Design of an optical content-addressable parallel processor for expert systems

Ahmed Louri and Jongwhoa Na

The slow execution speed of current rule-based systems (RBS's) has restricted their application areas. To improve the speed of RBS's, researchers have proposed various electronic multiprocessor systems as well as optical systems. However, the electronic systems still suffer in performance from the large amount of required time-consuming pattern-matching and comparison operations at the core of RBS's. And optical systems do not fully exploit the available parallelism in RBS's. We propose an optical content-addressable parallel processor for expert systems. The processor executes the three basic RBS operations, match, select, and act, in a highly parallel fashion. Additionally, it extracts and exploits all possible parallelism in a RBS. Distinctive features of the proposed system include the following: (1) two-dimensional representation of data (knowledge) and control information to exploit the parallelism of optics in the three RBS units; (2) capability of processing general-domain knowledge expressed in terms of variables, numbers, symbols, and comparison operators such as greater than and less than; (3) the parallel optical match unit, which performs the two-dimensional optical pattern matching and comparison operations; (4) a novel conflict-resolution algorithm to resolve conflicts in a single step within the optical select unit. The three units and the general-knowledge representation scheme are designed to make the optical content-addressable parallel processor for expert systems suitable for any high-speed general-purpose RBS.

Key words: Optical content-addressable parallel processor, rule-based system, inference engine, optical parallel conflict resolution.

1. Introduction

Rule-based systems (RBS's) are one of the problem-solving methodologies that have been developed by artificial-intelligence researchers.[1] RBS's have a vast potential in several application areas because of the modularity, maintainability, and expressibility of the knowledge base as well as the simplicity of control. The RBS types include classification, selection, diagnosis, design, planning, and interpretation systems for such problems as computer-aided design, medicine, configuration tasks, manufacturing, and oil exploration.[2–4] In spite of these large potential application areas, RBS's have not been used as widely as conventional problem-solving methods that use a programming language such as C. One major reason is the slow execution speed of the underlying architectures implementing the RBS. The problem stems from the many pattern-matching and comparison operations necessary to solve a given query in a RBS.

To overcome this fundamental speed problem, researchers have developed new algorithms and new architectures tailored for RBS implementations. The new algorithms include the TREAT optimizing compiler for sequential RBS's,[5] the PSM-E parallel compiler for RBS's,[6] and the SWARM parallel-programming environment.[7] The new architectures include shared memory and message-passing multiprocessor systems such as the DADO multiprocessor,[8] the NON-VON multiprocessor,[9] and data-flow computers such as the data-driven parallel-production system.[10] Although these proposed systems incorporate new algorithms and new hardware designed to improve performance over that of sequential RBS's, they do not show any significant performance increase at present owing to the communication overhead and synchronization problems.[11,12]

Optics has been introduced as an alternative to improve the speed of RBS's because of the ability to represent knowledge in two-dimensional (2-D) space and because of natural implementation of the parallel pattern-matching and comparison operations.[12–15]

The authors are with the Department of Electrical and Computer Engineering, University of Arizona, Tucson, Arizona 85721.

Received 21 March 1994; revised manuscript received 12 September 1994.

0003-6935/95/235053-11$06.00/0.

Previously proposed optical expert systems include a matched-filter inference engine using a classical VanderLugt matched filter[14] and an optical expert system based on an optical vector–matrix multiplier.[15] However, these systems do not fully utilize all the available parallelism in a RBS, particularly rule-level parallelism in which more than one rule is fired at a time. Recently the authors introduced a new optical system called the electro-optical rule-based system (EORBS) for the parallel implementation of RBS's.[12] Although the EORBS fully exploits the available parallelism in RBS's, the system still lacks generality.

In this paper we extend the concept of the optical content-addressable parallel processor[16] (OCAPP) to a novel architecture designed specifically for parallel EORB's, known as the optical content-addressable parallel processor for expert systems (OCAPP-ES). The OCAPP-ES executes the three basic RBS operations (match, select, and act) in a highly parallel fashion. Additionally, it extracts and exploits all possible parallelism in a RBS. Distinctive features of the proposed system include the following: (1) 2-D representation of data (knowledge) and control information; (2) capability of processing general-domain knowledge expressed in terms of variables, numbers, symbols, and comparison operators such as greater than and less than; (3) the parallel optical match unit, which performs the 2-D optical pattern matching and comparison operations; (4) a novel conflict-resolution algorithm to resolve conflicts in a single step within the optical select unit. The three units and the general-knowledge representation scheme are designed to make OCAPP-ES suitable for any high-speed general-purpose RBS.

2. Background

An expert system can be defined as an intelligent system that can mimic some part of human intelligence. As shown in Fig. 1, an expert system is composed of (1) a RBS and knowledge-acquisition facility, (2) an explanation facility, and (3) a user interface.[1] The RBS performs inferencing using the knowledge base and the inference engine. The knowledge-acquisition facility and the user interface act as an interface unit between the RBS and the user. The explanation facility explains to the user the way results have been obtained. The knowledge base is composed of rules representing universal knowledge and facts representing current knowledge. A rule conjoins condition elements C_i and action elements A_i with the following format:

$$\overbrace{\text{if } (C_1 \wedge C_2 \wedge C_3 \ldots \wedge C_n)}^{\text{condition elements}} \Rightarrow \overbrace{\text{then } (A_1, A_2, \ldots, A_m)}^{\text{action elements}},$$

where $\wedge$ is a logical AND operator. The condition elements and the action elements are represented by facts, which are the basic units of knowledge in a RBS.

The inference engine uses the modus ponens theorem as an underlying principle. The modus ponens theorem states that, if there is an axiom of the form $E_1 \Rightarrow E_2$ and there is another axiom of the form E_1, then E_2 logically follows.[17] The inference engine applies modus ponens as follows:

$$\overbrace{(A \text{ is true})}^{\text{current fact}} \wedge \overbrace{(\text{if } A \text{ then } B)}^{\text{rule}} \Rightarrow \overbrace{(B \text{ is true})}^{\text{inferred fact}}.$$

Thus with known facts and rules the inference engine can produce new facts in either a forward-chaining system or a backward-chaining system or both.[17] In the forward-chaining system the system tries to find final (goal) states by comparing the known facts, which describe the initial states, with the condition-specifying if parts of the rules. In the backward-chaining system the system tries to find initial hypotheses by comparing the known facts, which now describe the final (goal) states, with the action-specifying then parts of the rules.

The basic operations of a RBS are as follows:

Match. For each rule, determine whether the condition part of the rule matches the current facts. If a rule satisfies the condition part, the rule is added to the conflict set. Otherwise, the rule is discarded. A conflict set is a set of triggered rules that have satisfied condition parts.

Select. If the conflict set is empty, the inference engine stops inferencing and reports the failure to the user. If the conflict set is not empty and contains more than one rule, the inference engine selects one rule from the conflict set by applying a conflict-resolution strategy such as rule ordering.[18]

Act. The inference engine fires the selected rule by executing its action. Because the current facts are changed by the fired rule, it is important to check whether the changes agree with predefined goals. If the goals are satisfied, the inference engine stops inferencing and reports results to the user. Otherwise, the inference engine must continue until the goal is satisfied or there are no more matching rules.

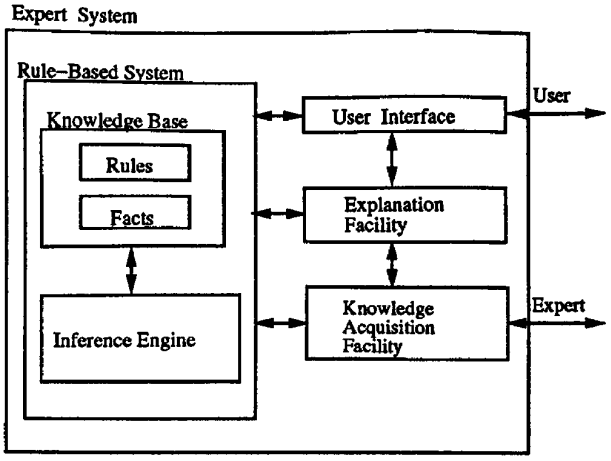

Fig. 1. Logical block diagram of an expert system.

In the following we discuss knowledge representation and operation of the OCAPP-ES.

3. Overview of the Optical Content-Addressable Parallel Processor for Expert Systems

The main objective of the OCAPP-ES is to exploit the maximum possible parallelism in a RBS. Because optical devices are two dimensional in nature, 2-D data can be processed simultaneously with optics. To exploit the parallelism available in optics fully, we represent knowledge-base and control information in a 2-D optical plane. In what follows we describe the OCAPP-ES and a knowledge-representation scheme for OCAPP-ES.

A. Description of the Optical Content-Addressable Parallel Processor for Expert Systems

Figure 2 shows a block diagram of the OCAPP-ES. The OCAPP-ES consists of an optical subsystem and an electronic subsystem. The optical subsystem is intended to implement the most time-consuming operations of RBS, namely, pattern-matching and comparison operations, and the electronic subsystem implements the flexible control of RBS's. The two subsystems complement each other. The optical subsystem consists of an optical match unit and an optical select unit. The optical match unit, utilizing an OCAPP, compares in a single step all the input facts to the input conditions of the rules and outputs a list of selected rules. The optical select unit then resolves conflicts among the selected rules and sends a list of conflict-free triggered rules back to the electronic subsystem. The electronic subsystem consists of a detector controller, an electronic act unit, and a spatial-light-modulator (SLM) controller. The detector controller converts the optical signal from the optical select unit into an electronic format. The act unit then executes the action elements of the triggered rules and sends the execution results to both the front-end computer and the SLM controller. The SLM controller is used to convert the electronic information into an optical signal. The front-end computer determines whether the desired solution has been obtained by checking the newly inferred facts from the OCAPP-ES. If the desired state is reached, the front-end computer stops further operation of the OCAPP-ES and returns control to the user.

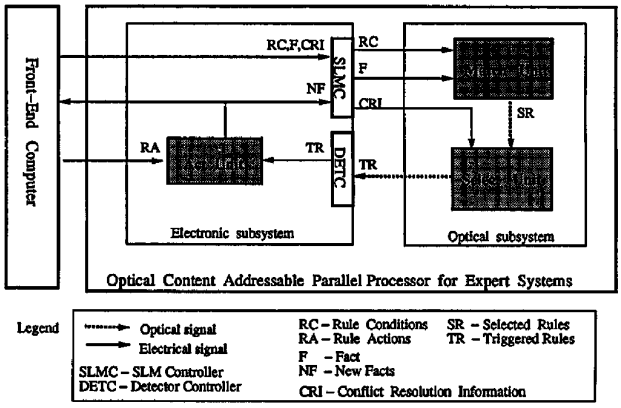

Fig. 2. Block diagram of the OCAPP-ES.

B. Optical Knowledge Representation

For the optical subsystem of the OCAPP-ES, facts are represented in a one-dimensional (1-D) fact vector (FV) and the condition parts of rules are represented in a 2-D plane called the condition template (CT), as shown in Fig. 3. The CT and the FV are used as inputs to the optical match unit. As an illustration of knowledge representation and the operation of the OCAPP-ES throughout this paper, consider the following going-to-the-theater example knowledge base:

```
/* 'Going to the theater' input rulebase */
Rule 1: If distance > 5 miles
        Then means = drive
Rule 2: If distance > 1 mile and time < 15min
        Then means = drive
Rule 3: If distance > 1 mile and time > 15min
        Then means = walk
Rule 4: If distance > 3 miles and time > 30min
        Then means = walk
Rule 5: If means = drive and location = down-
          town
        Then action = take_a_cab
Rule 6: If means = drive and location = suburb
        Then action = drive_your_car
Rule 7: If means = walk and weather = bad
        Then action = take_a_coat_and_walk
Rule 8: If means = walk and weather = good
        Then action = walk
```

This knowledge base decides how one can go to the theater under various circumstances. In this knowledge base the rule condition parts require three different logical operations: equality, greater than, and less than comparisons. It has been shown that the equality comparison can be easily done with optical devices performing an XOR operation.[19] However, optical implementation of the magnitude comparison operations such as greater than and less than turns out to be a difficult task. Although these comparison operations have been shown to be feasible for optical implementation, they require a fair number of optical logic devices,[16,20] which can dominate cost and propagation delay.

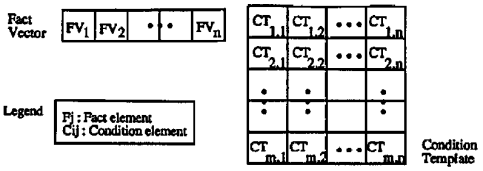

Fig. 3. Templates for optical knowledge representation: Each cell of the fact vector (FV) represents a fact variable, and the contents of the cell represent the value of the variable. In the condition template (CT) each row represents a rule and each column represents a condition element. Each fact variable, FV_j, is used as a condition-element variable $CT_{(i,j)}$, where i represents the rule number and j represents the FV index.

To reduce the number of required optical logic devices and the design complexity while enhancing system feasibility, we introduce a solution that can evaluate the greater-than and less-than operations with only the equality-checking optical hardware. In the proposed method, instead of performing a greater-than or less-than comparison with a specific number at the left-hand side of each condition element, we can perform a logically equivalent operation. As an illustration, consider the condition element (*distance* $>$ *1 mile*) of rule R1 of the going-to-the-theater example rule base. This condition element becomes true when the variable *distance* receives any number greater than 1. In other words the condition element compares whether the input value of the variable *distance* is in the range from 1 mile to ∞ miles. Thus by assigning a unique symbolic interval constant *distance_set_A* for the interval from 1 mile to ∞ miles and by performing the equality comparison with the symbolic interval constant *distance_set_A*, we can obtain the greater-than comparison result.

The above translation method can be generalized as follows: First, we assign an appropriate symbolic constant for both the condition element related to the greater-than or less-than comparison and the value of the fact element used as the condition element. Next, we perform an equality comparison instead of a greater-than or less-than comparison between the input value of the variable (left-hand side of a condition element) and the prescribed symbolic constant (right-hand side of a condition element). With the translation method the greater-than and less-than checking condition elements of the going-to-the-theater example rule base are translated into the condition elements with the symbolic interval constants and equality comparators, as shown in Table 1.

This method replaces greater-than and less-than comparison operations using exact values for a variable with equality comparison with a symbolic interval constant representing the corresponding interval. However, owing to the nature of expert systems (i.e., symbolic computation in the decision-making environment rather than numerical computation), the translation method will affect neither the overall performance of the system nor the generality of the proposed system. The number of symbolic interval constants created for a variable will be small considering that the major application domain of expert systems is in business and management[3] rather than numerical computation. Even if the number of created symbolic interval constants exceeds the capacity that one variable can represent, the front-end computer can easily create another variable to share the remaining symbolic interval constants. The translation method requires preprocessing in the host computer during the rule compile time. Hence this will not affect the execution time of the OCAPP-ES. On the other hand, the additionally created variable will take up one additional variable slot in the optical match/select unit. However, owing to the small number of original symbolic interval constants and the nonnumerical nature of expert systems, the additionally occupied variable slot should not cause any problem. This translation process can be regarded as a special case of fuzzy logic.[21,22]

Table 1. Translation of Condition Elements with Greater-Than or Less-Than Comparison Operators into Condition Elements with Corresponding Symbolic Constants

Input Condition Elements with Magnitude Comparison Operators ($>$, $<$)	Translated Condition Elements with Symbolic Interval Constant
distance $>$ 1 mile	distance = dist_set_A
distance $>$ 3 mile	distance = dist_set_B
distance $>$ 5 mile	distance = dist_set_C
time $<$ 15 min	time = time_set_A
time $>$ 15 min	time = time_set_B
time $>$ 30 min	time = time_set_C

From Table 1, the original going-to-the-theater example rule base is modified into the translated rule base with equality-checking elements only as follows:

```
/* Translated rulebase for the 'Going to the theater' example */
Rule 1:  If distance = distance_set_C
         Then means = drive
Rule 2:  If distance = distance_set_A and time =
           time_set_A
         Then means = drive
Rule 3:  If distance = distance_set_A and time =
           time_set_B
         Then means = walk
Rule 4:  If distance = distance_set_B and time =
           time_set_C
         Then means = walk
Rule 5:  If means = drive and location = down-
           town
         Then action = take_a_cab
Rule 6:  If means = drive and location = suburb
         Then action = drive_your_car
Rule 7:  If means = walk and weather = bad
         Then action = take_a_coat_and_walk
Rule 8:  If means = walk and weather = good
         Then action = walk
```

Once the translated rule base is ready, fact variables related to condition elements with a greater-than or less-than operator can be modified. For these variables the values of facts are translated into the corresponding symbolic constants instead of using numbers given by the user. For example, when the fact variable *distance* receives the number 2 as input, the translated value of the fact becomes *dist_set_A*.

When the knowledge base is ready, the electronic host computer modifies the translated knowledge base into the execution format for the optical inference engine by using templates, shown in Fig. 3. In Fig. 4 each cell of the CT and the FV represents the name of a variable. The cell contains the value of the variable. As an illustration, assume that we have a fact (*location* = *downtown*) and that it is assigned to

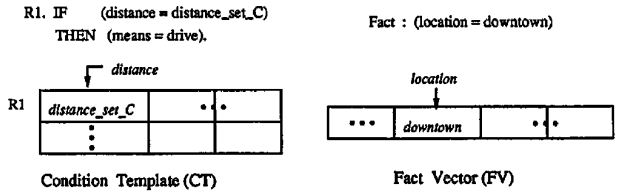

Fig. 4. Optical knowledge-representation example using a fact (*location* = *downtown*) for the FV and a condition element (*distance* = *dist_set_C*) of the rule R1 for the CT.

the *i*th cell of the FV, FV_i. Then FV_i represents the variable name *location*, and the content of FV_i becomes *downtown*. Next, assume that the condition element of rule R1 (*distance* = *distance_set_C*) is to be assigned to the *j*th column of the *i*th row of the CT. Then, cell $CT_{(i,j)}$ will represent the variable *distance*, and the contents of $CT_{(i,j)}$ will have the number *distance_set_C*.

Each cell of Fig. 4 is further divided into *n* pixels, creating an *n*-bit word-representation system. Figure 5 shows the 8-bit binary assignment scheme for numbers and symbols (including symbolic interval constants, symbolic variables, and values) used in the going-to-the-theater example. We create the numbers and symbols by distinguishing the most-significant bit, *b*7. If *b*7 is 1(0), the word represents a symbol (number). Thus a number in the range from $00000000_{(2)}$ to $01111111_{(2)}$ can be represented. We further divide the symbols into the symbolic variable, the symbolic value, and the symbolic interval constant by distinguishing the next-most-significant bit, *b*6. If *b*6 is 1(0), the word represents either a symbolic variable or a symbolic value (a symbolic interval constant). Then, we represent each unique symbol by assigning a unique binary number to the symbol, using the remaining bits.

Once the symbol and number representations are set, the host computer builds the FV and the CT, as shown in Fig. 6. Each fact is assigned to a unique cell of the FV and a unique column of the CT. Then each cell of the FV is filled with the current value of the facts known to the user. In this example, the

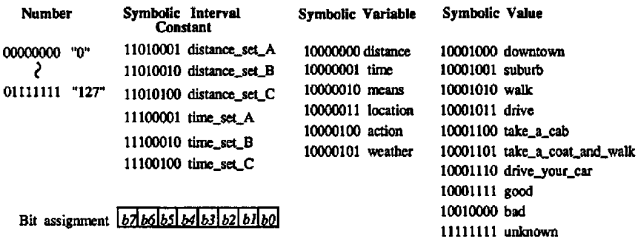

Fig. 5. Example of 8-bit binary value assignment for the variables, symbolic values, symbolic interval constants, and numbers for the going-to-the-theater example. Numbers are identified by the 0 in the most-significant bit (bit *b*7). Symbolic interval constants for intervals are identified by the 11 in bits *b*7 and *b*6. *b*5 and *b*4 are used to identify the name of symbolic interval constants for intervals, and the remaining bits are used to specify various intervals in a set. Symbolic variables and values are identified by the 10 in bits *b*7 and *b*6. The rest of the bits are used to identify various symbolic variables and values in this case.

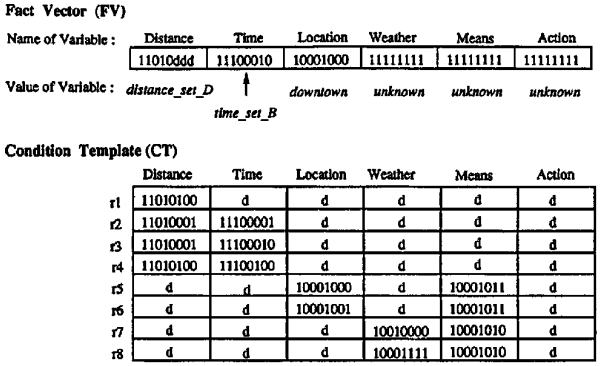

Fig. 6. FV and CT with data for the going-to-the-theater example. It is assumed that the input value of the fact *distance* from the user is 10, *time* is 20, *location* is *downtown*, and the rest are *unknown*. The host computer translates the interval evaluating variables into corresponding symbolic interval constants. For example, the input number 10 of the fact *distance* is translated into a symbolic interval constant, *distance_set_D (11010ddd), d:don't care. Distance_set_D* represents the union of the three symbolic interval constants *distance_set_A (11010001), distance_set_B (11010010),* and *distance_set_C (11010100),* as shown in Table 1. Likewise, the input number 20 of the fact *time* is translated into a symbolic interval constant, *time_set_B (11100010).* If a fact does not belong to the interval evaluating variables, the binary number for the symbol is written in the FV with the assignment of Fig. 4. We construct the CT by writing the value specified at the right-hand side of each condition element of the translated rule base at the cell specified by the variable of the left-hand side of each condition element. The *d* bits in the CT represent the variables that are not used in the rule.

currently known value of fact *distance* is 10, the fact *time* is 20, the fact *location* is *downtown*, and the rest are *unknown*. Because the variables *time* and *distance* belong to the interval evaluating variables, the host computer must assign symbolic constants for the values of these variables. For example, the value 20 of the variable *time* can be translated into *time_set_B (11100010)* because *time_set_B* becomes true for input value of *time* >15. On the other hand, the input number 10 of the variable *distance* will be translated into *distance_set_D (11010ddd),* where *d* represents *don't care.* The three *d* bits represent the symbolic interval constant, *distance_set_D,* as a union of the three sets, *distance_set_A (11010001), distance_set_B (11010010),* and *distance_set_C (11010100).* The reason why we use the *don't care* bit for a union of multiple symbolic interval constants is discussed later in this paper. The variable *location* keeps its original value, *downtown*.

In the CT the values described in the condition parts of the translated rule base are written to cells of the CT. For example, in the case of the condition element (*distance* = *distance_set_C*) of rule R1, *dist_set_C* is assigned to the cell corresponding to the variable *distance* of the first row of the CT. For a variable that is not used as a condition element of a rule, *d* is assigned to the corresponding cell. Next, we discuss the OCAPP-ES units.

C. Description of the Optical Match Unit

The optical match unit of the OCAPP-ES is designed around the original OCAPP design.[16] The OCAPP is an optical parallel data-base/knowledge-base processor based on an optical content-addressable memory. The system implements symbolic computing tasks such as searching, sorting, and information retrieval in a highly parallel fashion. The OCAPP is composed of a selection unit, a match/compare unit, a response unit, an output unit, and a control unit. The match/compare unit of an OCAPP is used as the optical match unit of the OCAPP-ES. A detailed explanation and implementation of each OCAPP unit and the algorithms implemented on the OCAPP are presented in Refs. 16 and 23.

The optical match unit compares an input FV of given facts with a CT of the condition parts of given rules to generate a selected-rule list. As shown in Fig. 7, the comparison is performed by a vector–matrix multiplier performing a 2-D XOR logic function followed by a masking operation.

Figure 8 explains how the selected-rule vector (SRV) is obtained for a given FV and CT. First, each bit of FV_i is vertically expanded to cover a column of the CT. The expanded 2-D FV is then XOR'ed with the CT to produce a 2-D intermediate XOR result plane. On the 2-D intermediate XOR result plane, matches between the FV and the CT form dark output pixels, whereas unmatched pixels create a bright output.

This intermediate XOR result plane is then imaged onto the mask. The latter blocks the contributions from the unused condition elements of a rule as well as the bits specified as *d* bits in the mask. Recall from the example that the *d* bits in the *distance_set_D (11010ddd)* are used to represent a union of the three symbolic interval constants *distance_set_A (11010001), distance_set_B (11010010),* and *distance_set_C (11010100).* Now, by blocking of the three *d* bits, the output *d* bits will become dark (matched). Thus if the *distance_set_D* is compared with any of the three symbolic interval constants, the three *d* bits will become dark (matched) for any input.

The XOR result after the mask is then logically OR'ed to produce a selected-rule vector (SRV). In the 2-D XOR result image, if there is at least one unmatched condition element in a rule (represented by a row), there should be some light in the row of that rule. On the other hand, if the condition elements consist of only either matched or *don't care* pixels, then the absence of light indicates the row is matched. Therefore in the SRV, bright pixels indicate unmatched rules and dark pixels represent matched rules. The match operation is performed in parallel with an execution time, independent of the number of matches to be performed.

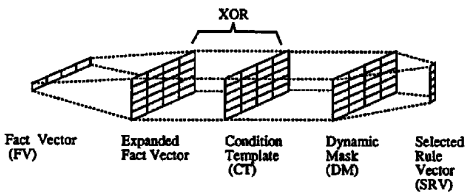

Fig. 7. Logical block diagram of the OCAPP-ES match unit: The FV is expanded vertically and XOR'ed with the CT. The 2-D XOR result is then masked by the DM to filter out unnecessary data. The XOR result after the DM is then OR'ed rowwise to produce a selected-rule column vector.

D. Description of the Optical Select Unit

Once the SRV is available, the optical select unit resolves conflicts among the selected rules and produces a triggered-rule vector (TRV), which represents all the rules that can be fired in parallel. In the SRV and the TRV each pixel represents a rule. Whereas the SRV represents candidate rules that can be fired, the TRV represents a list of rules to be fired. Instead of traditional conflict-resolution strategies that enable only one rule among the selected rules to be fired at a time, the optical select unit of the OCAPP-ES utilizes a new parallel conflict-resolution scheme to maximize performance by firing as many rules as possible.

The new parallel conflict-resolution scheme is based on a dependency analysis among the rules so that rules can be fired simultaneously without any undesirable side effects.[24] For optical implementation of this parallel conflict-resolution scheme, an algorithm creates a conflict-resolution control matrix (CRCM) for any given rule base. The CRCM performs parallel rule selection by taking a 1-D SRV as an input, controlling specific SRV rules according to the dependencies, and generating a 1-D output TRV. The test necessary for detecting dependency between rules is exhaustive, as each rule of the rule base must be tested against all the remaining rules. However, because these analyses can be done at compile time for a given rule base, the dependency testing should not incur any run-time overhead. For a detailed explanation of the proposed new algorithm and for an example of constructing a CRCM, please refer to Ref. 30.

Figure 9 shows the CRCM and an execution example of the optical select unit for the eight-rule going-to-the-theater example knowledge base. The figure illustrates how the 1×8 TRV $(1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$ is generated with the 8×1 $(1\ 0\ 1\ 0\ 0\ 0\ 0\ 0)^T$ SRV from the optical match unit and the 8×8 CRCM. First, the input selected-rule column vector is expanded horizontally so that each column of the CRCM can receive the SRV. Each CRCM pixel is configured such that the pixel will perform an XNOR (equivalence) operation between incoming input data and the control data specified in the CRCM pixel. If a control datum is set to *don't care,* the pixel should produce a 1 regardless of input data. Then the intermediate image, as shown in Fig. 9, is AND'ed columnwise to produce a row vector representing the desired TRV, which is $(1\ 0\ 0\ 0\ 0\ 0\ 0\ 0)$ in this example. Again, the conflict resolution (rule selection) is per-

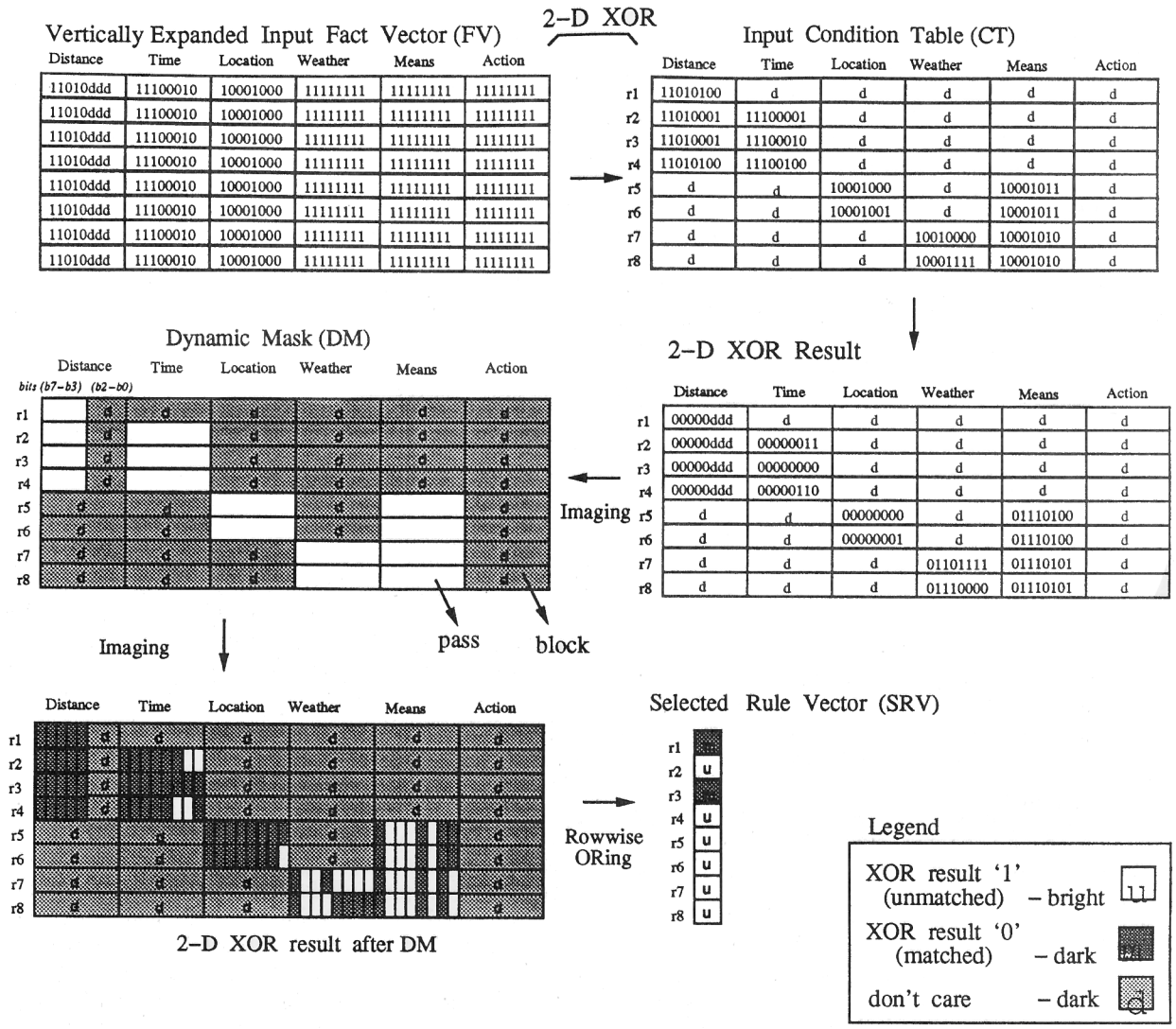

Fig. 8. Match operation result for the going-to-the-theater example. The input FV is expanded to cover the CT. The XOR logic function is then performed with the expanded FV and CT to produce a 2-D XOR result image. The XOR result is imaged onto the dynamic mask (DM), which blocks the d bits, $b2 - b0$, and the unused cells (condition elements) in the rule. Finally, the masked XOR result image is OR'ed rowwise to produce a selected-rule vector (SRV).

formed in parallel and is independent of the number of rules to be fired.

E. Description of the Act Unit

When the triggered-rule vector (TRV) becomes available, the front-end computer converts the optical TRV into an electronic form by means of the detector array. Then the electronic act unit executes the action part of the triggered rules and updates the changes caused by the rule-firing operation for the front-end computer as well as for the optical match unit. The act unit is implemented with electronics because of the following reasons: whereas the match and select operation requires computationally intensive operations (e.g., comparing each rule of the rule base to all other rules), the act operation requires simple assertion operations for only the triggered rules of the TRV. Also, the electronic act unit will ease the implementation of extra services such as explanations about the decisions being made. In addition, an optical act unit will add more hardware complexity while providing only simple operations that can be easily performed with electronics. In our example we have only one triggered rule, R1, which asserts the value of the action variable *means* to *drive*.

Next, we discuss the scalability problem in which the given knowledge-base (rules and facts) size exceeds the capacity of a given optical implementation. We solve this scalability problem in the RBS by taking advantage of the modular characteristic of the RBS.[18] This characteristic stems from the fact that the structure of human knowledge can be modeled in a modular and hierarchical structure. Thus if the size of the given knowledge base is greater than the size of the available hardware, the problem should be partitioned into a set of subproblems whose size fits the available hardware.

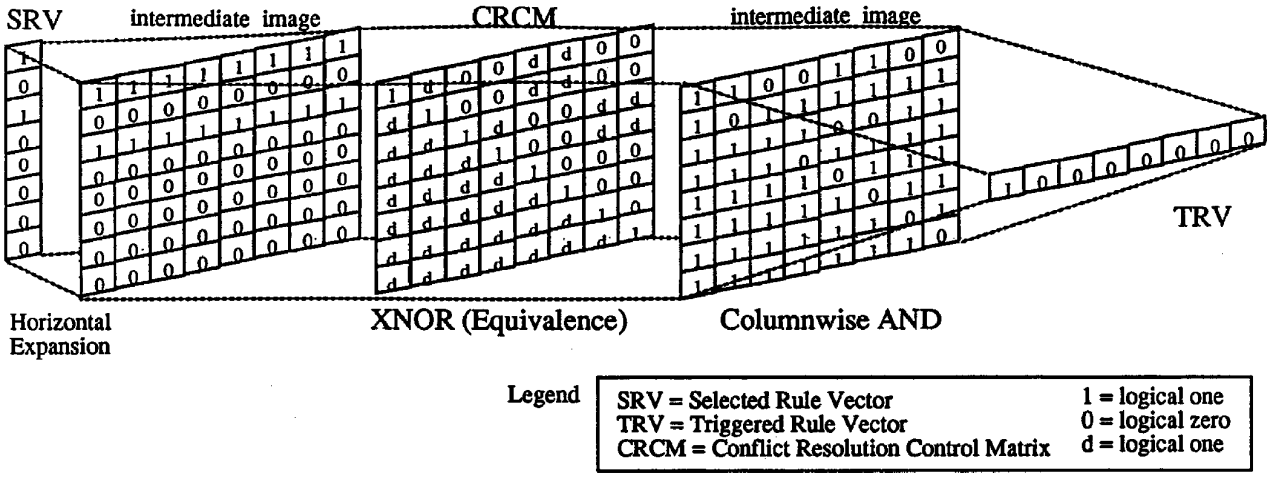

Fig. 9. 8 × 8 CRCM array for the eight-rule going-to-the-theater example.

4. Implementation of the Optical Content-Addressable Parallel Processor for Expert Systems

The OCAPP-ES consists of an optical match unit, an optical select unit, an electronic act unit, and 2-D optical sources and detectors for input–output interfacing with the front-end computer. A 1-D optical fact vector and a 2-D optical rule plane are created either by use of a 2-D laser diode array or modulation of a single laser with a 2-D spatial light modulator (SLM). The SLM can be a ferroelectric liquid-crystal SLM,[25] an electrically addressed microchannel SLM,[26] or a silicon lead-lanthanum-zirconate-titanate SLM.[27] Because an OCAPP-ES must also return results to the electronic host, an optical detector array is needed to convert optical signals into electronic ones. An OCAPP-ES also needs a 2-D optical logic gate array to perform the XOR logic function. The XOR logic functions are performed by a pair of liquid-crystal SLM's and by control of the polarization. Optical implementation of the match and select units constituting the OCAPP-ES and electrical implementation of the act unit are discussed in the following.

A. Implementation of the Optical Match Unit

The optical match unit consists of three SLM's (SLM1, SLM2, and SLM3), three cylindrical lenses (CL1, CL2, and CL3), and two polarizers (P1 and P2), as shown in Fig. 10. The three SLM's are pixellated, electrically addressed, ferroelectric liquid-crystal SLM's configured to rotate the incident light by 90° in bit positions containing a logical value of 1 and 0° for those containing a logical value of 0. SLM1, SLM2, and P1 of Fig. 10 are configured as a vector–matrix multiplier to perform 2-D XOR logic functions, as shown in Table 2. A uniform beam of vertically polarized light illuminates SLM1. The logical values of SLM1 are encoded such that vertically polarized light emanating from the device represents a 0 and horizontally polarized light represents a 1. The combinations of possible polarization rotations experienced by a beam passing through SLM1 and SLM2 are functionally equivalent to a Boolean XOR operation. Because CL1 and CL2 image a bit position of SLM1 onto a bit slice (column) of SLM2, the resulting 2-D data plane after SLM2 expresses the result of a bitwise matrix of XOR gates. Polarizer P1 then darkens any horizontally polarized light so that 1's are represented by the absence of light and the 0's continue to be represented by vertically polarized light.

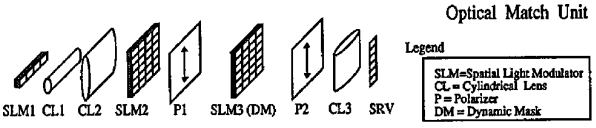

Fig. 10. Implementation of the optical match unit: The image of SLM1 is vertically expanded and XOR'ed with the data of SLM2 by means of cylindrical lenses CL1 and CL2. The XOR result image is then filtered with polarizer P1. The DM disables the transmission of unnecessary pixels. The intermediate image after the DM is then focused into a vertical line (SRV). Each pixel of the SRV represents a logical OR of all the pixels in a row of the image after the DM.

Table 2. Truth Table for the Optical XOR Logic Function Using SLM1 and SLM2 and Polarizer P1of Fig. 10[a]

SLM1 Input	SLM2 Input	XOR Result	After P1: Pass (↕)
0 (↕)	0 (No rotation)	0 (↕)	Bright (↕): unmatched
0 (↕)	1 (Rotate 90°)	1 (↔)	Dark: matched
1 (↔)	0 (No rotation)	1 (↔)	Dark: matched
1 (↔)	1 (Rotate 90°)	0 (↕)	Bright (↕): unmatched

[a]A logical value of 0 is assigned to a vertically polarized beam (↕), and a logical value of 1 is assigned to a horizontally polarized beam (↔). SLM's in the system are assumed to be electrically addressable liquid-crystal SLM's. In the SLM's the pixels with a logical value of 1 rotate the incoming light polarization by 90°, and the pixels with a logical value of 0 pass light without rotation. First, SLM1 is illuminated with a vertically polarized beam so that the pixels with a logical value of 0 have vertical polarization, and the pixels with a logical value of 1 have horizontal polarization. SLM2 then performs the XOR logic function by rotating the pixels with a logical value of 1 (0) by 90° (0°). Then, the pixels with horizontally polarized light (logical 1: matched) of the XOR result are blocked by use of polarizer P1, which passes vertically polarized light (logical 0: unmatched) only.

The vector–matrix multiplier is then followed by SLM3, which behaves as a dynamic mask (DM), shown in Fig. 10. The purpose of the DM is to block the transmission of the *don't care* pixels while passing the other pixels, as shown in Table 3. After the DM, the matched pixels and the *don't care* pixels are dark, and the unmatched pixels will remain bright. CL3 focuses the image after the DM into a vertical line, which represents the SRV. The SRV is routed to the optical select unit.

B. Implementation of the Optical Select Unit

Figure 11 shows the implementation of the optical select unit. The optical select unit consists of three SLM's (SLM1, SLM2, and SLM3), three cylindrical lenses (CL1, CL2, and CL3), two beam splitters (BS1 and BS2), and two polarizers (P1 and P2). In the optical select unit the three SLM's of Fig. 11 are configured to rotate the polarization of the incident light by 90° in bit positions containing a logical value of 0 and 0° for those containing a logical value of 1. SLM1 (an optically addressable SLM) and BS1 are used to convert the intensity-encoded SRV_i from the optical match unit into the polarization-encoded SRV_p, as shown in Table 4. The vertically polarized collimated beam going into BS1 is sent into the reflective side of SLM1 to modulate the incoming SRV_i, which simultaneously writes its value to the photoconductive side of SLM1. If the input pixel of SRV_i is bright (logical 0), SLM1 will rotate the polarization of the vertically polarized light incident upon it at the photoreflective side by 90° to produce a horizontal polarization. On the other hand, if the input is dark (logical 1), the output pixel will have a vertical polarization. Thus SRV_p is encoded such that a logical value of 0 (an unselected rule) is represented by horizontal polarization, whereas a logical value of 1 (a selected rule) is represented by vertical polarization.

Next, SLM2 and SLM3 of Fig. 11 work together to realize the CRCM. The logical values of SLM2 are encoded such that vertically polarized light emanating from the device represents a 1 and horizontally polarized light represents a 0, as shown in Table 5.

Table 3. Truth Table for the DM and P2 of Fig. 10[a]

After P1: Pass (↕)	DM Input	After P2: Pass (↕)
↕	0 (No rotation)	Bright (↕)
↕	1 (Rotate 90°)	Dark
dark	0 (No rotation)	Dark
dark	1 (Rotate 90°)	Dark

[a]The DM is assumed to be an electrically addressable liquid-crystal spatial light modulator (SLM), and ↕ represents a vertically polarized beam. In the table a logical value of 1 is assigned to the *don't care* pixels of the DM, and a logical value of 0 is assigned to the other pixels. In the DM the *don't care* pixels rotate the incoming light polarization by 90°, and the other pixels remain unchanged. With vertically polarized input light a vertical polarizer, P2, placed behind the DM selectively blocks the *don't care* pixels. After this unit, the matched pixels and the *don't care* pixels will be dark and the unmatched pixels will be bright.

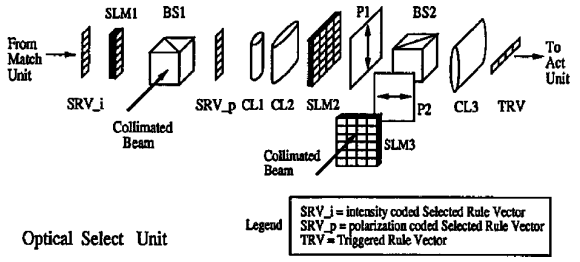

Fig. 11. Implementation of the optical select unit: The polarization-encoded selected-rule vector (SRV) is expanded and XOR'ed with the CRCM in SLM1. Then the two data planes from SLM2 and SLM3 are spatially added by the use of beam splitter BS2. The added image is then collimated with cylindrical lens CL3 to form a TRV.

The combinations of possible polarization rotations experienced by a beam passing through SLM2 are functionally equivalent to a Boolean XNOR operation. Because CL1 and CL2 image a bit position of SRV_p onto a bit slice (column) of SLM2, the resulting 2-D data plane after SLM2 expresses the result of a bitwise matrix of XNOR gates. Polarizer P1 then darkens any horizontally polarized light so that, after P1, 0's are represented by the absence of light and the 1's continue to be represented by vertically polarized light.

SLM3 provides light for the *don't care* pixels of the CRCM. With a vertically polarized collimated input beam, SLM3 controls the polarization angle of each pixel as follows: If the pixel corresponds to the *don't care* pixel, it will have 90° polarization rotation to have a horizontal polarization. On the other hand, if the pixel does not correspond to the *don't care* pixel, it will preserve the vertical polarization. Then, by use of P2, which passes only horizontally polarized light, only the *don't care* pixels will become bright. Finally, the two intermediate data planes from SLM2 and SLM3 are merged by the use of BS2 and focused into a horizontal line to form a TRV.

C. Implementation of the Act Unit

The act unit is implemented with software because of the reasons given in Subsection 3.E. One implementation method for the act unit is keeping a simple table that records the rule number, the name of the variable to be triggered, and its value. To perform

Table 4. Conversion of the Intensity-Encoded SRV_i to Polarization-Encoded SRV_p at SLM1 of Fig. 11[a]

Input	SRV_i (before SLM1)	SRV_p (after SLM1)
Logical 1	No light	↕
Logical 0	Light	↔

[a]Assuming a vertically polarized collimated beam is incident upon the photoreflective side of SLM1, the reflected light will be controlled by the intensity of the input pixel at the photoconductive side (SRV_i) of SLM1. If the input pixel of SRV_i is bright (logical 0), the SLM1 will rotate the polarization by 90° to produce a horizontal polarization. If the input is dark (logical 1), the pixel polarization at the photoreflective side of SLM1 remains vertical.

Table 5. Summary of XNOR Logic Function at SLM2 of the Optical Select Unit of Fig. 11[a]

SLM1 Output	SLM2 Setup	XNOR Result	After P1: Pass ↕
0 (↔)	0 (Rotate 90°)	1 (↕)	Bright: ↕
0 (↔)	1 (No rotation)	0 (↔)	Dark
1 (↕)	0 (Rotate 90°)	0 (↔)	Dark
1 (↕)	1 (No rotation)	1 (↕)	Bright: ↕

[a]The logic performed is similar to that of Table 2 except for the assignment of polarization for the logical values and for the polarization rotation angle control at the second SLM.

the act operation, the detector unit of the electrical subunit of the OCAPP-ES converts the optical TRV into an electrical TRV, which is a set of the triggered-rule numbers. Then executing the action parts of the rules will be a matter of accessing the table with the rule numbers as indices and of updating the values of the corresponding variables.

5. Theoretical Estimation of Execution Time and Number of Processed Rules per Second

In this section the execution time and the maximum number of rules that can be processed per second in the OCAPP-ES are estimated. In what follows we use the following terms and assumptions for estimating the execution speed:

- The dimension of available 2-D SLM's is $n \times n$.
- The response time of the SLM's and optical logic gate arrays is T_r.
- The setup time for a SLM of size $n \times n$ is T_{s2}.
- The setup time for a SLM of size n is T_{s1}.
- The detector response time is T_d.
- The propagation delay of optical passive devices such as lenses and mirrors is negligible compared with that of the SLM's.

A. Execution Time

In general the execution time of a system can be regarded as the sum of the system initialization time T_i and the system processing time T_p. T_i includes the SLM setup times in the optical match and select units. Although these two units have six SLM's in total, these SLM's can be accessed simultaneously: hence T_i becomes T_{s2}. In an OCAPP-ES, T_p can be further divided into the optical system processing time T_{op}, the conversion time from optical signal to electrical signal T_{oe}, the act operation processing time T_{ap}, and the conversion time from electrical signal to optical signal T_{eo}. T_{op} is equal to $5T_r$ because there are five SLM's in the major optical signal path in the OCAPP-ES. T_{oe} is the detector readout time T_d; T_{eo} is the $n \times n$ SLM modulation time T_{s2}. Thus T_p becomes $5T_r + T_d + T_{s2} + T_{\text{ap}}$. Therefore the overall execution time of the OCAPP-ES, T_e, becomes

$$T_e = T_i + T_p = T_{s2} + x(5T_r + T_d + T_{s2} + T_{\text{ap}}), \quad (1)$$

where x represents the number of iterations necessary to solve a given query.

B. Number of Processed Rules per Second

Next, the maximum number of rules that can be processed per second is estimated. For the $n \times n$ SLM the number of rules that can be represented in the OCAPP-ES becomes n. Because the processing time of the OCAPP-ES, T_p, is equal to $5T_r + T_d + T_{s2} + T_{\text{ap}}$, the maximum number of rules R that can be processed per second is

$$R = \frac{n}{5T_r + T_d + T_{s2} + T_{\text{ap}}}. \quad (2)$$

As an example, assuming that the technology permits, $n = 256$, $T_r = 10^{-6}$ s, and $T_{s2} = T_{\text{ap}} = T_{\text{d}} = 10^{-3}$; then an OCAPP-ES can execute roughly 8.52×10^4 rules/s. As a comparison, the NON-VON is a multiprocessor system that is composed of 32 powerful large processing elements and 16,000 small processing elements and is estimated to process 850 rules/s.[9] Another multiprocessor called RUBIC (rule-based inference computer) is estimated to process 4×10^3 rules/s.[28] With the processing capability of 8.52×10^4 rules/s, the OCAPP-ES is expected to achieve a system throughput an order of magnitude better than any electronic RBS can achieve.

6. Conclusion

Although many optical systems have been proposed to improve the performance of electronic rule-based expert systems, these systems still suffer from their limited exploitation of the available parallelism in RBS's or from being too application specific. To overcome these limitations and to take advantage of optics parallelism, we have proposed an optical content-addressable parallel processor for expert systems (OCAPP-ES) in this paper. Distinctive features of the OCAPP-ES include the following: (1) 2-D representation of data (knowledge) and control information; (2) capability of processing general-domain knowledge expressed in terms of variables, numbers, symbols, and comparison operators such as greater than and less than; (3) the parallel optical match unit (designed around the match/compare unit of the OCAPP), which performs the 2-D optical pattern matching and comparison operations; (4) a novel conflict-resolution control-matrix (CRCM) algorithm to resolve conflicts in a single step within the optical select unit. We have detailed the implementation of various units of the system. We have also shown that the estimated maximum number of rules that can be processed per second is 8.52×10^4. It is expected that further developments in optical devices technology (logic, memory, smart SLM's) will further increase the performance of the OCAPP-ES.

This research was supported by National Science Foundation grant MIP-9113688.

References

1. P. Harmon and D. King, *Expert System: Artificial Intelligence in Business* (Wiley, New York, 1985), Sec. 1, pp. 13–76.

2. H. Schorr and A. Rappaport, *Innovative Applications of Artificial Intelligence* (AIAA Press, Menlo Park, Calif., 1989), p. 137.
3. K. Pedersen, *Expert Systems Programming* (Addison-Wesley, Reading, Mass., 1989), Chap. 14, pp. 124–139.
4. R. I. Levine, *AI and Expert Systems: a Comprehensive Guide, C Language* (McGraw-Hill, New York, 1990), Chap. 9.
5. D. P. Miranker, "TREAT: a new and efficient algorithm for AI production systems," Ph.D. dissertation (Columbia University, New York, N.Y., 1987).
6. A. Gupta, C. Forge, D. Kalp, A. Newell, and M. S. Tambe, "Parallel OPS5 on Encore Multimax," in *Proceedings of the 1988 International Conference on Parallel Processing,* F. A. Briggs, ed. (Pennsylvania State Univ. Press, University Park, Pa., 1988), 271–280.
7. R. Gamble, "Transforming rule-based programs: from the sequential to the parallel," in *International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems* (Association of Computing Machinery, Charleston, S.C., 1990), pp. 854–863.
8. S. J. Stolfo and D. P. Miranker, "DADO: a parallel processor for expert systems," in *Proceedings of the International Conference on Parallel Processing,* R. M. Keller, ed. (Institute of Electrical and Electronics Engineers, New York, 1984), pp. 74–82.
9. B. W. Wah and G. J. Li, "Design issues of multiprocessors for artificial intelligence," in *Parallel Processing for Supercomputers,* K. Hwang and D. Degroot, eds. (McGraw-Hill, New York, 1989), Chap. 4, pp. 107–159.
10. J. Gaudiot and A. Sohn, "Data-driven parallel production systems," IEEE Trans. Software Eng. **16,** 281–293 (1990).
11. S. Kuo and D. Moldovan, "The state of the art in parallel production systems," J. Parallel Distrib. Comput. **15,** 1–26 (1992).
12. A. Louri and J. Na, "Parallel electro-optical rule-based system for fast execution of expert systems," Appl. Opt. **32,** 1863–1785 (1993).
13. Y. Li and G. Eichmann, "Conditional symbolic modified signed-digit arithmetic using optical content-addressable memory logic elements," Appl. Opt. **26,** 2328–2333 (1987).
14. A. B. VanderLugt, "Signal detection by complex spatial filtering," IEEE Trans. Inf. Theory **IT-10,** 139–145 (1964).
15. J. Y. Jau, F. Kiamilev, Y. Fainman, and S. H. Lee, "Optical expert system based on matrix-algebra formulation," Appl. Opt. **27,** 5170–5175 (1988).
16. A. Louri, "Optical content-addressable parallel processor: architecture, algorithms, and design concepts," Appl. Opt. **31,** 3241–3258 (1992).
17. P. H. Winston, *Artificial Intelligence* (Addison-Wesley, Reading, Mass., 1984), Chap. 6, pp. 159–204.
18. L. Brownston, R. Farrell, E. Kant, and N. Martin, *Programming Expert Systems in OPS5* (Addison-Wesley, Reading, Mass., 1985), Chap. 1, pp. 4–18.
19. A. D. McAulay, *Optical Computer Architectures* (Wiley, New York, 1991), Chap. 8, pp. 193–222.
20. A. Louri and J. Hatch, "Optical implementation of a single-iteration thresholding algorithm with application to parallel data-base/knowledge-base processing," Opt. Lett. **18,** 992–995 (1993).
21. M. Togai and H. Watanabe, "An inference engine for real time approximate reasoning: toward an expert on a chip," IEEE Expert **1**(3), 55–62 (1986).
22. C. C. Lee, "Fuzzy logic in control systems: fuzzy logic controller. Part 1," IEEE Trans. Syst. Man Cybern. **20,** 404–415 (1990).
23. A. Louri and J. Hatch, "An optical associative parallel processor for high-speed database processing: theoretical concepts and experimental results," IEEE Computer **27**(11), 65–72 (1994).
24. S. Kuo and D. Moldovan, "Implementation of multiple rule firing production systems on hypercube," J. Parallel Distrib. Comput. **13,** 383–394 (1991).
25. J. Liu, K. M. Johnson, and M. G. Robinson, "Room-temperature 10-MHz electro-optic modulation in ferroelectric liquid crystals," Appl. Phys. Lett. **62,** 934–936 (1993).
26. A. D. McAulay, X. Xu, and J. Wang, "Optical implementation of a multi-layer neural network by SLR and MSLM," in *IEEE International Conference on Systems Engineering,* (Institute of Electrical and Electronics Engineers, New York, 1991), pp. 197–200.
27. A. Ersen, S. Dasgupta, T. H. Lin, S. Esner, and S. H. Lee, "Dual-beam recrystallization of silicon on PLZT," in *Optical Computing,* Vol. 9 of 1989 OSA Technical Digest Series (Optical Society of America, Washington, D.C., 1989), pp. 54–57.
28. D. Moldovan, "RUBIC: a multiprocessor for rule-based systems," IEEE Trans. Syst. Man Cybern. **19,** 699–706 (1989).
29. J. A. Na, "Design and simulation of digital optical computing systems for artificial intelligence," Ph.D. dissertation (University of Arizona, Tucson, Ariz., 1994).