, resulting in efficient symbolic algorithms with execution times dependent only on the precision of the operands and not on the problem size. This makes OCAPP very suitable for applications where the number of data sets to be operated on is high e.g., massively parallel processing. A preliminary optical implementation of the architecture using currently available optical components is also presented.

1 Introduction

The "information explosion" seen in recent years has stimulated the development of computer-based information systems to assist in the creation, storage, modification, classification, and retrieval of mainly textual or symbolic data. For example, progress in database management systems, expert systems, and intelligent knowledge-based systems is increasing demand for symbolic information processing such as text editing, file processing, table sorting, searching, and retrieval. In fact a substantial proportion of the work-load of modern information processing systems involve searching and sorting symbolic data[1]. Nevertheless, a majority of today's computers are designed mainly for numerical computations, and suffer from a fundamental handicap, which stems from the principle of addressing the memory.

When a search for a value is made through a location-addressable memory, the entire memory may need to be searched one word at a time (if the data is not sorted in memory) which consumes a great deal of time. There is no logical reason why the search must be done sequential. The only reason stems from the fundamental handicap of separating processing and memory and addressing memory one word at a time. This fundamental flaw has forced system analysts and programmers to develop sophisticated software techniques for symbolic information processing such as hashing

*This research was supported by an NSF Grant No. MIP-8909216.

and indexing[2]. However, the implementation of such software techniques on location-addressed computers has lead to complex, expensive, and inefficient information processing systems.

Searching, retrieving, sorting and modifying symbolic data can be significantly improved by the use of content-addressable memory (CAM) instead of location-addressability. In a content-addressable memory data is addressed by its contents[2]. An associative processor is a parallel processing machine in which the data items are content-addressable with the added capability to write in parallel into words satisfying certain criterion. It may be that the entire contents of stored words may be changed or just a few bits of the words. Using this model, processing is carried out within the associative memory, without transfer to an independent processing unit. Since there is no addressing of data and no data movement, this implies the elimination of the fundamental von Neumann bottleneck encountered in conventional systems. Moreover, the amount of time required for searching, retrieving, and updating information is independent of the data set sizes.

However, this model of computing is not being largely used because of the difficulty and high cost of implementing it in conventional electronic technology. This can be seen from the following:

1. Each bit cell in an associative memory is much more complex and requires more circuitry than does a conventional cell. Even with the advent of VLSI technology, the single cell complexity still does not allow for the use of large associative memories.
2. The memory storage provides poor storage density compared with conventional memory.
3. The third major difficulty is the complexity of the interconnects. Recall that in order for all cells to compare their values to that of the comparand register, the control unit must broadcast the value to all cells involved in the comparison. However, using conventional technology, the time delays associated with the broadcasting function are very appreciable. Moreover, inter-cell interconnects become cumbersome for large array size.
4. The fourth difficulty is the lack of efficient means of implementing parallel access to the cells, namely parallel input and output.

There are two hypotheses underlying this paper:

1. that CAM-based processing provides a sound basis to uncover inherent parallelism in symbolic processing and information retrieval applications, and
2. that optics is, potentially, the ideal medium to exploit such parallelism by providing efficient implementation support for it.

2 Optical Content-Addressable Parallel Processor

Optical systems hold the promise for providing efficient support for future parallel processing systems. Optics advantages have been cited on numerous occasions[3, 4, 5, 6]. These include inherent parallelism, high spatial and temporal bandwidths, and non-interfering communications. For CAM-based processing, optics may be the ideal solution to the fundamental problems faced by electronic implementations, namely cell complexity, interconnects latency, difficulty of implementing information broadcasting and parallel access to the stored data. Optics can alleviate the cell complexity by migrating the implementation of wiring and logic into free-space. The multi-dimensional nature of optical systems allows for data storage and logic to be performed on two-dimensional planes while the third dimension can be used for interconnects. The high degree of connectivity available in free-space space-invariant optical systems (10^6 to 10^8), and the ease with which optical signals can be expanded (which allows for signal broadcasting) and combined (which allows for signal funneling) can also be exploited to solve the interconnects problems[7, 8]. Moreover, optical and electro-optical systems can offer a considerable storage capacity and parallel access than do pure electronic systems[9].

Figure 1 depicts a preliminary organizational structure for an optical content-addressable parallel processor called OCAPP. The architecture is organized in a modular fashion, and consists of a *selection unit*, a *match/compare unit*, a *response unit*, an *output unit*, and a *control unit*. The architecture is developed to meet four goals, namely: (1) exploitation of maximum parallelism; (2) amenability to optical implementation with existing devices; (3) modular design in that it can be scalable to bigger problems; and (4) ability to efficiently implement information retrieval, and symbolic computations. Moreover, the programming methodology for OCAPP is compatible with that of existing single-instruction multiple data (SIMD) systems. In what follows we describe the role of each unit. Detailed optical implementation of OCAPP will be presented in Sec.3.

The selection unit is schematically described in Fig.2. It is comprised of (1) a storage array of n words, each m bits long (in actuality, the storage array capacity is $n \times 2m$, since each bit position is comprised of a true bit w_{ij} and its complement $\bar{w}_{ij}$); and (2) word and bit-slice enable logic to enable/disable the words and/or the bit-slices that participate in the match operation, and reset the rest. It is assumed that the storage array can be loaded in parallel and (if need be) read in parallel.

The match/compare unit shown in Fig.3, contains a (1) $1 \times m$ interrogation register I; (2) logic hardware to perform parallel bitwise comparison between the bits of the interrogation register and the enabled bits of the storage array; (3) two $n \times 1$ working registers, G and L, which are used for magnitude comparisons (to be explained later); (4) a $n \times 1$ response register R for displaying the result of the comparison; and (5) a single indicator bit called the match detector MD, which indicates whether or not there is any matching words. This unit allows comparison of a single operand stored in the interrogation register and the words stored in the storage array. As such it is considered an SIMD (single-instruction-multiple data) unit. Bit position R_i of R is set to one when word W_i of the storage array matches the contents of I. The I register is a combination of the comparand register C and the mask register M as shown in table 1. As such, it holds the operand (depending on masking information if any) being searched for or being compared with. It is assumed that register I is available in dual-rail logic (both true and compliment bits available).

The response unit is responsible for selecting one or several matching words. It comprises several scratchpad registers and a priority circuit for selecting the first matching word. Depending on program control, the output of the response unit is routed either to the output unit for outputting the result or fed back to the selection unit for further processing of the matching words. All units are under the supervision of a conventional control unit with conventional storage (eg., a local RAM) which stores the program instruction. Its role is to load/unload the storage array, set/rest various registers such as the I, R, G and L of the match/compare unit, enable/disable memory words, perform conditional instructions, monitor the MD bit, and test program termination. In what follows, we describe the implementation of several parallel algorithms on the OCAPP in order to show its use and processing benefits.

Table 1: Formulation of the interrogation register

Search bit c_j	Mask bit m_j	Interrogation bits I_j I_j
0	0	0 1
1	0	1 0
0	1	1 1
1	1	1 1 (no comparison is performed at this bit position)

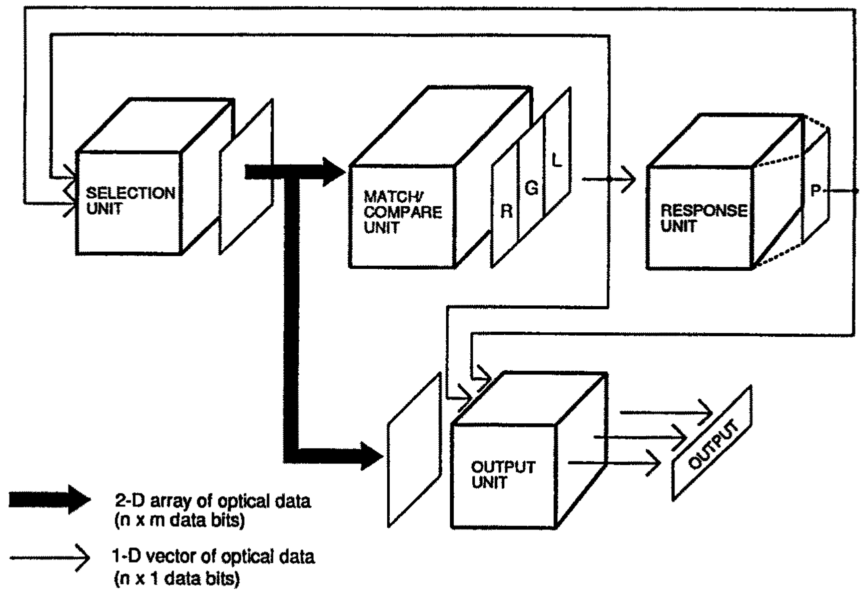

SELECTION UNIT
MATCH/ COMPARE UNIT
R
G
L
RESPONSE UNIT
P
OUTPUT UNIT
OUTPUT
2-D array of optical data (n x m data bits)
1-D vector of optical data (n x 1 data bits)

Figure 1 : A schematic organization of the proposed optical content-addressable parallel processor : OCAPP.

3 Parallel Search Algorithms on OCAPP

We classify search operations as basic and compound operations. A basic search operation is one which can be completed in one sweep over all the bit-slices of the storage array. It does not involve any feedback processing. A compound search operation requires a feedback from the response unit to the selection unit. As a consequence, it takes more than one sweep over the storage array to complete. Under basic search operations, we group the following operations:

- *Equivalence Search*: The equality search, the not-equal-to search, and the similarity search (search for a match within a masked field).
- *Threshold Search*: The smaller-than, the not-smaller-than, the greater-than, and the not-greater-than searches.
- *Extrema Search*: The greatest value search, and the smallest value search.

Compound search operations can be implemented in a series of basic search operations. Under the compound search, we group the following operations:

- *Adjacency Search*: Next-above search, and next-below search.
- *Between-Limits Search*: Search for words z, between two limits X and Y ($X < Y$): a) $X \leq z \leq Y$, b) $X < z \leq Y$, c) $X \leq z < Y$, and d) $X < z < Y$.

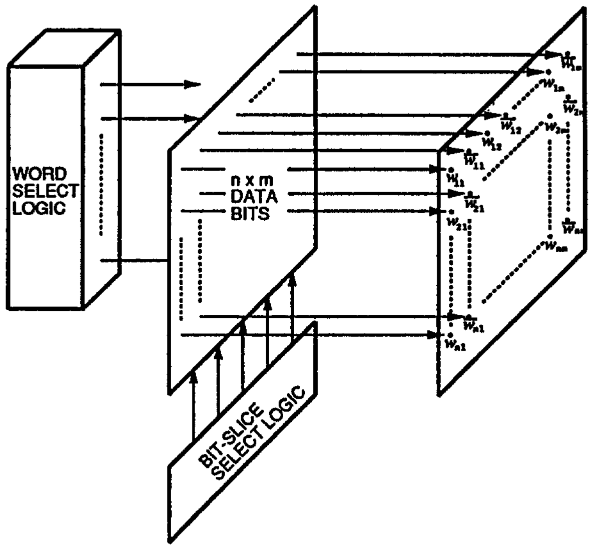


Figure 2 : Organization of the selection unit.

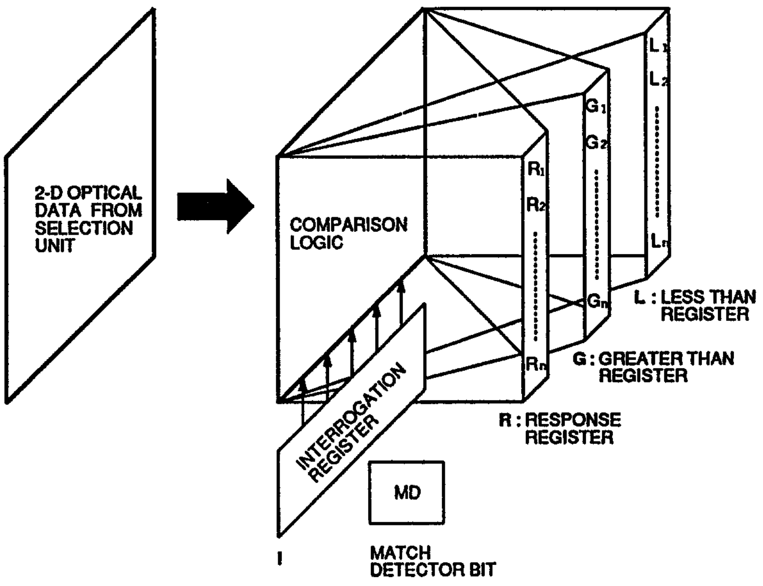


Figure 3 : Organization of the match/compare unit.

- *Outside-Limits Search*: Search for words z, outside two limits X and Y ($X < Y$): a) $X \leq z$ or $z \geq Y$, b) $X > z$ or $z \geq Y$, c) $X \geq z$ or $z > Y$, and d) $X > z$ or $z > Y$.
- *Ordered Retrievals (sorting)*: Ascending order retrieval, and descending order retrieval.

Of course many more compound search operations can be formulated using the basic search operations. The above search operations are the most frequently used in information retrieval applications.

3.1 Parallel Algorithms for Basic Search Operations on OCAPP

In what follows, we denote a memory word as $W_i = (w_{im}w_{im-1} \dots w_{i1})$ where w_{ij} is the jth bit cell of the word W_i. We denote the jth bit-slice by $B_j = (w_{1j}w_{2j} \dots w_{nj})$, which is made up of the jth bit of every word in the storage array. The interrogation and response registers are denoted by $I = I_m I_{m-1} \dots I_1$, and $R = R_1 R_2 \dots R_n$ respectively. The comparand word (search argument) and the mask register words are denoted by $C = (c_m c_{m-1} \dots c_1)$, and $M = (m_m m_{m-1} \dots m_1)$.

3.1.1 Equivalence Search

In this type of search, the memory is partitioned according to the magnitude of the search word C into two sets, namely, words which are equal to C and words which are not. The equality and masked search operations can be implemented by a bitwise match. For equality match all the bits of the search word need to be matched, whereas for the masked search, only a subset of the bits of the search word is compared with the respective bits of the memory words. For $m_j = 0$ means that c_j is not masked, while $m_j = 1$ means c_j is masked. These two search modes can be combined as shown in Table 1. Given an interrogation word I, a bit match denoted by b_{ij} on the jth cell of the ith word is given by:

$$b_{ij} = (I_j \wedge w_{ij}) \vee (\bar{I}_j \wedge \bar{w}_{ij}) \qquad \text{(equivalence)} \tag{1}$$

where the symbols $\wedge$, $\vee$, and the bar ($^-$) denote the logical AND, logical OR and logical NOT respectively. Now the exact matching of memory word W_i with interrogation vector I requires the logical product of the bits b_{ij} for $j = 1, \dots, m$, therefore:

$$R_i = \bigwedge_{j=1}^{j=m} b_{ij} = b_{im} \wedge b_{im-1} \wedge \dots \wedge b_{i1}. \tag{2}$$

where $\bigwedge$ denotes a logical AND over all bits. The above equation indicates that matching words in memory will be flagged by having their corresponding R bit set to one, and all mismatches will have their R bits set to zero. Equations 1 and 2 are space-invariant and can be implemented in bit-parallel as well as word-parallel fashion. Therefore, all R_is for $i = 1, \dots, n$, are computed at the same time with a single access to the storage array.

Equivalence Search Algorithm:

```
1) Initialization:
    a) Load I (this will depend on the search word and the masking condition);
    b) Clear R (clear all bits of the R register);
2) Perform comparison:
    a) $b_{ij} = (I_j \wedge w_{ij}) \vee (\bar{I}_j \wedge \bar{w}_{ij})$ ;
    b) $R_i = \bigwedge_{j=1}^{j=m} b_{ij}$ for $i = 1, \dots, n$. ($R_i = 1$ if and only if $W_i$ matches I).
```

3.1.2 Threshold Search

This mode of search partitions the memory according to the magnitude of the search word C into three sets, namely words which are equal to C, words which are less than C, and words which are greater than C. The result of the search is stored in the three registers of the response unit, namely R, G and L. Initially, all memory words are made active by making control registers RGL = 100. The memory is scanned from the most significant to the least significant bit position by enabling a single bit-slice at a time. When the comparand bit c_j is one, we select all active memory words with $w_{ij} = 0$ as "less than" by setting their corresponding bit position RGL = 001. These words are then disabled from further comparisons (the disabling process will be explained later). Similarly, when $c_j = 0$, we select all active memory words with $w_{ij} = 1$ as "greater than" by setting their corresponding bit position RGL = 010, and then disable them from further processing. At the end of the last bit position, words still in the state RGL = 100 are equal to the comparand, words in the state RGL = 010 are greater than the comparand, and words in the state RGL = 001 are less than the comparand. It is important to note that, even though we are scanning the memory from most significant bit to least significant bit, the search process can be terminated any time there are no matching words at a given bit position ($R_i = 0$ for all $i = 1, \ldots, n$). Such a condition is easily detectable by checking the MD bit. The detailed algorithm follows.

Threshold Search Algorithm:

1) *Initialization:*

    a) Load I (depending on the search word and masking condition);

    b) Enable memory words;

    c) Set R, clear G, clear L, set $j = m$ (the variable j is used by the control unit to scan the storage array);

2) *Perform Magnitude Search at bit-slice j:*

    a) $R_i = \bigwedge_{j=1}^{j=m} b_{ij}$, $G_i = \bigwedge_{j=1}^{j=m} I_i \wedge \bar{w}_{ij}$, $L_i = \bigwedge_{j=1}^{j=m} \bar{I}_i \wedge w_{ij}$ for $i = 1, \ldots, n$ (note that only the enabled bit-slice j determines the values of R_i, G_i and L_i, all other bit-slices are disabled at this time, and therefore have no influence);

    b) Test if MD = 1 (is there any words that match the I register at the current bit position j ?);

3) *If* $MD = 1$ *do :*

    a) Disable memory words whose corresponding bits in R are zero (memory words W_i with $R_i = 0$ have already been decided on);

    b) Decrement j: $j \leftarrow j - 1$, and test if $j = 0$?;

    c) If $j \neq 0$, go to step 2;

    d) If $j = 0$, go to step 4;

4) *If* $(MD = 0)$ *or* $(j = 0)$, then we are done and the search result is reported in R, G, and L.

The following example illustrates the algorithm for a magnitude search of 7 words, each 5 bits long:

Example 1: Threshold Search

Search word, S :	10110
Mask word:	00000
I register:	10110 (effective word search)

Memory word i	W_i	State of RGL at the end of the jth iteration				
		j = 5	j = 4	j = 3	j = 2	j = 1 (last iteration)
1	10111	100	100	100	100	010 ($W_1 > S$)
2	11000	100	010	010	010	010 ($W_2 > S$)
3	10010	100	100	001	001	001 ($W_3 < S$)
4	10110	100	100	100	100	100 ($W_4 = S$)
5	10101	100	100	100	001	001 ($W_5 < S$)
6	01101	001	001	001	001	001 ($W_6 < S$)
7	11101	100	010	010	010	010 ($W_7 > S$)

3.1.3 Extrema Search

This type of search refers to finding the maximum (or minimum) of a set of (or all) memory words. We consider first the search for maximum.

A. Maximum Search

To find the maximum, we scan memory words from the most to the least significant bit positions. As we scan the bit-slices, we determine if any of the enabled words have a one in the current bit position. If we find some, we disable all those words that do not have a one in this position. If none of the words at the current position possess a one, we do nothing. At any given time, all remaining candidates are equal as far as we have examined them, because for every bit position either everybody had a zero in that bit position, or whenever some words have ones, we disable the ones with zeros. Therefore, at bit position j, enabled words with $w_{ij} = 1$ are larger than enabled words with $w_{ij} = 0$. Since we are seeking the maximum, we disable the ones with $w_{ij} = 0$. This process is repeated until we exhaust all bit positions at which time the maximum word will be indicated by $R_i = 1$.

Algorithm for Finding the Maximum:

1) *Initialization:*
   a) Load I :$I \leftarrow 11\ldots11$ (I is loaded with all bits set to one);
   b) Clear MD, set $j = m$;
   c) Enable memory words, and set $R_i = 1$ for $i = 1, \ldots, n$;
2) *Perform equivalence search at bit-slice j:*
3) *Test if $MD = 1$* (is there any words with a one in the current bit position j ?);
4) *If $MD = 1$ do :*
   a) Disable all words which do not have a one in the current bit position (these words are indicated by $R_i = 0$).
   b) Clear R and MD;
   c) Go to step 5;
5) *Decrement j* : $j \leftarrow j - 1$, and test if $j = 0$?;
   a) If $j \neq 0$, go to step 2;
   b) If $j = 0$, output maximum value indicated by $R_i = 1$.

B. Minimum Search

The search for the minimum is very similar to the search for the maximum except that the I register is initially loaded with zeros and that if any enabled word has a zero in the current bit position (There exists a memory word W_i such that its corresponding $R_i = 1$), we disable the words with a one in the current bit position ($R_i = 0$). These words are bound to be greater than the minimum sought. The process is repeated until we exhaust all bits of the enabled words. The minimum value will also be indicated by a one in register R.

3.2 Parallel Algorithms for Compound Search Operations on OCAPP

Compound search operations such as the ones stated earlier can not be economically implemented by a single sweep over the memory words. We therefore choose to implement such operations as a series of basic searches. The rationale is to keep the architecture as simple as possible, and therefore making it highly amenable to optical implementation. Of course speed improvements can be gained by implementing these search operations as basic search, but the amount of logic circuits may be extensive.

3.2.1 Double Limits Search (Between and Outside Limits)

Given two numbers called HIGH and LOW, the double limits search consists of finding those words that are between this limits and/or words that are outside these limits. This gives rise to eight different searches which can be accomplished in a very similar manner. Let us consider the between limit search. Given the two numbers HIGH and LOW, we wish to find those words that are greater than LOW but less than HIGH namely, find all W_i such that $LOW < W_i < HIGH$. We can accomplish this search by using the magnitude comparison search as follows. First, we determine the words that are less than the comparand HIGH. These words will be indicated by a one in the L register. We then disable all other words except the ones that are less than HIGH, and perform another threshold search using the comparand LOW. After the second search, words that are less than HIGH and greater than LOW will be marked with a one in the G register, which could be routed to the output unit for outputting the search result.

3.2.2 Adjacency Search

To find the word that is next-above the comparand (the smallest word larger than the comparand), we search for all words that are larger than the comparand and then select their minimum. Similarly, to find the word that is next-below the comparand (the largest word smaller than the comparand), we search for all words less than the comparand and select their maximum. The search for the largest word smaller that the comparand (next-below search) can be carried out by a similar algorithm as the one above. In this case, step two of the next-above algorithm is replaced by a search for words that are less than the comparand, and step four is replaced by a maximum search.

3.2.3 Ordered Retrieval (Sorting)

The sorting or ordered retrieval of a set of data can be achieved by performing the extrema search repeatedly until all the data are retrieved. For the ascending order retrieval, we enable the memory words to be sorted, and determine their minimum (using the minimum search operation). We output the obtained minimum value and disable it from the storage array. We repeat these steps until we retrieve (in ascending order) all the enabled words. For descending order retrieval, we select the maximum value at each step.

4 Optical Implementation

In this section we identify the fundamental and basic operations required to implement the optical architecture, and describe possible optical components for achieving them. Detailed practical implementation issues and experimental setups will be the subject of a different publication.

4.1 Basic Operations and Hardware Components Required

An analysis of the conceptual OCAPP, the basic operations, and the algorithms reveals that in order to optically implement OCAPP, the following functions are required: (1) data is optical and must be available in dual-rail format (both the value and its complement is required); (2) parallel access for writing into and reading from the storage array as well as the various control registers; (3) disabling/enabling a memory word (or several memory words) based on certain criteria; (4) logical AND, and logical OR; (5) space-invariant optical transmission of information (one-to-one connections); (6) spreading a single bit (actually two bits due to the dual-rail format) of information to several spatial locations (one-to-many connection); (7) combining several bits of information into a single spatial location (many-to-one connection); and (8) dynamic routing of information (e.g., routing contents of register R to selection unit, or output unit, or response unit depending on the algorithm). The optical components required to accomplish the above operations can be divided into (1) logic elements, (2) storage elements, and (3) information transfer elements (or interconnects).

For optical logic and storage, many approaches are being investigated. One approach is the adaptation of the spatial light modulator (SLM) technology to optical logic[10]. Another approach for realizing optical components capable of performing logic, is to optimize the device from the beginning for digital operations. The recent emergence of the quantum-well self-electrooptic effect device (SEED) and its derivatives (S-SEED, T-SEED, D-SEED) is one such a product[11]. The SEED devices can be used to realize both logic operations such as NOR, OR, AND, NAND, etc. as well as for storage such as S-R latches[11]. Optical resonators are another family under this approach intended for optical logic[12]. Two similar bistable devices, etalons, and interference filters both based on the Fabry-Perot resonator are being actively pursued[13, 12]. All data movements and information transfer in OCAPP are space-invariant which may render their implementation easier. Classical optical components such as lenses, mirrors, beam splitters, holographic deflectors, and delay elements are most likely to be used for this purpose[14]. In addition, halfwave plates, shutters, and masks may be used for dynamic routing.

4.2 A Modular Implementation of OCAPP

In this paper, we present a modest design example of OCAPP, using existing optical hardware in order to highlight the potential implementation issues of a practicable realization. The implementation of this first version will make use of the SEED device operating as a NOR gate for optical logic, and of the S-SEED device operating as a S-R latch for storage[11]. The NOR gate is preferable to any other form of thresholding nonlinearity because it only requires distinguishing between the state where no light comes in and the state where light come in. Thus the NOR gate requires an SNR better than one only. In addition, a NOR function constitutes a complete logic set capable of implementing any boolean or arithmetic function[15]. The family of SEED devices seem to be easy to use, capable of high speed, low energy operation, and can be fabricated in 2-D format. Space-invariant optical interconnects, dynamic masking components, and beam spreading and combining devices are assumed for data routing[16].

The S-SEED device has two inputs, S, R, and two outputs Q and $\bar{Q}$. The state of the device is set by a pair of unequal signal beams labeled S (for setting the output $Q = 1$, $\bar{Q} = 0$) and R (for resetting the output $Q = 0$, $\bar{Q} = 1$). The device is set (Q = 1) when the power incident on the S input is much higher than the power incident on the R input. The state of the device is read by applying two equal-power (clock signal) beams to both inputs. During the setting of the device, the clock beams must be low, compared to the signal beams. The device holds its state when no clock signal is incident.. Thus the device can operate as a latch. Moreover, during the application of the clock signal (the reading process) the state of the device is unaltered. As described earlier, the optical processor can be constructed from several units: the selection unit, the match/compare unit, the response unit, the output unit, and the control unit. In what follows, we describe the optical implementation (architectural rather than experimental setup) of each of these units. Moreover, the details in the routing and imaging paths such as lenses, holographic elements, masks, beam splitters, and polarizers have been omitted in this version to assist the reader's conceptual understanding of these configurations.

4.3 The Optical Selection Unit

The optical selection unit of Fig.4 is composed of a storage array which consists of a 2-D $n \times m$ array of clocked S-SEED devices (each entry in the array at position i, j has two incoming bits S, R and two outgoing bits w_{ij}, $\bar{w}_{ij}$), a $n \times 1$ word register A which serves at setting and resetting data words in the storage array, a $1 \times (m + 1))$ bit-slice loading register B for loading a single bit-slice of the storage array. The first bit B_0 and its complement $\bar{B}_0$ are called set-E and reset-E respectively, since they are used for setting and resetting the $n \times 1$ enable register E which is used for the matching process (to be explained below). Memory words are disabled through the $n \times 1$ NOR gate array, representing the D register. The D register can be loaded from R, G or L registers. In addition, the E register can also be reset from the priority register P of the response unit (to be explained below).

A. Writing a Word/Bit-Slice into the Storage Array:

The storage array is assumed to be loaded in parallel at the beginning of the program. During program execution, the contents of the storage array can be altered by the use of the A and the B registers. To write a word in the storage array, say at word position i, the word is first written in the flip-flops of the B register. In the next clock cycle, the clock signals of B bits are pulsed high, and the contents of the B register is spread out vertically such that each bit B_j impinges on the set ports of the j-th column of the storage array. Next, bit A_i of A (corresponding to word position i) is pulsed high and spread out horizontally such that it impinges on the set/reset ports of the i-th row of the storage array. A one bit is written in bit position w_{ij} of the storage array if and only if a high A_i and a high B_j coincide at the set port of bit w_{ij}. Similarly, a zero bit is written in bit position w_{ij}, if and only if a high A_i and a high $\bar{B}_j$ coincide at the reset port of w_{ij}. This of course assumes that the set/reset thresholds of the S-SEED devices are so designed. Similar operations take place for writing a bit-slice in the j-th column of the storage array, with the exception of interchanging the roles of the B and A registers.

B. Enabling/Disabling Memory Words:

By enabling a memory word W_i, it is meant including it in the matching process. Similarly, by disabling it, it is meant excluding it from further matching operations. To allow a memory word W_i in participating in the matching process, its corresponding bit E_i in the E register must be set high. Similarly, to disallow a memory word w_i from participating in the matching process, its corresponding bit E_i in the E register must be made low. To enable/disable the entire memory words, the set-E/reset-E bits ($B_0/\bar{B}_0$) are spread out vertically and broadcast to all the set/reset ports of E. To selectively disable memory words whose R, or G or L bits are not asserted (R= 0, or G = 0 or L = 0), requires the routing of the appropriate register (R, G, or L) to the NOR gate array D. The output of D (which represents the complement of the routed register) is imaged onto the reset ports of register E. For example, to disable memory words whose R bits are not asserted (R = 0) from further matching operations, first contents of R is routed to D, which in turn image

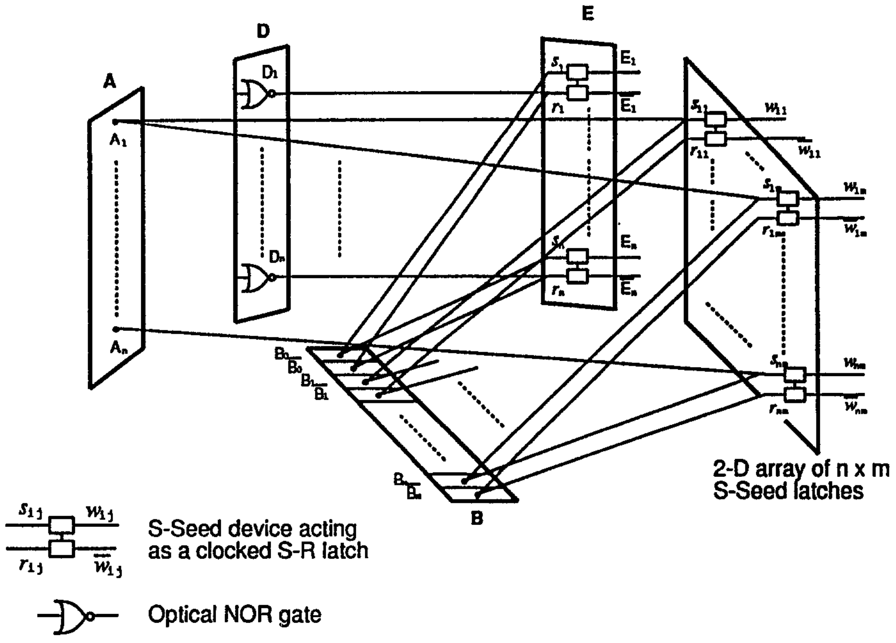


Figure 4 : Optical Implementation of the selection unit.

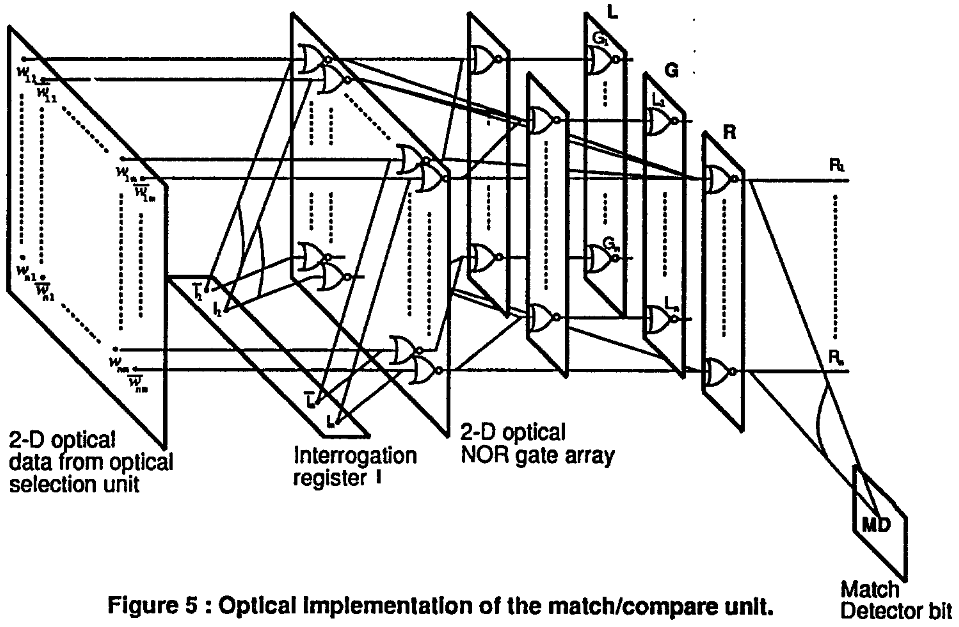


Figure 5 : Optical Implementation of the match/compare unit.

the complement of R onto the reset ports of E. Thus a low bit R_i of the R register will disable the i-th bit of the E register, which in turn disables memory word W_i from participating in further comparisons. The role of the E register in the match operation is explained next.

4.4 The Optical Match/Compare Unit

This unit performs exact match, and magnitude comparison searches between the interrogation register I and words of the storage array. As shown in Fig.5, It contains several SEED arrays operating as NOR gate arrays, and three registers, namely the response register, R, the greater than register G, and the less than register L. Parallel comparison takes place between memory words emanating from the storage array and the interrogation register I. A match at bit w_{ij} is detected by an exclusive-and principle as indicated in Eq.1. For that, register I needs to be spread out vertically so that each bit I_j impinges on one port of the NOR gates of the j-th column of the array, while data bits w_{ij} impinge on the second port of the NOR gates of the same j-th column. Matches between I_j and w_{ij} are reported in bit R_i of the R register. Otherwise, the G and L registers indicate the relative magnitude. The contents of R , G, and L are routed to the response unit as well as fed back to the selection unit.

As stated above, the enable register E determines whether or not a memory word participates in the matching process. Thus, the word match condition of Eq.2 is rewritten as follows:

$$R_i = [(I_{m+1} \wedge E_i) \vee (\bar{I}_{m+1} \wedge \bar{E}_i)] \wedge [(I_m \wedge w_{im}) \vee (\bar{I}_m \wedge \bar{w}_{im})] \wedge, \ldots, \wedge [(I_1 \wedge w_{i1}) \vee (\bar{I}_1 \wedge \bar{w}_{i1})] \quad (3)$$

where bit I_{m+1} is set to one ($\bar{I}_{m+1} = 0$) during a match operation. It can be seen from Eq.3 that a memory word W_i will participate in the match process if and only if its enable bit E_i is set to one. The R register bits are logically ORed to form the Match Detector (MD) bit. The MD flip-flop indicates if there is any match between I and memory words.

The optical match/compare unit of Fig.5 consists of a single interrogation register I, and therefore allows comparison of one search argument with the words stored in the storage array. However, using the multi-dimensionality of optical systems, this unit can be extended to perform multiple search operations in a single step. That is, several search arguments are compared simultaneously with the words of the storage array. An extended MIMD match/compare unit would have a $k \times m$ two-dimensional array of k search arguments, a two-dimensional storage array of n words each m bits long, and a $m \times k$ two-dimensional response array as shown in Fig.6. Each response register R_l ($l = 1, \ldots, k$) would indicate the match between interrogation register I_l ($l = 1, \ldots, k$) and the words of the storage array. The two-dimensional match operation can be thought of as an optical binary matrix-matrix multiplication which can be implemented using several optical techniques [17].

4.5 The Optical Response Unit

The response unit, contains a combinational priority circuit, and a priority register P for indicating the first matching word in memory (It may also contain few scratchpad registers for temporary storage). The priority circuit allows only the first responder (the first memory word W_i whose R_i is one) to pass to the priority register P. The priority circuit can be implemented using several stages of the NOR gate arrays in the form of a binary tree with space-invariant interconnections between them[18]. Contents of the P register are routed to the output unit, and also fed back to the selection unit.

4.6 The Optical Output Unit

The output unit outputs memory word whose corresponding bit in the priority register P, R, G or L is set to one (Fig.7). These latter registers are routed to a $n \times 1$ NOR gate array, denoted by N in Fig.7, whose sole purpose is to invert their values. Each bit N_i of N is logically NORed with memory word W_i using a 2-D NOR gate array. Next, each column of the NOR gate array is logically ORed to form output bit O_i of the output register. This latter could be a photosensitive device which only detects the presence of light and outputs electrical signals, or a 1-D array of SEED devices acting as OR gates, and outputting optical signals. It should be noted that parallel readout of selected memory words is also achievable by replacing P with a 2-D output device and eliminating the OR function.

4.7 The Control Unit

OCAPP is under the control of a memory control unit which comprises a local memory for storing programs and a program sequencer for executing instructions that control the optical hardware such as the S-R latches, the NOR gate arrays, the routing shutters, and splitters, etc. The instruction set is composed of conventional assignment and conditional statements, and additional instructions required to implement associative parallel processing. This includes data movement between units, comparison operations, memory loading and unloading, monitoring the MD bit, etc. These additional instructions are very few in nature and are derived from the required fundamental operations described above. It should be noted that application programs for OCAPP can be written in conventional high-level languages such as Pascal or C, with few calls to external procedures which support parallel associative processing.

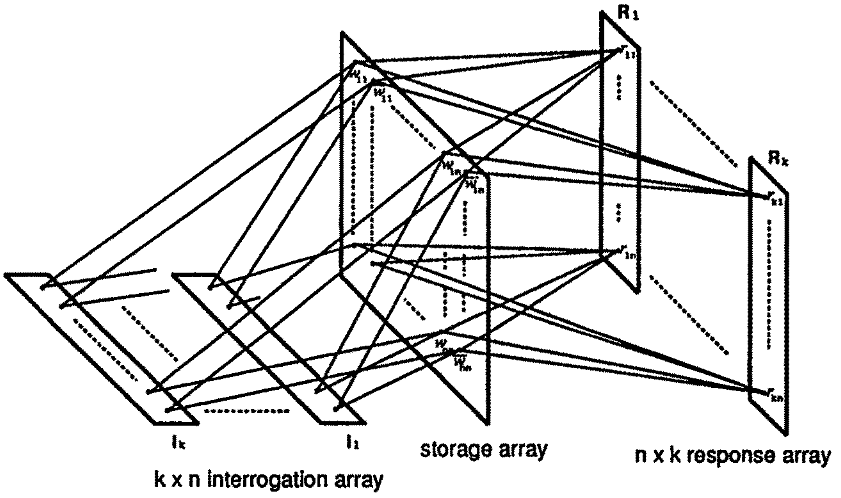
4.8 Estimated Execution time

An exact performance analysis of the proposed OCAPP including cost and power budget breakdown is currently not feasible due to the lack of optical S-R latches and thresholding devices with reasonable size (e.g., 500×500 gates), low operating energy, and fast response time. However, major efforts are being pursued[10, 11] in developing these devices in larger sizes. These efforts will soon culminate in the required devices and components for OCAPP as well as for other digital optical computing models.

We therefore try to theoretically estimate the execution time in terms of gate delay of the various basic search and arithmetic algorithms presented. This time does not include memory loading and unloading. We assume that the response times of the S-R latches (S-SEED) and that of the NOR gate arrays (SEED) are comparable and are both equal to T_s, and T_p is light propagation time through the processing loop (from the selection unit and back). It is assumed that the reading of memory words from the storage array and the I register is done at the same time, and takes one gate delay T_s; enabling/disabling of memory words is achieved in one gate delay; testing of the MD bit takes one gate delay, and the priority circuit takes $\log_2 n$ gate delays, where n is the number of words participating in the matching process.

Table 2 presents the estimated execution time of the algorithms presented. It can be seen that the execution time in equivalence search is a constant factor, and independent of the number of words in memory. The time in threshold search, double limits search, adjacency search, and extrema search, ordered retrieval (minimum time only) is proportional to the precision (number of bits) of the operands, and is independent of the number of words involved in the operation.

Note that the availability of the Match Detector (MD) bit provides major speed improvements to the above algorithms, in that certain conditions to terminate the computation as early as possible can be easily detected. Take for example, the threshold search algorithm. After the first comparison operation, if there are no words that match the comparand at that bit position (a condition that can be easily be detected by checking the MD bit), then all the words have been decided on in only $4T_s$ delay time, and the result is obtained in a much shorter time. The same considerations take place for double limits search, and adjacency search.



Register R_i indicates the match/mismatch of memory words with interrogation registers I_i(for i=1 to k).

Figure 6 : Optical implementation of a 2-D match/compare unit : the interrogation as well as the response registers of figure 5 are replaced by two-dimensional arrays of search arguments and response registers respectively.

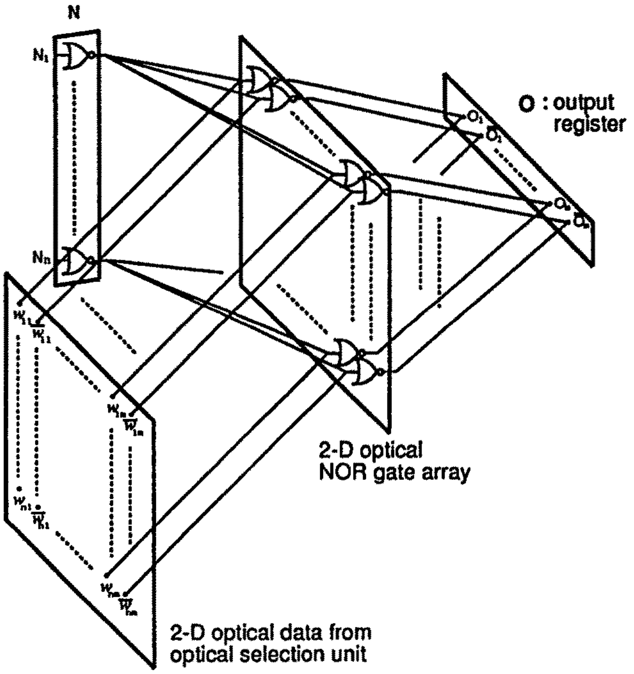


Figure 7 : Optical implementation of the output unit.

Table 2: Estimated execution time of the parallel algorithms on OCAPP

Search Algorithm	Minimun Execution Time	Maximum Execution Time
Equivalence Search	$3T_s$	$3T_s$
Threshold Search	$4T_s$	$(5T_s + T_p) \times m$
Minimum Search	$(6T_s + T_p) \times m$	$(6T_s + T_p) \times m$
Maximum Search	$(6T_s + T_p) \times m$	$(6T_s + T_p) \times m$
Double Limits Search	$10T_s + 2T_p$	$T_s + T_p + 2(5T_s + T_p) \times m$
Adjacency Search	$5T_s + T_p + (6T_s + T_p) \times m$	$T_s + T_p + (11T_s + 2T_p) \times m$
Ordered Retrieval Search	$((6T_s + T_p) \times m + T_p + 3T_s)$	$((6T_s + T_p) \times m + (\log_2 n)T_s + T_p + 3T_s) \times n$

The parameters m and n in the above table represent the word length and the number of operands respectively.

5 Conclusions

CAM-based processing has been argued to be the natural solution for non-numerical information processing applications. Unfortunately, the implementation requirements of these architectures using conventional electronic technology have been very cost prohibitive. This paper presented the principles and initial design concepts of an CAM-based parallel processing architecture that matches well with optics advantages, and therefore is highly amenable to optical implementation. The architecture relies heavily on the use of space-invariant interconnections, optical signal broadcasting and funneling (combining), and the simultaneous application of the same operation to many data points (SIMD mode of computing). The motivations behind this is the ease with which these operations can be realized with optics. A representative set of search algorithms have been presented to show the use and merits of the architecture. These algorithms are key components which occur in large computing tasks. It is important to note that these fundamental search algorithms are implemented on the optical architecture with an execution time independent of the problem size (the number of words to be processed). This indicates that the architecture would be best suited to applications where the number of data sets to be operated on is high. Some of the applications being investigated are: (1) real-time information retrieval, (2) database management, (3) knowledge-base and expert system implementation, and (4) list and string processing.

We presented a preliminary and simple version of an optical implementation of OCAPP. This version is only meant to show the feasibility of the architecture with existing optical devices. No optimization attempts were made. Nevertheless, this preliminary version reveals several key design issues that will determine the physical realization of such an optical architecture. Even if we assume the availability of optical nonlinear devices (latches, and NOR gates) in large sizes, the effective memory size will be critically determined by the beam spreading/combining optics, the contrast ratio and the fan-in/fan-out factors of the logic elements to be used. These practical implementations issues will be fully detailed in a follow-up paper.

References

[1] K. Hwang and D. Degroot, *Parallel Processing for Supercomputers and Artificial Intelligence*, McGraw-Hill, New York, 1988.

[2] T. Kohonen, *Content-addressable memories*, Springer-Verlag, 1980.

[3] A. A. Sawchuk and T. C. Stand, "Digital optical computing," *Proceedings of The IEEE*, vol. 72, no. 7, pp. 758–779, July 1984.

[4] W. T. Cathey, K. Wagner, and W. J. Miceli, "Digital computing with optics," *Proceedgins of the IEEE*, vol. 77, pp. 1558 - 1572, Oct. 1989.

[5] A. Louri, "A parallel architecture and algorithms for optical computing," *Optics Communications*, vol. 72, no. 1, pp. 27 - 37, July 1, 1989.

[6] A. Louri, "3-D optical architecture and data-parallel algorithms for massively parallel computing," *IEEE MICRO*, April 1991.

[7] B. K. Jenkins, P. Chavel, R. Forchheimer, A. A. Sawchuk, and T. C. Strand, "Architectural implications of a digital optical processor," *Applied Optics*, vol. 23, no. 19, , October 1984.

[8] J. W. Goodman, F. J. Leonberger, S. Y. Kung, and R. A. Athale, "Optical interconnections for VLSI systems," *Proceedings of the IEEE*, vol. 72, no. 7, pp. 850–866, July 1984.

[9] P. B. Berra, A. Ghafoor, M. Guizani, S. J. Marcinkowski, and P. A. Mitkas, "Optics and supercomputing," *Proceedings of the IEEE*, vol. 77, pp. 1797 - 1815, Dec. 1989.

[10] J. A. Neff, R. A. Athale, and S. H. Lee, "Two-dimensional spatial light modulators: a tutorial," *Proceedings of the IEEE*, vol. 78, pp. 836 - 855, May 1990.

[11] A. L. Lentine, H. S. Hinton, D. A. B. Miller, J. E. Henry, J. E. Cunningham, and L. M. F. Chirovsky, "Symmetric self-electrooptic effect device: optical set-reset latch, differential logic gate, and differential modulator/detector," *IEEE J. of Quantum Electron.*, vol. 25, pp. 1928 - 1936, Aug. 1989.

[12] J. L. Jewell, M. C. Rushford, and H. M. Gibbs, "Use of a single non-linear Fabry-Perot etalon as optical logic gate," *Appl. Phys. Lett.*, vol. 44, pp. 172 - 174, Jan. 1984.

[13] S. D. Smith, J. G. H. Mathew, M. R. Taghizadeth, A. C. Walker, B. S. Wherret, and A. Hendry, "Room temprature, visible wavelength optical bistability in ZnSe interference filters," *Optics Communications*, vol. 51, pp. 357 - 362, Oct. 1984.

[14] A. W. Lohmann, "What classical optics can do for the digital optical computer," *Applied Optics*, vol. 25, no. 10, pp. 1543 - 1549, 15 May 1986.

[15] A. Louri and K. Hwang, "A bit-plane architecture for optical computing with 2-d symbolic substitution algorithms," In *Proc. 15th Int'l. Symp. on Computer Arch.*, Honolulu, Hawaii, May 30 - June 4, 1988.

[16] K. Hwang and A. Louri, "Optical multiplication and division using modifed signed-digit symbolic substitution," *Optical Engineering, Special issue on Optical Computing*, vol. 28, no. 4, pp. 364 - 373, April 1989.

[17] R. A. Athale, "Optical matrix processors," *In Proc. SPIE, Optical and Hybrid Computing*, vol. 634, pp. 96 - 111, 1986.

[18] C. C. Foster, "Determination of priority in associative memories," *IEEE Transactions on Computers*, vol. C-17, pp. 788 - 789, Aug. 1968.