ARRAY PROCESSORS WITH PIPELINED OPTICAL BUSSES

Zicheng Guo, Rami G. Melhem, Richard W. Hall,
Donald M. Chiarulli and Steven P. Levitan

Departments of Electrical Engineering and Computer Science
The University of Pittsburgh
Pittsburgh, PA 15260

Abstract

A synchronous multiprocessor architecture based on pipelined optical bus interconnections is presented. In this architecture the processors are placed in a square grid and are interconnected to one another through horizontal and vertical optical busses. This architecture has an effective diameter as small as 2 due to its orthogonal bus connections, and allows all processors to have simultaneous access to the busses due to its capability for pipelining messages. Although the resulting architecture is mesh like and uses bus connections, it has a substantially higher bandwidth than conventional and bus augmented mesh computers. Moreover, it has a simple control structure and is universal in that various well known multiprocessor interconnections can be efficiently embedded in it. This architecture appears to be a good candidate for hybrid optical-electronic systems in the next generation of parallel computers.

1. Introduction

Two-dimensional meshes of processing elements (PE) have been extensively studied [17, 20], and large scale implementations of two-dimensional meshes have been built [7, 12]. However, since the communication diameter of an $n \times n$ mesh is $O(n)$, different approaches have been considered to augment the communication capabilities of the mesh to reduce this diameter. Meshes have been augmented with global busses [1, 28] and row and column busses [23], yielding both small communication diameter and higher efficiency for certain classes of algorithms. Interconnection networks, including trees [21, 31] and compounded graphs [13, 14], have also been considered for augmenting rows and columns in a mesh. In this context the binary hypercube can also be viewed as a two-dimensional mesh with horizontal and vertical hypercube interconnections [13, 14].

One of the simplest mesh augmentation schemes is the row and column bus augmentation. However, exclusive write access to busses is a major contributor to the low bandwidth of bus interconnections. A unique property of optics provides an alternative to this exclusive access; namely, the ability in optics to pipeline the transmission of signals through a channel. In electronic busses, signals propagate in both directions from the source, while optical channels are inherently directional and have predictable delays per unit distance. Hence, a pipeline of optical signals may be created by the synchronized directional coupling of each signal at specified locations along the channel. This property has been used to parallelize access to shared memory [3], to enhance the bandwidth in bus connected multiprocessor systems [16], and to minimize the control overhead in networking environments [30].

This work was, in part, supported under Air Force grant AFOSR-88-198, and under NSF grant MIP-8901053.

In this paper, we study a synchronous communication model which employs pipelined optical busses in computational arrays. In Section 2 we review the basic principle of pipelining messages on optical busses. Section 3 studies communication patterns and embedding results for a linear array of PE's interconnected with optical busses. In Section 4 we introduce the basic architecture of our two-dimensional array of PE's interconnected with horizontal and vertical optical busses. Here, we discuss message routing issues and embedding issues for this new architecture. We show how binary tree interconnections can be effectively embedded and identify key design issues for effective embeddings of arbitrary interconnections. Section 5 proposes a structural variation of the basic two-dimensional architecture, which is capable of switching messages from horizontal (vertical) busses to vertical (horizontal) ones, and thus increases the efficiency of the basic architecture. In Section 6, we compare the efficiency of the pipelined bus communication model with that of non-pipelined busses and of store and forward communications in nearest neighbor structures. Finally Section 7 contains concluding remarks.

2. Message Pipelining on Optical Busses

Consider the linear array of processors shown in Fig 2.1(a), where n processors, each having a constant number of registers, are connected through a single optical waveguide (bus). Each processor is coupled to the optical bus with two couplers, one for injecting (writing) signals on the waveguide, and the other for receiving (reading) signals from the waveguide [15, 32]. As in the case of electronic busses, each node j communicates with any other node i by sending a message to i through the common bus. However, because optical signals propagate in one direction, a node j in the system of Fig 2.1(a) may send signals to another node i only if $i > j$.

Assume that a message on an optical bus consists of a sequence of optical pulses, each having a width (or duration) w in seconds. The existence of an optical signal of length w

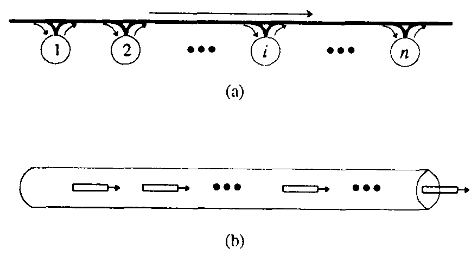


Fig 2.1. (a) A linear array of processors connected with a waveguide. (b) Message pipelining on the waveguide.

represents a binary bit 1, and the absence of such a signal represents a 0. For analytical convenience, we let D_o be the optical distance between each pair of adjacent nodes and τ be the time taken for an optical pulse to traverse the optical distance D_o. To transfer a message from a node j to node i, $i > j$, the sending node j writes its message on the bus. After a time $(i-j)\tau$, the message will arrive at the receiving node i, which then reads the message from the bus.

The property of uni-directional propagation of optical signals may be used advantageously. Specifically, unlike the electronic case, where the writing access to the bus by each node must be mutually *exclusive*, all nodes in the system of Fig 2.1(a) can write messages on the bus *simultaneously*, provided that the following collision-free condition [16] is satisfied:

$$D_o > bwc_g \qquad (1)$$

where b is the number of binary bits in each message, and c_g is the velocity of light in the waveguide. Clearly if this condition is satisfied and the network is synchronized such that every node starts writing a message on the bus at the same instant, then no two messages injected on the bus by any two distinct nodes will collide. Here by colliding we mean two opitcal signals injected on the bus by any two distinct nodes arrive at some point on the bus simultaneously. This kind of synchronized pulse generation is restrictive but it can be met in several ways. An optically distributed clock can be broadcast without skew to each node [5], or electro-optical switches [22] can be used in place of sources to "switch in" pulses generated from a single source. With this condition satisfied, every node can, in parallel, send a message to some other node, and the messages will all travel from left to right in a pipelined fashion as shown in Fig 2.1(b), thus the term "*pipelined bus*". In the rest of this paper we will always assume that this collision-free condition is satisfied.

To facilitate our discussion in subsequent sections we define some terms. Let τ be defined as before, and n be the number of nodes on the pipelined optical bus. We define $n\tau$ as a *bus cycle*, and correspondingly τ as a *petit cycle*. Note that a *bus cycle* is the time taken for an optical signal to traverse the entire length of the optical bus. For the time being, we do not include in a bus cycle the time taken to prepare and process the message before it can be injected on the bus. If *every* node is writing a message simultaneously on the bus, then each node has to wait for at least a *bus cycle* to inject its next message.

Before moving to the next section, let us look at a simple example where each node transmits a message and each node is programmed to receive a message from the k^{th} node (if it exists) to its left. All nodes start injecting messages at the beginning of a bus cycle, and all the messages will then travel on the optical bus in pipelined fashion without collision. By waiting for a specific interval of time, a node can selectively read the message intended for it as that message passes by the node. In our example, each node i is to receive a message from node $i-k$, and thus must read that message from the bus after $k\tau$ time relative to the beginning of the bus cycle. In this way, a message routing pattern in which each node sends a message to the k^{th} node to its right has been realized. In fact, as will be seen, we can efficiently realize various message routing patterns in a simple and straightforward way.

3. Linear Array Processors with Pipelined Optical Busses

The system of Fig 2.1(a) supports only message transmition from left to right. To allow message passing from right to left, we add another optical bus to the system in Fig 2.1(a). Now our system looks as in Fig 3.1(a). There, we have two optical busses, the upper one is used for sending messages from left to right, and the lower one is used for sending messages from right to left. Each node can write and read messages on either bus as desired. Obviously signals on different busses do not disturb one another, that is, the two busses can support two separate pipelines. The system in Fig 3.1(a) is our architecture of linear *Array Processors* with *Pipelined Busses* (APPB). For convenience this linear APPB will be schematically drawn as in Fig 3.1(b).

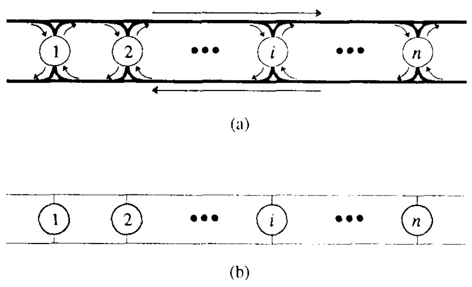


Fig 3.1. (a) A linear array processor with pipelined optical busses (linear APPB). (b) A schematic drawing of (a).

To specify the time at which a node should receive a message, we define a control function $wait(i)$ as the time that node i should wait, relative to the beginning of the bus cycle, before reading the message sent to it from some other node j. Thus

$$wait(i) = (i-j)\tau.$$

For convenience, if τ is considered as a time unit, then *wait* can be interpreted in terms of the number of such time units and thus be written as $wait(i) = i-j$. Clearly if $wait(i) > 0$, the message is to be received from the left; if $wait(i) < 0$, then the message is to be received from the right. If $wait(i) = 0$, then no message should be received by node i. The value of $wait(i)$ can be stored in a *wait register*, and more than one such register may be used if a node is to receive more than one message.

3.1. Message Routing in Linear APPB

Various message routing patterns can be realized in linear APPB's in a simple, straightforward way. Since a routing pattern is determined by the *wait* functions, we need only determine these *wait* functions for each routing pattern. The most commonly used message routing patterns are:

One-to-one: The system executes a $SEND(j, i)$ instruction, which means that a message is to be transferred from node j to node i. Thus, $wait(i) = i-j$, where i is a *single* specific node.

Broadcast: the system executes $BROADCAST(j)$, which means that node j broadcasts a single message, and all other nodes i will be receiving that message. In this case, $wait(i) = i-j$ for *all* $i \neq j$.

Semigroup Communication [2]: The system executes a $SEMIGROUP(i)$ instruction, which means that some global information, e.g., extrema and sum, is to be computed and stored at node i. This task can be accomplished by having the linear APPB function logically as a tree with root being node i. Later in this section we will present some embeddings of binary trees which facilitate such tree emulation task.

Permutations: For each node j to send a message to a node $i = PERM(j)$, where $PERM()$ is an arbitrary permutation, we set $wait(i) = i-j$ for *all* i.

It is clear that all these routing tasks can be performed using a single bus cycle, except the semigroup communication, which takes $\log(n)$ bus cycles. Such efficient accomplishment of these commonly used message routing patterns can significantly improve the efficiency of many parallel algorithms. Note that, in linear APPB, message passing between two non-adjacent nodes is nearly as efficient as that between two adjancet nodes. Specifically, a message takes τ more time to pass one more node on the bus. This is not the case in conventional linear arrays, where to pass a node, en route to another node, a message has to go through a router, which takes a much longer time. In this sense we may say that the APPB is communication efficient, and in particular global-communication efficient.

3.2. Embedding Binary trees in Linear APPB

In addition to efficiently accomplishing the above commonly used message routing patterns, many well known communication structures can be efficiently realized by embedding them in APPB's. Once such embeddings are obtained, all algorithms designed for these structures can be efficiently executed on APPB's. In the remaining of this section we use the complete binary tree network to illustrate how to embed a communication structure in the linear APPB. To show that the binary tree network can be embedded in the linear APPB it is sufficient to find the *wait* function for each processor in the linear APPB such that the desired message routing patterns in the binary tree are accomplished.

Let L be the number of levels of a binary tree and let the root of the tree be node 1. Each node i, $i \geq 1$, which is not a leaf node, has two children, $2i + \delta$, where $\delta = 0, 1$, corresponding to i's left and right child, respectively (see Fig 3.2(a)). Consider an embedding in which node i in the tree is mapped to node i in the APPB. That is, our embedding is defined by the mapping $M_1(i) = i$. For convenience, we will call this embedding E_{t1} (see Fig 3.2(b)). In E_{t1}, the wait functions for node i to receive messages from its children are:

$$wait_{c,\delta}(i) = \begin{cases} i - (2i + \delta) = -(i + \delta), & i < 2^{L-1} \\ 0, & otherwise \end{cases}$$

Thus, to realize children-to-parent message routing each parent should wait for $wait_{c,0}(i)$ and $wait_{c,1}(i)$ time to read the messages from its left and right child, respectively. Clearly this routing task can be performed using one bus cycle.

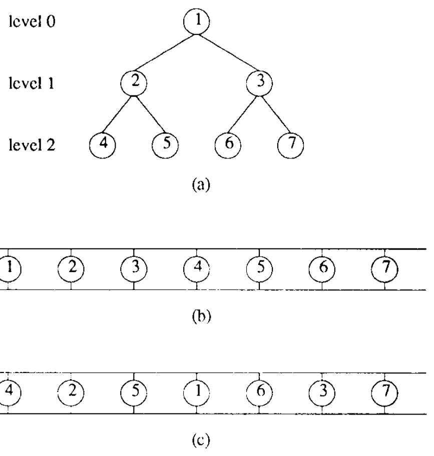


Fig 3.2. Embeddings of binary trees in the linear APPB.
(a) A binary tree. (b) Its first embedding, E_{t1}.
(c) Its second embedding, E_{t2}.

For parent-to-children message transfer in E_{t1}, each parent has two messages to send to its two children, respectively. To avoid message collision, two bus cycles are needed for carrying out such a routing task, one to send messages to left children and the second to send messages to right children. Let $wait_{p,\delta}(i)$ be the wait function for a child node i to receive the message from its parent. Then we have, during the first bus cycle,

$$wait_{p,0}(i) = i - \frac{i}{2} = \frac{i}{2}, \qquad i \ even$$

and during the second cycle we have

$$wait_{p,1}(i) = i - \frac{i-1}{2} = \frac{i+1}{2}, \qquad i = odd\ , i \neq 1$$

where, in each cycle, $wait_{p,\delta}(i) = 0$ if not specified.

Mapping each node i in the binary tree network onto node i in the APPB, as we have just shown above, is a straightforward approach. Using this straightforward approach we can embed any type of networks in the APPB. This approach, however, may not give a good embedding in the sense that it may take more time than needed, in number of bus cycles, to accomplish a given communication task. As will be seen, another tree embedding, E_{t2}, has a better communication efficiency than E_{t1}.

Let i, $1 \le i < 2^L$, be a node at level l, $0 \le l < L$, in the L-level binary tree, and $M_2(i)$, be the node in the linear APPB to which i is mapped in embedding E_{t2}. Then E_{t2} is defined by

$$M_2(i) = 2^{L-l}(i \ mod \ 2^l) + 2^{L-l-1}.$$

The wait functions for i to receive the messages from its two children are:

$$wait_{c,\delta}(i) = \begin{cases} (-1)^{\delta}\, 2^{L-l-2}, & i < 2^{L-1} \\ 0, & otherwise \end{cases}$$

Embedding E_{t2} may be viewed as being obtained by pressing the binary tree from the root down until all the nodes fall in the level of the leaf nodes (see Fig 3.2(c)). For this embedding, the two children of a node i are on opposing sides of i. Thus the parent-to-children message routing pattern in E_{t2} is different from the one in E_{t1} in that the two messages from a parent will travel on two different busses. Then the two messages from each parent node can be simultaneously injected on the two busses, respectively. As a result, the parent-to-children, as well as the children-to-parent, routing pattern can be accomplished in one bus cycle. $wait_p(i)$ can be obtained by noting that $wait_{p,\delta}(i) = -\, wait_{c,\delta}(I)$, where I is the parent of i. That is, the wait functions for children-to-parent message transfer are

$$wait_{p,\delta}(i) = \begin{cases} (-1)^{\delta+1} 2^{L-l-1}, & i > 1 \\ 0, & i = 1 \end{cases}$$

4. Two-Dimensional Array Processors with Pipelined Optical Busses

Long linear optical busses have the disadvantage that a message transfer may incur $N\tau$ time delay in an N-processor system. To reduce this delay to $O(\sqrt{N})$, we consider two dimensional APPB's. Assume that we have $N = m \times n$ processor nodes optically interconnected as a two-dimensional $m \times n$ array. In this two-dimensional APPB, each node is coupled to four busses as shown in Fig 4.1, where the two horizontal busses are used for passing messages horizontally in the same way as before, and the two vertical busses are used for passing messages vertically in a similar way. Each node in the array will be given two identifications, one being a pair of numbers (x, y), $0 \le x < m$, $0 \le y < n$, indicating the row-column position of the node in the two-dimensional $m \times n$ APPB, and the other being the row-major index, $i = xn + y$, of the node. Corresponding to the *bus cycle* defined for the linear case, we define, in the two-dimensional APPB, $n\tau$ and $m\tau$ as a *row bus cycle* and a *column bus cycle*, respectively, where τ is a *petit cycle* as defined previously. When there is no confusion, e.g., while talking about message transmissions in a row, we will simply say a *bus cycle* instead of a *row bus cycle*.

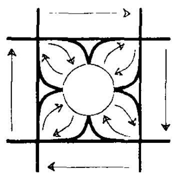


Fig 4.1. A node coupled to 4 waveguides in the two-dimensional APPB.

4.1. Message Routing in Two Dimensional APPB

A unique issue that arises in two-dimensional APPB is the relay of messages. Specifically, suppose a message is to be transferred from node (x_1, y_1) to node (x_2, y_2), with $x_1 \ne x_2$ and $y_1 \ne y_2$, and it has been decided that the row transfer is done first. Thus the message will go from (x_1, y_1) to (x_1, y_2), which is the node at the intersection of row x_1 and column y_2, in the first bus (a row bus cycle), and from (x_1, y_2) to (x_2, y_2) in the second bus cycle (a column bus cycle). That is, at the end of the first bus cycle, the message has to be buffered at node (x_1, y_2). For the purpose of relaying the message, we define a control function *relay* for node (x_1, y_2) as follows

$$relay\,[(x_1, y_2)] = y_2 - y_1,$$

which indicates that node (x_1, y_2) will read a message from a row bus at time $y_2 - y_1$ (relative to the start of the row bus cycle), and then write that message on the proper column bus at the beginning of the following column bus cycle. If $relay\,[(x_1, y_2)] = 0$, then no message is to be relayed by node (x_1, y_2). Clearly, in the worst case, up to n messages have to be relayed and, therefore, n relay buffers are needed at the relaying node. Now we are ready to show how the four most commonly used message routing patterns discussed in the previous section can be realized in the two-dimensional APPB.

One-to-one: The system executes a $SEND\,[(x_1, y_1), (x_2, y_2)]$ instruction, which requires that node (x_1, y_1) sends a message to node (x_2, y_2). Thus, we have $relay\,[(x_1, y_2)] = y_2 - y_1$ (in row bus cycle), and $wait\,[(x_2, y_2)] = x_2 - x_1$ (in column bus cycle). This communication takes 2 bus cycles.

Broadcast: The system executes a $BROADCAST\,[(x, y)]$ instruction, which states that node (x, y) is to broadcast the same message to all other nodes (x_j, y_j). In the row bus cycle, (x, y) broadcasts its message to all nodes (x, y_j), $y_j \ne y$. Then in the following column bus cycle all (x, y_j), including (x, y) broadcast the message in their corresponding columns. Thus $relay\,[(x, y_j)] = y_j - y$, and $wait\,[(x_j, y_j)] = x_j - x$. This communication also takes 2 bus cycles.

Semigroup Communication This corresponds to the execution of $SEMIGROUP\,[(x, y)]$, which says that some global information is to be computed and stored at node (x, y). Clearly this task can be accomplished using two linear semigroup communications, one in rows and the other in a column. That is, first we view each row as a linear APPB and do *SEMIGROUP* (y) in all rows. Then in column y, we perform *SEMIGROUP* (x). Thus $2\log(n)$ bus cycles are needed for this task.

Permutations: Let $PERM\,[(x, y)]$ be an arbitrary permutation. To avoid using up to n relays at each node, we can use a three-phase routing approach [18,25] or equivalently a three-bus-cycle approach in the two-dimensional APPB. In this approach the first bus cycle is a "preprocessing" step which distributes the messages in each row such that the messages going to the same row will occupy different columns. Then the second and third bus cycles will route the messages to their destination row and destination node, respectively. Although this approach is efficient, it is pointed out that the approach inherently requires a central arbiter for arbitrary permutations. This is because the preprocessing step requires the construction of a bipartite graph and the partition of the bipartite graph into complete matchings.

4.2. Embedding Binary Trees in Two-Dimensional APPB

As mentioned in the previous section, arbitrary message routing and permutations in two-dimensional APPB may require either three bus cycles or up to n relay buffers in each node (in the worst case). In this subsection, we present an embedding for binary tree networks in which only one relay buffer is needed to route messages. Our embedding of a binary tree network into the two-dimensional APPB is defined by a mapping $M_3(i) = (M_{3,x}(i), M_{3,y}(i))$, which maps each node i, $1 \le i \le 2^L - 1$, in the tree to a node $(M_{3,x}(i), M_{3,y}(i))$ in the two-dimensional APPB. Let i be a node at level l, $0 \le l < L$, in the binary tree. The mapping is defined by

$$M_{3,x}(i) = \begin{cases} 0, & 1 \le i < 2^k \\ 2^{l-k} + i \bmod 2^{l-k}, & 2^k \le i < 2^L \end{cases}$$

and

$$M_{3,y}(i) = \begin{cases} i, & 1 \le i < 2^k \\ \left\lfloor \dfrac{i \bmod 2^l}{2^{l-k}} \right\rfloor, & 2^k \le i < 2^L \end{cases}$$

As an example, the embedding for tree in Fig 4.2(a) is shown in Fig 4.2(b). Let us call this embedding E_{t3}. E_{t3} has these properties: (i) Parent nodes i, $1 \le i < 2^{k-1}$, and their children are in row 0; (ii) Parent nodes i, $2^{k-1} \le i < 2^k$, which are in row 0, have their children in row 1; and (iii) Parent nodes i, $2^k \le i < 2^{L-1}$, and their children are in the same column. Properties (i) and (ii) are obvious. Here we only prove (iii). Since in a binary tree network each parent node i has two children $2i + \delta$, $\delta = 0, 1$, to prove (iii) we need only show that $M_{3,y}(i) = M_{3,y}(2i + \delta)$ for $2^k \le i < 2^{L-1}$. Let i be a parent node at level l, where $k \le l < L-1$ and $i = p2^l + q$ for some integers p and q such that $0 \le q < 2^l$. Then we have

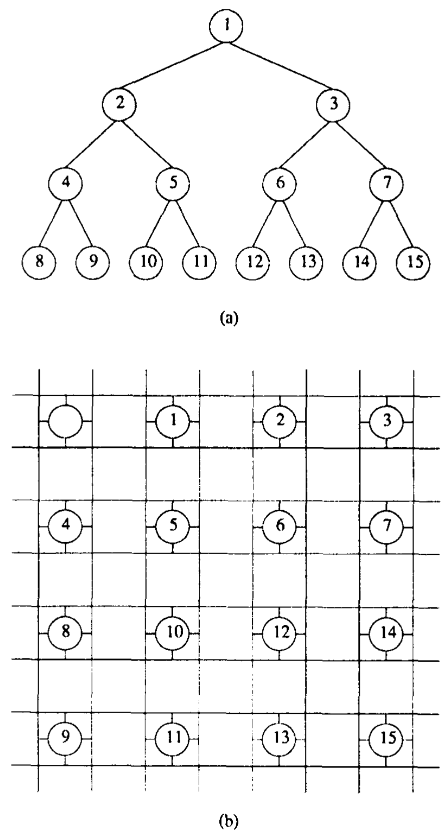


Fig 4.2. (a) A binary tree. (b) Its embedding, called E_{t3}, in the two-dimensional APPB.

$$M_{3,y}(2i + \delta) = \left\lfloor \frac{(2(p2^l + q) + \delta) \bmod 2^{l+1}}{2^{l+1-k}} \right\rfloor$$

$$= \left\lfloor \frac{(p2^{l+1} + 2q + \delta) \bmod 2^{l+1}}{2^{l+1-k}} \right\rfloor$$

$$= \left\lfloor \frac{2q + \delta}{2^{l+1-k}} \right\rfloor = \left\lfloor \frac{q}{2^{l-k}} \right\rfloor$$

$$= M_{3,y}(i).$$

It is now clear that the relay function is not needed for message transfers between parent nodes i and their children if $1 \le i < 2^{k-1}$ or $2^k \le i < 2^{L-1}$. However, such a relay is needed if $2^{k-1} \le i < 2^k$. The wait and relay functions for E_{t3} are obtained next.

Let $wait_{c,\delta}[(x,y)]$, where $(x,y)=M(i)$, be the *wait* functions for a parent node i to receive a message from its left and right child for $\delta=0,1$, respectively. For the case $1\le i<2^{k-1}$, the results for E_{t1} in the linear APPB directly gives $wait_{c,\delta}[(x,y)]=-(y+\delta)$. For the case $2^k\le i<2^{L-1}$, let i be at level l, $k\le l<L-1$, and $i=p2^{l-k}+q$. Then it can be shown that

$$M_x(i)=2^{l-k}+q,$$

$$M_x(2i+\delta)=2^{l+1-k}+2q+\delta,$$

and thus

$$wait_{c,\delta}[(x,y)]=M_x(i)-M_x(2i+\delta)=-(x+\delta).$$

$wait_{p,\delta}(i)$ can be easily obtained by noting that $wait_{p,\delta}(i)=-wait_{c,\delta}(I)$, where I is the parent of i.

For the case $2^{k-1}\le i<2^k$, both wait and relay functions are needed. Let $relay_{c,\delta}[(0,y)]$, $\delta=0,1$ and $0\le y<2^k$, be the *relay* function of node $(0,y)$ for relaying the message from a left and right child node, respectively, to its parent. Then we can show that

$$relay_{c,\delta}[(0,y)]=-1,\quad 0\le y<2^k,$$

$$wait_{c,\delta}[(0,y)]=2^k-y-\delta,\quad 2^{k-1}\le y<2^k.$$

Note that each node $(0,y)$ needs to relay only *one* child-to-parent message with the message from left (right) child being relayed by $(0,y)$ with y even (odd), and that although node 0 is not a node in the tree it helps relaying messages. Also note that $relay_{c,\delta}$ is applicable to column bus cycles. Now let $relay_{p,\delta}[(0,y)]$, y even (odd), be the relay function for node $(0,y)$ to relay the message from a parent $(0,Y)$ to its left (right) child for $\delta=0$ (1). Then $relay_{p,\delta}$ and $wait_{p,\delta}$ are easily obtained from $relay_{p,\delta}[(0,y)] = -wait_{c,\delta}[(0,Y)]$. $wait_{p,\delta}[(0,Y)]$ is determined as in the linear case.

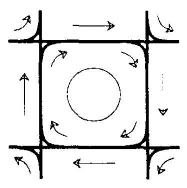
5. A Structural Variation of Array Processors with Pipelined Optical Busses

The basic two-dimensional APPB architecture in the previous section was proposed to reduce the length of a bus cycle for an N-processor system from N in linear APPB to $\sqrt{N}$. Although the reduction in the length of a bus cycle is significant, in general it takes 2 bus cycles, a row bus cycle and a column bus cycle, for two processors to communicate with each other, while only 1 bus cycle is necessary for the same communication in the linear case. Such 2-bus-cycle communication requires a message relay which involves an optical-electronic-optical information conversion, reducing the communication efficiency.

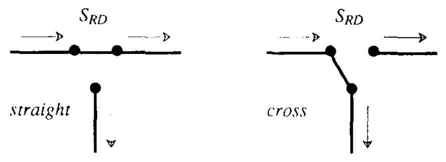
Two approaches may be used to deal with this disadvantage: a "software" approach and a hardware approach. The software approach relies on designing algorithms that require only communications between two processors on the same bus. For example this approach has been used in [10, 11] to obtain embeddings of many interconnection networks, e.g., binary trees, hypercubes, and pyramids, into the basic APPB such that any two neighboring nodes in the source network are mapped to the same bus in the two-dimensional APPB. In this paper we consider a hardware approach. In particular, we propose a structural variation of the basic APPB, called **APPB with switches**. In this structural variation of the basic APPB some electronically controlled optical switches are used to switch an optical signal, say, from a row bus to a column bus, eliminating the optical-electronic-optical conversion in the basic APPB. In an $m\times n$ APPB with switches, a bus cycle is defined as $(m+n)\tau$.

5.1. The Architecture of APPB with Switches

In APPB with switches the connection of the switches at each node is as shown in Fig 5.1(a), where each switch is a Ti:LiNbO$_3$ switch [22, 26] of the type used to implement the 4×4 nonblocking interconnection network in [8]. A switch may assume one of the two states *straight* and *cross* as defined in Fig 5.1(b). Initially all the switches are at state *straight*, which corresponds to the case where there is no message switching. That is, the APPB with straight switches is identical to the basic APPB architecture. When a message switching is desired at some node, a switch at that node must be set to the *cross* state.



(a)



(b)

Fig 5.1. (a) Switch connections at each node in APPB with switches. (b) Difinition of switch states.

To determine a switch state at a node (x,y) in an $m\times n$ array, we define a variable $S_{ij}(x,y)$, $0\le x<m$, $0\le y<n$, and $i,j\in\{R,L,D,U\}$, where R,L,D, and U stand for rightward, leftward, donwward, and upward, respectively. For example, $S_{RD}(x,y)$ is used to specify the control of the switch which guides optical signals in rightward-to-downward direction at node (x,y). The value of $S_{ij}(x,y)$ is a tuple (λ,μ), where the integer λ specifies the time, in number of petit cycles and relative to the beginning of a bus cycle, at which the switch is set to *cross*, and the integer μ determines the period,

again in number of petit cycles, during which the switch should remain *cross*. Note that although the length of a bus cycle is $m+n$, a switch must be set to *cross* before time n so that a message can be switched from a row bus to a column bus, that is, $0 \le \lambda < n$. For the same reason if messages are to be switched from column busses to row busses, then $0 \le \lambda < m$. In both cases, $0 \le \mu < m+n-\lambda$.

For example, if $S_{RD}(x, y) = (0, 2)$, then switch $S_{RD}(x, y)$ should be set to *cross* at time 0, i.e., the beginning of the first petit cycle of a bus cycle (which is also the beginning of that bus cycle) and should remain *cross* for 2 petit cycles thereafter in that bus cycle. If $S_{RD}(x, y) = (\lambda, 0)$, switch $S_{RD}(x, y)$ should remain *straight* throughout an entire bus cycle, as in its initial state. Clearly $S_{ij}(x, y) = (\lambda, m+n-1-\lambda)$ means that a switch should remain *cross* until the end of a bus cycle once it is set to *cross* at time λ, and $S_{ij}(x, y) = (0, m+n-1)$ means that a switch should be set to *cross* at the beginning of a bus cycle and then remain *cross* throughout the entire bus cycle.

In the basic APPB architecture messages will not collide as long as the collision-free condition (1) in Section 2 is satisfied. This condition, however, is not sufficient for APPB with switches, as can be seen in Fig 5.2 where two messages from source nodes (s, t) and (u, v), respectively, are colliding on the downward bus at node (x, y). Thus additional condition must be imposed to ensure collision-free communications in APPB with switches. For this, we present the following Lemma.

Lemma Assume that the collision-free condition (1) in Section 2 is satisfied and that all nodes start writing their messages simultaneously. Then, two messages from two distinct nodes (s, t) and (u, v), respectively, passing node (x, y) on the same bus will collide iff

$$|s-x| + |t-y| = |u-x| + |v-y|$$

Proof Noting that two messages from (s, t) and (u, v) will collide at node (x, y) iff they arrive at (x, y) simultanenously, i.e., their Manhattan distances to (x, y) are equal, the proof can be completed by simple distance arguments and is thus omitted.

5.2. Embedding Binary Trees in APPB with Switches

In this section we use binary trees as an example to illustrate how to embed interconnection networks in APPB with switches. The embedding, called E_{t4}, is obtained by modifying embedding E_{t3} for the basic APPB to facilitate switch states in APPB with switches. (In the following we use the same notation as that in the definition of E_{t3}. So the reader may refer to Section 4.2 for the meaning of each variable.) The embedding is defined by a mapping $M_4(i) = (M_{4,x}(i), M_{4,y}(i))$ with the modification that for $0 \le i < 2^k$, we have $M_{4,y}(i) = 2^{l-k}(i \ mod \ 2^l) + 2^{l-k-1}$ instead of $M_{3,y}(i) = i$ as in the definition of E_{t3}. That is our embedding E_{t4} is defined by the mapping $M_4(i) = (M_{4,x}(i), M_{4,y}(i))$, where

$$M_{4,x}(i) = \begin{cases} 0, & 1 \le i < 2^k \\ 2^{l-k} + i \ mod \ 2^{l-k}, & 2^k \le i < 2^L \end{cases}$$

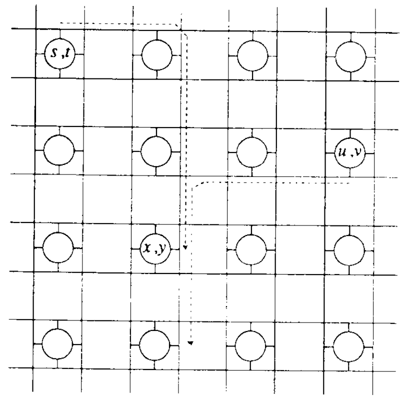


Fig 5.2. Message collision in APPB with Switches. Two messages from source nodes (s, t) and (u, v), respectively, are colliding on the downward bus at node (x, y).

and

$$M_{4,y}(i) = \begin{cases} 2^{k-l}(i \ mod \ 2^l) + 2^{k-l-1}, & 1 \le i < 2^k \\ \left\lfloor \dfrac{i \ mod \ 2^l}{2^{l-k}} \right\rfloor, & 2^k \le i < 2^L \end{cases}$$

Note that if we view levels 0 through $k-1$ of the L-level binary tree to be embedded as a k-level subtree, then in both E_{t3} and E_{t4} this subtree is embedded into row 0 of the two-dimensional APPB. The difference is that in E_{t3} this subtree was embedded into row 0 using embedding $M_1(i)$, while in E_{t4} it is embedded into row 0 using $M_2(i)$. As an example Fig 5.3 shows the embedding of the binary tree of Fig 4.2(a) into APPB with switches.

In E_{t4}, communications between parents i and their children are the same as those in E_{t2} and E_{t3} for $1 \le i < 2^{k-1}$ and $2^k \le i < 2^{L-1}$, respectively. For communications between the parents i, $2^{k-1} \le i < 2^k$, which are mapped to row 0, and their children, which are mapped to row 1, the switch states and the *wait* functions are determined in the following.

Assume that in E_{t4}, node i, $2^{k-1} \le i < 2^k$, in the L-level binary tree is mapped to node $(0, y)$, y odd, in the APPB with switches. Then it can be shown that the two children $2i + \delta$ of i are mapped to nodes $(1, y-1+\delta)$. That is, the right child of i is mapped to the same column y as i, and the left child of i is mapped column $y-1$. Therefore switching is needed only for the communication between i and its left child. We obtain the following switch state for i to send a message to its left child.

$$S_{LD}(0, y) = \begin{cases} (0, 2), & \text{if } y \text{ even} \\ (0, 0), & \text{otherwise} \end{cases}$$

Similarly the switch control for i to receive a message from its

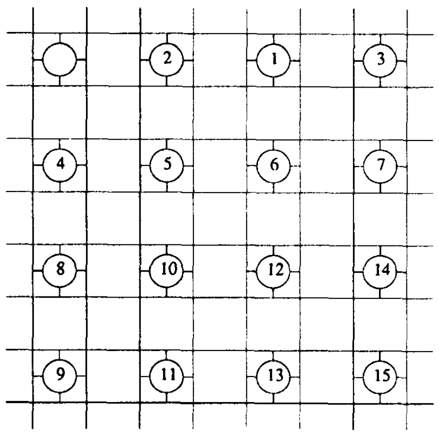


Fig 5.3. Embedding E_{t4} of the binary tree of Fig 4.2(a) in APPB with switches.

left child is

$$S_{UR}(0,y) = \begin{cases} (0,2), & \text{if } y \textit{ even} \\ (0,0), & \textit{otherwise} \end{cases}$$

For all other switches not specified above, we have $S_{ij}(x,y) = (0,0)$. It should be easy to check, using the Lemma, that these switch controls do not cause message collision in E_{t4}.

The *wait* functions for i to receive a message from its left and right child are, respectively,

$$wait_{c,0}[(0,y)] = \begin{cases} 2, & \text{if } y \textit{ odd (on row bus)} \\ 0, & \textit{otherwise} \end{cases}$$

and

$$wait_{c,1}[(0,y)] = \begin{cases} -1, & \text{if } y \textit{ odd (on column bus)} \\ 0, & \textit{otherwise} \end{cases}$$

The *wait* functions for i's children to receive a message from i can be obtained similarly.

In E_{t4}, the k-level subtree, which consists of levels 0 through $k-1$, is embedded into row 0 using E_{t2} instead of E_{t1}. As pointed out in Section 3.2, E_{t1} has the disadvantage that the parent-to-children message routing in linear APPB requires two bus cycles so that message collision will not occur. Should we use E_{t1} for the subtree embedding in row 0, it would also take two bus cycles for such message routing. Using E_{t2} for the subtree embedding in row 0 successfully solves the problem.

6. Performance Analysis

In this section, we evaluate the merit of our pipelined communication model by comparing it with linear arrays with nearest neighbor connections and exclusive access busses. We evaluate the different models irrespective of the technology used to implement these models. That is, we assume that the transmission rate and the propagation delay are the same for both optical and electronic communication links.

Consider a linear array of n processors with nearest neighbor connections as shown in Fig 6.1 and assume that the physical separation between each pair of neighboring processors is D. Such an array can emulate one cycle of a pipelined bus in a time $n(T_p + T_D)$, where T_D is the propagation time required for a signal to travel a distance D and T_p is the time required to process a message at the sending and the receiving ends of a communication link. T_p includes synchronization, message generation, buffering and routing. The bandwidth of the array, B_a, defined as the maximum number of messages that may be transmitted per second, is thus given by

$$B_a = \frac{n}{n(T_p + T_D)} = \frac{1}{T_D}\,\frac{1}{\rho + 1}$$

where $\rho = T_p \,/\, T_D$.

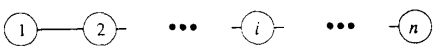


Fig 6.1. A linear array with nearest neighbor connections.

For the pipelined linear APPB, the optical distance, D_o, between consecutive processors should be larger than the message length bwc_g (see equation (1) in Section 2). In other words, if $D \geq bwc_g$, then $D_o = D$, otherwise, D_o should be made equal to bwc_g (for example by coiling an optical fiber) so that each processor can inject a message into the bus without collision. Thus, the signal propagation time between two consecutive processors, T_{D_o}, is $\max\{T_D, \alpha T_D\}$, where $\alpha = (bwc_g)/D$. The pipelined bus cycle time is then $T_p + nT_D \max\{1, \alpha\}$. Given that n messages may be transmitted during a pipelined bus cycle, the bandwidth of the pipelined bus is

$$B_p = \frac{n}{T_p + n\, T_D \max\{1,\alpha\}} \tag{2}$$

and thus,

$$\frac{B_p}{B_a} = \frac{n(\rho + 1)}{\rho + n \max\{1,\alpha\}} \tag{3}$$

In Fig 6.2, a parametric plot showing the relation between B_p / B_a and ρ is given in terms of n for $\alpha \leq 1$ and $\alpha > 1$. The curve for $\alpha \leq 1$ corresponds to the case where the message length is less than or equal to the physical separation between processors. The curve for $\alpha > 1$ reflect the cases where message length is longer than the physical separation D between processors and thus the optical path has been extended to accommodate the entire message. By taking the limit of equation (3) as $\rho \to \infty$, it is clear that, for fixed α and large ρ, the bandwidth ratio B_p/B_a approaches n. Also, when $\rho = 1$ and $\alpha \leq 1$, we obtain $B_p/B_a \approx 2$.

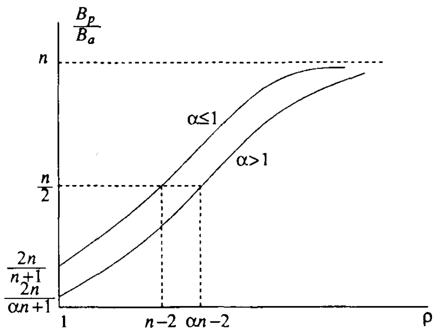


Fig 6.2. The ratio, B_p/B_a, of the bandwidth of a pipelined bus to that of a linear array with nearest neighbor connections as a function of ρ, α and n.

For *multiprocessor interconnections*, D is determined by placement and routing within VLSI chips, by PC board connections, or by back plane interconnections. In all cases, D is relatively small, and thus T_D is small. Given that T_p is, at least, on the order of microseconds, the ratio, ρ, of processing to communication times should be much larger than one (on the order of 10-1000). Also, with current technology, it is reasonable to assume that α is relatively small (between 1 and 10). For example, for board to board communications ($D \approx 10$ cm), it is possible to drive an optical communication line at the speed of 10 GHz. Assuming that the speed of light in optical fibers is $c_g = 2 \times 10^8$ m/sec, and that each message contains $b = 16$ bits, we obtain $\alpha \approx 3$. The same value of α is obtained if optical communications are implemented on GaAs wafers at 100 GHz and a physical processor separation of 1 cm. Note that the value of α may be reduced if parallel busses are used to reduce b.

Next we compare the bandwidth of pipelined busses with that of exclusive access busses. Given that the bandwidth of exclusive access busses is $B_e = 1/(T_p + nT_D)$, we have

$$\frac{B_p}{B_e} = \frac{n(\rho + n)}{\rho + n\max\{1,\alpha\}}$$

This shows that, as α approaches one, the pipelined bus can accommodate n messages in the same cycle time as the exclusive access bus. For $\alpha > 1$, the pipelined bus cycle will be stretched to accommodate the length of the messages, and thus, the performance gain due to pipelining will be less than n. However for any given $\alpha > 1$, as ρ goes to infinity B_p/B_e will approach n.

The above analysis is independent of the media used for communication. If optical pipelined busses are to be compared with electronic busses, then the physical constraints on the electronic propagation speed should be taken into account. Specifically, the effect of capacitive loading and mutual inductance on the signal propagation speed (the transmission line effect) should be considered in the electronic case. Thus, message pipelining using electro-optical technology offers a potential for substantially enhancing bandwidth utilization. Further, message pipelining techniques will be of increasing effectiveness because this technology offers the capability of generating very short pulses [9, 27] thus reducing w and decreasing α.

7. Concluding Remarks

We have presented efficient communication models which exploit the property of unidirectional propagation of optical signals to pipeline messages on optical busses. As shown in Section 6, the pipelined model has its merits irrespective of the technology in which it is implemented. Although the presentation in this paper is based on an optical model in which delays inherent in optical fibers serve as slots for space multiplexing, it is possible to use shift registers as buffer memories for these slots [29]. Thus pipelined busses may be implemented in either optics or electronics. However, for the electronic implementation, the signal propagation delay, T_D, will depend on the speed of the shift registers, resulting in a relatively small value for the ratio of processing to communication times, ρ.

In this paper we used only the binary tree networks to illustrate how to embed a communication structure into our APPB architectures. Optimal embeddings for other well known interconnection networks, including pyramids, X-trees [6], hypercubes, and shuffle-exchange networks have also been obtained [10, 11].

We have not considered in this paper issues that are relevant to the implementation of the proposed architecture. Such issues include the synchronization of the processing elements to the accuracy implied by the speed of optics, temporal pulse positioning, optical fanout and the distribution of optical power in a way that allows the detector at each processor to detect the optical signals correctly. These issues must be addressed with reguard to the reliability, scale, and device technology which is appropriate for computing applications. Results related to these issues may be found in [4, 19, 24].

References

1. S.H. Bokhari, "Finding Maximum on an Array Processor with a Global Bus," *IEEE Trans Comput*, vol. C-32, no. 2, pp. 133-139, 1984.
2. Y.C. Chen, W.T. Chen, G.H. Chen, and J.P. Sheu, "Designing Efficient Paralell Algorithms on Mesh-Connected Computers with Multiple Broadcasting," *IEEE Trans Parallel Distributed Systems*, vol. 1, no. 2, pp. 241-245, Apr 1990.
3. D.M. Chiarulli, R.G. Melhem, and S.P. Levitan, "Using Coincident Optical Pulses for Parallel Memory Addressing," *IEEE Computer*, pp. 48-57, Dec 1987.
4. D.M. Chiarulli, S.P. Levitan, and R.G. Melhem, "Optical Bus Control for Distributed Multiprocessors," *Journal of Parallel and Distributed Computing*, to appear.

5. B.D. Clymer and J.W. Goodman, "Optical Clock Distribution to Silicon Chips," *SPIE Proceedings*, vol. 625, pp. 134-138, 1986.

6. A.M. Despain and D.A. Patterson, "X-Tree: A Tree Structured Multi-Processor Computer Architecture," *5th Symp on Computer Architecture*, pp. 144-151, 1978.

7. M.J.B. Duff, D.M. Watson, T.J. Fountain, and G.K. Shaw, "A Cellular Logic Array for Image Processing," *Pattern Recognition*, vol. 5, pp. 229-237, 1973.

8. J.R. Erickson and H.S. Hinton, "Implementing a Ti:$LiNbO_3$ 4 × 4 Nonblocking Interconnection Network," *SPIE Integrated Optical Circuit Engineering*, vol. 578, pp. 201-206, 1985.

9. J. Fujimoto, A. Weiner, and E. Ippen, "Generation and Measurment of Optical Pulses as Short as 16 fs.," *Applied Physics Letters*, vol. 44, no. 9, 1984.

10. Z. Guo and R.G. Melhem, "Embedding Pyramids in Array Processors with Pipelined Busses," *International Conf on Application Specific Array Processors*, Princeton, NJ, 1990, to appear .

11. Z. Guo, "Array Processors with Pipelined Busses and Their Implication in Optically and Electronically Interconnected Multiprocessor Architectures," , Ph.D. Thesis, Dept of Electrical Engineering, University of Pittsburgh, in preparation.

12. D.J. Hunt, "The ICL DAP and Its Application to Image Processing," in *Languages and Architectures for Image Processing*, ed. Duff & Levialdi, 1981.

13. A.M. Jrad and R.W. Hall, "The OFC Enhanced Mesh Architecture: A Performance Study," *Proc of the 1987 Workshop on Comput Arch for Pattern Anal and Machine Intelligence*, pp. 184-191, 1987.

14. A.M. Jrad and R.W. Hall, "Orthogonal Fast Channels: An Enhanced Mesh Architecture," *International Conf on Parallel Processing*, pp. 828-831, 1987.

15. B.S. Kawasaki, K.O. Hill, and R.G. Lamont, "Biconical-Taper Single-Mode Fiber Coupler," *Optics Letters*, vol. 6, no. 7, pp. 327-328, July 1981.

16. R.G. Melhem, D.M. Chiarulli, and S.P. Levitan, "Space Multiplexing of Waveguides in Optically Interconnected Multiprocessor Systems," *The Computer Journal*, vol. 32, no. 4, pp. 362-369, 1989.

17. R. Miller and Q.F. Stout, "Mesh Computer Algorithms for Computational Geometry," *IEEE Trans Comput*, vol. C-38, no. 3, pp. 321-340, 1989.

18. M. Misra and V.K. Prasanna-Kumar, "Efficient VLSI Implementation of Iterative Solutions to Sparse Linear Systems," *Technical Report, Univ of Southern California*, no. IRIS #246, May 1989.

19. M. Nassehi, F. Tobagi, and M. Marhic, "Fiber Optic Configurations for Local AREA Networks," *IEEE Journal on Selected Areas in Communications*, vol. SAC-3, no. 6, pp. 941-949, Nov. 1985.

20. D. Nassimi and S. Sahni, "Data Broadcasting in SIMD Computers," *IEEE Trans Comput*, vol. C-30, pp. 101-107, 1981.

21. D. Nath, S.N. Maheshwari, and P.C.P. Bhatt, "Efficient VLSI Networks for Parallel Processing on Orthogonal Trees," *IEEE Trans Comput*, vol. C-32, no. 6, pp. 569-581, 1983.

22. A. Neyer, "Electro-Optic X-Switch Using Single-Mode Ti:$LiNbO_3$ Channel Waveguides," *Electronics Letters*, vol. 19, no. 14, pp. 553-554, July 1983.

23. V.K. Prasanna-Kumar and D. Reisis, "Image Computations on Meshes with Multiple Broadcast," *IEEE Trans PAMI*, vol. PAMI-11, no. 11, pp. 1194-1202, 1989.

24. P. Prucnal, D. Blumenthal, and P. Perrier, "Self Routing Photonic Switching Demonstration with Optical Control," *Optical Engineering*, vol. 26, no. 5, pp. 473-477, 1987.

25. C.S. Raghavendra and V.K. Prasanna-Kumar, "Permutations on ILLIAC-IV Type Networks," *IEEE Trans Comput*, vol. C-37, no. 7, pp. 662-669, 1986.

26. R.V. Schmidt and R.C. Alferness, "Directional Coupler Switches, Modulators, and Filters Using Alternating $\Delta\beta$ Techniques," *IEEE Trans on Circuits and Systems*, vol. CAS-26, no. 12, pp. 1099-1108, Dec 1979.

27. C. Shank, "The Role of Ultrafast Optical Pulses in High Speed Electronics," in *Picosecond Electronics and Opto-electronics*, ed. Morou G., Bloom D. and Lee C., Springer Verlag, 1985.

28. Q.F. Stout, "Mesh Connected Computers with Broadcasting," *IEEE Trans Comput*, vol. C-32, pp. 826-630, 1983.

29. A.S. Tanenbaum, *Computer Networks*, Prentice-Hall, Englewood Cliffs, NJ, 1981.

30. F. Tobagi, F. Borgonovo, and L. Fratta, "Expressnet: A High-Performance Integrated-Services Local Area Network," *IEEE Journal on Selected Areas in Communications*, vol. SAC-1, no. 5, pp. 898-912, 1983.

31. J.D. Ullman, *Computational Aspects of VLSI*, Computer Science Press, Rockville, MD, 1984.

32. M.S. Whalen and T.H. Wood, "Effectively Nonreciprocal Evanescent-Wave Optical-Fibre Directional Coupler," *Electronics Letters*, vol. 21, no. 5, pp. 175-176, Feb 1985.