Three-Dimensional Optical Architecture

and Data-Parallel Algorithms for Massively Parallel Computing

The parallel nature of optics and free-space propagation, together with its freedom from communication interference, makes it ideal for designing massively parallel computers. Our architecture is highly amenable to optical implementations and aims at data-parallel applications.

Ahmed Louri

University of Arizona

Optics, due to its inherent parallelism and noninterfering communications, is under serious consideration for designs of massively parallel processing systems of the future. To contribute to this undertaking, designers at the University of Arizona's Department of Electrical and Computing Engineering explored a three-dimensional optical computing architecture under a grant from the US National Science Foundation.

This model—a single-instruction, multiple-data system, or SIMD—exploits spatial parallelism and processes 2D binary images as fundamental computational entities based on symbolic substitution logic. A better alternative than electronic mesh computers, this system effectively implements highly structured data-parallel algorithms, such as signal and image processing, partial differential equations, multidimensional numerical transforms, and numerical supercomputing. The model includes a hierarchical mapping technique that helps design the algorithms and maps them onto the proposed optical architecture.

We estimated the theoretical performance of the optical system and compared it with electronic SIMD array processors. Preliminary results show that the system provides greater computational throughput and efficiency than its electronic counterparts.

Background

The tremendous progress in science and technology introduced an increase in the processing of large amounts of data in real time in a wide variety of scientific applications. For example, real-time computer vision requires processing images of 1,000 × 1,000 data elements within a time frame of 16.7 milliseconds. This time suggests processing rates of 10 to 1,000 GOPS (10^9 operations per second) and input data rates approaching 1 Gbyte/s.

A common factor of these applications is a high degree of data parallelism in which simple arithmetic and logic operations must simultaneously take place across all data points.[1] Computing these applications with high-throughput rates requires massively parallel processing; however, traditional electronic technology faces major limitations in achieving massive parallelism. A key feature of this type of processing is the large amount of communication required among the processing elements (PEs). While the design of high-performance PEs has progressed significantly, the progress in designing high-performance interconnection networks has not been satisfactory. The major bottlenecks in today's massively parallel processing systems include the limited communication bandwidth and the lack of cost-effective means of achieving parallel I/O.[2-4]

0272-1732/91/0400-0024$01.00 © 1991 IEEE

Several researchers[5-9] suggest optics as a complementary technology for breaking major performance barriers faced by conventional electronic technology. Optics has many unique features that can be exploited for high-speed parallel processing. They include speed, parallelism, adequate communications, and architectural flexibility.

Optical systems are inherently multidimensional. Lenses, prisms, and mirrors can transfer planes comprising over a million data points simultaneously. This fact implies that a cost-effective parallel means of achieving I/O and multidimensional architectural topologies may be possible. The rate at which data moves through an optical processing system is essentially limited by the rate at which data enters the system and its detection at the output. The actual computation time consists mainly of light propagation through optical devices (provided that the switching rates of these active devices are comparable to optical signal propagation). Thus, we can obtain higher throughput and processing rates than we do with current systems.

Perhaps the most attractive feature of optics for massively parallel processing is communications.[4,10,11] Transmission of information via photons requires no physical conducting material, but relies on low-loss dielectric material for waveguide propagation or free space. As a result, optics-based interconnections potentially offer a freedom from mutual effects not afforded by electronic interconnections. This advantage becomes more important as the bandwidth of the interconnections increases, for the effect of mutual coupling associated with electrical interconnections is proportional to the frequency of the signals propagating on the interconnect lines. Therefore, optics-based communications offers higher temporal and spatial bandwidths.

The noninterfering nature of optical interconnections offers extra flexibility in routing, which in turn offers more architectural flexibility. Since electrical interconnections cannot cross, they must be routed under one another. Optical interconnections can cross one another without negative effects. Moreover, since optics-based interconnections require no mechanical contacts, we simply change the directions of optical beams to reroute interconnections. Various sources provide more details on optical interconnections.[10-14]

While the justifications for using optics for interconnections as well as mass storage are well established, the justification for using optics in digital processing remains in an embryonic stage, since digital optical device developments are in their infancy. However, if data must be converted to optical form to use an optics-based communication medium, using an optical computing engine might keep up with the rate of communications, without resorting to signal conversions (electronic-optical-electronic). These conversions cause major performance degradation and increase power consumption.

The possibility of using optics for building new parallel computing systems tailored to the requirements of data-intensive applications has been an objective of several researchers. Recent technological advances in optical devices raised hopes for the practical realization of new parallel optical computers. These advances include the development of compounds in multiple quantum wells[15] for high-efficiency injection lasers, the development of nonlinear materials for optical switching devices,[16-20] and the development of optical logic devices capable of implementing logic functions and serving as memory storage.[21-24]

The 3D optical architecture

The driving features of optical systems—the massive fine-grain parallelism and the high degree of communication flexibility—and our ability to move around large optical images of bright and dark spots with great ease using optical components suit many applications. Such applications require the processing of large amounts of structured data (multidimensional arrays) and favor the SIMD mode of computation. The attractiveness of these attributes is evidenced by the number of SIMD optical architectures that have been proposed in the past.[25-40] The optical model we present also exploits optics advantages for parallel processing.

The 3D optical architecture. Figure 1 depicts a block diagram of the basic components of the optical architecture. Unlike conventional computers that manipulate individual 0s and 1s as basic computational objects, the optical architecture

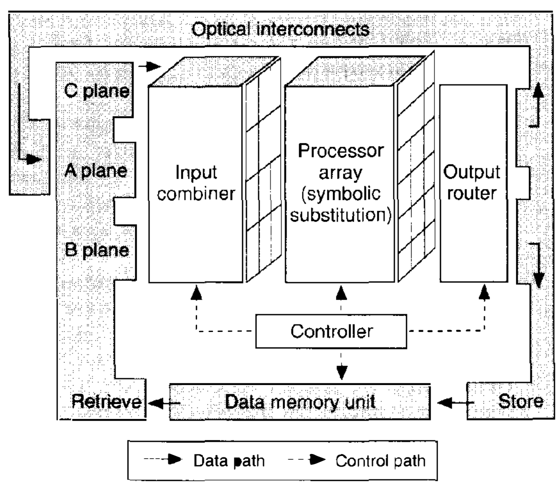


Figure 1. A schematic diagram of a 3D optical architecture for massively parallel computing.

manipulates bit planes as basic computational entities. Each bit plane i corresponds to a weight factor 2^i in the binary representation, and up to three bit planes can be processed simultaneously. For images of $n \times n$ elements, up to $3n^2$ operations process concurrently.

The heart of the architecture is the parallel processor 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 1-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 interconnections move the images around the system. We conceived the architecture as being built with optical hardware that manipulates entire images simultaneously both at I/O and processing. In this way, the 2D parallelism is sustained throughout various stages of the computation.

Processor array organization. The processor array operates in the SIMD mode of computation—the same operation applies to all data entries. Processing is based on optical symbolic substitution logic, or SSL,[41] described in detail later. The processor array uses three fundamental operators: a logical Not, a logical And, and a full Add as defined below, along with some other basic terms:

- **Definition 1.** We define a bit plane as $I \times I \to \{0,1\}$, where I is a set of integers. Hence, we denote it as $A = \{a_{i,j}\}$, where i, j represents the Cartesian coordinates of the binary value $a_{i,j} \in \{0,1\}$. For an $n \times n$ bit plane, $i, j = 1, \ldots, n$. We define a 0 plane as an $n \times n$ bit plane A, such that $a_{i,j} = 0$ for all $i, j = 1, \ldots, n$. Similarly, we define a 1 plane as an $n \times n$ bit plane such that $a_{i,j} = 1$ for all $i, j = 1, \ldots, n$.
- **Definition 2.** We define a data plane of length q as a stack of q-bit planes, denoted by the boldface notation $\mathbf{A} = A_{q-1}, A_{q-2}, \ldots, A_0$, where A_{q-1} and A_0 are the most significant and the least significant bit planes respectively. We will also denote $\mathbf{A} = \{\mathbf{a}_{ij}\}$, where $\mathbf{a}_{ij}$ is an integer number.
- **Definition 3.** We define a plane negation operator denoted by P-Not(A) as one that takes a bit plane A as input and produces an output bit plane A' as follows: P-Not(A) = A' where $A' = \{a'_{i,j}\}$ for $i, j = 1, \ldots, n$.
- **Definition 4.** We define a plane logical And operator, denoted by P-And, as one that takes two bit planes A,B as arguments and produces an output bit plane X as follows: P-And(A,B) = X, such that $x_{i,j} = a_{i,j} \wedge b_{i,j}$, where $\wedge$ is the conventional logical And applied to single bits.
- **Definition 5.** We define a plane full Add operator, denoted by P-Add, as one that adds three bit planes A,B,C and produces two output planes X and Y, defined as follows: P-Add(A,B,C) = X, Y where $x_{i,j} = a_{i,j} \oplus b_{i,j} \oplus c_{i,j}$ (sum bits) and $y_{i,j} = (a_{i,j} \wedge b_{i,j}) \vee (a_{i,j} \oplus b_{i,j}) \wedge c_{i,j}$ (carry bits).

The signs $\oplus$ and $\vee$ denote the conventional logical exclusive Or, and the logical Or operations respectively. The three fundamental operators constitute a complete logic and arithmetic set capable of computing any arithmetic or logic function using bit-serial algorithms.

Data-routing functions. A distinctive feature of the bit-plane architecture is that it provides parallel data movement along with the parallel processor array. We can load the binary images at the input in plane format, either from the data memory or from the external world such as a television scanner or a remote-sensing device (Figure 1 again).

The data enters the processor array through three input planes A,B,C, which are necessary for bit-serial arithmetic. Planes A and B hold the operands, while plane C holds the carry-bit plane required in bit-serial arithmetic. Depending on the primitive operator needed at a given computational step, the input combiner performs three data movement functions as elaborated next.

For the logical P-Not operator, the input combiner latches onto the relevant input plane with the data to be inverted into the processor array without any change in the spatial position of the data. The logical P-And operator is applied to two bit planes in which the logical And operation proceeds on overlapping bits from the two bit planes. The data movement function required in this case, the 2D perfect shuffle, performs the perfect shuffle[42] function on the rows of the two relevant input planes, while leaving the column positions unchanged.

Given two $n \times n$ input planes, $A = \{a_{i,j}\}$ and $B = \{b_{i,j}\}$, where $a_{i,j}, b_{i,j} \in \{0, 1\}$. The 2D perfect shuffle of A and B, denoted by 2D PS(A,B), results in a bit plane of size $2n \times n$ as follows:

$$\text{2D PS}(A,B) = X = \begin{bmatrix} x_{1,1} & \cdots & x_{1,n} \\ x_{2,1} & \cdots & x_{2,n} \\ x_{3,1} & \cdots & x_{3,n} \\ x_{4,1} & \cdots & x_{4,n} \\ \vdots & & \vdots \\ x_{2n-1,1} & \cdots & x_{2n-1,n} \\ x_{2n,1} & \cdots & x_{2n,n} \end{bmatrix} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,n} \\ b_{1,1} & \cdots & b_{1,n} \\ a_{2,1} & \cdots & a_{2,n} \\ b_{2,1} & \cdots & b_{2,n} \\ \vdots & & \vdots \\ a_{n,1} & \cdots & a_{n,n} \\ b_{n,1} & \cdots & b_{n,n} \end{bmatrix} \quad (1)$$

The P-Add operator adds the overlapping bits of three bit planes. The permutation function required for the data, the 2D 3-shuffle, is similar in function to the 2D perfect shuffle just described, except that the 2D 3-shuffle alternates rows of three bit planes. Given three input planes A,B,C, of size $n \times n$, the resulting 3-shuffled image D measures $3n \times n$ defined as:

$$
\text{2D 3-PS }(A,B,C) = D =
\begin{bmatrix}
d_{1,1} & \cdots & d_{1,n} \\
d_{2,1} & \cdots & d_{2,n} \\
d_{3,1} & \cdots & d_{3,n} \\
d_{4,1} & \cdots & d_{4,n} \\
d_{5,1} & \cdots & d_{5,n} \\
d_{6,1} & \cdots & d_{6,n} \\
 & \cdot & \\
 & \cdot & \\
 & \cdot & \\
d_{3n-2,1} & \cdots & d_{3n-2,n} \\
d_{3n-1,1} & \cdots & c_{3n-1,n} \\
d_{3n,1} & \cdots & d_{3n,n}
\end{bmatrix}
=
\begin{bmatrix}
a_{1,1} & \cdots & a_{1,n} \\
b_{1,1} & \cdots & b_{1,n} \\
c_{1,1} & \cdots & c_{1,n} \\
a_{2,1} & \cdots & a_{2,n} \\
b_{2,1} & \cdots & b_{2,n} \\
c_{2,1} & \cdots & c_{2,n} \\
 & \cdot & \\
 & \cdot & \\
 & \cdot & \\
a_{n,1} & \cdots & a_{n,n} \\
b_{n,1} & \cdots & b_{n,n} \\
c_{n,1} & \cdots & c_{n,n}
\end{bmatrix}
\quad (2)
$$

The shorthand expression 2D 3-PS (A,B,C) means the 2D 3-shuffle of planes A, B, and C. Note that the bits $(a_{1,1}, b_{1,1}, c_{1,1})$, $(a_{2,1}, b_{2,1}, c_{2,1})$, and so on become spatially adjacent after the 2D 3-shuffle permutation. The need for these data permutations will become clear when we present the optical implementation of the three fundamental operators.

The output router directs the processed data to its appropriate destination. It performs three data movement functions:

- feeding back to the input combiner a partial result needed in the next iteration, such as the carry bit plane resulting from a full Add operation;
- sending a final result to memory for storage; and
- shifting the processor array output in the X and Y directions by a programmable integer number of pixels.

The shifting functions enable communication between pixels. By means of this spatial shifting, data moves among widely and arbitrarily separated locations in the image. Furthermore, these shift functions add the flexibility of executing recursive data-parallel algorithms in which the same processing steps are applied to a reduced set of data at each iteration. The shifting functions considered here are logical shifts, in which rows or columns of zeros enter from the opposite direction of shift (rows of zeros for shifting along the Y axis, and columns of zeros along the X axis).

Symbolic substitution logic

SSL is an optical computing technique that was introduced[41] to take advantage of the massive parallelism and high speed in optics. In this method, optical patterns within a 2D binary image represent information. An optical pattern is a spatial arrangement of dark and bright spots corresponding to binary values 0 and 1. Using SSL, we can consider data as optical patterns, and processing as transformation rules. Computation proceeds by transforming optical patterns into other patterns according to predefined SSL rules. This computing technique is sensitive not only to the values of pixels carrying information but also to their spatial locations in the image.

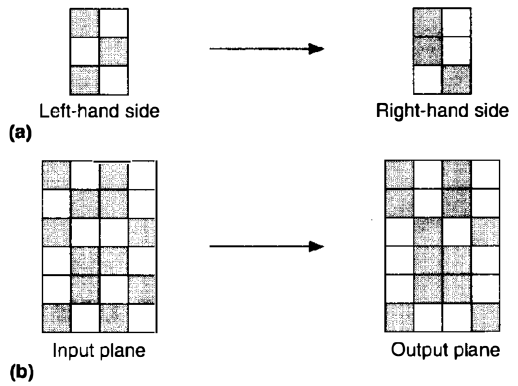


Figure 2. The concept of optical symbolic substitution logic: an example of an SSL rule (a) and its application to this input plane (b).

In its operation, SSL consists of two processing phases. A recognition phase detects the presence of a specific search pattern within an optical binary image. A following phase substitutes a different pattern in all locations in which the search pattern was found. Note that searching for all occurrences of the search pattern and substituting of the replacement pattern occur in parallel.

Figure 2a shows an example of an SSL rule and Figure 2b illustrates its application to a 2D image. The left-hand side pattern (search pattern) of the SSL rule is searched in the input image and then replaced by the right-hand side (replacement pattern). All locations of the search pattern are recognized in parallel.

Similarly, all replacements proceed in parallel. Since the input image can be very large (say, 1,000 × 1,000 pixels), over a million data items can be processed simultaneously. In optics, parallel search and parallel replacement of optically encoded data proceed relatively easily in parallel. Hence, an optical processor based on SSL introduces a huge amount of parallelism with little overhead of communication, data addressing, and loop indexing. Many researchers[31,44-54] have investigated optical implementations of the two processing phases of SSL.

Two-dimensional symbolic substitution rules. To implement the fundamental operators (P-Add, logical P-And, logical P-Not) optically, we need an optical property to represent the logical values 0 and 1. We can use several properties of light: intensity, polarization, and optical signal phase. One representation would encode the logic value 0 by two

continued on p. 65

continued from p. 27

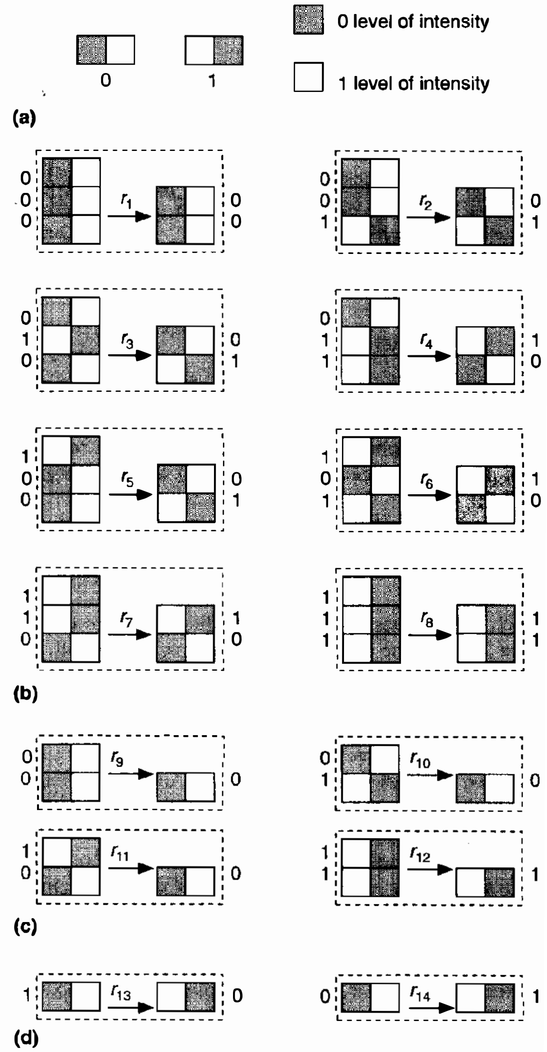


Figure 3. Light-intensity encoding of the binary values 0 and 1 (a); optical SSL rules for primitive operators: the full Add (b), the logical And (c), and the logical Not.

pixels, dark and bright, and the logic value 1 by the inverse pattern, bright and dark, as shown in Figure 3a. The dark and bright pixels represent increasing levels of light intensity. In this dual-rail coding scheme, the intensity of the bright pixel and its position represents a logic value, which has some implementation advantages.[43] We refer to the optical encoding of the binary values 0 and 1 as the fundamental patterns.

We next implement the fundamental operators as SSL rules, specifying how to manipulate information represented by optical patterns. These optical patterns are combinations of the fundamental patterns.

We derive the SSL rules from the truth-table specifications of each operator and the fundamental patterns shown in Figure 3a. The input combinations of the truth tables represent the search patterns of the SSL rules, while the table entries represent the replacement patterns. The P-Add operator truth table manipulates three bits, which gives rise to the eight combinations shown in Figure 3b. If we place the bit symbols on top of each other, we produce eight SSL rules. Note that a separate input plane provides each bit. These bits have the same coordinates *i, j* in each plane.

The 2D 3-shuffle function described earlier groups bits of the same coordinates. Similarly, the logical P-And and P-Not give rise to four and then two SSL rules, as shown in Figure 3c,d. Thus, we need a total of 14 SSL rules to implement the three fundamental operators.

Implementation of SSL. We briefly illustrate the implementation of one SSL rule (r_3 in Figure 3b) using an additive logic implementation method[43,55] to assist the conceptual understanding of this technique. The required optics have two parts: pattern recognition followed by pattern replacement. The recognition phase locates the presence of the search pattern in the input image, while the substitution phase uses this information to substitute the replacement pattern. The recognition optics applies a thresholding operation to a composite of shifted replicas of the input image. See Figure 4.

Using dual-rail coding (Figure 3a) and assuming dark-pixel recognition, the pattern-recognition optics replicate the input image as many times as there are dark pixels in the search pattern (Figure 4b). Then each replica shifts horizontally and/or vertically by an amount that brings a corresponding dark pixel to a designated reference pixel (Figure 4c). The shifted replicas are then superimposed optically (Figure 4d), and an optical Nor-gate array inverts the resulting image (Figure 4e). We mask the output of the Nor-gate array to eliminate erroneous overlapping patterns.

The masked image constitutes the recognition plane, since each bright pixel in it indicates the presence and location of the search pattern (Figure 4f). This image enters the pattern replacement phase (Figure 5), which replicates the recognition image for each bright pixel in the replacement pattern (Figure 5b). Then each replica shifts by an amount corresponding to the position of the bright pixel associated with it in the substitution pattern (Figure 5c). These shifted replicas then form the final image, shown in Figure 5d, through optical superimposition. We omitted optical hardware for image

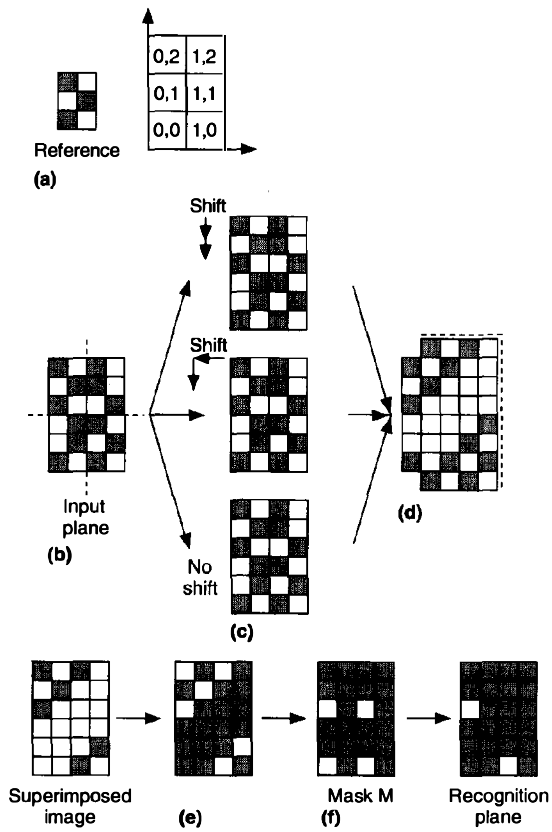


Figure 4. Processing steps needed to implement the recognition phase of SSL using additive logic: search pattern (a), replication (b), shift (c), superimposition (d), inversion (e), and masking (f).

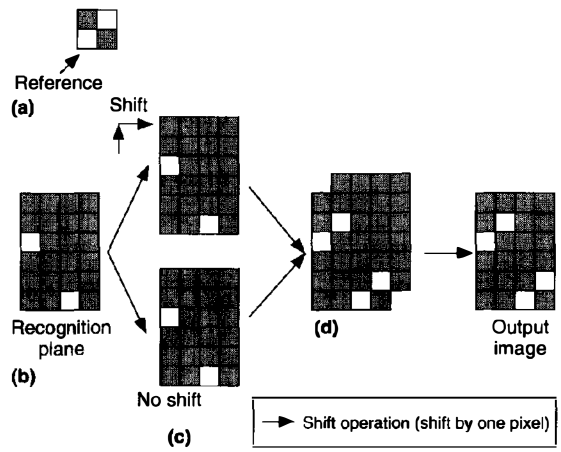


Figure 5. Optical processing steps needed to implement the substitution phase of SSL using additive logic: replacement pattern (a), replication (b), shift (c), and superimposition (d).

replication, shift, combination, and masking from Figure 5 for clarity. Brenner et al.[43] provides a detailed description of this particular method, including the optical setup.

Implementation of the processor array. Each of the three fundamental operators comprises several SSL rules that need to be fired simultaneously. To do so, we replicate the output of the input combiner a number of times equating the number of SSL rules to be activated at a given stage of computation.[43]

For example, to perform the P-Add operator, we need to replicate the input plane eight times corresponding to the eight SSL rules associated with the P-Add operator. Passive optical components such as beam splitters, or holographic elements can replicate the input. However, we would need a binary treelike replication scheme to equalize the optical path for each copy.

Each copy moves to one of the eight SSL rules r_1 to r_8 of Figure 3b. After the necessary substitutions, we optically superimpose the outputs of every active SSL rule to form the processed result. The optical superimposition represents a logical Or of light patterns in which a bright pixel overwrites a dark pixel. Thus we can implement the processor array with three modules, namely, an Add, an And, and a Not.

Each module comprises the SSL rules of the corresponding operator, as illustrated in Figure 6. A dynamic beam-steering element (an acousto-optic or electro-optic deflector along with some mirrors) under program control deflects the input plane to the desired module. Within each module, a static beam-steering element directs the processed output to the output router as shown in Figure 7. Recently, I[9,53] introduced an alternative dynamic method to implement the processor array that does not require a dynamic steering device.

Implementation of data-routing functions. The input combiner and output router assume only data movement functions; they do not require data processing. The input combiner assumes the three functions already described. Transmitting a bit plane to the processor array does not involve permutation of the data, and therefore we can use any imaging system.

The 2D perfect shuffle and 3-shuffle functions permute the row position of the input data. The literature proposes a wide variety of methods for realizing these functions.[56-60] These

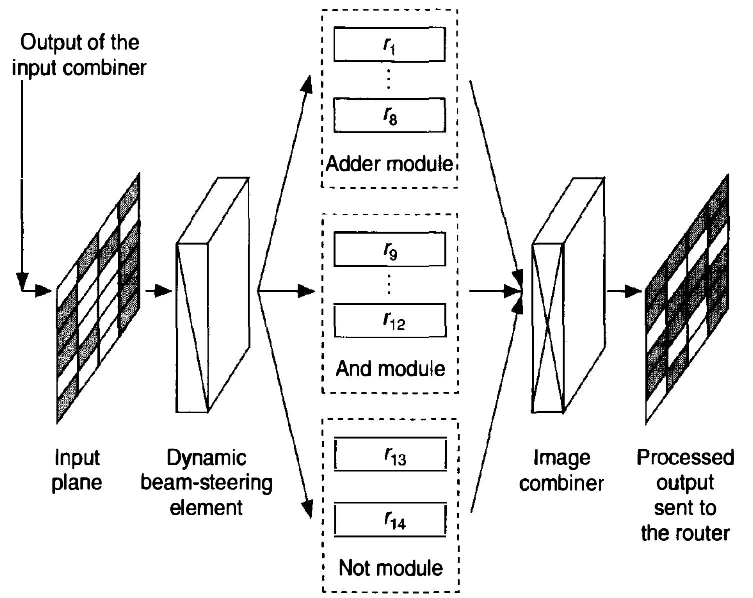


Figure 6. Logical structure of the optical processor array.

masking. The same principle can be extended to implement the 2D perfect shuffle and 3-shuffle functions described here.

Figure 8 on the next page illustrates an optical implementation of the 2D perfect shuffle, extended from Lohmann et al.[56] Given two bit planes, we implement the 2D perfect shuffle by first magnifying each bit plane to the total size of the two input planes and then achieve the 2D permutations by appropriate masking. Polarization-based devices controlling the flow of data control this setup. Similar considerations take place for the 3-shuffle function.

Polarization control devices can also implement the data movement functions assumed by the output router (feedback, routing data to memory, and shifting). We can use polarizing beam splitters and halfwave plates to control the pathways of the light beam. Two sources[61,62] suggest using birefringent prisms and acousto-optic cells to perform uniform shifts of a bit plane. Drabik and Lee[28] also proposed real-time holograms (discussed later) in photorefractive media to accomplish space-invariant shifts. The output-input optical feedback is one-to-one mapping that does not require permutation of the data; furthermore it does not need to be reconfigurable, which renders its optical implementation very simple. In fact an imaging system with some control (polarization-based devices) can implement the optical feedback.

schemes can be made controllable (whether to shuffle) using electro-optical and polarization-based devices.[61]

Lohmann et al.[56] proposed an optical setup for performing perfect shuffle permutations of a one-dimensional column of optical data using geometrical optical devices. These devices use the inherent speed of light and dissipate very little power. The basic principle is to divide the input into two upper and lower portions, magnify the two halves to the original size of the input, and then obtain the shuffled output by appropriate

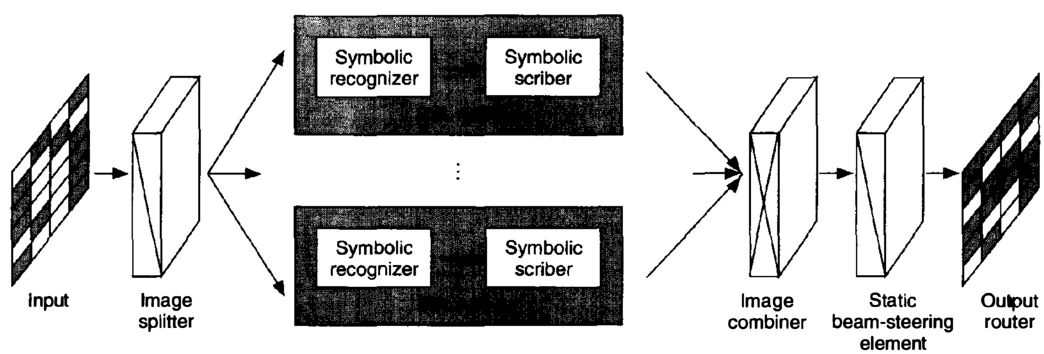


Figure 7. The logical implementation of each functional module in Figure 6.

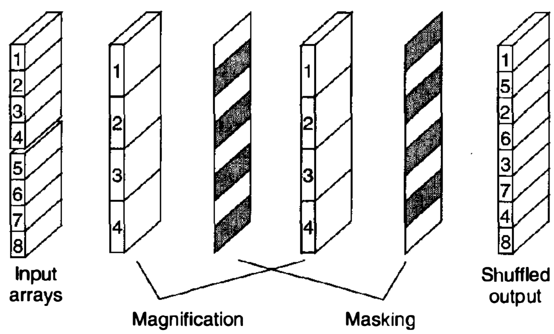


Figure 8. Optical implementation of the 2D perfect shuffle permutations. The numbers in the boxes represent the row positions within the plane. We achieve the 2D shuffle function by interlacing the upper and lower halves of the input.

Memory organization and control. To maintain the 2D processing throughout the system, the data memory must be addressable in bit planes. For single-plane storage, such as the input and output planes, and temporary buffers, we can use spatial light modulator (SLM) technology[63,64] and bistable optical latches.[15,16,21,22,65] SLMs are real-time optical active devices capable of spatially or temporally modifying some characteristic (polarization, phase, amplitude, intensity) of an optical signal beam. SLMs have a broad range of applications in optical information processing, including image amplification, inversion, incoherent-to-coherent conversion, analog multiplication, wavelength conversion, and short-term storage. However, SLMs would not be sufficient to build a data memory unit capable of holding a large number of bit planes.

Volume holograms, with the capability to store information in three dimensions, show the potential for a dramatic increase in optical storage density (10^{11} to 10^{12} bits). By recording stacked holograms in photorefractive crystals, we can achieve high storage density in random-access optical memory.[66-69] Bragg angular selectivity allows superpositions of holographic planes by slightly changing the angle between the recording beams.[62] Erasing such recorded holograms usually takes place by uniformly illuminating the storage crystal or by heating.

A holographic memory unit would be a superposition of volume holograms, one for each bit plane. Figure 9a schematically indicates how information moves into the optical memory in plane format. The bit planes from the output router form the data to be stored. Each bit plane is encoded in a volume hologram in which spatial frequencies are characteristic of a unique reference beam as well as the light distribution on the bit plane. The controller generates the memory address indicating where the data should be stored. The beam

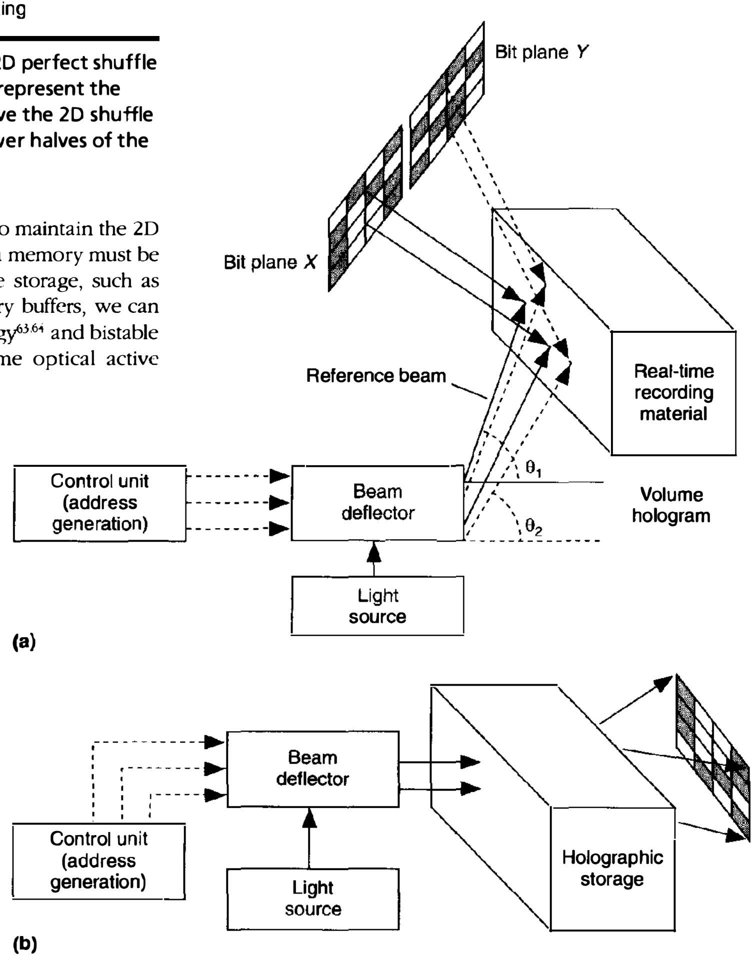


Figure 9. Real-time storage in a volume holographic medium (a) and retrieval from a volume holographic memory (b) of a light-encoded bit plane. The angular positions θ_1 and θ_2 correspond to the physical addresses of bit planes X and Y.

deflector device deflects the light by an amount proportional to the address generated by the controller. The interference of the image plane and the reference beam is recorded in the volume hologram.

Figure 9b depicts data retrieval from the holographic storage. The controller generates the address of the bit plane to be read and sends it to the beam deflector. This device in turn illuminates the volume holographic unit by a reference beam with an angular direction corresponding to the address generated by the controller. It should be noted that this angular position is the same as the one used to record the bit plane in the volume hologram.

Although this technology is very promising for real-time optical storage, it is far from being perfected for use in main memory applications. Several severe problems have to be solved before it becomes practical. These include cross talk between multiple holograms stored in the medium, low-diffraction efficiency of multiple volume holograms, fast selective writing and erasure of data, and data volatility. Nevertheless, Redfield and Hesselink[70] report making major efforts to overcome these limitations.

Another way of implementing the data memory would be to extend the optical disk technology. A 15-cm optical disk may have as many as 40,000 tracks and contain up to 10 Gbits of storage,[71,72] which corresponds to one thousand 1,000 × 1,000 bit planes. Recently, research efforts have concentrated on read/write optical disks based in magneto-optic combinations.[8,71]

A thin layer of vertically oriented material (a rare-earth, transition-metal alloy such as gadolinium and terbium) that is sandwiched between a transparent polycarbonate protective coating and a reflective substrate constitutes the recording medium. Initially, the orientation of all data bits is the same (corresponding to logic level 0). To write a bit on the disk, we use a high-intensity laser beam to heat the spot on the disk, dropping its coercivity with increasing temperature. This drop makes it possible to magnetize the heated spot easily with a weak magnetic field when applied.

Readout, based on the Kerr effect, results from using a lower beam to illuminate the location of each bit on the disk individually. Based on the reflected or transmitted intensity detected, the bit is decoded as logical 0 or 1. For writing, we can use the same optical setup for writing data on the disk. The key feature of magneto-optical disks is their use as parallel-access (read/write) memories. By illuminating a large portion of the disk during a write cycle or a read cycle, we can access several bits of information for writing or reading at once.[8,73]

Conceivably, an optical bit-plane addressable memory unit based on the transmissive optical disk technology could be built as shown in Figure 10. A controller sends the address of the bit plane to be read to the disk controller, which translates it into a mechanical motion of the disk. Meanwhile a different signal simultaneously moves to a laser deflection device that shines a light beam on the exact location of the data. Thus an entire bit plane can be read at once. According to the Faraday effect, the polarization of a laser beam passing through a magnetized medium experiences a rotation determined by the direction of the magnetic field of the material. Hence, magneto-optical disks can operate in the transmissive mode, provided that we take appropriate measures to eliminate diffraction effects.

Although the optical disk technology is commercially available for secondary storage (the Next computer uses a 256-Mbyte, read/write magneto-optical disk as a secondary storage device), it is limited. The main limitations of this technology for real-time main memory storage required for optical computing are the high access time and the relatively immature optical read/write schemes and devices (typical

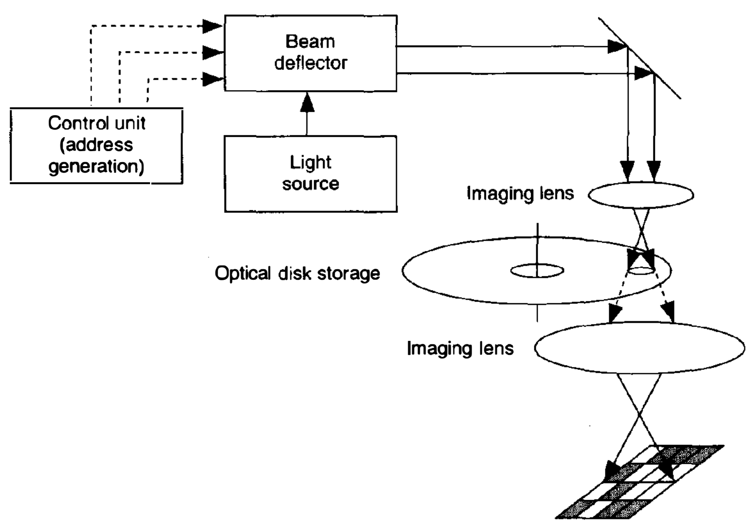


Figure 10. Using an optical disk as a real-time read/write memory device. We read the address of the bit plane from memory and access the desired bit plane of data at once.

access time for read/write optical disk is 100 ms).[8,62] The high access time results from the rotational latency or the delay incurred while waiting for the desired data to rotate to the proper location.

A potential solution to the mechanical motion problem may be the development of 2D optical beam deflectors that can provide tens of thousands of beam positions and very fast deflection time. In this manner, the disk will become stationary, and disk access will take place through the optical beam-deflector device. Henshaw and Todtenkopf[74] report several techniques under consideration: wavefront tilt, phased array, polarization modulation, interferometric switching, and photorefractive beam steering.

Another alternative for implementing the optical bit-plane addressable memory is the use of the two-photon-based, 3D optical memories.[75] Researchers claim that the two-photon effect provides a means of storing data into separate bit locations throughout the volume without affecting the neighboring bit locations. Thus the effect provides the highest storage density and the largest bandwidth of any existing storage device. In addition, this process permits a higher accessing speed than that found in volume holographic storage. These claims have been demonstrated.

2D computing substructures. The optical architecture exploits spatial parallelism at the hardware level, which enables it to process an entire data plane at once. This capability is opposed to task (or function) parallelism in which the data plane is decomposed into subplanes that are processed sequentially in a pipelined fashion.[76]

To enforce this capability at the algorithm design level, we view the design and mapping process as a hierarchical structure, as shown in Figure 11. At the highest level of the hierarchy is the application we wish to solve (signal and image processing, vision, radar). The next level identifies the various algorithms we can use to compute these applications. This level includes matrix algebra, numerical transforms, and solutions of partial differential equations among others. 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 operators such as the P-Add, the logical P-And, and P-Not.

The rationale behind this mapping technique is that most of the data-parallel algorithms share common attributes such as regularity, localized and intensive computations, recursiveness, and matrix operations. So the mapping process starts by identifying a set of substructures that captures most of these features. We then must efficiently map these substructures onto the architecture and build parallel algorithms upon them. This makes the mapping process more systematic and hence efficient. In this article we concentrate on a representative set of these computing substructures to show the methodology.

We will denote data transfer by A (B or C) $\leftarrow X_k$ by which we mean the transfer of bit plane X_k to the input plane A (B or C). Similarly, the expression $X \leftarrow Y$ denotes the transfer of bit plane Y to bit plane X. This step involves loading Y from memory, going through the processor array and the output router without any effect, and storing it in X.

We use $C \leftarrow 0$ plane to clear all the entries of input plane C. Similarly, $C \leftarrow 1$ plane indicates the setting of all the entries of input plane C to 1, and $P \leftarrow 0$ plane denotes a transfer of a zero-bit plane to memory location P. Loop constructs such as $k := a$ to $\log_2 n$ and indices such as a and $\log_2 n$, and parameter calculations should be interpreted as control instructions that the control unit executes.

2D addition/subtraction. This substructure refers to the addition (or subtraction) of corresponding elements of two $n \times n$ data planes $\mathbf{X}$ and $\mathbf{Y}$ of integers. The result is a data plane $\mathbf{S} = \{\mathbf{s}_{ij}\}$, whose elements $\mathbf{s}_{ij} = \mathbf{x}_{ij} \pm \mathbf{y}_{ij}$ for $i, j = 1, \ldots, n$. This step is similar to conventional matrix addition (subtraction). Let $\mathbf{X}$ be an $n \times n$ q-bit planes, $X_{q-1}, X_{q-2}, \ldots, X_0$. Here 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 $\mathbf{Y}$.

The 2D addition substructure adds the corresponding elements of the data planes bit serially, starting from the least

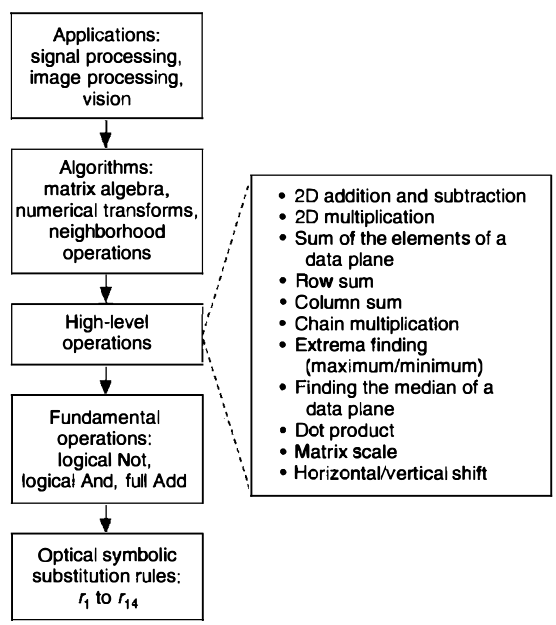


Figure 11. A top-down approach to mapping algorithms onto the bit-plane architecture.

significant bit planes. The substructure starts by initializing the C plane to zero and loading bit planes X_0, Y_0 into the A plane and the B plane respectively. The input combiner performs the 3-shuffle function of the three input planes. The processor array applies the P-Add SSL rules simultaneously to the resulting image. Thus the addition proceeds on all the operand pairs in parallel.

The sum bits are extracted from the output plane and stored in memory location S_0, and the carry bits are extracted and fed back to the C plane for the next iteration. Meanwhile the memory unit loads 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 $S_0, S_1, \ldots, S_q$ is stored as stacks of bit planes in the memory.

```
Procedure 2D Addition(X,Y)
  begin
     C <-- 0 plane ;
     /* initial carry is zero */
     for k := 0 to q - 1 do
        /* begin k loop */
           A <-X_k;
           /* load contents of X_k into A plane */
           B <-- Y_k;
           /* load contents of Y_k into B plane */
           S_k, C <-- P-Add(A,B,C)
           /* Perform the full addition of input planes
                 A, B, C*/
     endfor          /* end k loop */
     S_q <-- C;
     /* transfer the last carry to the most significant bit
                 plane S_q of S */
  end 2D Addition
```

The notation S_k, $C_{out} \leftarrow$ P-Add(A,B,C) in this procedure designates the addition of bit planes A and B together with the previous carry C_{in}. The sum bit plane moves to storage in S_k, and the resulting carry bit plane C_{out} returns to input plane C. We add two q-bit planes in q iterations, regardless of the number of operands to be added.

Representation of numbers in two's-complement form allows 2D subtraction by adding few additional steps to the 2D addition procedure. To subtract two data planes **X,Y**, we first form the two's complement of the subtrahend **Y** and then add it to **X** using the 2D addition procedure. We obtain the two's complement of data plane **Y** by first negating all the bit planes of **Y** ($Y_i \leftarrow Y'_i$ for $i = 0, \ldots, q-1$ using the P-Not operator). We then add it to a data plane whose least significant bit plane is a 1 plane; the remaining $q-1$ bit planes are all 0 planes.

2D multiplication. This operation refers to the multiplication of corresponding elements of two data planes. Let **X** and **Y** be $n \times n$ q-bit planes. The product **P** forms as $2q$-bit planes $\mathbf{P} = P_{2q-1}\, P_{2q-2}, \ldots, P_0$, where $\mathbf{P}_{ij} = \mathbf{x}_{ij} \times \mathbf{y}_{ij}$. As an example, let q be equal to 3 (we assume the same precision for both data planes to simplify the example). With $\mathbf{X} = X_2X_1X_0$ and $\mathbf{Y} = Y_2Y_1Y_0$, the resulting product then becomes $\mathbf{P} = P_5P_4P_3P_2P_1P_0$. The multiplication process starts by clearing the product bit planes to zero:

$$P_k \leftarrow 0 \text{ plane for } k = 0, \ldots, 5. \qquad (3)$$

This step represents the initial partial product $\mathbf{P}^0$ (the superscript 0 indicates the initial partial product). Next, we calculate the first partial product $\mathbf{P}^1$:

$$\begin{array}{ll}
P_0^1 & \leftarrow \text{P-And}(X_0, Y_0) \\
P_1^1 & \leftarrow \text{P-And}(X_1, Y_0) \\
P_2^1 & \leftarrow \text{P-And}(X_2, Y_0) \\
P_3^1 & \leftarrow P_3^0 \\
P_4^1 & \leftarrow P_4^0 \\
P_5^1 & \leftarrow P_5^0
\end{array}$$

The notation P_j^i ($j = 0, \ldots, 5$) means the jth bit plane of the ith partial product. The second partial product $\mathbf{P}^2$ is generated from $\mathbf{P}^1$ in the following manner:

$$\begin{array}{ll}
P_0^2 & \leftarrow P_0^1 \\
C & \leftarrow 0 \text{ plane} \\
T & \leftarrow \text{P-And}\,(X_0, Y_1) \\
P_1^2, C & \leftarrow \text{P-Add}\,(P_1^1, T, C) \\
T & \leftarrow \text{P-And}\,(X_1, Y_1) \\
P_2^2, C & \leftarrow \text{P-Add}\,(P_2^1, T, C) \\
T & \leftarrow \text{P-And}\,(X_2, Y_1) \\
P_3^2, P_4^2 & \leftarrow \text{P-Add}(P_3^1, T, C) \\
P_5^2 & \leftarrow P_5^1
\end{array}$$

The variable T here is a temporary bit plane. We obtain the final product after three iterations $\mathbf{P} = \mathbf{P}^3$. This product is produced as:

$$\begin{array}{ll}
P_0^3 & \leftarrow P_0^2 \\
P_1^3 & \leftarrow P_1^2 \\
C & \leftarrow 0 \text{ plane} \\
T & \leftarrow \text{P-And}(X_0, Y_2) \\
P_2^3, C & \leftarrow \text{P-Add}(P_2^2, T, C) \\
T & \leftarrow \text{P-And}(X_1, Y_2) \\
P_3^3, C & \leftarrow \text{P-Add}(P_3^2, T, C) \\
T & \leftarrow \text{P-And}(X_2, Y_2) \\
P_4^3, P_5^3 & \leftarrow \text{P-Add}(P_4^2, T, C)
\end{array}$$

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. The complete procedure is as follows:

Procedure 2D Multiplication(X,Y)
begin
for $k := 0$ **to** $2q - 1$ **do**
/* this loop clears the bit planes of the product to 0 */
$P_k \leftarrow 0$ plane;
for $l := 0$ **to** $q - 1$ **do**
/* this loop generates the successive partial products */
$C \leftarrow 0$ plane;
for $m := 0$ **to** $q - 1$ **do**
$A \leftarrow X_m$;
$B \leftarrow Y_l$;
$B \leftarrow$ P-And(A,B);
$A \leftarrow P_{m+l}$;
$P_{m+l}, C \leftarrow$ P-Add (A,B,C);
endfor; /* end of m loop */
$P_{q+l} \leftarrow C$;
endfor; /* end of l loop */
end 2D Multiplication

It takes q^2 full additions and q^2 logical And operations to generate the final product **P**. Therefore, the time complexity of the 2D multiplication is $O(q^2)$, independent of the number of pairs to be multiplied.

2D data-shifting operations. We define two operations for shifting a data plane by a variable number of pixels in either direction. The logical shift involves columns (or rows) of 0s that enter from the opposite side of the shift direction. Given a data plane **P** of q-bit planes $P_{q-1}, P_{q-2}, \ldots, P_0$, 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). See Figure 12a.

The amount of shift is sequentially applied to every bit plane P_i of the data plane **P**. The shifted plane can either be stored in itself or in a different data plane in memory. For the latter case, we introduce the notation $\mathbf{X} \leftarrow \mathbf{H}_\alpha(\mathbf{P})$, by which we mean that the shifted plane **P** is stored in plane **X**. Similarly, we define two other operations, denoted by $\mathbf{V}_\alpha(\mathbf{P})$ and $\mathbf{X} \leftarrow \mathbf{V}_\alpha(\mathbf{P})$ for vertical shifting. An illustration of vertical shift apperars in Figure 12b.

We now have an optical register-transfer language, comprising the 2D operations just described, with which we can describe parallel algorithms without referring to the machine hardware. Hereafter, the following shorthand notations **2D add(X, Y), 2D sub(X, Y), 2D multiply(X, Y),** $\mathbf{H}_\alpha(\mathbf{P})$, $\mathbf{V}_\alpha(\mathbf{P})$ denote the 2D addition, 2D subtraction, and 2D multiplication of the two data planes **X**, **Y**, and horizontal and vertical shift of a data plane **P** respectively.

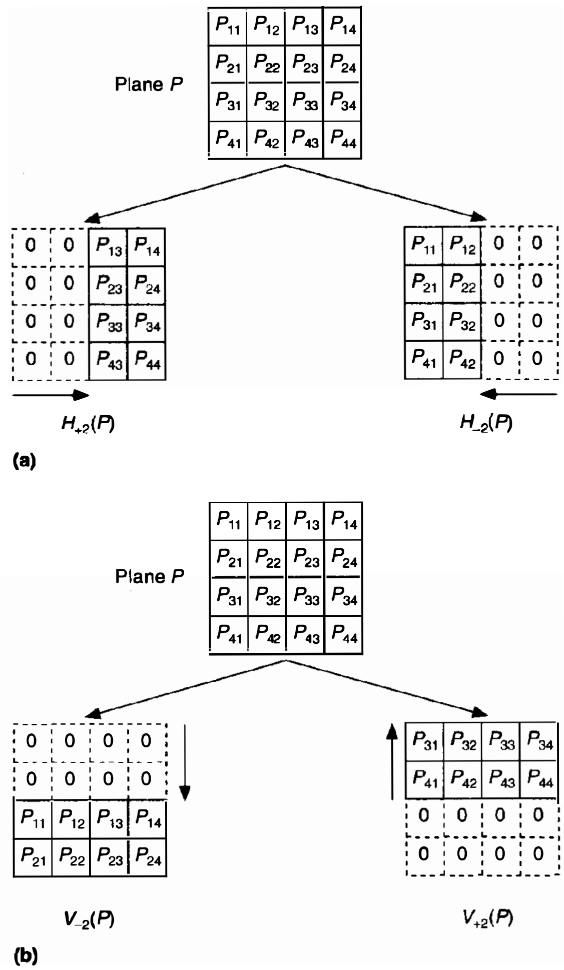


Figure 12. Horizontal (a) and vertical (b) shift functions (shown for $\alpha = \pm 2$).

Mapping data-parallel algorithms

The key feature of data-parallel algorithms is that their parallelism comes from simultaneous operations across large sets of data, rather than from multiple thread of control.[1] A large portion of scientific computing algorithms fall into this category because of the enormous amounts of structured data that need to be processed. Various sources propose SIMD machines as the most suitable class of computers for dealing with these algorithms. These machines include image processing systems such as the MPP,[75] the Clip,[34] and the DAP 610,[78] as well as fine-grained parallel systems such as the Connection Machine.[79]

The SIMD optical architecture potentially offers a larger array size (larger number of processing elements) than existing counterparts. In addition, its unrestricted interconnections give it a greater flexibility in handling data-parallel algorithms that require local as well as global communications. In the following, we show the mapping of several algorithms onto the architecture. We chose these algorithms to represent a broad range of complexity. They are also important key algorithms that occur as subproblems in larger programming tasks. Many more numerical algorithms have been mapped onto the optical architecture.[55]

Row/column accumulation. In calculating the sum of all the elements of a data plane columnwise (rowwise), we sum all the elements of a particular row; the final sum resides in the first entry of that row. Similarly, for column accumulation, we sum all the elements of a particular column, and the final sum occupies the first entry of that column. For a given data plane **S** of $n \times n$ elements, we proceed as follows. We split the initial plane **S** horizontally using the vertical shift operation (or vertically for rowwise accumulation using the horizontal shift operation) into two planes **X** and **Y**. Each plane contains half the data entries of **S**. Next we add these planes using the 2D addition substructure. We repeat this split-and-add process for $\log_2 n$ iterations, after which, the first row (first column) of **S** holds the accumulated sums of each column (row).

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

We can combine the Row-Sum and Column-Sum substructures to compute the sum of all the elements of a data plane. To find the sum of the elements of a data plane, say **S**, we first apply the Row-Sum substructure to produce one column of accumulated sums. Next, we apply the Column-Sum substructure to accumulate the elements of that column. Figure 13 shows an example of computing the sum of the 16 elements stored in a 4×4 data plane **S**. We compute the sum after $2\log_2 16 = 8$ steps and store it in location $\mathbf{s}_{1,1}$. Similarly, we can compute the product of all the elements of a data plane (chain multiplication) using the 2D multiplication and vertical and horizontal shift substructures. The product of n^2 elements with q precision each can be found in $O(q^2\log_2 n)$ time.

Matrix multiplication. Let **X**,**Y** be $n \times n$ matrices (assuming the same size for simplicity). Then their product $\mathbf{X} * \mathbf{Y} = \mathbf{Z}$ is an $n \times n$ matrix (the multiply asterisk denotes matrix multiply) whose elements are given by:

S_{11}	S_{12}	S_{13}	S_{14}
S_{21}	S_{22}	S_{23}	S_{24}
S_{31}	S_{32}	S_{33}	S_{34}
S_{41}	S_{42}	S_{43}	S_{44}

(a)

Step 1: Split S into X and Y

X

S_{11}	S_{12}	S_{13}	S_{14}
S_{21}	S_{22}	S_{23}	S_{24}
0	0	0	0
0	0	0	0

Y

S_{31}	S_{32}	S_{33}	S_{34}
S_{41}	S_{42}	S_{43}	S_{44}
0	0	0	0
0	0	0	0

Step 2: 2D add (x,y)

S'

S'_{11}	S'_{12}	S'_{13}	S'_{14}
S'_{21}	S'_{22}	S'_{23}	S'_{24}
0	0	0	0
0	0	0	0

Step 3: Split S' into X and Y

X

S'_{11}	S'_{12}	S'_{13}	S'_{14}
0	0	0	0
0	0	0	0
0	0	0	0

Y

S'_{21}	S'_{22}	S'_{23}	S'_{24}
0	0	0	0
0	0	0	0
0	0	0	0

Step 4: 2D add (x,y)

S''

S''_{11}	S''_{12}	S''_{13}	S''_{14}
0	0	0	0
0	0	0	0
0	0	0	0

Step 5: Split S'' into X and Y

X

S''_{11}	S''_{12}	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Y

S''_{11}	S''_{12}	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Step 6: 2D add (x,y)

S'''

S''_{11}	S''_{12}	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Step 7: Split S''' into X and Y

X

S'''_{11}	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Y

S'''_{12}	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

Step 8: 2D add (x,y)

S''''

S'''_{11}	0	0	0
0	0	0	0
0	0	0	0
0	0	0	0

(b)

Figure 13. Sum of 16 integers using the 2D addition substructure on the optical architecture: set S of 16 integers (a) and summation algorithm (b). $S''''_{11} = \Sigma S_{i,j}$ for $i, j = 1, \ldots, 4$.

$$\mathbf{z}_{ij} = \sum_{k=1}^{k=n} \mathbf{x}_{ik} \times \mathbf{y}_{kj}, \qquad i, j = 1, \ldots, n \qquad (4)$$

We assume matrix $\mathbf{X}$ is being transposed and stored as $\mathbf{X} = \mathbf{X}'_1\mathbf{X}'_2, \ldots, \mathbf{X}'_n$, where $\mathbf{X}'_i$ is an $n \times n$ matrix formed by replicating the ith column of the transposed matrix $\mathbf{X}'$ n times. The basic approach for computing the jth row of the product matrix $\mathbf{Z}$ is to first generate the point-by-point multiplication of the elements of matrix $\mathbf{Y}$ by the elements of matrix $\mathbf{X}'_j$, using the 2D multiplication substructure. Then we sum the columns of this matrix, using the Row-Sum substructure. As an example, consider matrices $\mathbf{X}$ and $\mathbf{Y}$ to be 3 × 3 of integers:

$$\mathbf{X} = \begin{bmatrix} \mathbf{x}_{11} & \mathbf{x}_{12} & \mathbf{x}_{13} \\ \mathbf{x}_{21} & \mathbf{x}_{22} & \mathbf{x}_{23} \\ \mathbf{x}_{31} & \mathbf{x}_{32} & \mathbf{x}_{33} \end{bmatrix} \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_{11} & \mathbf{y}_{12} & \mathbf{y}_{13} \\ \mathbf{y}_{21} & \mathbf{y}_{22} & \mathbf{y}_{23} \\ \mathbf{y}_{31} & \mathbf{y}_{32} & \mathbf{y}_{33} \end{bmatrix}$$

We assume matrix $\mathbf{X} = \mathbf{X}_1^t\mathbf{X}_2^t\mathbf{X}_3^t$ where $\mathbf{X}_i^t$ is the ith row of matrix $\mathbf{X}$ transposed and replicated as follows:

$$\mathbf{X}_1^t = \begin{bmatrix} \mathbf{x}_{11} & \mathbf{x}_{11} & \mathbf{x}_{11} \\ \mathbf{x}_{12} & \mathbf{x}_{12} & \mathbf{x}_{12} \\ \mathbf{x}_{13} & \mathbf{x}_{13} & \mathbf{x}_{13} \end{bmatrix} \quad \mathbf{X}_2^t = \begin{bmatrix} \mathbf{x}_{21} & \mathbf{x}_{21} & \mathbf{x}_{21} \\ \mathbf{x}_{22} & \mathbf{x}_{22} & \mathbf{x}_{22} \\ \mathbf{x}_{23} & \mathbf{x}_{23} & \mathbf{x}_{23} \end{bmatrix} \quad \mathbf{X}_3^t = \begin{bmatrix} \mathbf{x}_{31} & \mathbf{x}_{31} & \mathbf{x}_{31} \\ \mathbf{x}_{32} & \mathbf{x}_{32} & \mathbf{x}_{32} \\ \mathbf{x}_{33} & \mathbf{x}_{33} & \mathbf{x}_{33} \end{bmatrix}$$

The elementwise multiplication of $\mathbf{X}_1^t$ and $\mathbf{Y}$ using the 2D multiplication results in a matrix:

$$\mathbf{T} = \begin{bmatrix} \mathbf{t}_{11} & \mathbf{t}_{12} & \mathbf{t}_{13} \\ \mathbf{t}_{21} & \mathbf{t}_{22} & \mathbf{t}_{23} \\ \mathbf{t}_{31} & \mathbf{t}_{32} & \mathbf{t}_{33} \end{bmatrix} = \begin{bmatrix} \mathbf{x}_{11} \times \mathbf{y}_{11} & \mathbf{x}_{11} \times \mathbf{y}_{12} & \mathbf{x}_{11} \times \mathbf{y}_{13} \\ \mathbf{x}_{12} \times \mathbf{y}_{21} & \mathbf{x}_{12} \times \mathbf{y}_{22} & \mathbf{x}_{12} \times \mathbf{y}_{23} \\ \mathbf{x}_{13} \times \mathbf{y}_{31} & \mathbf{x}_{13} \times \mathbf{y}_{32} & \mathbf{x}_{13} \times \mathbf{y}_{33} \end{bmatrix}$$

We accumulate the rows of matrix $\mathbf{T}$ using the Row-Sum substructure, to generate a matrix $\mathbf{Z}^1$ whose first row is the first row of the product matrix $\mathbf{Z}$:

$$\textbf{Row-Sum(T)} = \mathbf{Z}^1 = \begin{bmatrix} \mathbf{z}^1_{11} & \mathbf{z}^1_{12} & \mathbf{z}^1_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where

$$\mathbf{z}^1_{1j} = \sum_{k=1}^{k=3} \mathbf{t}_{kj} = \sum_{k=1}^{k=3} \mathbf{x}_{1k} \times \mathbf{y}_{kj}.$$

In a similar manner, we use $\mathbf{X}_2^t$, and $\mathbf{X}_3^t$ to generate two matrices $\mathbf{Z}^2$ and $\mathbf{Z}^3$ respectively:

$$\mathbf{Z}^2 = \begin{bmatrix} \mathbf{z}^2_{11} & \mathbf{z}^2_{12} & \mathbf{z}^2_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad \mathbf{Z}^3 = \begin{bmatrix} \mathbf{z}^3_{11} & \mathbf{z}^3_{12} & \mathbf{z}^3_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

where

$$\mathbf{z}^2_{1j} = \sum_{k=1}^{k=3} \mathbf{x}_{2k} \times \mathbf{y}_{kj}$$

and

$$\mathbf{z}^3_{1j} = \sum_{k=1}^{k=3} \mathbf{x}_{3k} \times \mathbf{y}_{kj} \text{ for } j = 1,2,3.$$

Note that the first row of $\mathbf{Z}^2$ and the first row of $\mathbf{Z}^3$ are the second and last rows of the product matrix $\mathbf{Z}$ respectively. We generate the product matrix $\mathbf{Z}$ by shifting $\mathbf{Z}^2$ by one row, and $\mathbf{Z}^3$ by two rows downward, and sequentially adding all the three matrices $\mathbf{Z}^1$, $\mathbf{Z}^2$, $\mathbf{Z}^3$ using the 2D addition substructure:

$$\mathbf{Z} = \begin{bmatrix} \mathbf{z}^1_{11} & \mathbf{z}^1_{12} & \mathbf{z}^1_{13} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ \mathbf{z}^2_{11} & \mathbf{z}^2_{12} & \mathbf{z}^2_{13} \\ 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ \mathbf{z}^3_{11} & \mathbf{z}^3_{12} & \mathbf{z}^3_{13} \end{bmatrix}$$

The plus sign refers to the 2D addition substructure. The detailed algorithm is given as follows:

Procedure Matrix Multiply(Z,X,Y)
begin
  for k := 1 **to** n **do**
    T ← **2D multiply(** $\mathbf{X}^k$,**Y)**
    $\mathbf{Z}^k$ ← **Row-Sum(T)**
  endfor
  for k := 1 **to** n **do**
    $\mathbf{V}_{(1-k)}(\mathbf{Z}^k)$;
  endfor
  for k := 1 **to** n **do**
    Z ← **2D add(Z,**$\mathbf{Z}^k$**)**;
  endfor
end Matrix Multiply

The time complexity of the algorithm presented is $O[n(q \log_2 n + q^2)]$, where q is the operand length. Thus, the time complexity of this algorithm is $O(n \log_2 n)$, as opposed to $O(n^3)$ for the conventional triple-loop matrix multiplication.

Extrema finding. Extrema finding is the search for the maximum (minimum) value of a set of elements. The computations involved are global since the result is a function of all the data entries. These measures are very useful in many applications including signal/image processing, searching, and sorting.

Let the set of data be represented by a data plane **S** of n^2 elements. The algorithm for finding the maximum value of **S** proceeds by folding **S** repeatedly in half and selecting the largest value of overlapping elements from each half at each step. Initially we fold **S** in half by storing its first $n/2$ rows in the first $n/2$ rows of a matrix, say $\mathbf{S}_u$, and its last $n/2$ rows in the first rows of a second matrix $\mathbf{S}_l$. We subtract $\mathbf{S}_l$ from $\mathbf{S}_u$ using the 2D subtraction substructure, retain the maximum value of each pair of elements, and store it back into $\mathbf{S}$. The new $\mathbf{S}$ contains half the data points of the original set.

We find the maximum value by repeating the folding and subtraction process for $2\log_2 n$. During the first $\log_2 n$ iterations, we fold the data plane $\mathbf{S}$ along the horizontal direction at each iteration. At the end of the first $\log_2 n$ iterations, each entry in the first row of $\mathbf{S}$ holds the maximum value of the corresponding column. Next, we fold $\mathbf{S}$ vertically and perform comparison for another $\log_2 n$ iterations after which the maximum value of the entire data plane $\mathbf{S}$ is located in the first entry $\mathbf{s}_{11}$.

```
Procedure Maximum(S,Su,Sl)
  begin
    for k = 1 to log2n do
        α := n/2^k;
        β := Σ_{i=1}^{i=k} (n/2^k);
        Su ← V-β(S);
        V+β Su ;
        Sl ← V+α(S);
        T ← 2D sub(Su, Sl);
        if Tq-1 = 0 then S ← Su else S ← Sl;
    endfor
    for k = 1 to log2n do
        α := n/2^k;
        β := Σ_{i=1}^{i=k} (n/2^k);
        Su ← H-β(S);
        H+β(Su );
        Sl ← H-α(S);
        T ← 2D sub(Su, Sl);
        if Tq-1 = 0 then S ← Su else S ← Sl;
  endfor          /*end k loop */
end Maximum
```

In this procedure, $\mathbf{T} = T_{q-1}T_{q-2}, \ldots, T_0$ is a temporary data plane used to hold the subtraction result. T_{q-1} is the most significant bit plane of $\mathbf{T}$. Since we are using two's-complement subtraction, the most significant bit of the subtraction result indicates the relative magnitude of the operands. Each entry $T_{q-1}(ij)$ represents the sign bit of the subtraction operation $\mathbf{S}_u(ij) - \mathbf{S}_l(ij)$. Therefore, for $T_{q-1}(ij) = 0$, $\mathbf{S}_u(ij)$ is greater than or equal to $\mathbf{S}_l(ij)$; otherwise $\mathbf{S}_l(ij)$ is greater than $\mathbf{S}_u(ij)$. We achieve the selection of the largest value (noted by the simple conditional statement: if $T_{q-1} = 0$, then $\mathbf{S} \leftarrow \mathbf{S}_u$ else $\mathbf{S} \leftarrow \mathbf{S}_l$) by the following Boolean expression:

$$S_k = \text{P-And(P-Not}(T_{q-1}), S_{uk}) \vee \text{P-And}(T_{q-1}, S_l) \quad \text{for } k = 0, \ldots, q. \tag{5}$$

where the sign $\vee$ is the logical Or and S_{uk}, S_{lk} refers to the kth-bit plane of arrays $\mathbf{S}_u$ and $\mathbf{S}_l$ respectively. We can express Equation 5 using only logical P-Not and P-And operators (using De Morgan's theorem $A \vee B = \overline{\bar{A} \wedge \bar{B}}$:

$$S_k = \text{P-Not (P-And (P-Not (P-And(P-not}(T_{q-1}), S_{uk})), \text{P-Not(P-And}(T_{q-1}, S_{lk})))) \tag{6}$$

Several iterations through the system carry out Equation 6. The time complexity of the algorithm is $\Theta(q\log_2 n)$. This algorithm is representative of many neighborhood algorithms such as those that find the minimum, the average, the median, the sum of a data plane, histogramming, counting, and so on. All can be implemented in $\Theta(q\log_2 n)$ time. For example, we can apply the same algorithm to find the minimum of a set. In this case, we retain the minimum value at each iteration.

Projected performance

We estimated the theoretical performance of the optical architecture by evaluating several performance measures and compared them to the ones of existing SIMD array processors. The optical implementation of SSL follows the method briefly described earlier.[43,55]

We used the following key parameters in the analysis:

- T_p: Propagation time of a light beam through passive optical devices such as lenses, beam splitters, and holograms required for image replication and spatial shifting;
- T_{sw}: Response time of the optical switching devices such as the optical Nor-gate arrays, and the optical latches used in the processor array;
- T_{proc}: Response time of the optical processor array;
- T_{comb}: Response time of the input combiner;
- T_{rout}: Response time of the output router; and
- q: Precision of the operands (word length).

Optical cycle time. The optical cycle time, denoted by T_{array}, equals the time elapsed between inputting the data in the input planes and outputting the result at the output router. This time includes formatting the data at the input combiner, processing the formatted data in the processor array, and routing it to the appropriate destination. Therefore:

$$T_{array} = T_{comb} + T_{proc} + T_{rout} \tag{7}$$

Please note that the definition of optical cycle time does not include memory access. Currently, we do not have a quantitative measure of response time of plane-addressable optical memories to include it in the analysis. The time needed to process data in the processor array (T_{proc}), using SSL hardware, is attributed to the time needed to:

1) deflect the formatted image to the active module (only one module can be active at a time);

2) replicate the formatted image, spatial shift the replicas, and combine the shifted copies;
3) activate the Nor-gate array for inverting the light intensity of the combined image;
4) propagate the inverted image though the mask;
5) replicate, shift, and combine images for the substitution phase; and
6) combine the outputs of all the activated SSL rules of the active module.

Using the above parameters, we derive T_{proc}:

$$T_{proc} = \overbrace{T_{sw}}^{(1)} + \overbrace{3T_p}^{(2)} + \overbrace{T_{sw}}^{(3)} + \overbrace{T_p + T_{sw}}^{(4)} + \overbrace{3T_p + T_{sw}}^{(5)} + \overbrace{T_p + T_{sw}}^{(6)} \quad (8)$$

The numbers over the braces indicate the times needed to accomplish each subtask as enumerated above. T_p can be in the range of 0.1 to 1 ns (light propagates at 1 ft/ns in free space). Presently, the status of active optical devices is much less mature than the passive components, and is the subject of intense research. The available optical switching devices have response times orders-of-magnitude higher than T_p (see Table 1). Thus, the dominant factor in Equation 8 becomes the switching time of the optical device used, T_{sw}. Consequently, we approximate T_{proc} as $T_{proc} = 5T_{sw}$.

The input combiner and the output router require optical switching devices along with some passive devices. We can also approximate the time spent in these units by the switching time of the active optical devices. We assume that $T_{comb} = T_{rout} = T_{sw}$. Then the total optical processing time is approximated by:

$$T_{array} = 7T_{sw} \quad (9)$$

Table 1 lists estimated values for the optical cycle time using different optical logic devices. The cycle time of the optical architecture is very much dependent upon the maturity of optical switching devices and their comparative performance with respect to electronic devices. Several approaches are being pursued for performing optical logic. One approach involves the adaptation of the SLM technology to optical logic. While current SLMs can be fabricated in large 2D arrays, their major drawback is the extremely slow response time (≥ 1 ms). However, considerable research is being performed to increase their spatial resolution (number of resolvable points on the array) and reduce their response time. The rationale is that large arrays of these devices will be fabricated to exploit the spatial parallelism of optics, and therefore compensate for their current slow response time.[24]

Another approach for realizing optical components capable of performing logic is to optimize the device from the beginning for digital operations. The recent emergence of the quantum-well self-electro-optic effect device—or SEED, and its derivatives (S-SEED, T-SEED, D-SEED)—is one such product.[18,19] We can use the SEED devices to realize both logic operations such as Nor, Or, And, and Nand as well as to store (S-R latches).[22] The family of SEED devices seems to be easy to use, capable of high-speed, low-energy operation, capable of being fabricated in 2D format, and cascadable.[36] Streibl et al.[60] reports that several hundred devices have been fabricated in 2D arrays on a chip and demonstrated with good uniformity. SEEDs show great potential for large parallel, digital optical computing systems (the optical architecture) because they are integrable low-energy devices (1-20 femtojoules per square centimeter), cascadable, and operate at high speed.

Table 1. Estimation of optical cycle time with respect to the physical characteristics of optical switching devices.*

Optical logic devices	Array size Available	Array size Future	Response time for entire array Available	Response time for entire array Future	Switching power Available	Switching power Future	Cycle time (T_{array}) Available	Cycle time (T_{array}) Future
SLMs, LCLV[24,84]	500 × 500	500 × 500	20 ms	50 μs	μj/sq cm	NA**	140 ms	350 μs
Sight modulators[24,85]	128 × 128	1,024 × 1,024	1 μs	NA	NA	NA	7 μs	NA
FLC-SLMs[86,87]	400 × 400	10,000 × 10,000	155 μs	15 μs	0.1 pj/pixel	NA	1 ms	NA
SEED[19,22,23]	64 × 64 (within 10 years[13])	10,000 × 10,000	30 ns	1 ns	4 fj /sq μm	NA	210 ns	7 ns
Optical logic etalons[15,82,83]	100 × 100	10,000 × 10,000	150 ps	1 ps	100 pj	NA	1 ns	NA

* 1990 data
** Not available

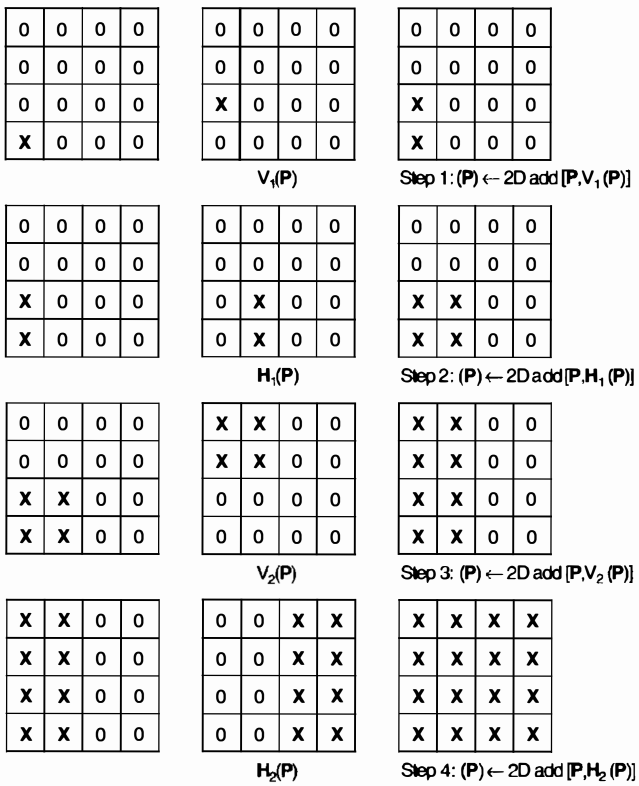


Figure 14. Broadcasting example on the optical architecture: The value *x* in the lower left-corner processing element is broadcast to all other PEs in the array in $2\log_2 4 = 4$ steps. The original plane P appears in the upper left array.

Optical resonators are another family under this approach intended for optical logic.[81,82] Two similar bistable devices, optical logic etalons (OLEs), and interference filters, both based on the Fabry-Perot resonator, are being actively pursued.[20,83] Although these devices can be fabricated in large 2D arrays (10,000 × 10,000), and have a comparatively small response time, they are not cascadable at the present time.

OLEs are pulsed devices. In their operation, these devices require two wavelengths, one for the input signal and one for a bias signal (clock cycle). The two inputs, data and clock cycle, are separated in both time and wavelength. The modulated output signal has the same wavelength as the bias signal, and therefore the input and output signals have different wavelengths.[15] Hinton reports research efforts under way to implement a device composed of two OLE devices interconnected in such a way that the second OLE changes the wavelength of the output signal from the first device to the original input wavelength.[13]

Communication and I/O capabilities. Communication plays a crucial part in determining the performance of a parallel computer. Many communication metrics appear in the literature.[88,89] In the following analysis, we choose the most widely used.

The 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 receive and send data at a time. Data transmits in the MPP and the Clip at 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 features 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 case, this factor is 1, since we allow any number of shifts in either direction 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 is essentially wired in the pattern of a Boolean n cube, therefore its diameter is $O(\log_2 n)$.

Broadcasting sends the value in a certain PE to all the other PEs. The amount of communication cycles needed to achieve this is considered a measure of communication performance. This value is $O(n)$ for the MPP, DAP, 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 takes $O(\log_2 n)$ steps. The example in Figure 14 shows the broadcasting of a value **x** residing in the lower left-corner PE to all other PEs in $2\log_2 4 = 4$ steps.

In current implementations of the MPP and the Clip, I/O processes in column-parallel fashion, while the row-parallel DAP loads data into the processor array one column or one row at a time. In contrast with the optical system, I/O activities process in a plane-parallel manner. This capability gives the optical system an I/O speedup of n, for an $n \times n$ input image, over the MPP, Clip, and the DAP. It could be a tremendous speed advantage, considering the large potential value of n. We note that the Connection Machine can also handle plane-parallel data, loading through a very expensive I/O system called the data vault.

Table 2 (on the next page) summarizes the various performance measures. Note that the processing time listed for the

Table 2. Performance comparison of the optical architecture with electronic array processors.

Computing system	Performance metrics: No. of PEs	Cycle time (ns)	Diameter	Bandwidth*	Broadcasting
Optical architecture	$10{,}000 \times 10{,}000$ **	1	1	n^2	$O(\log_2 n)$
MPP[13]	128×128	100	$O(n)$	n	$O(n)$
DAP 610[78]	64×64	100	$O(n)$	n	$O(n)$
Clip[34]	96×96	2,500	$O(n)$	n	$O(n)$
Connection Machine[90]	65,536	100	$O(\log_2 n)$	n^2	$O(\log_2 n)$

*We assume the total number of PEs for the optical architecture to be the projected size of the SEED device.
**The bandwidth column refers to the communication bandwidth.

optical system is the estimated time when the implementation uses projected values for the SEED devices.[13]

Table 3 summarizes some performance measures for the parallel algorithms we have presented. These measures are theoretically evaluated by orders of magnitude only. From Table 3 we see that the theoretical efficiency of algorithms that are totally parallel and use every PE during the execution (such as 2D addition, 2D subtraction, and 2D multiplication) approaches 100 percent. (We assume the dimension of the problem matches the processor array size.) However, the theoretical efficiency is less than ideal for recursive algorithms because at each iteration only a fraction of the PEs are active. These algorithms include extrema finding, summation, and chain multiplication.

Table 3. Estimated performance of data-parallel algorithms on the proposed optical architecture.

Algorithm	Time complexity	Speedup*	Efficiency (speedup/ number of PEs)
2D addition ($n \times n$ arrays)	$O(q)$**	$\frac{qn^2}{q} = n^2$	1
2D multiplication ($n \times n$ arrays)	$O(q^2)$	$\frac{q^2n^2}{q^2} = n^2$	1
Sum of n^2 integers	$O(q\log_2 n)$	$\frac{q(n^2-1)}{q\log_2 n} \approx \frac{n^2}{\log_2 n}$	$\frac{1}{\log_2 n}$
Chain multiplication (n^2 number)	$O(q^2\log_2 n)$	$\frac{q^2(n^2-1)}{q^2\log_2 n} \approx \frac{n^2}{\log_2 n}$	$\frac{1}{\log_2 n}$
Matrix multiplication ($n \times n$ matrices	$O[n(q\log_2 n + q^2)]$	$\approx \frac{q^2n^3}{n(q\log_2 n+q^2)}$	$\approx \frac{q^2}{(q\log_2 n+q^2)}$
Maximum of n^2 integers	$O(q\log_2 n)$	$\frac{q(n^2-1)}{q\log_2 n}$	$\approx \frac{1}{\log_2 n}$

* The speedup is the ratio of the execution time on one 1-bit processing element to time taken on n^2 PEs.
** q is the precision (or operand length).

THE EVER-INCREASING DEMANDS for speed, throughput, and computing power, coupled with the limitations of electronic technology for massively parallel systems, brought about a major research impetus to explore optical technology for developing future massively parallel computers. Optics offers several advantages, including parallelism, high bandwidths, and noninterfering communications. Researchers seriously consider these advantages for breaking major bottlenecks imposed by conventional technology in storage, communications, and processing for high-performance computers. Justifications for the near-term inclusion of optics in parallel computing systems are already established for mass storage and interconnections. Optical disks begin to replace magnetic disks in some commercial systems. Moreover, optics is being used for processor-to-processor and processor-to-memory interconnections. Buses, optical backplanes, crossbar switches, reconfigurable interconnections (neural nets), and clock distribution networks are but a few applications of optical interconnections.

With the advances in optical storage and optical interconnections, optical information processing systems will have a major impact on overall system performance. If the data is already being stored and transmitted in optical form, optical computing processors might be the best alternative for processing the data rather than resorting to optical-to-electronic-to-optical conversions, which are major sources of performance degradation. Thus a more uniform technology for storage, communication, and processing of data in optical form will significantly impact the future of high-performance computing systems.

Digital optical information processing is the least developed at this time due to the immaturity of optical switching and logic devices. These devices are in their first generations. Considerable research and development efforts presently under way will lead to optical devices with lower switching energy, higher switching speeds, and higher resolutions. As these devices mature, optical computing systems will be highly competitive with existing electronic systems.

Here, we contribute to the ongoing efforts in building the foundations of new optical computing systems. We introduced a 3D optical computing architecture based on symbolic substitution logic. Although we focused on architectural and algorithmic issues plus some performance projections, we provide an extensive and up-to-date reference list that covers all aspects of the field. We showed that with a few symbolic substitution rules one can build a massively parallel optical computer. After introducing a hierarchical mapping technique for mapping parallel algorithms onto the optical computing model, we mapped several parallel algorithms onto the architecture. We chose these algorithms to represent a wide range of computational complexity.

We have assessed the theoretical performance of the proposed optical system. Although the system is not competitive at the present time with electronic array processors, it is quite attractive. It can potentially deliver a throughput at least 100 times higher than that of its electronic counterparts (owing to its multidimensional nature and high speed). Moreover, the communication flexibility and the parallel I/O of the optical system seems to be unmatchable with electronic array processors.

This preliminary performance analysis suggests that the proposed optical system is potentially a better alternative than current computing systems. The best applications are those that require the processing of large amounts of structured data such as remote sensing, signal/image processing, vision, weather modeling, and seismic data processing. We projected the performance under conservative assumptions and did not include power and budget breakdowns. A thorough power and cost analysis will require exact characteristics of the optical devices chosen for implementing different parts of the system. Such optical devices are just beginning to emerge. A more accurate assessment is needed once these optical devices are in wide use.

Acknowledgments

I thank the anonymous referees for their objectivity and constructive comments and gratefully acknowledge their help in the revision process. I also thank Kai Hwang for his valuable suggestions and technical review of the manuscript.

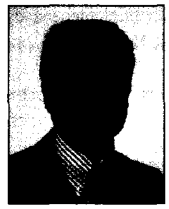
NSF Grant MIP-8909326 supported this research. I presented a preliminary version of this work at the 15th International Symposium on Computer Architecture in Honolulu on June 1, 1988.

References

1. W.D. Hillis and G.L. Steele, Jr., "Data Parallel Algorithms," *Comm. ACM*, Vol. 29, No. 12, Dec. 1986, pp. 1170-1183.
2. K. Hwang, "Advanced Parallel Processing with Supercomputer Architectures," *Proc. IEEE*, Oct. 1987, pp. 1348-1379.
3. G.S. Almasi and A. Gottlieb, *Highly Parallel Computing*, Addison Wesley, Reading, Mass., 1989.
4. A. Guha et al., "Optical Interconnections for Massively Parallel Architectures," *Applied Optics*, Vol. 29, Mar. 10, 1990, pp. 1077-1093.
5. A.A. Sawchuk and T.C. Stand, "Digital Optical Computing," *Proc. IEEE*, Vol. 72, No. 7, July 1984, pp. 758-779.
6. A. Huang, "Architectural Considerations Involved in the Design of an Optical Digital Computer," *Proc. IEEE*, Vol. 72, No. 7, July 1984, pp. 780-787.
7. P.B. Berra et al., "Optics and Supercomputing," *Proc. IEEE*, Vol. 77, Dec. 1989, pp. 1797-1815.
8. P.B. Berra et al., "The Impact of Optics on Data and Knowledge-Base Processing Systems," *IEEE Trans. Knowledge and Data Eng.*, Vol. 1, Mar. 1989, pp. 111-131.
9. W.T. Cathey et al., "Digital Computing with Optics," *Proc. IEEE*, Vol. 77, Oct. 1989, pp. 1558-1572.
10. J.W. Goodman et al., "Optical Interconnections for VLSI Systems," *Proc. IEEE*, Vol. 72, No. 7, July 1984, pp. 850-866.
11. A.A. Sawchuk et al., "Optical Crossbar Networks," *Computer*, Vol. 20, No. 6, June 1987, pp. 50-62.
12. L.D. Hutcheson and A. Husein, "Optical Interconnections Replace Hardware," *IEEE Spectrum*, 1987, pp. 30-35.
13. H.S. Hinton, "Architectural Considerations for Photonic Switching Networks," *IEEE J. Selective Areas in Comm.*, Vol. 6, Aug. 1988, pp. 1209-1226.

14. Special issue, "Optical Interconnections," *Applied Optics*, Vol. 29, Mar. 1990.
15. J.L. Jewell et al., "3-pj, 82-MHz Optical Logic Gates in a Room Temperature GaAs-AlGaAs Multiple Quantum-Well Etalon," *Applied Physics Letters*, 1985, pp. 918-925.
16. H.M. Gibbs et al., "Optical Bistable Devices: The Basic Components of an All-Optical System," *Optical Eng.*, Vol. 19, 1980, pp. 463-468.
17. A.R. Tanguay, "Material Requirements for Optical Processing and Computing Devices," *Optical Eng.*, Vol. 24, No. 1, Jan./Feb. 1985, pp. 2-17.
18. D.A.B Miller et al., "Large Room-Temperature Optical Nonlinearity in GaAs/$Ga_{1-x}Al_xAs$ Multiple Quantum Well Structures," *Applied Physics Letters*, Vol. 44, No. 10, 1982.
19. D.A.B. Miller et al., "The Quantum Well Self-Electro-optic Effect Device: Opto-electronic Bistability and Oscillation, and Self-Linearized Modulation," *IEEE J. Quantum Electronics*, Vol. QE-21, Sept. 1985, pp. 1462-1476.
20. S.D. Smith, "Room-Temperature, Visible-Wavelength Optical Bistability in ZnSe Interference Filters," *Optics Comm.*, Vol. 51, Oct. 1984, pp. 357-362.
21. G. Livescu et al., "Spatial Light Modulator and Optical Dynamic Memory Using a 6 × 6 Array of Self-Electro-optic Effect Devices," *Optics Letters*, Vol. 13, Apr. 1988, pp. 297-299.
22. A.L. Lentine et al., "Symmetric Self-Electro-optic Effect Device: Optical Set-Reset Latch, Differential Logic Gate, and Differential Modulator/Detector," *IEEE J. Quantum Electronics*, Vol. 25, Aug. 1989, pp. 1928-1936.
23. A.L. Lentine et al., "Optical Logic Using Electrically Connected Quantum Well PIN Diode Modulators and Detectors," *Applied Optics*, Vol. 29, No. 14, May 10, 1990, pp. 2153-2163.
24. J.A. Neff et al., "Two-Dimensional Spatial Light Modulators: A Tutorial," *Proc. IEEE*, Vol. 78, May 1990, pp. 836-855.
25. B.K. Jenkins et al., "Sequential Optical Logic Implementation," *Applied Optics*, Vol. 23, No. 19, Oct. 1984, pp. 3473-3474.
26. K.S. Huang et al., "Optical Cellular Logic Architectures Based on Binary Image Algebra," *Proc. Workshop on Computer Architecture Pattern Analysis and Machine Intelligence*, Oct. 1987, pp. 19-26.
27. J. Tanida and Y. Ichioka, "Optical Logic Array Processor Using Shadowgrams," *J. Optical Soc. Am.*, Vol. 73, No. 6, June 1983, pp. 800-809.
28. T.J. Drabik and S.L. Lee, "Shift-Connected SIMD Array Architectures for Digital Optical Computing Systems, with Algorithms for Numerical Transforms and Partial Differential Equations," *Applied Optics*, Vol. 25, No. 22, Nov. 1986., pp. 4053-4064.
29. F. Kiamilev et al., "Programmable Opto-electronic Multiprocessors and Their Comparison with Symbolic Substitution for Digital Optical Computing," *Optical Eng.*, Vol. 28, No. 4, 1989, pp. 396-409.
30. J. Taboury et al., "Optical Cellular Logic Architecture 1: Principles," *Applied Optics*, Vol. 27, No. 9, May 1, 1988, pp. 1643-1650.
31. D.P. Casasent and E.C. Botha, "Multifunctional Optical Processor Based on Symbolic Substitution," *Optical Eng.*, Vol. 28, No. 4, Apr. 1989, pp. 425-433.
32 K.H. Brenner, "Programmable Optical Processor Based on Symbolic Substitution," *Applied Optics*, Vol. 27, No. 9, May 1, 1988, pp. 1687-1691.
33. B.S. Werrett et al., "Construction and Tolerancing of an Optical-CLIP," *Proc. SPIE*, Vol. 1215, 1990, pp. 264-273.
34. M.J. Duff, *CLIP 4: Special Computer Architecture for Pattern Recognition*, Fu and Ichikawa, eds., CRC Press, 1982.
35. M.J. Murdocca et al., "Optical Design of Programmable Logic Arrays," *Applied Optics*, Vol. 27, May 1988, pp.1651-1660.
36. M. Prise et al., "Module for Optical Logic Circuits Using Symmetric Self-Electro-optic Effect Devices," *Applied Optics*, May 10, 1990, pp. 2164-2170.
37. M. E. Prise, "Optical Computing Using Self-Electro-optic Effect Devices," *Proc. SPIE*, Vol. 1214, Jan. 1990.
38. H.J. Caulfield and G. Gheen, eds., "Selected Papers on Optical Computing," *SPIE Milestone Series*, Vol. 1142, 1989.
39. A.D. McAulay, *Optical Computing Architectures: The Application of Optical Concepts to Next-Generation Computers*, John Wiley & Sons, New York, 1990.
40. D.G. Feitelson, *Optical Computing: A Survey for Computer Scientists*, MIT Press, Cambridge, Mass., 1988.
41 A. Huang, "Parallel Algorithms for Optical Digital Computers," *Proc. IEEE 10th Int'l Optical Computing Conf.*, 1983, pp. 13-17.
42. H.S. Stone, "Parallel Processing with the Perfect Shuffle," *IEEE Trans. Computers*, Vol. C-20, No. 2, 1971, pp. 153-161.
43. K.H. Brenner et al., "Digital Optical Computing with Symbolic Substitution," *Applied Optics*, Vol. 25, No. 18, Sept. 15, 1986, pp. 3054-3060.
44. Y. Li et al., "An And Operation-Based Optical Symbolic Substitution," *Optics Comm.*, Vol. 63, No. 6, Sept. 15, 1987, pp. 375-379.
45. F.T.S. Yu and S. Jutamulia, "Implementation of Symbolic Substitution Logic Using Optical Associative Memories," *Applied Optics*, Vol. 26, No. 12, June 15, 1987, pp. 2293-2294.
46. E. Botha et al., "Optical Symbolic Substitution Using Multichannel Correlators," *Applied Optics*, Vol. 27, No. 5, Mar. 1, 1988, pp. 817-818.
47. J.N. Mait and K.H. Brenner, "Optical Symbolic Substitution: System Design Using Phase-Only Holograms," *Applied Optics*, Vol. 27, No. 9, May 1, 1988, pp. 1692-1700.
48. K.H. Brenner, "New Implementation of Symbolic Substitution Logic," *Applied Optics*, Vol. 25, No. 18, Sept. 15, 1986, pp. 3061-3064.
49. A. Louri, "Efficient Implementation Method for Symbolic Substitution Logic Based on Shadow-Casting," *Applied Optics*, Vol. 28, No. 15, Aug. 15, 1989, pp. 3264-3267.
50. A. Louri, "Throughput Enhancement for Optical Symbolic Substitution Computing Systems," *Applied Optics*, Vol. 29, No. 20, July 10, 1990, pp. 2979-2981.
51. H.J. Jeon et al., "Digital Optical Processor Based on Symbolic Substitution Using Holographic Matched Filtering," *Applied Optics*, Vol. 29, May 10, 1990, pp. 2113-2125.

52. K.H. Brenner et al., "Symbolic Substitution Implemented by Spatial Filtering Logic," *Optical Eng.,* Vol. 28, No. 14, 1989, pp. 390-396.
53. A. Louri, "Parallel Implementation of Optical Symbolic Substitution Logic Using Shadow-Casting and Polarization," *Applied Optics,* Feb. 1991, pp. 540-548.
54. A. Louri, "Parallel Implementation of Optical Symbolic Substitution Logic Using Shadow-Casting and Wavelength Multiplexing," submitted to *Optics Letters.*
55. A. Louri and K. Hwang, "A Bit-Plane Architecture for Optical Computing with 2D Symbolic Substitution Algorithms," *Proc. 15th Int'l. Symp. Computer Architecture,* May 1988, pp. 18-27.
56. A.W. Lohmann et al., "Optical Perfect Shuffle," *Applied Optics,* Vol. 25, No. 10, May, 15 1986, pp. 1530-1531.
57. A.A. Sawchuk and B.K. Jenkins, "Dynamic Optical Interconnections for Optical Processors," *Proc. Soc. Photo-Optical Instrumentation Engineers,* Jan. 1986, Vol. 625.
58. C.W. Stirk et al., "Folded Perfect Shuffle Optical Processor," *Applied Optics,* Vol. 27, No. 2, Jan. 1988, pp. 202-203.
59. H.H. Brenner and A. Huang, "Optical Implementations of the Perfect Shuffle Interconnections," *Applied Optics,* Vol. 27, No. 1, Jan. 1988, pp. 135-137.
60. Y. Sheng and H. Arsenault, "Light Effective Perfect Shuffle Using Fresnel Mirrors," *Tech Digest, Topical Meeting on Optical Computing,* Vol. 9, Feb. 1989, pp. 154-157.
61. A.W. Lohmann, "What Classical Optics Can Do for the Digital Optical Computer," *Applied Optics,* Vol. 25, No. 10, May 15, 1986, pp. 1543-1549.
62. P.B. Berra et al. "Optical Database/Knowledge Base Machines," *Applied Optics,* Vol. 29, Jan. 10, 1990, pp. 195-205.
63. C. Warde and A. Fisher, "Spatial Light Modulators: Applications and Functional Capabilities," *Optical Signal Processing,* J. Horner, ed., Academic Press, New York, 1987, pp. 478-524.
64. A.D. Fisher and J.N. Lee, "Current Status of Two-Dimensional Spatial Light Modulators," *Proc. SPIE Optical and Hybrid Computing,* Vol. 634,1986, pp. 352-372.
65. S.D. Smith, "Optical Bistability, Photonic, Logic and Optical Computation," *Applied Optics,* Vol. 25, May 15, 1986, pp. 1550-1564.
66. L. Solymar and D. J. Cooke, *Volume Holography and Volume Gratings,* Academic Press, New York, 1981.
67. G.C. Valley and M.B. Klein, "Optimal Properties of Photorefractive Materials for Optical Data Processing," *Optical Eng.,* Vol. 22, Dec. 1983.
68. D. Psaltis et al., "Adaptive Optical Networks Using Photorefractive Crystals," *Applied Optics,* Vol. 27, No. 9, May 1988, pp. 1752-1759.
69. D. Psaltis et al., "Optical Neural Nets Implemented with Volume Holograms," *Tech. Digest, OSA Topical Meeting on Optical Computing,* 1987, pp. TuA3.1-TuA3.4.
70. S. Redfield and L. Hesselink, "Data Storage in Photorefractives Revisited," *SPIE Proc. Optical Computing 88,* J.W. Goodman et al., eds., Vol. 963, 1988, pp. 35-45.
71. R.P. Freese, "Optical Disks Become Erasable," *IEEE Spectrum,* Vol. 25, Feb. 1988, pp. 41-45.
72. T. Murakami et al., "Magneto-optic Erasable Disk Memory with Two Optical Heads," *Applied Optics,* Vol. 25, Nov. 1986, pp. 3986-3989.
73. D. Psaltis et al., "Parallel Readout of Optical Disks," *Tech. Digest, Topical Meeting on Optical Computing,* Vol. 9, Feb. 1989, pp. 58-62
74. P.D. Henshaw and A. Todtenkopf, "Artificial Intelligence Applications of Fast Optical Memory Access," *Proc. SPIE, Optical and Hybrid Computing,* Vol. 634, 1986, pp. 422-438.
75. S. Hunter et al., "Potentials of Two-Photon-Based 3D Optical Memories for High-Performance Computing," *Applied Optics,* Vol. 29, May 10, 1990, pp. 2058-2066.
76. K. Hwang, "Computer Architecture for Image Processing," *Computer,* Vol. 16, Jan. 1983, pp. 10-12.
77. K.E. Batcher, "Design of a Massively Parallel Processor," *IEEE Trans. Computers,* Vol. C-29, Sept. 1980, pp. 836-884.
78. C.G. Winckless, "Massively Parallel Computer for Signal and Image Processing," *Proc. IEEE Int'l Symp. Circuits and Systems,* May 1989, pp. 1396-1399.
79. W.D. Hillis, *The Connection Machine,* MIT Press, Cambridge, Mass., 1985.
80. N. Streibl et al., "Digital Optics," *Proc. IEEE,* Vol. 77, Dec. 1989, pp. 1954-1970.
81. H. Gibbs and N. Peyghambarian, "Nonlinear Etalons and Optical Computing," *Proc. SPIE, Optical and Hybrid Computing,* Vol. 634, Mar. 1986, pp. 142-148.
82. J.L. Jewell et al., "GaAs-AlAs Monolithic Microresonator Arrays," *Applied Physics Letters,* Vol. 51, July 1987, pp. 94-96.
83. J.L. Jewell et al., "Use of a Single Nonlinear Fabry-Perot Etalon as an Optical Logic Gate," *Applied Physics Letters,* Vol. 44, Jan. 1984, pp. 172-174.
84. M.S. Welkowski et al., "Status of Charge-Coupled-Device-Addressed Liquid Crystal Light Valve," *Optical Eng.,* Vol. 26, 1987, pp. 414-417.
85. B. Hill, "The Current Status of Two-Dimensional Spatial Light Modulators," *Optical Computing: Digital and Symbolic,* R. Arrathoon, ed., Marcel Dekker, New York, 1989, pp. 1-40.
86. G. Moddel et al., "High-Speed Binary Optically Addressed Spatial Light Modulator," *App. Phys. Lett.,* Aug. 1989.
87. K.M. Johnson and G. Moddel, "Motivations for Using Ferroelectric Liquid Crystal Spatial Light Modulators in Neurocomputing," *Applied Optics,* Vol. 28, Nov. 1989, pp. 4888-4899.
88. T.Y. Feng, "A Survey of Interconnection Networks," *Computer,* Vol. 14, Dec. 1981, pp. 12-27.
89. S.P. Levitan, "Measuring Communications Structures in Parallel Architectures and Algorithms," *The Characteristics of Parallel Algorithms,* Chap. 4, L.H. Jamieson et al., eds., MIT Press, 1987, pp. 101-139.
90. "Connection Machine Model CM-2 Technical Summary," Tech. Report Series HA87-4, Thinking Machine Corporation, Cambridge, Mass., 1987.



Ahmed Louri is an assistant professor in the Department of Electrical and Computer Engineering at the University of Arizona. Previously, he worked with the Computer Research Institute at the University of Southern California in Los Angeles, conducting research into parallel processing, multiprocessor system design, and optical computing. His technical interests include high-speed parallel processing, optical computing, and optical interconnections.

Louri earned the Engineer Degree diploma in electrical engineering from the University of Science and Technology, Algeria, and MS and PhD degrees in computer engineering from USC. He is a member of the IEEE, Association of Computing Machinery, Optical Society of America, and the Society of Photo-Optical Instrumentation Engineers.

Address questions regarding this article to the author at the Department of Electrical and Computer Engineering, University of Arizona, Tucson, AZ 85721; or e-mail at louri@pairs.ece.arizona.edu.

Reader Interest Survey

Indicate your interest in this article by circling the appropriate number on the Reader Service Card.

Low 159 Medium 160 High 161