An Optical Data-Flow Computer [1]

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

Abstract

For many applications, such as signal and image processing, computer vision, and artificial intelligence, the current achievable performance is much lower than that needed. Von Neumann models cannot achieve computational rates equivalent to billions of operations per second that will be required for these applications. Therefore, parallel models of computation must be used. One attractive concept for future computer architecture is the data-flow model of computation. Theoretically, maximal concurrency can be exploited using such a model. However, the lack of efficient implementation has prevented its wide use. This paper examines how optical systems can be used to break the bottlenecks that conventional electronic implementation imposes and presents a high-performance and scalable optical data-flow architecture. The architecture exploits the high degree of connectivity and inherent parallelism in optics for implementing a highly parallel instruction-level data-flow multiprocessing system. Both architectural and implementation issues are considered.

1 Introduction

The ever increasing demands for speed and computational power from a wide variety of scientific and symbolic applications are putting stringent demands on computer design. To achieve the computational rates equivalent to billions of operations per second that will be required from the processing systems of the future[1], improvements will have to be made on all fronts of computer system design. System speed may be increased either by using faster circuit and packaging technologies or by altering the system architecture to allow for more operations to be performed simultaneously.

The first approach appears to be leveling off due to fundamental physical limits[2]. Recent developments point in the direction of innovative architectures that exploit parallelism to achieve higher throughput. Within the second approach (exploitation of concurrency), there are two schools of thoughts that have been proposed at the system organization level[3]. The first one insists on retaining conventional sequential languages and architecture models and depends strongly on the use of concurrency between instructions. This concurrency must be detectable in high-level language programs and managed at the hardware level. In order to exploit concurrency in this approach, a program must be partitioned and allocated to processors in such a manner that the parallelism available in the program can be used effectively by the system organization. However, programs based on the conventional languages such as FORTAN, are essentially sequential and used the concept of updatable storage which introduces side effects and often completely prevent an effective extraction of parallelism. In addition, conventional architecture models all inherit the von Neumann bottleneck[4] and therefore restrict maximum use of parallelism.

The second approach, and most promising one, relies on the use of unconventional (or non-von) architectures based on parallel models of computation. This approach promotes both programmability and

[1]This research was supported in part by a Grant from the Vice President Office for Research, University of Arizona.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

performance. For programmability, new languages (functional languages) that are not dependent on the sequential model of computation, free from side-effects, and allow implicit and explicit representation of concurrency are desirable. For performance, highly concurrent systems that avoid centralized control and allow exploitation of maximum parallelism in user programs are more desirable. Data-Flow model of computing[5] is one alternative in which the issues of programmability and performance are not treated separately. In data-flow, an operation is performed as soon as its operands are available, thus, no centralized program sequencer is needed. An operation is purely functional and produces no side effects. Data-Flow programs are usually described in terms of directed graphs used to illustrate the flow of dependency of data between operations. The nodes of the data-flow graph represent operations and the arcs (or edges) represent data dependency. Theoretically, maximal concurrency can be exploited in such a model of computation, by directly mapping the data-flow graph onto a parallel machine, without loss of parallelism, subject only to resource availability.

Although, the data-flow approach offers a possible solution of efficiently exploiting concurrency of computation, and eases control of computation by distributing it, it places stringent demands upon communications and resource availability. Until now, data-flow appears to be failing to achieve the proclaimed performance due primarily to (1) the lack of adequate communication support to satisfy the high data traffic between communicating operations and (2) to contention of ready instructions for the limited resources available. A major part of the communication problem is due to the underlying VLSI technology. Conventional electronic technology seems to be unable to provide the required communication support to exploit maximum parallelism. Therefore, a design goal of the data-flow community is to minimize the number of communications and manage parallelism with limited resources. While this compromise alleviates the heavy demands on system organization, it also leads to loss of parallelism.

This paper seeks to initiate research into optics as an advantageous medium for data-flow computing. There are two parts of our hypothesis; one, that data-flow paradigm provides a sound basis to uncover inherent parallelism in user applications and two, that optics is, potentially the ideal medium to exploit such parallelism by breaking the bottlenecks that conventional electronics imposes. A brief description of data-flow computing is given in section 2. the advantages of optics for parallel processing are briefly expounded upon in section 3. Section 4 presents a preliminary optical architecture that can fully support the data-flow model of computation.

2 Data-Flow Computing

In data-flow computing, an instruction is enabled when its operands have arrived[5]. This model of computation has two major consequences: first, it lets the data dependencies determine the sequence in which operations may be executed. Thus there is no need for a program counter or central control. This implies that there may be many operations that can execute simultaneously. Thus the support of concurrent computations. Second, data-flow operations are purely functional and produce no side-effects. This functionality is due to the fact that operations operate on values rather than on calls of memory and that their effect is limited to create another value rather than to update a memory location. A data-flow program can be represented in a graph form made up of nodes, representing operations and arcs representing data dependency. Figure 1 shows a data-flow graph that corresponds to the following expression:

$$R = \frac{(x+y) \times (z-w)}{(t \times y)}$$

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

Values are carried by tokens which move along the lines between operations. Such a data-flow graph represents the machine language of a data-flow computer. Operations are distibuted over a network of homogeneous processing elements (PEs). These PEs operate concurrently constrained only by the operational dependencies of the graph.

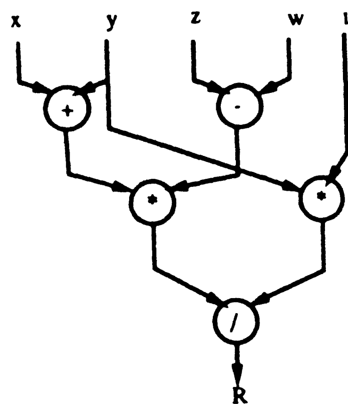


Fig.1. An example of a data-flow graph.

According to the enabling rules of the operation, data-flow computers are divided into the static model[6] and the dynamic model[7]. In the static execution, the subject of this paper, an operation is declared executable when it has tokens on all input arcs and there are no tokens on its output arcs. In other words, there can be only one token at any one time on each arc connecting two operations. This rule is enforced by requiring an operation to acknowledge the consumption of the input operands to its predecessors. This is known as the acknowledgment scheme[6]. In the dynamic model, each token has a tag associated with it that indicates its context. An operation is enabled when its sets of operands with the same tag have arrived. The advantage of the dynamic model is the ability for one PE to support several simultaneous activations of one operation, a feature not available in the static model. Its major drawback is the runtime overhead required to manage the tag operations and the relatively costly associative mechanism needed to implement the matching store.

3 How can Optics Help?

The advantages of optics for high-speed parallel processing have been expounded upon on numerous occasions[8, 9]. The advantages of using optical systems for data-flow computing rather than electronic systems can be seen as follows:

- **Communication:** The use of optical techniques has been widely recognized as a solution to overcome the fundamental problems of data communications at various levels, i.e. processor level, board level, as well as gate level. Some of these problems include limited temporal bandwidth, clock skew between different signals, and limited number of I/O pins between chips[9]. The parallel nature of optics and free-space propagation, together with their relative freedom from interference make them ideal for parallel computations. Optics may be able to provide a larger degree of connectivity and higher temporal bandwidth between processing elements than electronics[10].

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

- **Time-consuming operations:** Moreover, optics may be a better alternative for some of the basic and time-consuming operations such as operand searching and replacing, matching, comparison, duplication etc. These operations are often sources of major bottlenecks in conventional systems.

In this paper, we explore optics as a means of implementing the data-flow model of computation. This includes exploring optics not only for providing communications as was done elsewhere[11] but also for computations.

4 Preliminary Version of an Optical Data-Flow Computer

The proposed preliminary optical architecture has been designed to satisfy the architectural requirements of data-flow computing, and to take advantage of the unique features of optics while avoiding its limitations. Three basic principles have guided the design of this initial architecture.

First, is concurrency achieved through distributed control. This first principle serves two purposes, namely, to maximize parallelism extraction from the running application and to circumvent the lack of flexibility in optical systems in order to implement MIMD-type (multiple instruction multiple data) of computations. For optics, it is easy to perform the same operation on all the pixels of a 2-D plane. This property has made optics very attractive for designing SIMD (single instruction multiple data) array processors for the implementation of highly structured and regular algorithms. This fact is evidenced by the proliferation of the various proposals for SIMD optical architectures that have surfaced recently[12, 13, 14, 15, 16]. However, the essence of data-flow computing is based on asynchrony of computations, which means that many operations can be executed simultaneously and asynchronously. Therefore, MIMD-type of control is more natural for data-flow computing than SIMD one. Because operation execution is purely functional in data-flow, no notion of the state of the machine or computation is added. The result of an operation execution is exclusively a function of the input operands. This implies that control can be fully distributed. This distributed nature of the control mechanism eases its optical implementation by circumventing the flexibility issue in optical systems.

The second principle is distributed memory. Having realized that random-access memory is a problem at this time for optics[17], the second principle is the use of message-passing communications and data tagging to shift the burden from memory-based to register-based system with a high degree of connectivity. This will alleviate the need for addressable memory since data can be located by sending messages containing tags to where it resides rather than fetching it from a central memory.

The third principle is scalability achieved through free-space programmable optical interconnects. This implies that the size of the optical machine can be increased or decreased to fit the size of real applications without major architectural and software changes.

4.1 Mapping a Data-Flow Graph onto the Optical Architecture

The optical architecture implements instruction-level data-flow (fine-grain data-flow). In this scheme, a high level language program is compiled into a data-flow graph where nodes represent instructions and arcs represent messages that need to be sent between these instructions. Instructions include primitive operations like the basic arithmetic and logical operations, i.e., addition, multiplication, subtraction, division, logical and, logical or, etc, and control operations like decider, gates, and boolean operations (greater than, less than, etc). The control operations are required for implementing complex control structures such as iteration, recursion and conditionals[6].

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

In pure data-flow computing the number of edges that are incident onto a node or that emerge from a node are not limited. Such an unlimited fan-in and fan-out can not be mapped in a straight-forward fashion onto the processing plane because the hardware complexity required for representing each node as well as the time to perform search would linearly increase with increasing node fan-in and fan-out. One solution is to restrict the fan-in and fan-out of an node to some manageable number and use special nodes, which we call fan-out nodes, whose functions is to create a manageable graph. Figure 2 shows the representation of the execution nodes from which instruction-level data-flow programs are constructed and mapped onto the optical architecture. The circles in Fig.2 represent instructions which can be any primitive arithmetic, boolean, control (logical gates, mergers, deciders), or fan-out operation. Due to the hardware consideration above, the executing nodes are confined to fan-in and fan-out factors of 2. This implies that each node can receive two data tokens form predecessor nodes and two acknowledgment signals, represented by dotted lines in the Figure, from successor nodes. Moreover, it can send out two data tokens and two acknowledgment signals. A data-flow compiler translates an instruction-level data-flow graph into an intermediate form composed of primitive nodes and fan-out nodes. When a node has more than two successors and more than two predecessors, fan-out nodes are used as illustrated in Fig.3.

To exploit the multi-dimensional nature of optical systems, we represent nodes of the graph as rows of 2-D planes of say 1000×1000 pixels of optical gates. Each node of the data-flow graph is located on a row of some plane and has slots (or registers) for its operands as well as optically interpretable code for its operation (to be described later). Node-to-node communications are carried out through message-passing protocols via free-space optical interconnects. Functionally, each row of the processing plane represents an optical processor (OP) capable of handling one node (or instruction). Each optical processor is comprised of three components: a processing element (PE), a communication element (CE) and a communication network (CN). The PEs execute the primitive operations and data-flow sequencing control. The CEs are responsible for implementing message-passing protocols and for transferring messages between communicating PEs. The CNs represent the physical layer (buffers, registers, control mechanisms, etc.) that is needed for transferring messages between PEs as shown in Fig.4. The separation of the OPs into PEs and CEs allows for more concurrent processing. By shifting the burden of message handling onto the CEs, communication and computation can proceed simultaneously. This optical representation is inspired by the thorough analysis of the application of optics to symbolic processing conducted by Derstine and Guha[18].

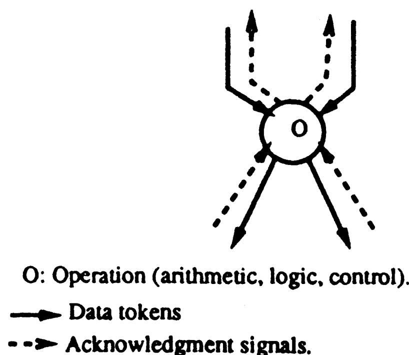


Fig.2. Format of a basic data-flow node used in the optical architecture.

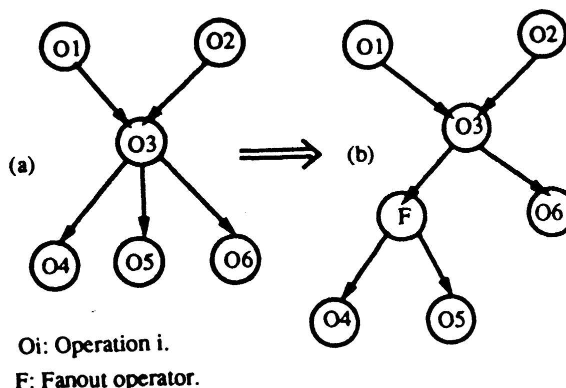


Fig.3. Insertion of a fanout node to limit the out-degree of a data-flow node: (a) original graph; (b) refined graph.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

4.2 Processing Element

Each PE in the processing plane can be considered as a finite state machine (FSM) capable of his own control. The PE represents the processing engine of the optical processor. It executes primitive instructions and performs data-flow sequencing control. Instruction enabling is performed when all input operands have arrived. Actually, instructions are enabled as soon as their data operands arrive. Results are sent out as messages when all acknowledgment signals have arrived. This is a slight modification of the original acknowledgment scheme introduced by Dennis[6]. In this latter, instruction enabling takes place only when all input operands (data and acknowledgment tokens) have arrived. The benefits of the modified scheme is a reduced waiting time, since an operation can fire while still receiving acknowledgment signals. Consequently, the PE structure consists of two parts: an instruction execution section that contains information used to execute the instruction, and a control section that contains control information for enabling the instruction and routing the results as shown in Fig.5.

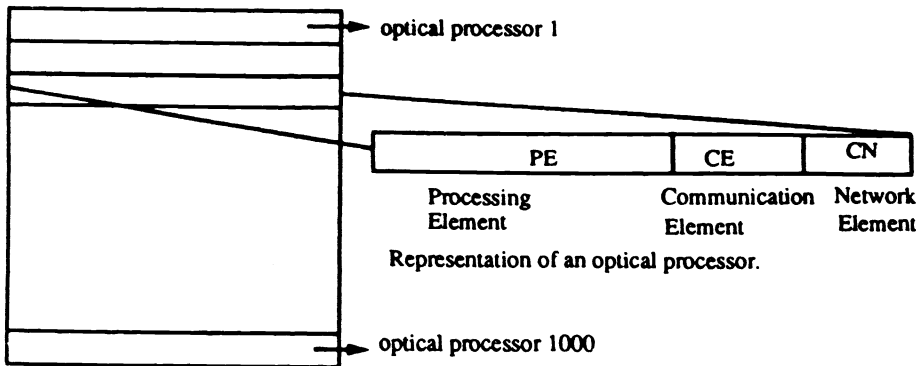
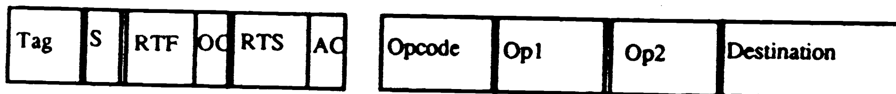


Fig.4. A 2-D processing plane (1000 × 1000) where each row represents an optical processor (OP).

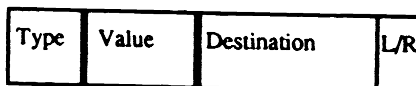


Tag: PE address
S: PE status
RTF: Ready-to-fire field
OC: Operand count
AC: Acknowledgment count

(a) Control section.

Opcode: instruction type
Op1: Left operand
Op2: Right operand
Destination: address of successor and predecessor nodes

(b) Instruction execution section.



Type: Data token/acknowledgment signal
L/R : Left/Right operand

(c) Data token format (message format).

Fig.5. The format of the (a) control section, (b) instruction execution section, and of the (c) data token (message).

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

The control section (Fig.5.a) is composed of 4 fields: (1) unique tag representing the address of the PE (node-to-PE allocation is done at compile time, which implies that the node of the data-flow graph and the PE assigned to it carry the same tag); (2) a status field indicating the status of the PE (whether executing, waiting for acknowledgment signals, or idle); (3) a ready-to-fire (RTF) field that indicates the status of the instruction (ready to fire or waiting for operands), and a count of the number of operands required that will be used to reset the RTF field after the instruction executes; and (4) a ready-to-send (RTS) field that monitors the arrival of acknowledgment signals and indicates when to send the result to the successor instructions. It also contains a count of the successors that is used to reset the RTS field once the results are sent out. The instruction execution section (Fig.5.b) consists of four fileds: (1) an opcode field that indicates the type of the instruction; (2) two operand fields for storing the incoming data operands; (3) a destination field that indicates where the instruction execution results should go, and the address of the predecessor nodes for which acknowledgment signals are to be sent upon instruction execution.

Initially, the RTF bits are cleared to zero for instructions expecting two operands, and set to one for instructions expecting one operand or have already one operand as a literal (literals are allocated at compile time); the operand count field is set to zero for a dyadic instruction and to one for a unary instruction; the RTS bits are set to one indicating that ready instructions can send out results upon firing. Upon receiving a data token, the PE performs the following : (1) it checks the RTF bit if set, indicating that one of the operands has already arrived, if so, (2) it executes the instruction and (3) forms a data token from the result and the destination address. If not, it stores the data value of the incoming token in its appropriate field and sets the RTF bit to one. During the execution, the status field of the PE is set indicating to the CE that the PE is busy. When the PE is in the busy state, the CE makes sure that incoming messages are not lost nor do they override current information in the PE. Upon instruction completion, the PE checks if the RTS bit is set, in which case the PE sends the result data token to the CE which in turn will route it to its proper destination via the communication network. In fact, up to four messages are sent (according to the successor and predecessor field counts). Two messages carrying data tokens are sent to the successor PEs, and two messages carrying acknowledgment signals are sent to the predecessor PEs signaling that the instruction is available for further use. Once the data tokens and acknowledgment signals are sent, the RTF and RTS bits are reset to the operand and successor counts.

For the fan-out nodes, the execution step consists of replicating the data token just received and sending it to appropriate destinations. The data token consists of 3 fields: (1) a type field indicating whether the token is carrying data or an acknowledgment signal; (2) a value field for holding data; and (3) a destination field that indicates where the token should be sent and consists of the PE address and the instruction port as depicted in Fig.5.c. For acknowledgment signals, the value and port field is irrelevant. When a PE gets a token carrying an acknowledgment signal, no instruction execution takes place. Instead, the PE updates the RTS field and takes further action accordingly.

4.3 Communication Element and Network Topology

As described above, the optical architecture implements instruction-level data-flow, which requires a flexible high-bandwidth communication network that operates efficiently for small messages (data tokens). The CE part of the optical processor is responsible for implementing the message-passing protocols required. It continually monitors the network to look for incoming messages. It accepts tokens addressed directly to the corresponding PE and reroutes tokens addressed to other PEs. The requirements we impose on the interconnection network both for node-to-node and plane-to-plane communications are as follows:

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

- Compatibility and expandability: the network topology should be compatible with the linear layout of the optical architecture. In addition, if the machine is to the scalable to the size of the problem by adding more optical processors as needed, the network should be accordingly expandable.
- Parallel transmission of messages: the network should be able to transmit several messages in parallel so as to enforce concurrency.
- Nonblocking: the network should be able to establish a connection between communicating PEs without disturbing other connections.
- Speed: the topology of the network must allow for easy and fast transfer of messages between communicating PEs.
- Reconfigurability: for plane-to-plane communication, the network must be reconfigurable with a reasonable reconfiguration time.

The ideal network for such demands is a crossbar. However, the implementation and control of large crossbar is not currently feasible even for optics[10, 19]. Thus networks with limited connectivity must be used. Candidate networks for in-plane communications include ring structures (unidirectional, dual counter-rotating, and cordal rings) as shown in Fig.6. In such a network, messages are passed synchronously and simultaneously between OPs. The scheme functions as a slotted conveyor belt. The conveyor belt would be a set of registers representing the CNs and the CEs will consist of a set of in-registers and out-registers for buffering messages and a control section for managing data transfer. After each time unit the OPs are presented with a new slot of the conveyor belt. The CE checks the destination of the message, if the destination is this PE then the data portion of the message is off-loaded from the CN to the PE. Otherwise, the message is transmitted to the next available slot. This operation will be occurring simultaneously in all the OPs on the plane. The CE will manage contention by buffering messages when the destination PE is in the busy state. In addition, the CE will also receive out-going messages from the PE and routes them onto the network.

The merits of such a scheme are simplicity of control, parallel communication and compatibility with the linear layout of the architecture. However, this simple network is slow and does not offer redundancy. Other variations of this simple network would be a dual counter-rotating ring network. In this case, Two conveyor belts are available to the PEs for communication with both neighboring OPs. This brings

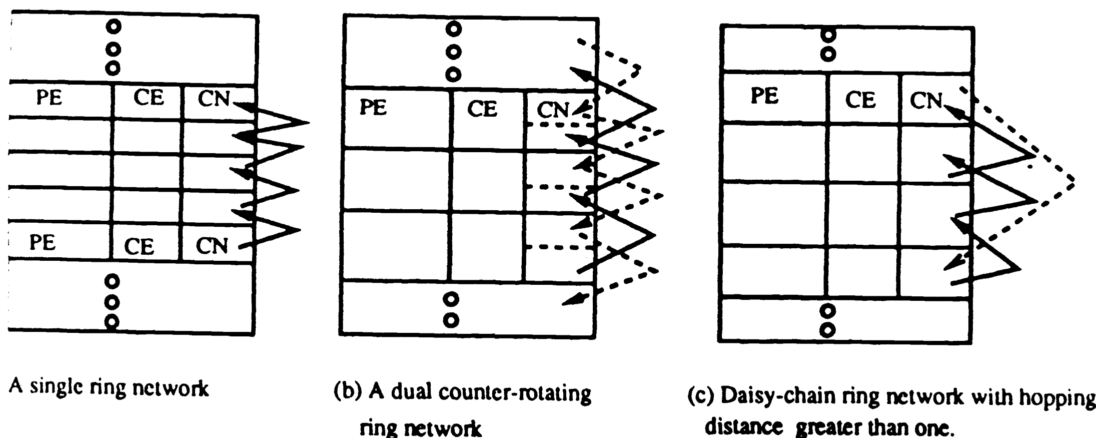


Fig.6. Candidate network topologies that are compatible with the linear layout of the optical architecture.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

the advantage that the distance between two neighbors is always 1 while it was either 1 or N-1 in the single ring scheme (N represents the total number of OPs on the plane). Moreover, message propagation time is reduced and redundancy is introduced since two paths exist for communication between any two OPs. Simplicity of the control is still maintained. Other refinements can also be brought to the counter-rotating ring network. Instead of stringing the second ring between immediate neighbors, the second ring can be made to hop over several OPs and connect non-adjacent OPs. Other refinements would be to allow the network to access several OPs on the plane in parallel (broadcasting).

4.4 Optical Implementation

This section is intended to provide possible optical implementations of the different components of the architecture. It should be noted that these descriptions represent an initial attempt and are by no means optimal or final. The optical architecture consists of processing planes which in turn, consist of optical gates that provide the nonlinear functions required to implement the optical processors described before. The implementation of such a highly parallel architecture can be divided into two parts: the implementation of the processing elements which execute instructions and data-flow control, and the implementation of communication protocols and interconnects (the CEs and CNs). This division implies that memory is distributed and is part of the PEs. At the high level, the PEs and the CEs are considered finite state machines with a finite number of states. There are several optical techniques that can be used to implement an optical FSM. These include optical sequential logic[20], optical associative processing[21], optical array logic[13], and optical symbolic substitution[22] (OSS), among many other techniques[23]. Among these methods, OSS appears to offer more parallelism, flexibility, and ease of implementation.

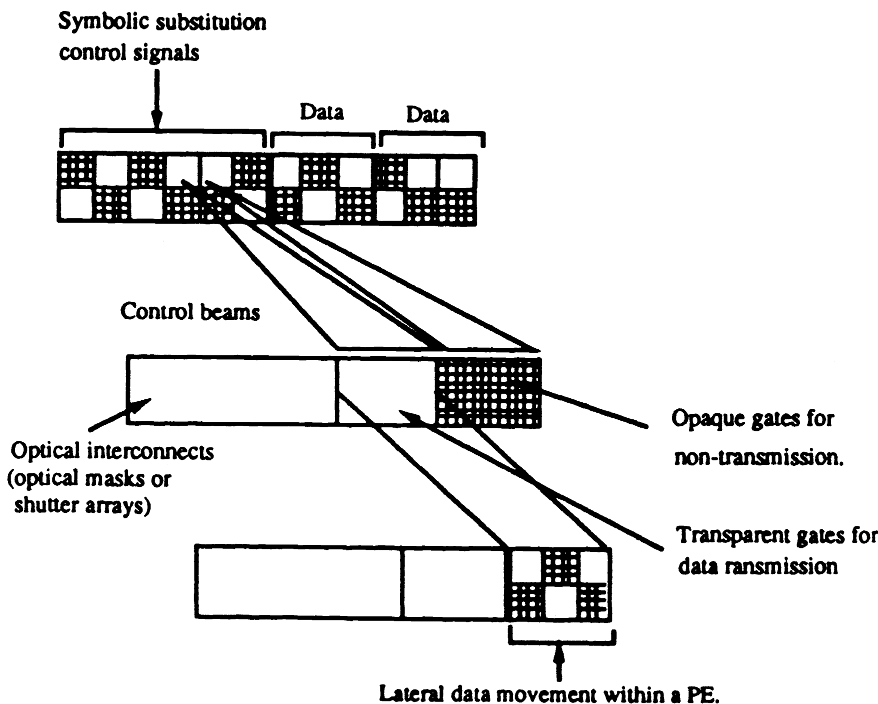


Fig.7. Data movement within a PE and between PEs using symbolic substitution control signals and programmable shutter arrays.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

Optical Symbolic substitution is a computing technique that was introduced by Huang [22] to take advantage of the massive parallelism and high speed in optics. In this method, information is represented by optical patterns within a two-dimensional binary image. Computation proceeds in transforming these patterns into other patterns according to predefined OSS rules. It consists of two processing phases: a *recognition phase* where the presence of a specific pattern, called the *search pattern*, is detected within a binary image and a *substitution phase*, where the present pattern is replaced by another pattern, called the *replacement pattern* according to a predefined OSS rule. This computing technique can be used to implement instruction execution, sequencing control, and data movement within the node and between nodes. The use of OSS is as follows: (1) states and state transitions will have to be determined, next (2) assembly-like instruction set will be derived that implement the state transitions, (3) finally, symbolic substitution rules will be developed to optically carry out the instruction set. For data movement, pixels form the control section of the PE can be used to control optical gates that implement the interconnection network as shown in Fig.7. Physically, each processing plane becomes a three-dimensional system as illustrated in Fig.8. The first plane is the initial state and the second plane is the next state after data transmission and computation. In between planes, lies the optical hardware for implementing symbolic substitution and interconnects.

Several processing planes can be configured into a cluster-based architecture as shown in Fig.9, to fit the need of real applications. These processing planes will communicate though programmable free-space interconnects. The global interconnects can be implemented using a real-time volume holographic medium such as photorefractive crystals[24]. The global inter-plane communication can be implemented by activating different interconnection holograms in a volume holographic material of large storage capacity. These holograms can be activated independently. The machine configuration can be programmed to best match the application at hand. This hierarchy of interconnects (linear layout for communication within a processing plane and volume interconnects between planes) is intended to manage diffraction effects while providing a high degree of connectivity.

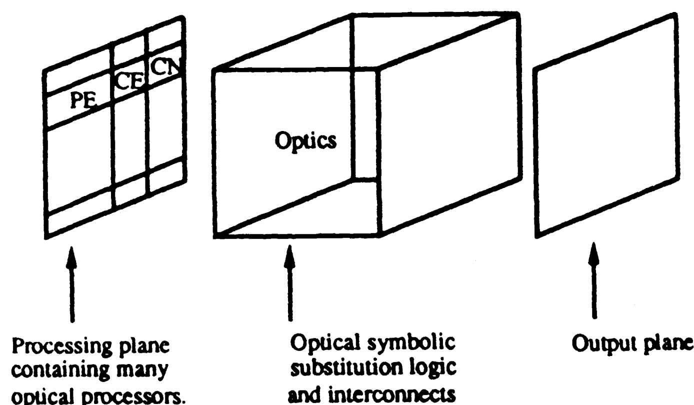


Fig.8. The physical layout of a single optical processing plane with corresponding optical components for computation and communication.

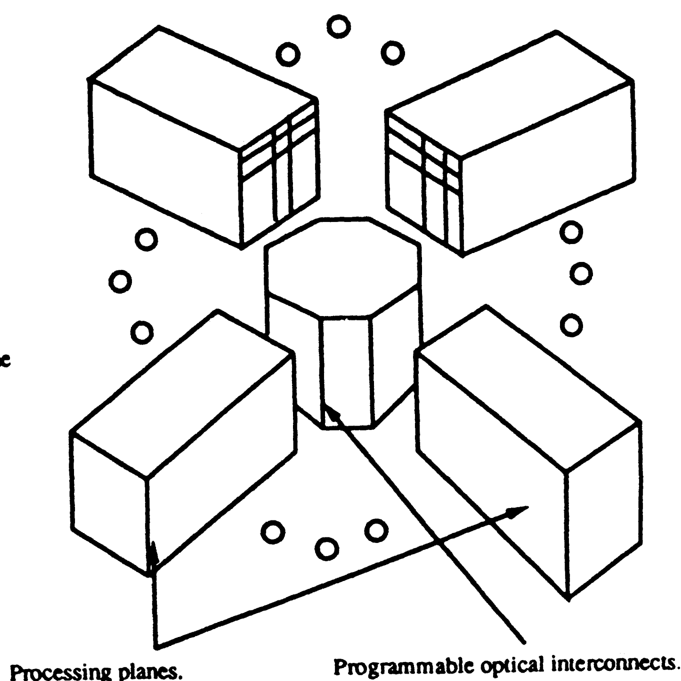


Fig.9. A scalable optical data-flow machine: the processing planes communicate via free-space programmable optical interconnects.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

5 Conclusion

In this paper, we have examined data-flow computing as a future alternative for computer architecture for achieving much higher computational rates than what is obtainable with today's computing systems. Data-Flow holds much promise because it allows exploitation of maximal concurrency. However, it places stringent demands on communications, which is making its implementation with electronic technology impractical.

Optics, due to its inherent parallelism and non-interfering communications, appears to be an ideal medium for implementing the data-flow model of computation. We have presented a preliminary high-performance and scalable to real-world problems architecture, that exploits the full potential of optical systems. The architecture combines the best features of data-flow and optics. The distributed control nature of data-flow computing is used to overcome the lack of flexibility in optical systems, while the high degree of connectivity and multi-dimensional nature of optics is used to provide adequate architectural and communication support for data-flow execution.

This architecture represents an initial attempt. Much more work remains to be done in order to conduct a thorough performance analysis on the proposed system and assess its validity to data-flow computing. Many architectural and algorithm issues, such as the control mechanism, parallel communications schemes, the handling of complex data structures[25], fault-tolerance, programmability, and program partitioning and allocation need to be fully elaborated.

References

[1] DARPA, "Strategic computing: New-generation computing technology," Technical report, Defence Advanced Research Project Agency, Oct. 1983.

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

[3] D. Gajski and J. K. Pier, "Essential issues in multiprocessor systems," *IEEE Computer*, vol. 18, no. 6, pp. 9 – 18, May 1985.

[4] J. Backus, "Can programming be liberated from the Von Neumann style?," *Comm. of the ACM*, vol. 8, pp. 613 – 641, 1978.

[5] J. B. Dennis and D. P. Misunas, "A preliminary architecture for a basic data-flow processor," In *Proc. 2nd Int'l. Symp. on Computer Arch.*, pp. 126 – 132, Jan. 1975.

[6] J. B. Dennis, "Data-flow supercomputers," *Computer*, pp. 48 – 56, Nov. 1980.

[7] Arvind and K. P. Gostelow, "A computer capable of exchanging processors for time," In *Proc. IFIP Congress 77*, pp. 849 – 853, Aug. 1977.

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

[9] A. Huang, "Architectural considerations involved in the design of an optical digital computer," *Proc. IEEE*, vol. 72, no. 7, pp. 780 – 787, July 1984.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms

[10] A. Hartmann and S. Redfield, "Design sketches for optical crossbar switches intended for large-scale parallel processing applications," *Opt. Eng.*, vol. 28, no. 4, pp. 315 – 328, May 1989.

[11] A. D. MacAulay, "Optical crossbar interconnected digital signal processor with basic algorithms," *Opt. Eng.*, vol. 25, no. 1, pp. 082 – 090, Jan. 1986.

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

[13] J. Tanida and Y. Ichioka, "Opals: Optical parallel array logic system," *Appl. Opt.*, pp. 1565 – 1570, 15 May 1986.

[14] K. S. Huang, B. K. Jenkins, and A. A. Sawchuk, "Optical cellular logic architectures based on binary image algebra," In *Proc. IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence*, pp. 19 – 26, Oct. 1987.

[15] K. H. Brenner, A. Huang, and N. Streibl, "Digital optical computing with symbolic substitution," *Appl. Opt.*, vol. 25, no. 18, , 15 Sept 1986.

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

[17] A. Guha, R. Ramnarayan, and M. Derstine, "Architectural issues in designing symbolic processors in optics," *In Proc. 14th Int'l. Symp. on Comput. Archi.*, 1987.

[18] M. W. Derstine and A. Guha, "Design considerations for an optical symbolic processing architecture," *Opt. Eng.*, vol. 28, no. 4, pp. 434 – 446, April 1989.

[19] A. R. Dias, R. F. Kalman, J. W. Goodman, and A. A. Sawchuk, "Fiber-optic crossbar switch with broadcast capability," *Opt. Eng.*, vol. 27, no. 11, pp. 955 – 960, Nov. 1988.

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

[21] A. D. Fisher, C. L. Giles, and J. N. Lee, "Associative processing architectures for optical computing," *Journal Opt. Soc. Am. A*, 1984.

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

[23] D. G. Feitelson, *Optical Computing : a survey for computer Scientists*, MIT Press, 1988.

[24] D. Psaltis, D. Brady, and K. Wagner, "Adaptive optical networks using photorefractive crystals," *Appl. Opt.*, vol. 27, no. 9, pp. 1752 – 1759, May 1988.

[25] J. Gaudiot, "Data-flow computers," *IEEE Trans. on computers*, vol. c-35, no. 6, pp. 489 – 502, June 1986.

Downloaded From: http://proceedings.spiedigitallibrary.org/ on 06/06/2014 Terms of Use: http://spiedl.org/terms