A Symbolic Substitution Based Parallel Architecture and Algorithms for High-speed Parallel Processing

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

Abstract

A new parallel architecture that is amenable to optical implementation is presented for massively data-parallel computing. The architecture is an SIMD model that exploits spatial parallelism and processes 2-D binary images as fundamental computational entities. Processing is based on a new technique called *symbolic substitution* logic. A hierarchical mapping technique is presented for designing data-parallel algorithms and mapping them onto the optical architecture. The mapping of several numerical algorithms onto the architecture is presented. Implementation issues as well as performance analysis of the optical system are also considered.

1 Introduction

Processing large amounts of data at high-speed has been increasingly required as progress in science and technology advances. This is manifested in data-intensive applications such as signal and image processing, weather forcasting and modeling, remote sensing among many others[1]. These applications exhibit a high degree of data-parallelism in which concurrency occurs naturally in structured data. To achieve the computational rates equivalent to trillions of operations per second that will be required for data-intensive applications, improvements will have to be made on all fronts of computer system design. With current technology, fast circuit and packaging techniques will improve performance by reducing the basic cycle time of conventional computing systems. However, to reach the rate of one trillion operations per second, would require a cycle time lower than a picosecond! This can not be expected from advances in electronic technology alone, because of fundamental physical limitations and therefore further speed-up will have to come from parallelism.

Optics has many unique features that can be exploited for high-speed parallel processing (see for example [2]). Optical systems are inherently parallel and multi-dimensional. The rate at which data is processed through an optical processing system is essentially limited by the rate at which data is placed in the system and detected at the output. This implies higher throughput and processing rates than current systems. Transmission of information via photons requires no physical conducting material, but relies on low-loss dielectric material for wave guide propagation or free space. Therefore higher temporal and spatial bandwidths can be obtained. This paper presents a hybrid parallel optical computing architecture that combines the high-speed and parallelism of optics with the programming flexibility of electronics for data-intensive computing. In addition, a technique for mapping parallel algorithms onto the architecture is presented.

2 The 3-D Optical Architecture

Figure 1 depicts a block diagram of the basic components of the optical architecture. Unlike conventional computers that manipulate individual 0's and 1's as basic computational objects, the optical architecture manipulates bit planes as basic computational entities. Each bit plane i corresponds to a weight factor 2^i in the binary representation. Up to 3 bit planes can be processed simultaneously. For images of $n \times n$ elements, it follows that up to $3n^2$ operations are performed concurrently. The heart of the architecture is the processing array. Locally, this array can be viewed as a bit-serial or a bit-slice processor, since it performs one logical operation, on one, two or three single-bit operands.

Globally, it can be viewed as a plane-parallel processor, since it simultaneously performs the same operation on a large set of operands encoded as bit planes. This bit-serial processing allows flexible data formats and almost unlimited precision. Optical interconnects are used to move the images around the system. The architecture is conceived to be built with optical hardware that manipulates entire images simultaneously both at I/O and processing, so that the 2-D parallelism is sustained throughout various stages of the computation.

2.1 The Processing Array

The processing array operates in the SIMD (single instruction multiple data) mode of computation, where the same operation is applied to all the data entries. In the proposed system, processing is based on the optical symbolic substitution logic[7] (SSL) which will be described in detail later. The processing unit is equipped with 3 fundamental operators: a *logical NOT* which inverts all the entries of an input plane, a *logical AND* , denoted by $\triangle$, that performs the logical and of the overlapping bits of two bit planes, and a *full ADD*, denoted by $\biguplus$, that performs the full addition of the overlapping bits of the three input planes. By overlapping bits, it is meant bits with the same Cartesian coordinates (i, j) in the input planes. These operators constitute a complete arithmetic and logic set capable of computing any arithmetic or logic function.

2.2 Input/Output Data Routing

The data represented as bit planes is fed to the system through three input planes, namely A-, B-, and C-plane as shown in Fig.1. Depending on the fundamental operator needed at a given computational step, the input combiner performs three data movement functions: for the logical NOT, it simply latches the relevant input plane to the processing unit. For the logical AND, the data movement required is called the *2-D perfect shuffle*. This function performs the shuffling of the row position, i, of the data such that the overlapping bits from the two planes become spatially adjacent. This function does not affect the column position, j. The data movement required for the full ADD operator is called the *2-D 3-shuffle*. This function is similar to the 2-D shuffle function except that it performs a 3-way shuffling of the rows of the three input planes.

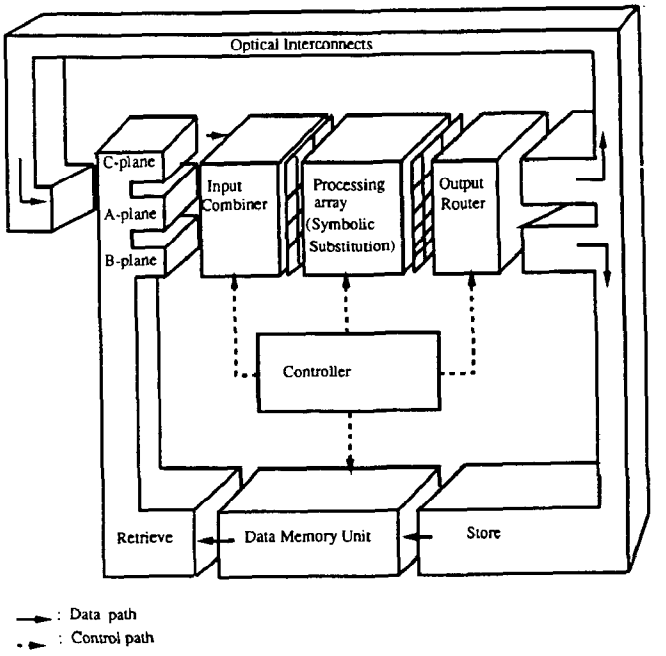

Figure 1: A Block diagram of a parallel optical architecture

The output router is responsible for directing the processed data to its appropriate destination. It also performs three data movement functions, namely, *feeding back* to the input combiner, a partial result such as a carry bit plane resulting from a full ADD operation, *sending* a final result to the data memory for storage, and *shifting* the output either in the X or Y direction by a variable number of pixels. This shift enables communication between pixels in the plane. By means of this spatial shifting, data can be moved among widely and at arbitrarily separated locations in the plane.

2.3 2-D Symbolic Substitution Rules for the Fundamental Operators

Symbolic substitution Logic[7] was introduced to take advantage of the massive parallelism and ultrahigh speed in optics. In this method, information is represented by optical patterns within a two-dimensional binary image and operators are seen as pattern transformation rules or substitution rules. Computation proceeds in transforming these patterns into other patterns. It consists of two processing phases: a *recognition phase* where the presence of a specific pattern, called the *search pattern*, is detected within a binary image and a *substitution phase*, where the present pattern is replaced by another pattern, called the *replacement pattern* according to a predefined substitution rule. All locations of the search pattern can be recognized in parallel (if present in the input image). Similarly, all replacements can be done in parallel.

In order to implement the fundamental operators (full ADD, logical AND, logical NOT) optically, we need an optical property to represent the logical 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 logical value 0 by two pixels dark-bright, and the logical value 1 by the inverse pattern, bright-dark as shown in Fig. 2a. The dark and bright pixels represent increasing levels of light intensity. In this coding scheme, a logical value is represented not only by the intensity of the bright pixel but also by its position, which has some implementation advantages[8].

We derive the substitution rules for the fundamental operators (full ADD, logical NOT and AND) from the truth-table specifications of each operator. The input combinations of the truth-tables represent the search patterns of the substitution rules, while the table entries represent the replacement patterns. The full ADD truth-table manipulates three bits which gives rise to eight combinations. If we put the bit symbols on top of each other, we produce eight substitution rules. Note that each bit is provided by a separate input plane. These bits have the same coordinates i, j in each plane. The grouping of bits of the same coordinates is accomplished by the 2-D 3-shuffle function described earlier. Similarly, the logical AND and NOT give rise to four and two substitution rules respectively as shown in Fig. 2(b)-(d). A total of 14 substitution rules is needed to implement the three fundamental operators.

In order to simultaneously process several substitution rules, the output of the input combiner is replicated a number of times equating the number of substitution rules to be activated at a given stage of computation. After the necessary substitutions, the outputs of all the active substitution rules are optically superimposed to form the processed result[9]. The optical superimposition represents a logical OR of light patterns where a bright pixel overwrites a dark pixel.

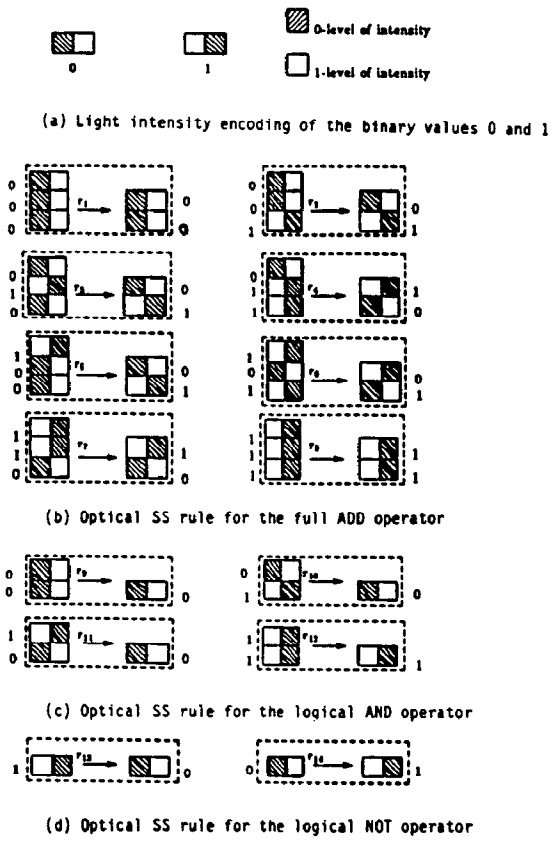

Figure 2: **Optical symbolic substitution rules for the primitive operators: full ADD, logical AND and logical NOT.**

3 Mapping Data-Parallel Algorithms on the Optical Architecture

The optical architecture exploits data parallelism at the hardware level, which enables it to process an entire data plane at once. To enforce this capability at the algorithm design level, we view the design and the mapping process as a hierarchical structure as shown in Fig. 3. At the highest level of the hierarchy is the application we wish to solve (i.e., signal and image processing, vision, radar application, etc). The next level identifies the various algorithms that can be used to compute these applications. These include matrix algebra, numerical transforms, solutions of partial differential equation, etc. A further analysis of these algorithms reveals that they share a common set of high-level operations, which we call *computing substructures*. These substructures can in turn be decomposed into a set of fundamental operations such as the full ADD, the logical AND and NOT.

The rationale behind this approach is that numerous data-parallel algorithms share common features such as localized operations, intensive computations, matrix operations and communication patterns. The high-level computing substructures are meant to capture these features. These substructures are directly mapped onto the hardware, and parallel algorithms are built upon these constructs so as to provide an efficient algorithm-architecture mapping. Due to the short length of the paper, we only focus on the implementation of a small sample of these computing substructures.

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 (B or C) $\leftarrow X$ is interpreted as data transfer from memory location X to input plane A (B or C). The notation $X \leftarrow Y$ 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. The notation $C \leftarrow 0$ 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.

3.1 2-D Addition/Subtraction

This substructure 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 $\mathbf{S} = \{ \mathbf{s}_{ij} \}$, where $\mathbf{s}_{ij} = \mathbf{x}_{ij} \pm \mathbf{y}_{ij}$ for $i, j = 1, \ldots, n$. Let $\mathbf{X} = X_{q-1}, X_{q-1}, \ldots, X_0$ a $n \times n$ $q-$bit planes, 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 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 substructure.

3.2 2-D Multiplication

This substructure refers to the multiplication of overlapping elements of two data planes. Let **X** and **Y** be as described previously, then the product **P** is a $2q-$bit planes $\mathbf{P} = P_{2q-1} P_{2q-2} \ldots P_0$, where $\mathbf{p}_{ij} = \mathbf{x}_{ij} \times \mathbf{y}_{ij}$. This substructure uses the logical AND and the full ADD operations. 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.

3.3 2-D Data Shifting

We define two substructures for shifting a data plane by a variable number of elements. The shift considered here is the logical shift, where columns (or rows) of 0s enter the opposite direction of the shift. Given two data planes **P** and **X**, we define a horizontal shift substructure, 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). The amount of shift α is applied to every bit plane P_i comprising the data plane **P**. 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 $\mathbf{P}$ by α columns and storing it in $\mathbf{X}$. Similarly, we define the vertical shifting operation, denoted by $\mathbf{V}_\alpha(\mathbf{P})$, to be the data plane $\mathbf{P}$ shifted along the Y-axis by α rows ($+\alpha$ for upward shift, and $-\alpha$ for downward shift).

3.4 Row/Column Accumulation

This refers to calculating the sum of all the elements of a data plane columnwise(rowwise). The initial plane $\mathbf{S}$ is split horizontally/vertically using the (vertical/horizontal) shift operations into two planes $\mathbf{X}$ and $\mathbf{Y}$, each with half the data entries of $\mathbf{S}$. Next, these planes are added using the 2-D addition procedure. This split and add process is repeated for $\log_2 n$ iterations, after which, the first row of $\mathbf{S}$ holds the sums of all the rows of $\mathbf{S}$. In other word, the elements of each column are accumulated and stored in the first entry of each column.

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

The Row-Sum and Column-Sum substructures can be combined to compute the sum of all the elements of a data plane. To find the sum of the elements of a data plane, we first apply the Row-Sum substructure to produce a single column of accumulated sums. Next, we apply the Column-Sum substructure to accumulate the elements of that single column.

3.5 Matrix Multiplication

As an example of algorithm mapping, we present a parallel algorithm for matrix multiplication which is based on the use of the computing substructures introduced above. 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:

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

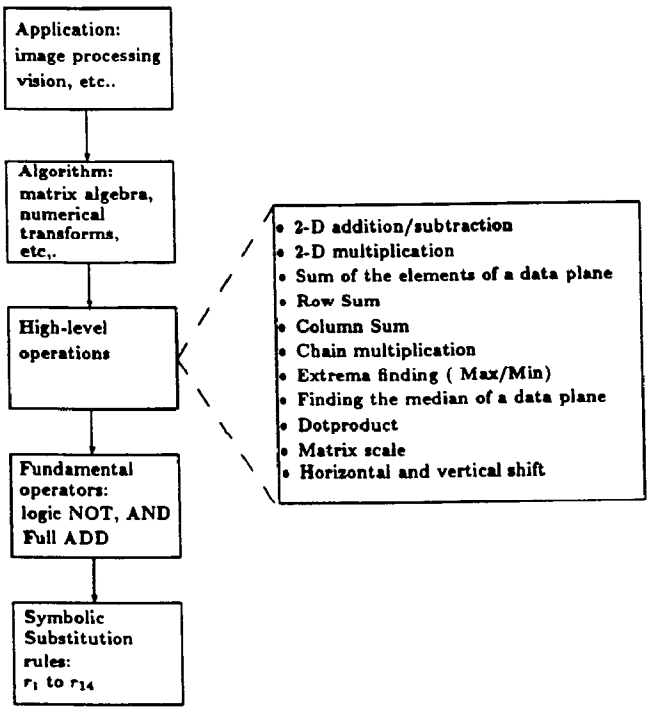

Figure 3: **A hierarchical top-down approach to the mapping of algorithms onto the optical architecture.**

We assume that the matrix $\mathbf{X}$ is stored as $n \times n$ matrices: $\mathbf{X}^n, \mathbf{X}^{n-1}, \ldots, \mathbf{X}^1$, where each matrix $\mathbf{X}^i$ is formed by transposing the i-th row of $\mathbf{X}$ and replicating it n times. Put differently, each column of $\mathbf{X}^i$ is equal to the i-th row of $\mathbf{X}$, for $i = 1, \ldots, n$. 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 $t_{ij} = x_{kj} y_{ij}$ for $i, j = 1, \ldots, n$. 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 k-th row of matrix $\mathbf{Z}$:

$$\mathbf{Z}_k = \sum_{i=1}^{i=n} t_{ij}, \quad j = 1, \ldots, n \tag{2}$$

where the first row of $\mathbf{Z}_k$ represents the k-th 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 product matrix $\mathbf{Z}$. Each matrix Z_k is shifted by $1 - k$ rows downward. All the shifted matrices are then added pairwise to produce the final matrix $\mathbf{Z}$. The complete 2-D matrix multiplication algorithm is as follows:

```
Procedure Matrix Multiply(Z,X,Y)
  begin
    for  k := 1 to n do
           T_k ←2-D Multiplication(X^k, Y);
           Z_k ← Row-Sum(T_k);
    endfor
    for  k := 1 to n do
           V_(1-k)(Z_k);
    endfor
    for k := 1 to n do
           Z ←2-D Addition(Z, Z_k);
    endfor
  end Matrix Multiply
```

It can be seen that the time complexity of the algorithm is logarithmic in n ($O(n \log n)$) as opposed to cubic in n ($O(n^3)$) for the conventional triple loop matrix multiplication.

4 Potential Performance

In this section, we estimate the theoretical performance of the optical architecture by evaluating several performance measures and compare them to the ones of existing SIMD array processors.

4.1 Asymptotic Performance

Hockney and Jesshope[10] have introduced two parameters $(r_\infty, n_{1/2})$ to give a first-order characterization of the *asymptotic performance* of a parallel computing system. The first parameter : r_∞ 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_∞ is evaluated as follows[10]:

$$r_\infty = \frac{n^2}{t_{array}} \tag{3}$$

where t_{array} is the time taken to execute one operation on all the PEs, this is usually taken as the processing rate, and n^2 is the total number of PEs. The *half-performance length*: $n_{1/2}$ characterizes the amount of hardware parallelism in a computer architecture. For a nonpipelined array processor, the factor $n_{1/2}$ is defined to be the vector length required to achieve half the maximum performance($r_\infty/2$)[10]. For an array processing system, $n_{1/2}$ is half the array size $n^2/2$ [10].

The NASA MPP[3] with an array of 128×128 PEs and 10 Mhz rate has achieved 6×10^9 8-bit operations /s. The CLIP[4] with a 96×96 array and a 25 μs cycle time has achieved 3.7×10^9 bit operations/s. The ICL DAP[5] with a 64×64 array and 0.2 μs cycle time, was described to achieve 10^8 32-bit operations/s. The Connection Machine[11] with 65,536 PEs and a 0.5 μs machine cycle time can achieve 13×10^{10} bit operations/s (the CM-2 model). For the optical case, if we assume that the processing unit is formed with NOR-gate arrays[8] of size 1000×1000, and 100 Mhz processing rate, $r_\infty = 10^{14}$ bit operations/s.

4.2 Communication and I/O Capabilities

Communication plays a crucial part in determining the overall system performance.

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 $O(n^2)$, since up to n^2 PEs can send and receive data at a time. The data transmission in the MPP and the CLIP is one column at a time, therefore their bandwidth is $O(n)$, the DAP on the other hand, transmits data in a row-parallel fashion which amounts to the same bandwidth factor $O(n)$. The Connection Machine has a maximum sustained communication bandwidth of $O(n^2)$.

The *diameter* is the maximum number of communication cycles (or links) needed for any two PEs to communicate. For the optical system, 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 Connection Machine, has a diameter of $O(\log_2 n)$.

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 $O(n)$ for the DAP, MPP, and CLIP, and $O(\log_2 n)$ for the Connection Machine. As far as the optical system is concerned, broadcasting a value in one PE to all other n^2-1 PEs can be done in $O(\log_2 n)$ steps.

In current implementations of the MPP and the CLIP, I/O is handled in column-parallel fashion while the DAP is row-parallel (data is loaded into the processing array one column or one row at a time). By contrast with the optical system, I/O activities are handled in plane-parallel manner. This

ability gives the optical system an I/O speedup of n, for an $n \times n$ input image, over the MPP, CLIP and the DAP which could be a tremendous speed advantage, considering the large potential value of n (eventually 1000).

5 Conclusions

In this paper, we have introduced a parallel architecture based on symbolic substitution logic and a hierarchical technique for mapping parallel algorithms onto it. The architecture is intended for applications that exhibit a high degree of data-level parallelism. The mapping technique is based on identifying basic computing substructures analogous to software routines that capture most of the characteristics of data-parallel algorithms. Parallel algorithms are then built upon these substructures. This will make the mapping process more systematic and hence efficient. The architecture is highly amenable to optical implementations using state-of-the-art optical and electro-optical technologies. A preliminary theoretical performance analysis of the proposed architecture has been conducted and it has been shown that the optical system has a great potential for outperforming conventional array processors. Hence, it may be a better alternative than current computing systems for high-speed data-parallel computing.

References

[1] K. Hwang, Z. Xu, and A. Louri, "Remps: An electro-optical supercomputer for parallel solution of PDE problems," *in Proc. 2nd Int'l. Conf. on Supercomputing (Santa Clara)*, May 5 - 8, 1987.

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

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

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

[5] S. F. Reddaway, "DAP - a distributed array processor," In *First Annual Symposium on Computer Architecture*, Florida, 1973.

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

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

[8] K. H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," *Applied Optics*, vol. 25, no. 18, pp. 3054 – 3060, 15 Sept 1986.

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

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

[11] Thinking Machine Corporation, "Connection machine model CM-2 technical summary," Technical Report Series HA87-4, Thinking Machine Corporation, 1986.