A BIT-PLANE ARCHITECTURE FOR OPTICAL COMPUTING WITH TWO-DIMENSIONAL SYMBOLIC SUBSTITUTION

Ahmed Louri and Kai Hwang
Computer Research Institute
University of Southern California
Los Angeles, California 90089-0781

ABSTRACT

A novel architecture based on optical technology is presented for constructing parallel computers. The architecture exploits optics for its ultra-high speed, massive parallelism, and dense connectivity. The processing is based on a new technique called *2-D symbolic substitution* which can be implemented with very fast optical components. Two-dimensional symbolic substitution algorithms are developed for arithmetic/logic operations as well as for complex scientific computations such as matrix algebra and FFT. The predicted performance of the system is compared with the performance of existing electronic array processors and is shown to be potentially superior. The *bit-plane* architecture is shown feasible and economical based on state-of-the-art optical and electro-optical technologies.

1 INTRODUCTION

Many scientific applications such as signal/image processing, finite-element modeling, vision, weather forecasting, and artificial intelligence are data intensive. Conventional computers may not be necessarily efficient in processing large amounts of structured data at high speed due to the limited processor-memory bandwidth, and the slow communication among cooperating processors in a multiprocessing system.

Optics can provide a quantum leap over electronics in high-speed computing and massive parallelism[1,2,3,4]. Optics has several important characteristics which make it more advantageous over electronics as far as massively parallel processing is concerned:

- Optical systems are inherently parallel due to their multi-dimensional and densely interconnected nature. Lenses, prisms, mirrors, etc. can transfer a large data plane consisting of millions of resolvable data points at once. All data points on the plane can be operated upon concurrently.
- The rate at which data is processed through an optical system is essentially limited by the rate at which data is placed in the system and detected at the output. The actual processing time is mainly light propagation through optical devices, which in turn implies higher system throughput.
- Transmission of information via photons requires no physical conducting material. Therefore, the bandwidth is limited only by the modulator technology, whereas in electronics the bandwidth is limited by the capacitance and inductance of the physical conductors used[5].

Although optics can provide the parallelism, high speed processing, and high communication bandwidth needed for scientific computing, it lacks the programmability, control, and decision-making features that are well developed in electronic computers. Thus a computer system that combines the processing power of optics and the programming flexibility of electronics is highly desirable. In this paper, we present a bit-plane architecture and an extended 2-D optical processing technique, called *symbolic substitution*[6], for supporting massively parallel computations. The system processes binary images (images of 0's and 1's) called *bit planes*, as fundamental computational entities. Several parallel computers have been constructed based on bit plane processing. These include the MPP[7] CLIP[8], ICL DAP[9], and to some extent, the Connection Machine[10]. The proposed optical system differs from these existing systems in many respects. The major differences lie in *parallel I/O, non-restricted interconnection pattern,* and *expandability* as elaborated below:

- Although existing array processors process binary images in parallel, the image is still loaded into the processing elements one column (or one row) at a time, due to the planar constraints imposed by the electronic implementation. In contrast, the proposed optical system handles the entire image in parallel, both at the I/O and processing levels.
- Electronic array processors have restricted interprocessor communication to certain patterns wired into

the hardware, typically, two-dimensional grid, so that an individual processing element (PE) can only communicate quickly with its four (or eight) nearest neighbors. This restricts the class of algorithms that can be efficiently mapped onto the array to those that emphasize local operations. This restriction is alleviated in the optical architecture since all communications are done optically, using free space.

- It is difficult to add more processing elements to an electronic array processor once constructed. Whereas in the proposed optical system, adding more processing power needs only an increase in the input image size as shall be seen in subsequent sections.

The paper is organized into six sections. In Sec.2 we describe the optical bit-plane architecture. In Sec.3 we introduce the 2-D symbolic substitution rules and present their optical implementations. In Sec.4 we present algorithms for 2-D arithmetic using symbolic substitution. In Sec.5 we develop parallel algorithms based on 2-D arithmetic. In Sec.6 we project the performance of the optical system.

2 THE OPTICAL BIT-PLANE ARCHITECTURE

Figure 1 depicts a block diagram of the basic components of the system. It consists mainly of: an *input image combiner*, a *parallel processing unit*, an *output router*, a *plane-addressable memory unit* and *control unit*. While conventional computers manipulate individual 0's and 1's as the basic computational objects, the optical architecture manipulates images of 0's and 1's (or bit planes). Each bit plane, i, corresponds to a weight factor 2^i in the binary representation as shown in Fig.2. Up to three bit planes can be processed simultaneously. Locally, the system is viewed as a bit-serial processor, since it involves one-bit arithmetic and logic operations. Globally, it is viewed as a plane-parallel processing system, since large sets of operands stored as bit planes are fed through and executed in parallel.

2.1 The Processing Unit

This unit operates in a SIMD mode, where the same operation is applied to all data entries. The actual processing is based on the optical symbolic substitution technique[6]. In this technique, information is represented by *optical patterns* within a two-dimensional image. An optical pattern is a spatial arrangement of dark and bright spots corresponding to the binary values 0 and 1. Computation proceeds in transforming these patterns into other patterns according to predefined *symbolic substitution rules.* This technique is sensitive not only to the values of bits carrying information but also to their spatial locations within the image.

Symbolic substitution consists of two processing steps: a *recognition phase* where the parallelism of optics is used to recognize all the occurrences of a pattern, called the *search pattern,* within an image, followed by a *substitution phase* where another pattern, called *replacement pattern,* is substituted in all the locations of the search pattern. The search and replacement patterns constitute one substitution rule. We equip the processing unit with three fundamental operators: the *full add*, the *logical NOT*, and the *logical AND* which are defined next. Given bit planes $A = \{a_{ij}\}$, $B = \{b_{ij}\}$, and $C = \{c_{ij}\}$, where $a_{ij}, b_{ij}, c_{ij} \in \{0,1\}$, and i,j represent the Cartesian coordinates, we define a logical NOT operator $\text{NOT}(A) = \bar{A}$ such that $\bar{A} = \{\bar{a_{ij}}\}$ (negation). We define the logical AND operator, denoted by $\triangle$, that takes two bit planes as arguments and produces an output bit plane as follows: $A \triangle B = X$, where $X = \{a_{ij} \wedge b_{ij}\}$. The symbol $\wedge$ represents the conventional logical AND applied to two binary values. We define the full add operator, denoted by $\uplus$, that adds three bit planes and produces two output bit planes as follows: $A \uplus B \uplus C = X, Y$, where $X = \{x_{ij}\} = \{a_{ij} \oplus b_{ij} \oplus c_{ij}\}$ (sum bits), and $Y = \{y_{ij}\} = \{(a_{ij} \wedge b_{ij}) \vee (a_{ij} \oplus b_{ij}) \wedge c_{ij}\}$ (carry bits). The signs $\oplus, \vee$ represent the logical exclusive or and logical or respectively. The three fundamental operators constitute a complete logic and arithmetic set. The optical implementation of these operators will be presented in the next section.

2.2 The Input Combiner

The data represented as bit planes is fed to the processing unit through three input planes A-, B-, and C-plane as shown in Fig.1. The objective of the input combiner is to perform spatial permutations of the input data in such a way that overlapping bits from the input planes form spatial patterns upon which the processing unit applies the relevant symbolic substitution rules. Depending on the fundamental operator needed at a given computational step, the input combiner performs three data movement functions as follows: for the case of the logical NOT, the input combiner needs only to transmit the input image to the processing unit without any change in the spatial position of the data. The logical AND operator is applied to two bit planes. The corresponding data movement function is called the *2-D perfect shuffle.* This function is an extension of the 1-D perfect shuffle [11] to bit planes. The 2-D perfect shuffle permutations affect only the row position of the data leaving the column position unchanged. A pictorial description of the 2-D perfect shuffle is shown in Fig.3 for a 3×3 input planes.

The full add operator is applied to the three input planes. The data movement function needed in this case is called the *2-D 3-shuffle* permutation. This function is also a 2-D extension of the 1-D 3-shuffle permutations[12]. The permuted output is formed by alternating the rows of the 3 input planes.

2.3 The Output Router

This unit directs the processed data to its appropriate destination. It accomplishes this through three data movement functions, namely: *feeding back* the carry bit plane

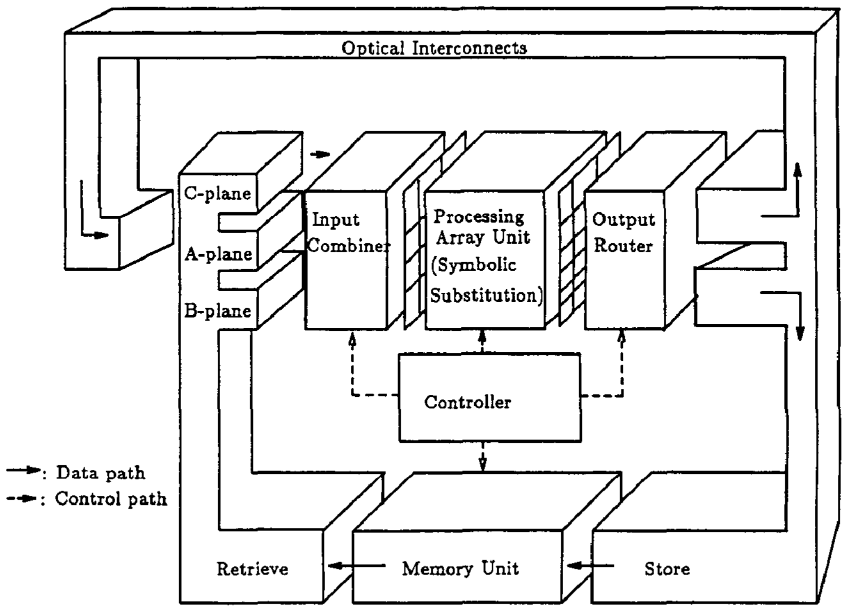


Fig.1 The Architecture of an optica bit-plane processor array

resulting from a full add operation, *transferring* the results to the memory for storage, and *shifting* the processing unit output vertically or horizontally by a variable number of pixels. The shifting functions enable communication between pixels. These shifting functions add a tremendous processing power and flexibility to the system. Many algorithms are recursively defined, where the same processing steps are applied to a reduced set of data points at each iteration. Such algorithms can be executed more efficiently by including the shift functions.

The memory unit is assumed to be random-access plane-addressable. It is organized such that data is retrieved and stored in plane format. The execution sequence and the data flow are controlled by a very fast controller that generates control signals to different units during the execution of a program code. It should be noted that the 2-D parallelism is maintained throughout all the processing stages of the system thus allowing it to support massive parallelism.

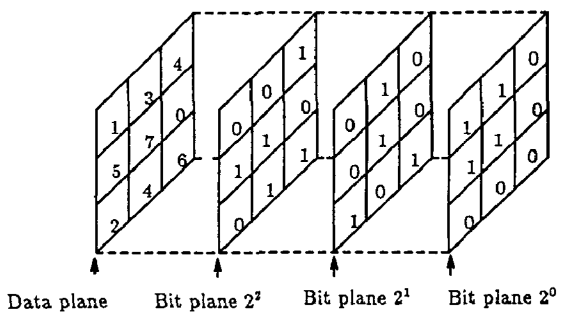


Fig.2 Data representation as a stack of bit planes

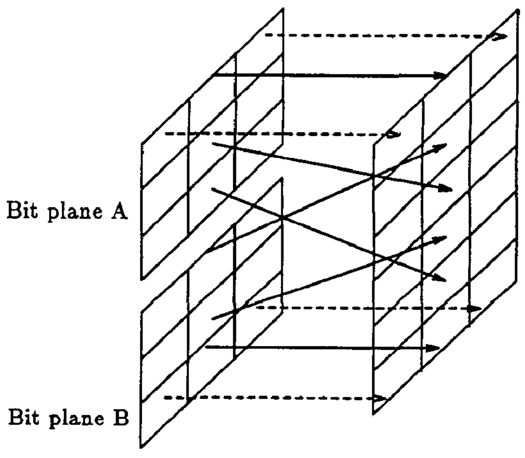


Fig.3 The 2-D perfect shuffle operation.

3 OPTICAL IMPLEMENTATION

The first consideration in implementing the system, is the optical encoding of the binary values 0 and 1. There are several properties of light that can be used. These include light intensity, polarization, and optical signal phase. A possible representation is to encode the logic value 0 by two pixels dark-bright, and the logic value 1 by the inverse pattern, bright-dark as shown in Fig.4. In what follows, we focus on the conceptual issues in implementing the system.

Fig.4 Light encoding of the binary values 0 and 1

3.1 The Input/Output Data Routing

The data movement functions assumed by the input combiner and the output router are all space-invariant. A wide variety of methods can be imagined for realizing these functions. Electrically controllable optical devices such as Wollaston prisms, halfwave plates, and polarizers can be used to implement the interconnection patterns required[13]. Lohmann[13] has given an optical setup for performing the 1-D perfect shuffle. The same principle can be extended to 2-D perfect shuffle. The 2-D 3-shuffle can be implemented in 3 steps as shown in Fig.5. We magnify the input arrays to the total size of the 3 arrays, and then by appropriate masking, we achieve the permutations needed. The setup shown in Fig.5 can be extended to 2-D by replicating the hardware n times for an $n \times n$ input bit planes. There are other methods that have been reported for implementing the perfect shuffle permutations [14].

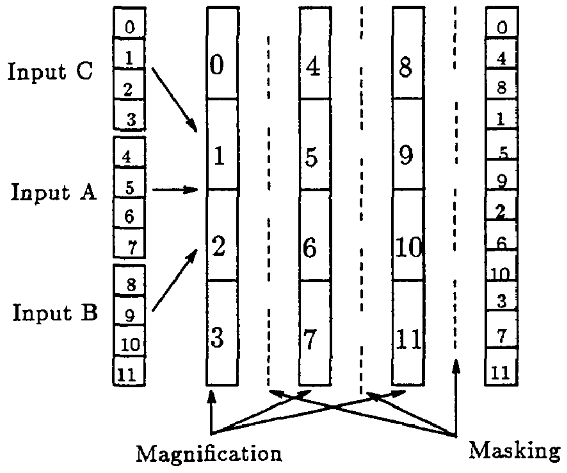


Fig.5 The principle of an optical 2-D 3-shuffle. The shuffling is done by interlacing the input arrays

Halfwave plates and polarizing beam splitters can be used to control the pathways of the light beam at the output router. The main component of such a unit is the *image shifter* which is able to shift a bit plane by an arbitrary number of pixels. The direction and amount of shift are program dependent, and are supplied to the shifter as control signals. This shifter can be implemented using birefringent prisms and electro-optical cell[13], or using real time holograms in a photorefractive media[15].

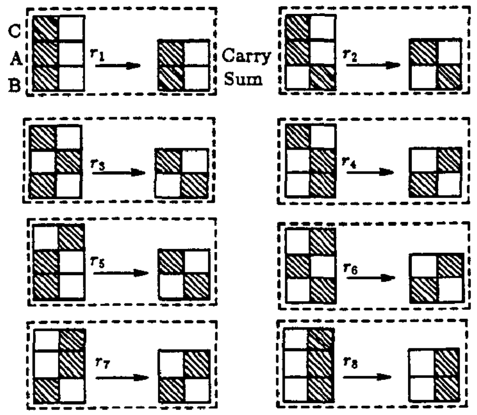
3.2 The Symbolic Substitution Processing Unit

This unit is based on the optical symbolic substitution for performing arithmetic and logic. Symbolic substitution can be formally defined by a three-tuple, $S = (R, P, Q)$ where R is a one-to-one mapping from the domain P to the range Q; $R : P \longrightarrow Q$. R is the set of substitution rules $R = (r_1, r_2, \ldots, r_n)$ where n is the total number of rules. P is the set of search patterns and Q is the set of replacement patterns. Each substitution rule r_i can be defined as:

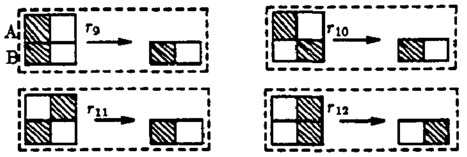
$$r_i(P_j) = \begin{cases} Q_j & \text{if } i = j \\ \phi & \text{if } i \neq j \end{cases} \tag{1}$$

for $i = 1, \ldots, n$ and ϕ representing an empty pattern (a number of dark pixels).

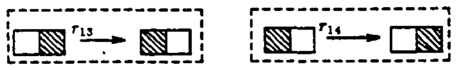
We derive the symbolic substitution rules required to implement the fundamental operators from their associated truth-table specifications. The input combinations of these truth tables constitute the search patterns, and the table entries represent the replacement patterns. The logical NOT operator manipulates one bit pattern, while the AND and the full add manipulate 2 and 3 bits respectively. Therefore, a total of 14 substitution rules are required to implement the fundamental operators as shown in Fig.6.



(a) Optical substitution rules for the full addition



(b) Optical substitution rules for the logical AND



(c) Optical substitution rules for the logical NOT

Fig.6 Optical symbolic substitution rules for full add, logical AND, and logical NOT operators

To illustrate the optical implementation of the two processing steps involved in symbolic substitution let us consider the implementation of substitution rule r_3 in Fig.6. The search pattern P_3 of r_3 can be described by :

$$P_3 = (P_3{}^0, P_3{}^1) \tag{2}$$

where $P_3{}^0$ and $P_3{}^1$ are the dark-pixel pattern and bright-pixel pattern of P_3 respectively (Fig.7a). These latter can in turn be described by a 2-D position vector with discrete components. Each component represents the location of a dark (or a bright) pixel in the Cartesian plane:

$$\begin{aligned} P_3{}^0 &= \{(x_1,y_1)(x_2,y_2)(x_3,y_3)\} \quad \text{and} \\ P_3{}^1 &= \{(x_1',y_1')(x_2',y_2')(x_3',y_3')\} \end{aligned} \tag{3}$$

where (x_i, y_i), (x_i', y_i') are the Cartesian coordinates of the i-th dark pixel and the i-th bright pixel respectively. Thus P_3 can be expressed by:

$$P_3{}^0 = (0,0)(1,1)(0,2), \quad P_3{}^1 = (1,0)(0,1)(1,2) \tag{4}$$

We extend the implementation method described in Ref. [16] to implement the new substitution rules introduced in Fig.6. Since we are using dual-rail coding (the value and its complement), it is sufficient to recognize the dark-pixel pattern or the bright-pixel pattern of a search pattern to completely recognize it[16]. We choose to recognize the dark-pixel pattern as this appears more convenient for practical realization. The recognition phase is illustrated in Fig.7. In this phase, the input plane must be replicated as many times as there are dark pixels in the search pattern. Each copy is associated with one dark pixel. Therefore the input plane of Fig.7 is replicated into 3 copies. Copy 1 is associated with dark pixel (x_1, y_1), copy 2 with (x_2, y_2) and copy 3 with (x_3, y_3) as shown in Fig.7b. Then each copy i is shifted by $-x_i$ along the X-axis and $-y_i$ along the Y-axis in such a way that its associated dark pixel coincides with the origin (Fig.7c). Next, the shifted copies are superimposed(Fig.7d). The superimposed image is then inverted using an optical NOR-gate array[16] and AND-ed with a mask M as shown Fig.7e. The presence and the location of the search pattern in the input image is indicated by a bright pixel in the masked output (or the recognition plane) as shown in Fig.7f.

The substitution phase is shown in Fig.8. In this phase, the recognition plane is replicated as many times as there are bright pixels in the replacement pattern. In our example, the recognition plane is replicated twice as shown in Fig.8b. Each replica i is associated with the i-th bright pixel of the replacement pattern. Then each replica is shifted by $+x_i'$ along the X-axis, and $+y_i'$ along the Y-axis such that its associated bright pixel overlays with its proper location in the replacement pattern (Fig.8c). Then all the shifted copies are superimposed to form the output image as shown in Fig.8d.

The method just outlined is used to process one substitution rule. Generally, several substitution rules are simultaneously applied. In order to process several rules concurrently, the output of the input combiner is replicated a number of times equal to the number of rules to be activated at a given stage of computation (for example, to add 3

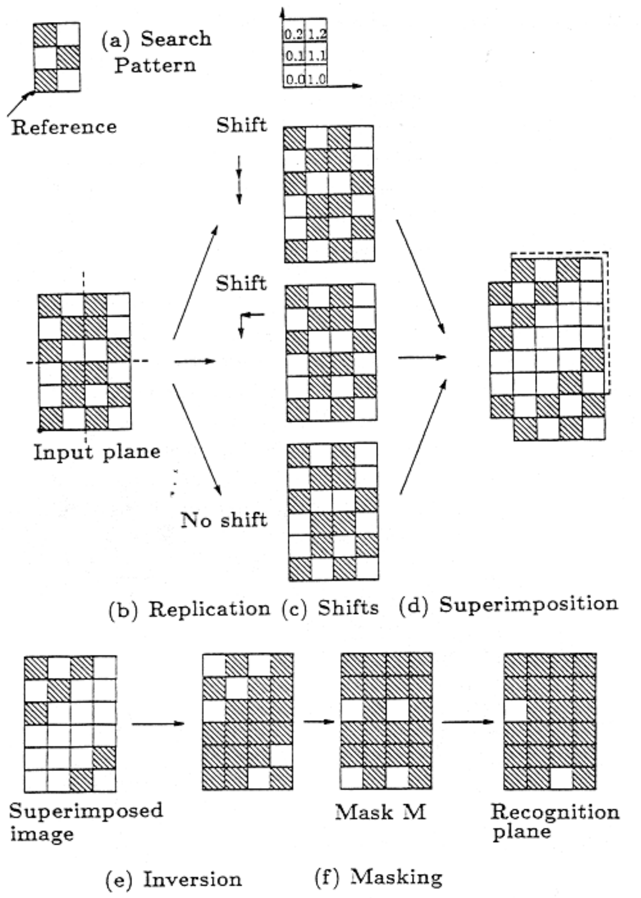
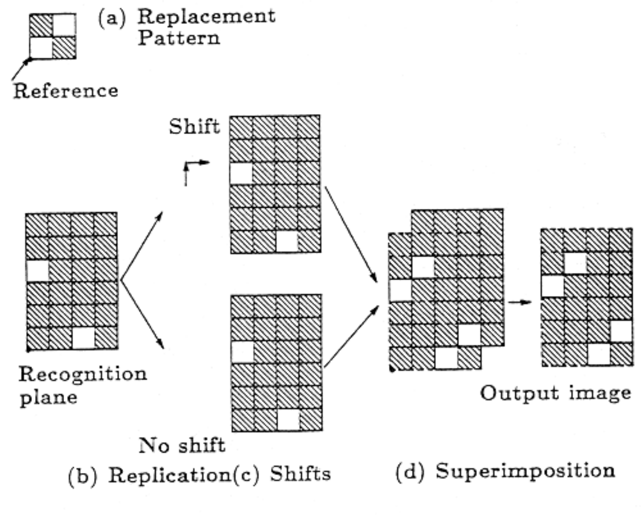


Fig.7 The Optical image processing steps needed to implement the recognition phase of symbolic substitution



→: Shift operation (shift by one pixel)

Fig.8 Optical processing steps needed to implement the substitution phase of symbolic substitution

bit planes, we need to replicate the arranged input 8 times corresponding to the 8 substitution rules associated with the full add operator). Each copy is sent to one rule, after the necessary substitutions, the outputs of all the active rules are optically combined to form the processed result as shown in Fig.9.

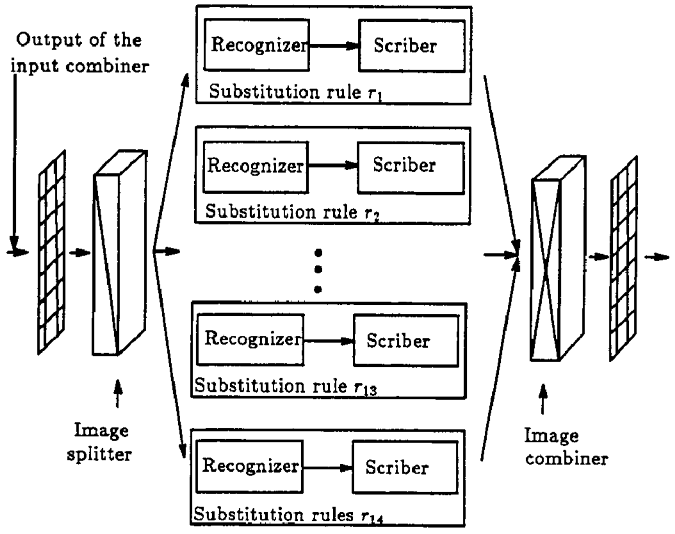


Fig.9 The optical symbolic substitution processing unit

3.3 The Memory Organization

Spatial light modulator or *bistable optical* technologies [1] can be used for single-plane temporary storage such as the input and output planes. However, this will not be sufficient to implement a memory unit able to hold a large number of bit planes. Volume holograms, with their ability to store information in 3-D, show a great potential for a dramatic increase in optical storage density. Photorefractive materials such as Lithium Niobate and Potassium Niobate are also attractive new candidates for real-time optical memory.

4 2-D ARITHMETIC/LOGIC ALGORITHMS

The proposed architecture exploits image (or spatial) parallelism, which means the simultaneous processing of the entire binary images at each computational step (or cycle). To enforce this capability, we identify high-level operations that can be simultaneously applied to large sets of data represented in planes. The use of such constructs to formulate parallel algorithms will provide an efficient algorithm-architecture mapping.

In what follows, the boldface notation i.e. **X**, **Y** etc. denotes a data plane (or a stack of bit planes), and the italic notation (i.e. X, Y) designates a single bit plane. The notation $A \leftarrow X$ is interpreted as data transfer from memory location X to input plane A. $X \leftarrow Y$ denotes data transfer from memory location Y to X. This involves loading Y, going through the processing unit without any effect, and storing it in X. $C \leftarrow 0$ is interpreted as loading the C-plane with a zero bit plane (all entries are 0). Loop indices and parameter calculations such as "for $i = a$ to b" should be interpreted as control instructions that are executed by the control unit.

4.1 2-D Addition/Subtraction

This operation refers to the addition (subtraction) of corresponding elements of two $n \times n$ data planes **X** and **Y** of integers. The result is a data plane $\mathbf{S} = \{ s_{ij} \}$, where $s_{ij} = x_{ij} \pm y_{ij}$ for $i, j = 1, \ldots, n$. Let **X** be an $n \times n$ q−bit planes, $X_{q-1}, X_{q-1}, \ldots, X_0$ where q is the precision of the operands, X_0 being the least significant and X_{q-1} being the most significant bit planes respectively. Similar considerations take place for the data plane **Y**, the procedure is as follows:

```
Procedure 2-D Addition(X,Y)
  begin
    $C \leftarrow 0$ ;
    for $k := 0$ to $q-1$ do
        $A \leftarrow X_k$;
        $B \leftarrow Y_k$;
        $S_k, C_{out} \leftarrow A \uplus B \uplus C_{in}$
    $S_q \leftarrow C$;
  end 2-D Addition
```

The notation $S_k, C_{out} \leftarrow A \uplus B \uplus C_{in}$, in the above procedure, designates the addition of bit planes A and B together with the previous carry C_{in}; the sum bit plane is to be stored in S_k and the resulting carry bit plane, C_{out}, is routed back to input plane C (C_{in} and C_{out} represent the same physical location). The procedure starts by initializing the C-plane to zero, and loading bit planes X_0, Y_0 into A-plane and B-plane respectively. The processing unit applies the full add substitution rules simultaneously to the 2-D 3-shuffled plane. The sum bit plane is stored in S_0, and the carry bit plane is fed back to the C-plane for the next iteration, while the memory unit loads the bit planes X_1 and Y_1 in the A-plane and B-plane respectively. The whole process continues until X_{q-1} and Y_{q-1} are added and the sum $S_0, S_1, \ldots, S_q$ stored as a stack of bit planes in the memory. The addition of two q-bit planes is done in q iterations, regardless of the number of operands to be added.

Representation of numbers in two's complement form allows 2-D subtraction by adding few additional steps to the 2-D addition procedure. The pairwise subtaction of two data planes **X**,**Y** is done by first forming the two's complement of the subtrahend **Y**, then add it to **X**, using the 2-D addition procedure. The two's complement of data plane **Y** is obtained by applying the substitution rules for the NOT operator and the 2-D addition procedure.

4.2 2-D Multiplication

This operation refers to the multiplication of corresponding elements of two data planes. Let **X** and **Y** be as described previously, then the product **P** is a $2q-$bit planes $\mathbf{P} = P_{2q-1}P_{2q-2}\dots P_0$, where $\mathbf{p}_{ij} = \mathbf{x}_{ij} \times \mathbf{y}_{ij}$. This operation uses the logical AND and the full add operators. The complete procedure is as follows:

```
Procedure 2-D Multiplication(X,Y)
    begin
        for $k := 0$ to $2q - 1$ do
            $P_k \leftarrow 0$;
        for $l := 0$ to $q - 1$ do
                $C \leftarrow 0$;
                for $m := 0$ to $q - 1$ do
                        $A \leftarrow X_m$;
                        $B \leftarrow Y_l$;
                        $B \leftarrow A \triangle B$;
                        $A \leftarrow P_{m+l}$;
                        $P_{m+l}, C \leftarrow A \uplus B \uplus C$;
                endfor;
                $P_{q+l} \leftarrow C$;
        endfor;
    end 2-D Multiplication
```

The time complexity of the 2-D multiplication is $O(q^2)$, independent of the number of pairs to be multiplied. Note that, unlike the conventional shift and add multiplication algorithm, we did not need to shift the previous partial product to generate the current one. Instead, we start the addition at the bit plane corresponding to the amount of shift required.

4.3 2-D Shift

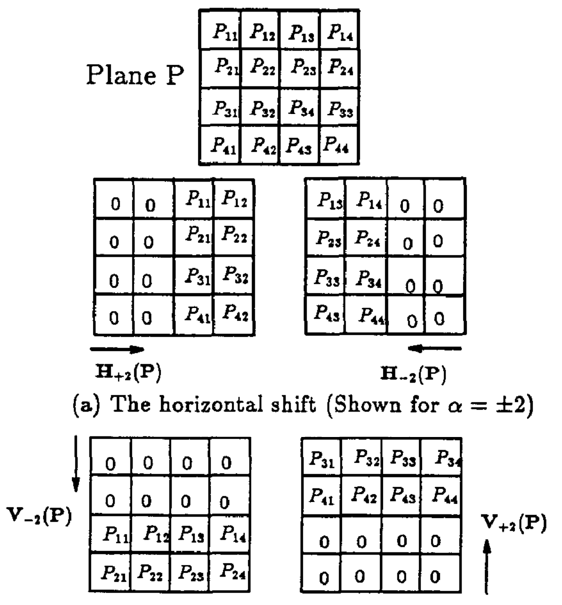
We define two operations for shifting a data plane by a variable number of pixels. The shift considered here is the logical shift, where columns (or rows) of 0s enter the opposite direction of the shift. Given $\mathbf{P} = P_{q-1}P_{q-2}\dots P_0$, and $\mathbf{X} = X_{q-1}X_{q-2}\dots X_0$, we define a horizontal shift operation, denoted by $\mathbf{H}_\alpha(\mathbf{P})$, to be the data plane **P** shifted in the X-axis by α columns ($+\alpha$ for positive shift, and $-\alpha$ for negative shift) as shown in Fig.10a. The shifted plane can be either stored in itself or in a different memory location, therefore the notation $\mathbf{X} \leftarrow \mathbf{H}_\alpha(\mathbf{P})$ is interpreted as shifting the data plane **P** by α columns and storing it in **X**. Similarly, we define the vertical shifting operation, denoted by $\mathbf{V}_\alpha(\mathbf{P})$, to be the data plane **P** shifted along the Y-axis by α rows ($+\alpha$ for upward shift, and $-\alpha$ for downward shift) as shown in Fig.10b. These shift operations are implemented by the shifter unit.

4.4 Summation of n^2 Numbers

This refers to calculating the sum of all the elements of an array. Let the initial array of data **S** contain n^2 elements and be stored as bit planes : $S_{q-1}S_{q-2}\dots S_0$. The algorithm is composed of two phases. In the first phase, we sum the elements of **S** along the rows to produce one row of accumulated sums. In the second phase, we sum the elements of this row along the columns to produce a single integer whose value is the sum of all the data entries of **S**. The detailed procedure is as follows:

```
Procedure Sum(S,X,Y)
    begin
        for $k = 1$ to $\log_2 n$ do        /* Loop 1 */
            $\alpha := n/2^k$;
            $\beta := \sum_{i=1}^{i=k} (n/2^k)$;
            $\mathbf{X} \leftarrow \mathbf{V}_{-\beta}(\mathbf{S})$;
            $\mathbf{V}_{+\beta}(\mathbf{X})$;
            $\mathbf{Y} \leftarrow \mathbf{V}_{+\alpha}(\mathbf{S})$;
            $\mathbf{S} \leftarrow$ 2-D Addition(X,Y);
        endfor      /* end Loop 1 */
        for $k = 1$ to $\log_2 n$ do        /* Loop 2 */
            $\alpha := n/2^k$;
            $\beta := \sum_{i=1}^{i=k} (n/2^k)$;
            $\mathbf{X} \leftarrow \mathbf{H}_{-\beta}(\mathbf{S})$;
            $\mathbf{H}_{+\beta}(\mathbf{X})$;
            $\mathbf{Y} \leftarrow \mathbf{H}_{+\alpha}(\mathbf{S})$;
            $\mathbf{S} \leftarrow$ 2-D Addition(X,Y);
        endfor      /* end Loop 2 */
    end Procedure Sum
```

There are two other useful operations that are similar to the Sum procedure namely, *Row-Sum* and *Column-Sum*. Row-Sum sums the elements of data plane along the rows, thus reducing a data plane to one row of accumulated sum, while all the other rows contain 0s. Column-Sum accumulates the data along the columns thereby producing a new plane whose first column represents an accumulation of all the columns, and 0s in the rest. In fact, "Loop 1" and "Loop 2" in the above procedure implement Row-Sum and Column-Sum respectively.



(a) The horizontal shift (Shown for $\alpha = \pm 2$)

(b) The vertical shift (Shown for $\alpha = \pm 2$)

Fig.10 The horizontal and vertical shift functions

5 MAPPING PARALLEL ALGORITHMS

Data movement can be used to characterize an algorithm running on a parallel system. Consequently, parallel algorithms can be classified in two major classes : those with *local* interconnections and those with *global* interconnections [12]. The optical architecture offers a great flexibility in handling both local as well as global computations. In what follows, we use the 2-D operations introduced previously to construct parallel algorithms that can be directly mapped onto the optical architecture. The first example is for finding the product of two matrices which exhibits local and intensive computations. The second example is for computing the 2-D fast Fourier transform which represents a typical example of global communication.

5.1 Matrix Multiplication

Let $\mathbf{X}$ and $\mathbf{Y}$ be $n \times n$ matrices (assuming same size for simplicity) then their product $\mathbf{X} \times \mathbf{Y} = \mathbf{Z}$ is an $n \times n$ matrix whose elements are given by:

$$z_{ij} = \sum_{k=1}^{k=n} x_{ik} y_{kj}, \quad i,j = 1,\ldots,n \tag{5}$$

We assume that the matrix $\mathbf{X}$ is stored as $n \times n$ data planes : $X^n, X^{n-1}, \ldots, X^1$, in which each column of $\mathbf{X}^i$ is equal to the i-th row of $\mathbf{X}$, for $i = 1,\ldots,n$. Let $\mathbf{X}^k = \{x_{ij}{}^k\}$, where $x_{ij}{}^k = x_{kj}$. Let $\mathbf{T}_k$ be the matrix formed by the 2-D multiplication of matrices $\mathbf{X}^k$ and $\mathbf{Y}$, then $\mathbf{T}_k = \{t_{ij}\}$, where $t_{ij} = x_{ij}{}^k y_{ij} = x_{kj} y_{ij}$. Thus summing the elements of each column of $\mathbf{T_k}$ using the Row-Sum procedure will produce a matrix say $\mathbf{Z}_k$ whose first row represents the first row of matrix $\mathbf{Z}$:

$$\mathbf{Z}_k = \sum_{i=1}^{i=n} t_{ij}, \quad j = 1,\ldots,n \tag{6}$$

where the first row of $\mathbf{Z}_k$ represents the first row of $\mathbf{Z}$ and all the other rows are 0s. By repeating these steps for all values of k ($k = 1,\ldots,n$), we produce n matrices $\mathbf{Z}_n$, $\mathbf{Z}_{n-1} \ldots, \mathbf{Z}_1$. The first row of each matrix $\mathbf{Z}_i$ represents the i-th row of the final product matrix $\mathbf{Z}$. We shift these matrices by appropriate shifts, and add them pairwise to produce the final matrix $\mathbf{Z}$:

```
Procedure Matrix Multiply(Z,X,Y)
  begin
    for  $k := 1$ to $n$ do
        $\mathbf{T}_k \leftarrow$ 2-D Multiplication($\mathbf{X}^k, Y$);
        $\mathbf{Z}_k \leftarrow$ Row-Sum($\mathbf{T}_k$);
    endfor
    for  $k := 1$ to $n$ do
        $\mathbf{V}_{(1-k)}(\mathbf{Z}_k)$;
    for $k := 1$ to $n$ do
        $\mathbf{Z} \leftarrow$ 2-D Addition($\mathbf{Z}, \mathbf{Z}_k$);
  end Matrix Multiply
```

The time complexity of the algorithm is $O(n(q\log_2 n + q^2))$, where q is the operand length. It can be seen that the time complexity of this algorithm is logarithmic in n, as opposed to cubic in n for the conventional triple loop matrix multiplication.

5.2 2-D Fast Fourier Transform

FFT is a transitive function since every output is a non-trivial function of every input. One would expect such functions to generate global communications. A two-dimensional Fourier transform can be mathematically defined as follows:

$$\mathbf{X}(k_1,k_2) = \sum_{n_1=1}^{n} \sum_{n_2=1}^{n} \mathrm{x}(n_1,n_2) \mathbf{W}^{n_2 k_2} \mathbf{W}^{n_1 k_1} \tag{7}$$

for $k_1, k_2 = 1,\ldots,n$. This is equivalent to :

$$X(k_1,k_2) = \mathrm{FFT}_{n_1}(\mathrm{FFT}_{n_2} x(n_1,n_2)) \tag{8}$$

This form gives rise to a 3-D signal flow graph, which if calculated in a conventional way, would take $O(n^2 \log_2 n)$, by first applying 1-D FFT rowwise(columnwise) n times and then applying, on the transformed sequence, 1-D FFT columnwise (rowwise) n times. In [11], a single stage perfect shuffle interconnect and a set of multiply-add modules were introduced for in-place computation of 1-D FFT. We present here an extended algorithm for computing 2-D FFT, using the 2-D operations introduced in Sec.4 and the 2-D perfect shuffle function described in Sec.2.

In the following algorithm, we assume the data points $\mathbf{X}$ are stored in two data planes, U, for the upper portion, and L *for the lower portion.* W_i *are data planes, corresponding* to the weights of the i-th stage as indicated in Eq.7, $\mathbf{S}_u$, $\mathbf{S}_l$ and $\mathbf{T}$ are scratchpad planes used to hold intermediate results.

```
Procedure 2-D FFT(X)
  begin
    for $k := 1$ to $\log_2 n$ do
        $\mathbf{U} \leftarrow \mathbf{V}_{-n/2}(\mathbf{X})$;
        $\mathbf{V}_{+n/2}(\mathbf{U})$;
        $\mathbf{L} \leftarrow \mathbf{V}_{+n/2}(\mathbf{X})$;
        $\mathbf{T} \leftarrow$ 2-D Multiplication($\mathbf{L}, \mathbf{W}_k$);
        $\mathbf{S_u} \leftarrow$ 2-D Addition($\mathbf{U}, \mathbf{T}$);
        $\mathbf{S_l} \leftarrow$ 2-D Subtraction($\mathbf{U}, \mathbf{T}$) ;
        $\mathbf{X} \leftarrow$ 2-D Perfect Shuffle($\mathbf{S_u}, \mathbf{S_l}$);
    endfor
  end 2-D FFT
```

The expression $\mathbf{X} \leftarrow$ 2-D Perfect Shuffle($\mathbf{S}_u, \mathbf{S}_l$), in the above procedure designates the 2-D perfect shuffle of the matrices $\mathbf{S}_u$ and $\mathbf{S}_l$. The resulting transform is in reverse binary order. The above algorithm computes the columnwise (rowwise) computations. We have to restore the normal order before we proceed for the rowwise (columnwise) computation. This is done by $O(\log_2 n)$ permutations involving only data movement. The algorithm computes the 2-D FFT in $O(q^2 \log_2 n)$ iterations. The inverse FFT can be expressed in the same formalism as the FFT, and therefore can be computed by the same algorithm.

6 PERFORMANCE PROJECTION

We project the performance of the optical architecture by estimating several performance metrics and compare them with those of commercial electronic array processors such as the MPP, CLIP, DAP and the Connection Machine.

6.1 Asymptotic Performance

Hockney[17] introduced a performance metrics called: r_{inf} to give a first-order characterization of the *asymptotic performance* of a parallel computing system. The parameter r_{inf} gives a quantitative measure of the maximum rate of computation in units of equivalent scalar operations performed per second. For an array processing system, r_{inf} is evaluated as follows[17]:

$$r_{inf} = \frac{n^2}{t_{array}} \quad (9)$$

where t_{array} is the time taken to execute one operation on all the PEs, this is usually taken as the clock period, and n^2 is the total number of PEs. Assuming an array size of 1000×1000, and 10 Mhz rate, r_{inf} is then 10^{13} bit operations/sec for the optical system. The MPP with an array of 128×128 PEs and 10 Mhz rate has achieved 6×10^9 8-bit operations /sec. The CLIP with a 96×96 array and a 25 μsec cycle time has achieved 3.7×10^9 bit operations/sec. The DAP with a 64×64 array and 0.2 μsec cycle time, was described to achieve 10^8 32-bit operations/sec. The Connection Machine with 65,536 PEs and a 0.5 μsec cycle time can achieve 13×10^{10} bit operations/sec (the CM-2 model[18]). It can be seen that the potential throughput of the optical system can be at least 3 orders of magnitude higher than any existing array processor.

The *half-performance length*: $n_{1/2}$ determines the amount of the hardware parallelism in a computer architecture[17]. For a nonpipelined array processor, the factor $n_{1/2}$ is defined to be the vector length required to achieve half the maximum performance($r_{inf}/2$). This can be easily calculated for all the systems since half the vector is simply half the array size, hence $n_{1/2} = 5 \times 10^5$ for the optical architecture, 8192 for the MPP, 4608 for the CLIP, 2048 for the DAP, and 32768 for the CM-2.

6.2 Communication and I/O Capabilities

The parameters r_{inf} and $n_{1/2}$ do not completely reflect the effectiveness of a parallel computer. Communication plays a crucial part in determining the overall system performance. There are many communication metrics in the literature[19] we choose the most widely used for our purposes:

Communication bandwidth is the maximum number of messages that can be simultaneously exchanged in one time step. Hence the bandwidth of the optical system is n^2, since up to n^2 PEs can receive and send data at a time. The data transmission in the MPP and the CLIP is one column at a time, therefore their bandwidth is n, the DAP on the other hand transmits data in a row-parallel fashion which amounts to the same bandwidth factor n. The Connection Machine has a maximum sustained communication bandwidth of $2n^2$.

The *diameter* is the maximum number of communication cycles (or links) needed for any two PEs to communicate. For the optical case, this factor is 1, since we allow any number of shifts in either directions in one cycle time. The MPP and the DAP are mesh-connected and therefore have a diameter of $2(n-1)$. The CLIP has a hexagonal connectivity and therefore has a diameter of $n\sqrt{2}$. The diameter of a Connection Machine of size n^2 PEs is $O(\log_2 n)$.

Broadcasting is the ability to send the value in a certain PE to all the other PEs. The amount of communication cycles to achieve this is considered a measure of communication performance. This value is $O(\log_2 n)$ for the optical system, $O(n)$ for the DAP, MPP, and CLIP, and $O(\log_2 n)$ for the Connection Machine.

Unfortunately no metrics exist that measure the I/O capability of a parallel computer. In current implementations of the MPP and the CLIP, I/O is handled in column-parallel fashion while the DAP is row-parallel. By contrast with the optical system, I/O activities are plane-parallel. This gives the optical system an I/O speedup of n over the electronic counterparts. Table 1 summarizes the various performance metrics values considered above.

7 CONCLUSIONS

We have proposed a bit-plane architecture that takes advantage of many useful properties of optics for parallel processing. We have discussed its implementation using state-of-the-art optical devices. We introduced 2-D symbolic substitution rules for arithmetic/logic operations. These rules are used for parallel SIMD computation. Furthermore, we presented data-parallel constructs to implement parallel algorithms on the proposed architecture. The performance of the proposed optical array processor has the potential to be at least thousand times higher than any existing electronic array processor.

ACKNOWLEDGMENTS

This research was supported by an ONR Contract No.N14-86-K-559 and in part by a NOSC Contract No.85-D-203. The authors would like to thank the referees for their valuable suggestions.

Table 1: Performance comparison of the optical bit-plane architecture with electronic array processors

Computing System	Performance Metrics						
	Maximum Performance (r_∞)	Parallelism ($n_{1/2}$)	Diameter	Bandwidth	Broadcasting	I/O Capability	Cycle time(μsec)
Optical Architecture (1000 × 1000) PEs	10^{13} bit op/sec	500,000	1	n^2	$O(\log_2 n)$	n^2	10^{-3} (potentially)
MPP (128 × 128)	6×10^9 8-bit op/sec	8192	$2n-2$	n	$O(n)$	n	10^{-1}
DAP (64 × 64)	3.7×10^8 32-bit op/sec	2048	$2n-2$	n	$O(n)$	n	2×10^{-1}
CLIP (96 × 96)	3.7×10^9 bit op/sec	4608	$\sqrt{2}n$	n	$O(n)$	n	25
CM-2 (64K PE's)	13×10^{10} bit op/sec	32768	$O(\log_2 n)$	n^2	$O(\log_2 n)$	n^2	5×10^{-1}

Note: n^2 is the array size (total number of PE's)

References

[1] T. E. Bell, "Optical computing: a field in flux," *IEEE Spectrum*, pp. 34–57, August 1986.

[2] K. Hwang and A. Louri, "New symbolic substitution algorithms for arithmetic using signed-digit representation," *In Proc. Soc. of Photo-Opt. Instr. Eng., High Speed Computing (Los Angeles)*, Jan. 10 - 14, 1988.

[3] A. Louri and K. Hwang, "Parallel architectures for optical computing," *Proceedings Third SIAM Conf. on Parallel Processing and Scientific Computing (Los Angeles)*, Dec. 1 - 4, 1987.

[4] A. A. Sawchuk and T. C. Stand, "Digital optical computing," *Proceedings of The IEEE*, vol. 72, pp. 758–779, July 1984.

[5] K. C. Saraswat and F. Mohammadi, "Effect of scaling of interconnections on the time delay of VLSI circuits," *IEEE Transaction on Electron Devices*, vol. ED-29, no. 4, pp. 645–650, 1982.

[6] A. Huang, "Parallel algorithms for optical digital computers," in *Proceedings IEEE Tenth Int'l Optical Computing Conf.*, pp. 13–17, 1983.

[7] K. E. Batcher, "Design of a massively parallel processor," *IEEE Transactions on Computers*, vol. C-29, pp. 836–884, Sept. 1980.

[8] M. J. Duff (Fu and Ichikawa, eds.), *CLIP 4 : Special Computer Architecture for Pattern Recognition.* CRC Press, 1982.

[9] S. F. Reddaway, "DAP - a distributed array processor," in *First Annual Symposium on Computer Architecture*, (Florida), IEEE/ACM, 1973.

[10] W. D. Hillis, *The Connection Machine.* MIT Press, Cambridge, Mass., 1985.

[11] H. S. Stone, "Parallel processing with the perfect shuffle," *IEEE Transactions on Computers*, vol. C-20, no. 2, pp. 153–161, 1971.

[12] K. Hwang and F. Briggs, *Computer Architecture and Parallel Processing.* McGrow-Hill, New York, 1984.

[13] A. W. Lohmann, "What classical optics can do for the digital optical computer," *Applied Optics*, vol. 25, pp. 1543 - 1549, 15 May 1986.

[14] K. H. Brenner and A. Huang, "Optical implementations of the perfect shuffle interconection," *Applied Optics*, vol. 27, pp. 135 - 137, 1 Jan. 1988 1988.

[15] T. J. Drabik and S. H. Lee, "Shift-connected SIMD array architectures for digital optical computing systems, with algorithms for numerical transforms and partial differential equations," *Applied Optics*, vol. 25, pp. 4053–4064, Nov. 1986.

[16] K. H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," *Applied Optics*, vol. 25, pp. 3054 - 3060, 15 Sept. 1986.

[17] R. W. Hockney and C. R. Jesshope, *Parallel Computers: Architecture, Programming and Algorithms.* Adam Hilger Ltd, Bristol, 1981.

[18] Thinking Machine Corporation, "Connection machine model CM-2 technical summary," Tech. Rep. Series HA87-4, Thinking Machine Corporation, 1986.

[19] T. Y. Feng, "A survey of interconnection networks," *Computer*, pp. 12–27, December 1981.