JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **12,** 269–282 (1991)

Pipelined Communications in Optically Interconnected Arrays*

ZICHENG GUO, RAMI G. MELHEM, RICHARD W. HALL, DONALD M. CHIARULLI, AND STEVEN P. LEVITAN

Departments of Electrical Engineering and Computer Science, University of Pittsburgh, Pittsburgh, Pennsylvania 15261

Two synchronous multiprocessor architectures based on pipelined optical bus interconnections are presented. The first is a linear pipeline with enhanced control strategies which make optimal use of the available communication bandwidth of the optical bus. The second is a two-dimensional architecture in which processors are placed in a square grid and interconnected to one another through horizontal and vertical pipelined optical buses. These architectures allow any two processors to communicate with each other using one (for the linear case) or two (for the two-dimensional case) pipelined bus cycles. Further, they permit all processors to have simultaneous access to the buses using slots within a pipelined cycle. We show that the architectures have simple control structures and that well-known processor interconnections, e.g., the complete binary trees and the hypercube networks, can be efficiently embedded in them. These architectures have an effectively higher bandwidth than conventional bus configurations and appear to be good candidates for a new generation of hybrid optical–electronic parallel computers. © 1991 Academic Press, Inc.

1. INTRODUCTION

Two-dimensional meshes of processors have been extensively studied in various forms and augmentations [23, 26, 37]. Large-scale implementations of two-dimensional meshes have been built [2, 10, 17]. 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 buses [3, 10, 11, 35], reducing the communication diameter but giving only very small bandwidth improvements. Row and column bus augmentations [29, 30] have yielded both a low communication diameter and adequate bandwidth for certain classes of algorithms. Interconnection networks have been considered for augmenting rows and columns in a mesh including trees [27, 28, 39] and compounded graphs [18, 19]. The binary hypercube can also be viewed in this context as a two-dimensional mesh with horizontal and vertical hypercube interconnections [18, 19].

* This work was, in part, supported by Air Force Grant AFOSR-89-0469 and by NSF Grant MIP-8901053.

One of the simplest mesh augmentation schemes is the row and column bus augmentation. However, exclusive write access to buses 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 buses, signals propagate in both directions from the source, while optical channels are inherently directional and have precise predictable path 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 [5], to enhance the bandwidth in bus-connected multiprocessor systems [22], and to minimize the control overhead in networking environments [38].

In this paper, we present two multiprocessor architectures, called *Array Processors* with *Pipelined Buses* (APPB), which employ optical bus interconnections in processor arrays. In Section 2 we review the basic principle of pipelining messages on optical buses. In Section 3 we introduce our linear APPB, where processors are connected with a single optical bus. We present efficient approaches to message routing and network embedding, for the linear APPB as well as techniques for enhancing the bus utilization through enhanced control functions. In Section 4 we introduce our two-dimensional APPB, where processors are interconnected with horizontal and vertical optical buses. We discuss routing and embedding issues for this new architecture. We show how binary tree and hypercube interconnections can be effectively embedded and identify key design issues for effective embeddings of arbitrary interconnections. In Section 5 we compare the efficiency of the pipelined bus communication model with that of nonpipelined buses and of store and forward communications in nearest-neighbor structures. Finally, Section 6 contains concluding remarks.

2. MESSAGE PIPELINING ON OPTICAL BUSES

Consider the system of Fig. 1a, 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 waveguide with two passive couplers, one for injecting (writing) signals on the waveguide and the other

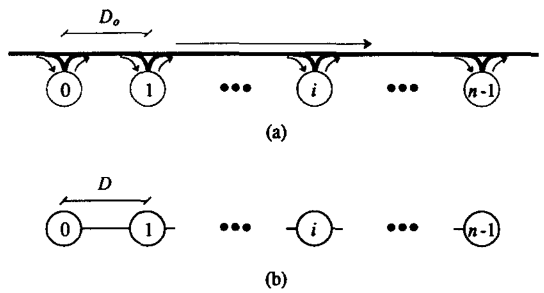


FIG. 1. (a) A system of n processors connected with a single optical waveguide (bus). (b) A linear array of n processors with nearest-neighbor connections.

for receiving (reading) signals from the waveguide [20, 40]. Each receiving coupler passively taps a percentage (typically 5–10%, depending on the coupling ratio) of the optical signal power available on the bus. Thus the couplers do not introduce any delay to the propagation of optical signals along the bus. However, the degradation of signal power does place an upper limit on the number of processors that can be connected on the bus [8]. As in the case of electronic buses, each processor j communicates with any other processor i by sending a message to i through the common bus. However, because optical signals propagate in one direction, a processor j may send signals to another processor i only if $i > j$.

Assume that a message on an optical bus consists of a sequence of optical pulses, each having a width w in seconds. The existence of an optical signal of width w represents a binary bit 1, and the absence of such a signal represents a 0. Note that w includes a time for electro-optical conversions, rise and fall times, and propagation delay in the latch of the receiver circuits [6]. For analytical convenience, we let D_o be the optical distance between each pair of adjacent nodes (it will become clear that the distance between two adjacent nodes need not be equal) 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 sender j writes its message on the bus. After a time $(i - j)\tau$ the message will arrive at the receiver i, which then reads the message from the bus.

The properties of unidirectional propagation and predictable path delays 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. 1a can write on the bus *simultaneously*, provided that the following collision-free condition [22] is satisfied,

$$D_0 > bwc_g, \qquad (1)$$

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 system 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 that two optical 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 [21]. An optically distributed clock can be broadcast without skew to each node, or electro-optical switches 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 on the bus in a pipelined fashion, as shown in Fig. 2. Thus we use the term *pipelined bus.* In the rest of this paper we always assume that the collision-free condition (1) 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 discussion in this section, we do not include in a bus cycle the time taken to prepare and process a message before it can be injected on the bus. This time is explicitly introduced in our performance analysis in Section 5. 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. Note that each cycle on the pipelined bus may be emulated by n cycles in a linear array with nearest-neighbor communications shown Fig. 1b. Comparison of the two interconnection schemes is made in Section 5.

Let us look at a simple routing task where each node transmits a message and each node is programmed to receive a message from the kth node (if it exists) to its left. All nodes start injecting messages at the beginning of a bus cycle, and all the messages 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 its message from the bus after $k\tau$ time from the beginning of the bus cycle. In this way, a message routing pattern in which each node sends a message to the kth node to its right has been realized. In fact, as will be seen, we can realize various message routing patterns in a simple, straightforward way.

3. LINEAR ARRAY PROCESSORS WITH PIPELINED BUSES

In the system of Fig. 1a, messages can be transmitted only from left to right. To allow message passing from right to

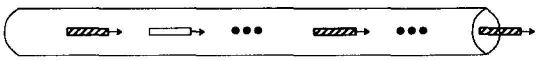


FIG. 2. Message pipelining on the optical bus. A blank rectangle indicates "no signal," implying that some processor is not sending a message.

left, another optical bus is used, as shown in Fig. 3a. In this figure, we have two optical buses; 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 in different buses do not disturb one another; that is, the two buses can support two separate pipelines. The system in Fig. 3a is our architecture of linear APPB. For convenience the linear APPB in Fig. 3a is schematically drawn as in Fig. 3b.

To specify the time at which a node should receive a message, we introduce a control function *twait*(i), which is defined 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

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

If τ is considered as a time unit, then *twait* can be interpreted in terms of the number of such time units and thus be written $twait(i) = i - j$. Clearly if $twait(i) > 0$, then the message is to be received from the left; if $twait(i) < 0$, then the message is to be received from the right. If $twait(i) = 0$, then no message should be received by node i. The value of *twait*(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 in one bus cycle.

This *twait* control function, however, has the disadvantages that it depends crucially on timing accuracy and is sensitive to the optical distance D_o between two adjacent nodes. An equivalent control function, *mwait*, that does not have these disadvantages may be defined if we require that each node inject a message, real or dummy, every bus cycle. In this case we define *mwait*(i) as the number of messages that node i should skip before reading its message. For example, if $mwait(i) = \gamma$, then node i should receive the $|\gamma|$th message that passes i on the bus. That is, it has to wait until $|\gamma| - 1$ messages have passed and then it reads its own message. The sign of γ determines on which bus the message should be received. Clearly *mwait* is equivalent to *twait* and

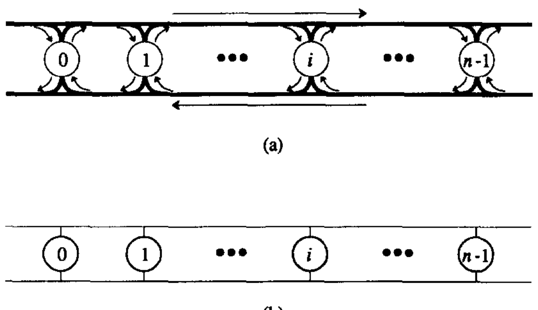


FIG. 3. (a) Linear array processors with pipelined buses (APPB). (b) A schematic drawing of (a).

either control function may be used. For convenience we simply write the control function as *wait*, and we assume that the optical distance between each pair of adjacent nodes i and $i + 1$ is constant.

The control function *wait* can only be used when the communication pattern is known to the receiver in the sense that the receiver knows from which node the message is to be received. In cases where the communication pattern is unknown to the receiver, the coincident pulse techniques [5, 21] may be used such that an addressing pulse and a reference pulse coincide at the detector of the receiver, thereby addressing it. In this paper we use *wait* for addressing since the communication patterns which we discuss are known to the receiver.

In the following we present techniques for message routing and network embedding in the linear APPB. For the purpose of evaluating the communication efficiency, we note that a lower bound on the number of bus cycles needed to transfer H messages in the linear APPB is $\lceil H/n \rceil$, where n is the number of nodes on the optical bus. This lower bound is obtained by assuming a perfectly even distribution of messages along the bus at each bus cycle, that is, *every* node has one message to send at *each* bus cycle.

3.1. *Message Routing in Linear APPB*

Various message routing patterns can be realized 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 common 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 message, and all other nodes i will receive that message. In this case, $wait(i) = i - j$ for *all* $i \neq j$.

Semigroup Communication [4]. The system executes a *SEMIGROUP*(i) instruction, which says 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 logically function as a tree with the root being node i. Later in this section we present embeddings of binary trees which facilitate such a 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.

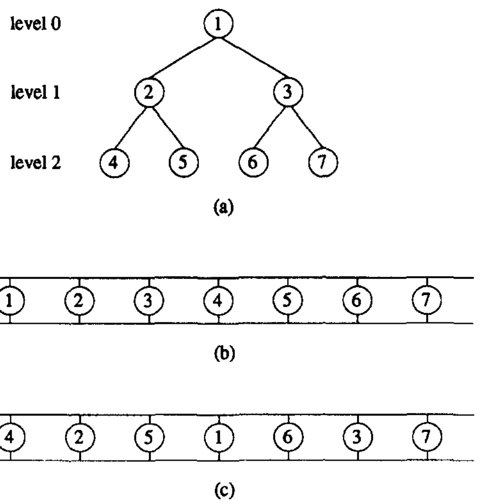
We see that the computation of *wait*(i) is very simple and uniform. The only difference among the *wait* functions for different message routing patterns is that the nodes involved are different. It is clear that all these communication tasks can be performed using a single bus cycle, except the semigroup communication, which takes $\log(n)$ bus cycles. Note that, in the linear APPB, message passing between two nonneighboring nodes is nearly as efficient as that between two neighbors. Specifically, a message takes τ more time to pass one more node on the optical bus. This is not the case in the linear array with nearest-neighbor connections shown in Fig. 1b, where to pass a node, en route to another node, a message has to go through a router. In this sense we may say that the APPB is communication efficient, and in particular global-communication efficient.

3.2. *Embedding Binary Tree and Hypercube Networks into Linear APPB*

In this subsection we show how to embed other interconnection networks into the linear APPB. Our first example is the embedding of complete binary tree networks. To show that a 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 routing pattern is accomplished.

Let L be the number of levels of a complete binary tree and let the root of the tree be node 1. Each node i, $i \geqslant 1$, 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. 4a for an example). Consider an embedding in which node i in the tree is mapped to node $i - 1$ in the linear APPB. For convenience, we call this embedding E_{t1} (see Fig. 4b). In E_{t1}, the wait functions for node i to receive a message from its children are:

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



(c)

FIG. 4. Embeddings of complete binary trees in the linear APPB. (a) A binary tree. (b) The first embedding, E_{t1}. (c) The second embedding, E_{t2}.

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.

For parent-to-children message transfer in E_{t1}, each parent has two messages to send to its two children, respectively. In this case, two bus cycles are needed to carry out such a routing task, one to send messages to left children and one to send messages to right children. Let $wait_{p,0}(j)$ and $wait_{p,1}(j)$ be the wait functions for a left child and right child, respectively, to receive a message from its parent. Then, during the first cycle we have

$$wait_{p,0}(j) = \begin{cases} j - \frac{j}{2} = \frac{j}{2}, & j = \text{even}, \\ 0, & \text{otherwise}, \end{cases}$$

and during the second cycle we have

$$wait_{p,1}(j) = \begin{cases} j - \frac{j-1}{2} = \frac{j+1}{2}, & j = \text{odd, and } j \neq 1, \\ 0, & \text{otherwise.} \end{cases}$$

Mapping each node i in the binary tree network onto node i (or $i - 1$ as was just done above) in the linear APPB is a straightforward approach. Using this straightforward approach we can embed any type of network in the linear 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 is seen next, another tree embedding, E_{t2}, has a better communication efficiency than E_{t1}.

Embedding E_{t2} may be viewed as pressing the binary tree from the root down until all the nodes fall in the level of the leaf nodes (see Fig. 4c). In this embedding the two children of a node i are on opposing sides of i. Thus the parent-to-children routing pattern, as well as the children-to-parent routing pattern, may be accomplished in one bus cycle. Specifically, if i is a node at level l, where l is the integer satisfying $2^l - 1 < i < 2^{l+1}$, then the wait functions for i to receive the messages from its two children are

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

The parent-to-children message routing pattern in E_{t2} is different from that in E_{t1} in that the two messages from a parent will travel on two different buses. Then the two messages from each parent node can be simultaneously injected on the two buses, respectively, in the same bus cycle. Hence, the parent-to-children routing pattern can be accomplished in one bus cycle. $wait_{p,\delta}$ can be determined by noting that

$wait_{p,\delta}(j) = -wait_{c,\delta}(i)$, where i is the parent of j. That is, the wait functions for parent-to-children message transfer are

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

Next, we consider a k-dimensional binary hypercube in which the nodes are numbered such that if nodes i and j are neighbors across dimension h, $1 \leqslant h \leqslant k$, then $|i-j| = 2^{h-1}$ (see Fig. 5a). Let E_{c1} be the embedding of this k-cube into a linear APPB such that each node i in the hypercube is mapped into node i in the linear APPB. With this embedding, a node in the hypercube may send a distinct message to each of its k neighbors if each node sends one message to one neighbor in each bus cycle. For example, at the hth bus cycle a message is sent from each node to its neighbor at distance 2^{h-1}. To accomplish this, the time that a node i has to wait during the hth bus cycle before receiving a message from its neighbor along the hth dimension is

$$wait_h(i) = \pm 2^{h-1}.$$

In our discussions so far, we have allowed each node to send only one message on each bus during each bus cycle. In other words after placing a message on the bus in the current cycle, all nodes must wait until the next cycle to initiate the next message. In the following subsection, we show that such a wait is not always necessary.

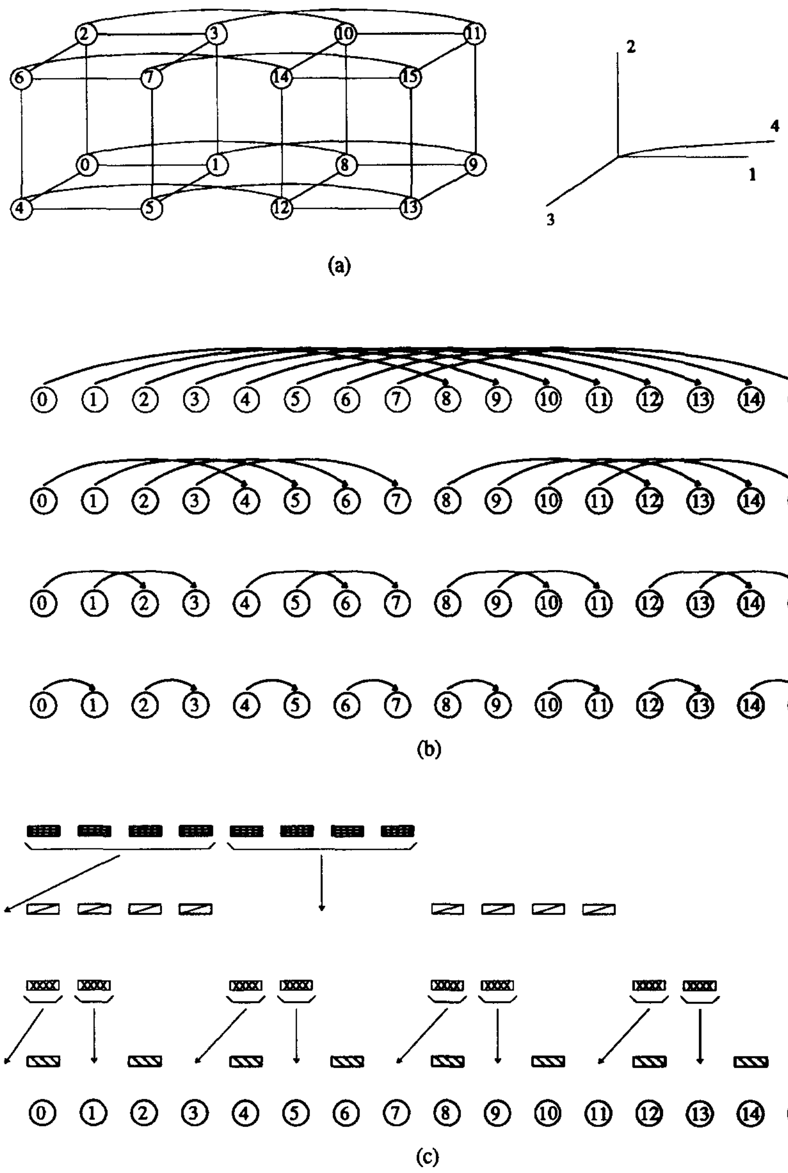


FIG. 5. (a) A binary hypercube and its dimension assignment. (b) Message routing patterns in the hypercube. (c) Message distribution in the hypercube.

3.3. *Interleaved and Overlapped Pipelining*

Up until now, we have required that each node send only one message on each bus in one bus cycle and that the transmission of messages be initiated at the beginning of a bus cycle. Given these two restrictions, no specific control function was needed for the initiation of messages. However, if some node does not have a message to send during a bus cycle, a slot of one petit cycle in duration will be created. Interleaved pipelining is a technique which tries to fully utilize the communication capacity of the pipelined bus by inserting a message into any available slot. This may be accomplished if a node is allowed to place more than one message on the same bus within a bus cycle, but at different petit cycles. To allow for this flexibility, a control function $send_g(j)$ must be used to specify the time, relative to the beginning of a bus cycle, at which node j should write its gth message on the bus.

To show how interleaved pipelining works, let us now examine the routing patterns in E_{c1}. Since message transfers in opposite directions on the two buses of the linear APPB form two separate and symmetric pipelines, we need to look at only one direction. Consider the left-to-right message transfer in E_{c1}, and define k sets, $S_h = \{j \mid 0 \leq j < n, 0 \leq (j \bmod 2^h) < 2^{h-1}\}$, $1 \leq h \leq k$, of nodes for the k-cube. That is, S_h is obtained by partitioning the n nodes of the hypercube into 2^h-node groups and including in S_h the first 2^{h-1} nodes in each group. For example, for the 4-cube in Fig. 5a, we have $S_1 = \{0, 2, 4, 6, 8, 10, 12, 14\}$, $S_2 = \{0, 1, 4, 5, 8, 9, 12, 13\}$, $S_3 = \{0, 1, 2, 3, 8, 9, 10, 11\}$, and $S_4 = \{0, 1, 2, 3, 4, 5, 6, 7\}$. Note that all the k sets, S_h, have the same cardinality 2^{k-1}, and each contains node 0. Hence, in the realization of the binary k-cube using a linear APPB, there are k routing patterns. In the hth pattern, $1 \leq h \leq k$, the nodes in set S_h send messages to their neighbors along the hth dimension in the hypercube, as indicated with the arrowed curves in Fig. 5b. Correspondingly, the messages can be divided into k sets, M_h, $1 \leq h \leq k$, which are sent by the k sets of nodes S_h, respectively. For the routing patterns in Fig. 5b, these message sets are shown in Fig. 5c.

Using interleaved pipelining, the messages in the two sets M_{2s-1} and M_{2s}, $1 \leq s \leq k/2$, are interleaved and sent in the same bus cycle. Let $send_1(j)$ and $send_2(j)$ be the times at which node j writes its messages in M_{2s-1} and M_{2s}, respectively, on the bus during bus cycle s. Correspondingly, let $wait_1(i)$ and $wait_2(i)$ be the wait functions for a node i to receive the messages in M_{2s-1} and M_{2s}, respectively, during bus cycle s. Then, for interleaved pipelining we have the following *send* functions for a node j at bus cycle s, $1 \leq s \leq k/2$:

$$send_1(j) = 0, \quad j \in S_{2s-1},$$

$$send_2(j) = \begin{cases} 0, & j \in S_{2s} \text{ and } 2^{2s-2} \leq (j \bmod 2^{2s}) < 2^{2s-1}, \\ 2^{2s-2}, & j \in S_{2s} \text{ and } 0 \leq (j \bmod 2^{2s}) < 2^{2s-2}. \end{cases}$$

The corresponding *wait* functions are

$$wait_1(i) = i - j, \quad j \in S_{2s-1},$$

$$wait_2(i) = \begin{cases} i - j, & j \in S_{2s} \text{ and } 2^{2s-2} \leq (j \bmod 2^{2s}) < 2^{2s-1}, \\ i - j + 2^{2s-2}, & j \in S_{2s} \text{ and } 0 \leq (j \bmod 2^{2s}) < 2^{2s-2}. \end{cases}$$

A node for which the *send* or *wait* function is not defined above should not send or receive any message. Note that the times determined by these *send* and *wait* functions are with respect to the beginning of each bus cycle s. Also note that since the receiving node i knows the id of the sending node j (since they are neighbors in the k-cube), it knows which of the two values of $wait_2(i)$ should be used. As an example, the interleaved pipelining for the messages in Fig. 5c is achieved by interleaving message sets M_1 and M_2 in the first bus cycle and M_3 and M_4 in the second bus cycle. The arrowed lines in Fig. 5c show how the messages are being interleaved, and the resulting message pipelines are shown in Fig. 6a.

It can be seen that using interleaved message pipelining, the total communication time taken for each node to send a message to each of its neighbors is $k/2 + 1$ bus cycles, where the last bus cycle is due to the time needed to clear out the first $n/4$ messages (sent by nodes 0 through $n/4 - 1$) in M_k that were inserted in front of M_{k-1}. Comparing with k bus cycles, the time needed if each node sends one message per bus cycle, our savings in the communication time is $(k - 2)/2$ bus cycles. Although this savings is significant there are still unused slots from the rightmost nodes on the bus, as can be seen from the message pipeline at time $t = 16$ in Fig. 6a. We next show how to utilize these empty slots using overlapped pipelining.

In overlapped pipelining, we pipeline the message pipelines obtained from interleaved pipelining by allowing the messages for bus cycle s to be initiated before bus cycle $s - 1$ terminates, as long as message collision does not occur. For this purpose we define a new control function, $step_s$, which specifies the time, with respect to the beginning of the first bus cycle, at which the messages for bus cycle s are initiated. Clearly, savings in communication time is possible if $step_s - step_{s-1} < n\tau$. In this case, we avoid confusion by calling the bus cycles *message transfer steps.*

In E_{c1}, the control function $step_s$, $1 \leq s \leq k/2$, specifies when S_{2s-1} and S_{2s} should start sending their messages. Specifically let $step_1 = 0$ and let $step_s$, $1 < s \leq k/2$, be the time interval in number of petit cycles between the initiations of steps 1 and s. Then, messages from step s and step $s - 1$ will not collide if

$$step_s = step_{s-1} + n - \frac{3}{4} 2^{2s-2}, \quad 1 < s \leq \frac{k}{2}.$$

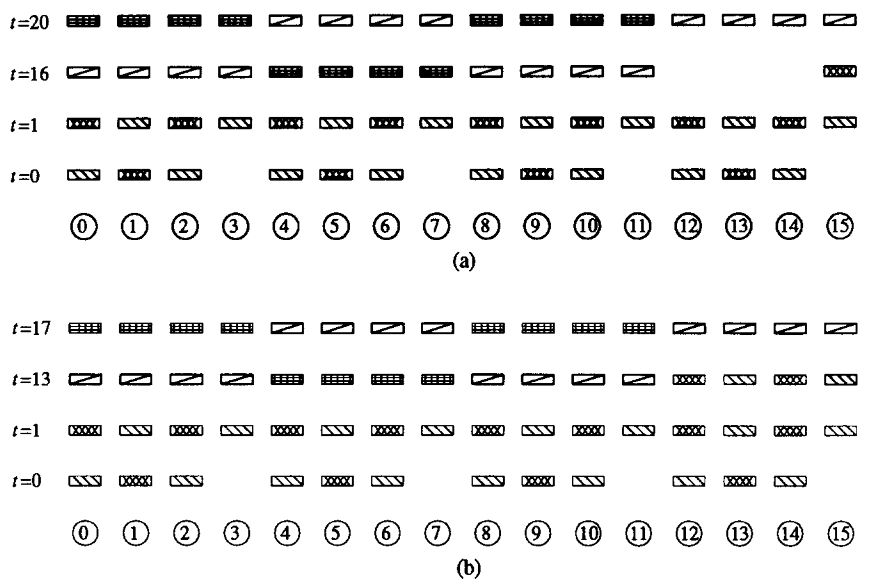


FIG. 6. (a) Interleaved pipelining and (b) overlapped pipelining of messages in the 4-cube. (t is measured in petit cycles.)

The *send* and *wait* functions defined in the previous subsection are still applicable here, but they are now defined with respect to the time determined by $step_s$, the beginning of transfer step s, rather than the beginning of each bus cycle s. Figure 6b shows the result of overlapped pipelining of the message pipelines in Fig. 6a. Note that in interleaved pipelining there was also some overlapping between the two message pipelines generated in two consecutive bus cycles, as can be seen from the message pipeline at time $t = 16$ in Fig. 6a. But, as has been mentioned previously, interleaved pipelining does not fully utilize the pipelined bus.

These control functions *step*, *send*, and *wait* together result in a minimized total communication time. To show this we first note that since the cardinality of M_h, $1 \leqslant h \leqslant k$, is $n/2$, the total number of messages is $kn/2$. Thus, if we assume that the message distribution over processors is perfectly even in each bus cycle (every processor has a message to send in each bus cycle), then the time needed for transferring these messages is at least $\lceil kn/2n \rceil = k/2$ bus cycles, or equivalently $kn/2$ petit cycles. In our case, however, such an assumption of even message distribution does not hold. For example, no message can be inserted on the bus at processor $n - 1$ in the first bus cycle, as can be seen from the message pipeline at time $t = 0$ in Fig. 6b. Now we compute the total time, in number of petit cycles, using the control functions determined above. It can be shown that

$$step_{k/2} = \sum_{s=2}^{k/2} \left(n - \frac{3}{4} 2^{2s-2} \right) = \left(\frac{k}{2} - \frac{5}{4} \right) n + 1.$$

The time due to $send_2$ at step $k/2$ is $2^{k-2} = n/4$. Finally it takes n petit cycles for the bus to clear out. Therefore the total time in number of petit cycles is

$$\left(\frac{k}{2} - \frac{5}{4} \right) n + 1 + \frac{n}{4} + n = \frac{k}{2} n + 1.$$

Finally, we note that interleaved message pipelining may also be applied to binary tree routing patterns. From our previous discussion we know that the parent-to-children message routing in E_{t1} has to be done in two bus cycles and that the same message routing task can be performed using a single bus cycle in E_{t2}. Communication efficiency in E_{t2} can be further improved by using interleaved message pipelining because during parent-to-children message transfer only every other node is sending a message. Thus each parent can send two messages to each child in one bus cycle.

4. TWO-DIMENSIONAL ARRAY PROCESSORS WITH PIPELINED BUSES

Linear optical buses have the disadvantage that message transfer may incur $O(N)$ time delay in an N-processor system. To reduce this delay to $O(\sqrt{N})$, we consider two-dimensional APPBs. In a two-dimensional APPB, each node is coupled to four buses as shown in Fig. 7a, where the two horizontal buses are used for passing messages horizontally in the same way as before, and the two vertical buses are used for passing messages vertically in a similar way. For convenience we diagram our two-dimensional APPB as in Fig. 7b. Each node in a two-dimensional APPB of size $N = m \times n$ will be given two identifications, one being a pair of numbers (x, y), $0 \leqslant x < m$, $0 \leqslant y < n$, indicating the row–column position of the node in the two-dimensional APPB, and the other being the row-major index, $i = xn + y$, $0 \leqslant i < N$, of the node. Corresponding to the bus cycle defined for the linear case, in the two-dimensional APPB we define $n\tau$ 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 simply say a *bus cycle* instead of a *row bus cycle*.

4.1. *Message Routing in Two-Dimensional APPB*

A unique issue that arises in the 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 $x_1 \neq x_2$ and $y_1 \neq y_2$. Then the message may first be sent 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 then from (x_1, y_2) to (x_2, y_2) in the second bus cycle (a column bus cycle). That is, the message has to be buffered at node (x_1, y_2) at the end of the first bus cycle and then relayed to its destination in the second bus cycle. For the purpose of relaying the message, we define a control function *relay* for node (x_1, y_2) as

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

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) send a message to node (x_2, y_2). 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 two bus cycles.

Broadcast. The system executes a $BROADCAST[(x, y)]$ instruction, which states that node (x, y) broadcasts the same message to all other nodes (x_j, y_j). In a row bus cycle, (x, y) broadcasts the message to nodes (x, y_j), $y_j \neq y$. 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 two bus cycles.

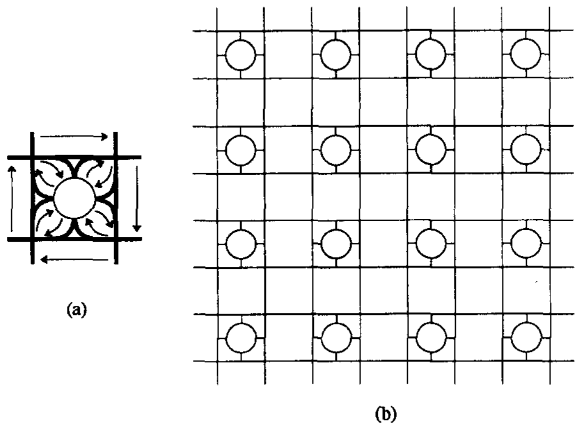


FIG. 7. Two-dimensional APPB. (a) A processor coupled to four waveguides in the two-dimensional APPB. (b) A schematic drawing of the two-dimensional APPB.

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). This task can be accomplished using two linear semigroup operations, 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 n relays at each node, we can use a three-phase routing approach [24, 32] 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 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. We note that for arbitrary permutations this approach implies the use of a centralized controller which would compute the message destinations for the preprocessing step. This calculation requires the construction of a bipartite graph and its partitioning into complete matchings, which would dominate the time complexity for the total task of computing and implementing an arbitrary permutation. In applications where a permutation can be precomputed, this time cost can be amortized over many subsequent applications of the permutation.

4.2. *Embedding Binary Trees in Two-Dimensional APPB*

As mentioned previously, arbitrary message routing and permutations in two-dimensional APPB may require n relaying buffers in each node in the worst case. In this subsection we present an embedding for a binary tree network in which only one relay buffer is needed to route messages. An embedding of an L-level complete binary tree into a two-dimensional APPB with $n = 2^k$ columns may be obtained by (i) mapping levels $0, \ldots, k - 1$ of the tree to row 0 of the two-dimensional APPB and (ii) mapping level l, $k \leq l < L$, of the tree to the 2^{l-k} rows, $2^{l-k}, 2^{l-k} + 1, \ldots, 2^{l-k+1} - 1$, of the APPB such that the two children of the same parent are mapped into two adjacent rows in the same column as the parent. Specifically we define our embedding of a binary tree network into the two-dimensional APPB by a mapping $F(i) = (F_x(i), F_y(i))$, which maps each node i, $1 \leq i < 2^L$, in the tree to a node $(F_x(i), F_y(i))$ in the two-dimensional APPB. Let i be a node at level l, $0 \leq l < L$, in the binary tree. The mapping is defined by

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

and

$$F_y(i) = \begin{cases} i, & 1 \leqslant i < 2^k, \\ \left\lfloor \dfrac{i \bmod 2^l}{2^{l-k}} \right\rfloor, & 2^k \leqslant i < 2^L. \end{cases}$$

As an example the embedding for the 4-level binary tree in Fig. 8a is shown in Fig. 8b. Let us call this embedding E_{t3}. E_{t3} has the following properties: (i) Parent nodes i, $1 \leqslant i < 2^{k-1}$, and their children are in row 0; (ii) parent nodes i, $2^{k-1} \leqslant i < 2^k$, which are in row 0, have their children in row 1; and (iii) parent nodes i, $2^k \leqslant i < 2^{L-1}$, and their children are in the same column. Properties (i) and (ii) are obvious. Here we prove only (iii). Since in the binary tree each parent node i has two children $2i + \delta$, $\delta = 0, 1$, to prove (iii) we need only show that $F_y(i) = F_y(2i + \delta)$ for $2^k \leqslant i < 2^{L-1}$. For that, let i be a parent node at level l, where $k \leqslant l < L - 1$ and $i = p2^l + q$ for some integers p and q such that $0 \leqslant q < 2^l$. Then

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

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

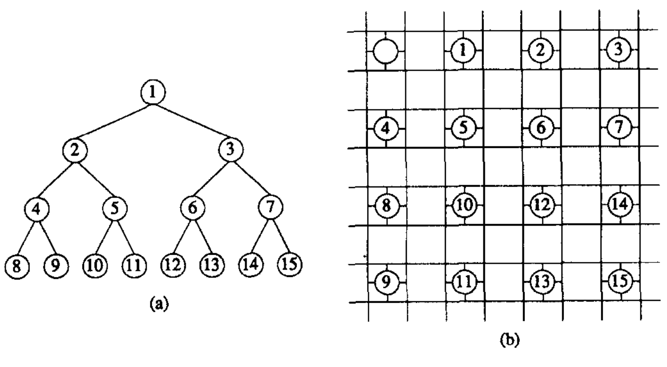


FIG. 8. (a) A 4-level binary tree. (b) Its embedding, E_{t3}, in the two-dimensional APPB.

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

$$F_x(i) = 2^{l-k} + i \bmod 2^{l-k} = 2^{l-k} + q;$$
$$F_x(2i + \delta) = 2^{l+1-k} + (2i + \delta) \bmod 2^{l+1-k}$$
$$= 2^{l+1-k} + (p2^{l+1-k} + 2q + \delta) \bmod 2^{l+1-k}$$
$$= 2^{l+1-k} + 2q + \delta;\ wait_{c,\delta}[(x, y)] = F_x(i) - F_x(2i + \delta)$$
$$= (2^{l-k} + q) - (2^{l+1-k} + 2q + \delta) = -(x + \delta).$$

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

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

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

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 even though node 0 is not a node in the tree, it helps relay 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 $relay_{p,\delta}$ is easily obtained from $relay_{p,\delta}[(0, y)] = -wait_{c,\delta}[(0, Y)]$. And $wait_{p,\delta}$ is determined as in the linear case.

4.3. *Network Embeddings Requiring No Relays*

Embedding E_{t3} still requires one message relay for communication between two neighboring nodes in binary trees. To further improve the communication efficiency, in this subsection we show how to obtain embeddings of binary trees as well as hypercubes such that no such message relay is needed. Two approaches may be used to eliminate message relays by intermediate nodes: a hardware approach and a "software" approach. In the hardware approach, optical switches are used at the intersections of row and column buses to switch an optical signal, say, from a row bus to a column bus, without requiring relay by an intermediate processor [15]. In this paper we consider the "software" approach, which relies on designing embeddings such that all neighboring processors in a network are mapped into the same row or column in the two-dimensional APPB. Thus, no message relay is needed and no *relay* function is required. This improves the communication efficiency significantly. However, it has the disadvantage that nodes in the APPB may not be fully utilized.

A basic measure that is usually used to evaluate the quality of an embedding of a source graph $G_1 = \{V_1, U_1\}$ with a set of nodes V_1 and a set of edges U_1 into a mesh architecture with a set of nodes V_2 is the *expansion cost,* which is defined as the ratio of the number of nodes in the target mesh to the number of nodes in the embedded graph. Another measure useful for such evaluation is the *dilation cost.* Specifically, the dilation of an edge $u \in U_1$, which is mapped to a path Q in the target mesh, is $|Q| - 1$, where $|Q|$ is the number of nodes on Q. However, the mesh model corresponding to that of APPBs is different from those studied previously [1, 13, 34] because the efficiency of the communication between any two nodes in the same row or column in an APPB does not depend on the distance between these two nodes. Therefore the criterion that is to be satisfied by an embedding is different from previously studied criteria. Specifically, it is desirable to obtain an embedding in which any two neighboring nodes in the source graph are mapped into either the same row or the same column in the two-dimensional APPB, thus allowing them to communicate with each other using a single bus cycle. An embedding which satisfies this requirement will be said to satisfy the *alignment condition.* Note that E_{t3} obtained in the previous subsection does not satisfy the alignment condition and thus requires message relays. That embedding, however, does have an optimal expansion cost of $2^L/(2^L - 1)$. In contrast, the binary tree embedding presented in the following satisfies the alignment condition, but its expansion cost is not optimal. This demonstrates a trade-off between the expansion cost and the dilation cost for network embeddings in the two-dimensional APPB.

Consider Fig. 9a and assume that we already have an embedding of an s-level binary tree with $N_s = 2^s - 1$ nodes into a two-dimensional APPB of size $a_s \times b_s$. The embedding is assumed to satisfy the alignment condition. That is, all the neighboring nodes in the s-level tree are mapped into the same row or column in the two-dimensional APPB. Using this level s embedding (starting level) as building blocks, the embedding for an $(s + 2)$-level tree is obtained as shown in Fig. 9b. Clearly in this embedding the neighboring nodes are again on the same row or column. A still larger tree is obtained by repeating this modular building procedure until the desired size is achieved. Let us call this embedding E_{t4}. Assuming that in E_{t4} the embedding of an L-level tree, $L = s + 2\sigma$, $\sigma = 0, 1, \ldots$, occupies an area, in number of nodes, equal to A_L in the two-dimensional APPB, we may inductively prove that

$$A_L = 2^{L-s}[A_s + (1 - 2^{-(L-s)/2})b_s].$$

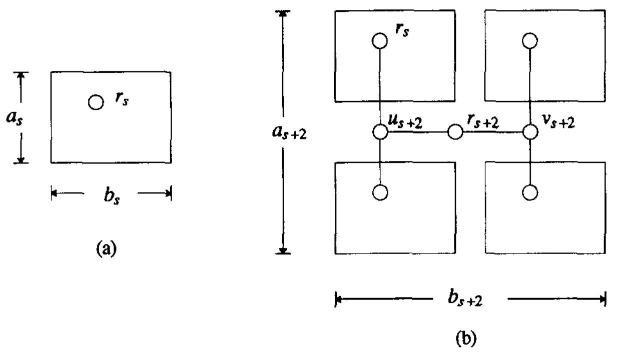


FIG. 9. Modular embedding of binary trees, E_{t4}, in the two-dimensional APPB. (a) A building block in which an s-level binary tree is embedded. (b) Embedding of an $(s + 2)$-level binary tree.

With this result, the expansion cost for the embedding of an L-level tree is

$$C_L = \frac{A_L}{N_L} = \frac{2^{L-s}[A_s + (1 - 2^{-(L-s)/2})b_s]}{2^L - 1}$$
$$= \frac{2^{L-s}[A_s + (1 - 2^{-(L-s)/2})b_s]}{2^{L-s}(N_s + 1) - 1}.$$

It can be checked that C_L is monotonically increasing with L. However, for large L, the value of C_L asymptotically equals

$$C_{L,\max} = \frac{A_s + b_s}{N_s + 1}.$$

Note that, if $N_s \gg 1$ and $A_s \gg a_s$, the value of C_L simplifies to $C_L \approx A_s/N_s = C_s \geq 1$. That is, the expansion cost for the entire embedding is determined by the expansion cost of the building block. Thus low expansion costs may be obtained if the starting building block satisfies $N_s \gg 1$, $A_s \gg b_s$, and $C_s \rightarrow 1$. Some examples of building blocks are shown in Fig. 10 with their corresponding expansion cost $C_{L,\max}$. Note that in this modular embedding scheme, as the embedding goes one level higher, the number of levels of the tree increases by 2. Thus if s is even (odd) then L is even (odd). Therefore according to whether the desired level L of the tree is even or odd, the starting level s must be chosen properly.

To determine the control functions for E_{t4}, let r_l be the root at an embedding level l, $l = s + 2, s + 4, \ldots, L$, and (x_l, y_l) be the coordinate, i.e., the row–column position, of r_l in the two-dimensional APPB. Then from Fig. 9b, the coordinate of r_l is

$$(x_l, y_l) = (a_{l-2}, b_{l-2} - 1),$$

where a_l and b_l can be found to be equal to $2^{(l-s)/2}(a_s + 1) - 1$ and $2^{(l-s)/2}b_s$, respectively. Thus,

$$(x_l, y_l) = (2^{(l-s-2)/2}(a_s + 1) - 1, 2^{(l-s-2)/2}b_s - 1).$$

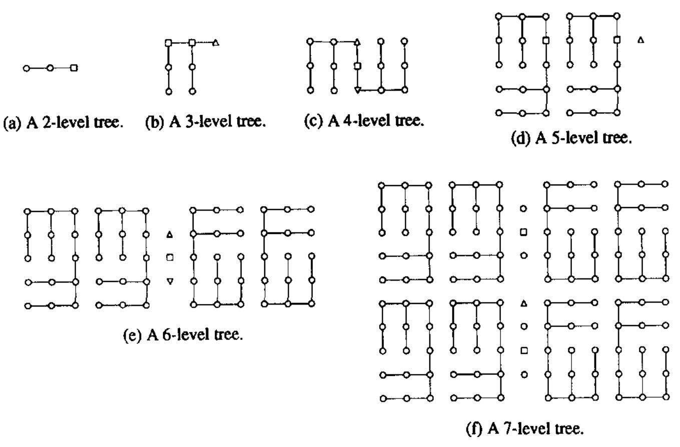


FIG. 10. Example building blocks for the modular embedding of binary trees, E_{t4}, and their corresponding expansion costs $C_{L,\max}$: (a) 1.5, (b) 1.5, (c) 1.25, (d) 1.31, (e) 1.22, (f) 1.12.

Now the control functions can be determined as follows. First, within the building block determine the *wait* functions according to the specific building block in use. Let (x_s, y_s) be the coordinate of r_s, the root in the building block. We then need only determine the *wait* functions for the new nodes which appear as we go to a higher-level embedding. For example, in Fig. 9b when we go from level s to $s + 2$, the new nodes are r_{s+2}, u_{s+2}, and v_{s+2}. By letting $wait_{c,u_l}(r_l)$ be the *wait* function for node r_l to receive a message from child node u_l, we have

$$wait_{c,u_l}(r_l) = y_l - y_{l-2},$$

$$wait_{c,v_l}(r_l) = -(y_l - y_{l-2} + 1),$$

$$wait_{c,r_{l-2}}(u_l) = wait_{c,r_{l-2}}(v_l) = \pm(x_l - x_{l-2}),$$

where the coordinate (x_l, y_l) is as determined previously. These are the *wait* functions for the new parents to receive messages from their children. The *wait* functions for the children to receive messages from these new parents are obtained by recalling that $wait_p = -wait_c$.

Next we show that the binary hypercube of 2^{2k} nodes can also be embedded in a two-dimensional APPB of size $2^k \times 2^k$ such that the alignment condition is satisfied. As in the case of binary trees, the embedding is again modular with the basic module being the binary 2-cube shown in Fig. 11a. A 3-cube embedding is obtained by putting together two such 2-cubes side-by-side as shown in Fig. 11b, and a 4-cube embedding is obtained by putting together two 3-cubes one above the other as shown in Fig. 11c and so on. Note that the nodes in Fig. 11c correspond to the cube nodes of Fig. 5a. In this way the embedding, denoted E_{c2}, of the binary hypercube of the desired size is obtained modularly.

It is observed that in embedding E_{c2}, each row and column is itself a binary k-cube. For example, if we take the column number y as the node id for the nodes in any row x, then row x is a binary k-cube consisting of nodes y, $0 \leqslant y < 2^k$. Let us call each row or column a *subcube*. Then we have 2^{k+1} such subcubes. For each subcube, if we use the column id y (or the row id x) to identify its nodes, all the control functions *step, send,* and *wait* are exactly the same as those derived for E_{c1} in the linear APPB. Thus the total communication time for emulating the hypercube can be minimized through overlapped pipelining as presented in the previous section. It can be seen that all the neighboring nodes in the hypercube are mapped to either the same row or the same column in the two-dimensional APPB. Therefore E_{c2} satisfies the alignment condition and thus requires no message relay for communications between neighboring nodes in the hypercube. Finally, since the number of nodes used in the two-dimensional APPB is equal to that of the hypercube, we achieve a minimal expansion cost of unity.

5. BANDWIDTH ANALYSIS

In this section, we evaluate the merit of the pipelined communication structure by comparing it with linear arrays which utilize nearest-neighbor and exclusive access bus interconnections. We evaluate the different models irrespective of the technology used to implement them. In other words, we assume that the transmission rate and the propagation delay are the same for both optical and electronic communication links.

Consider the linear array of n processors with nearest-neighbor connections as shown in Fig. 1b and assume that the physical separation between each pair of neighboring processors is D. Such an array may 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 the 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. We note that for the cases of interleaved and overlapped pipelining discussed in Section 3.3, at most two messages might be processed in this time. The bandwidth of the nearest-neighbor connected 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 $\rho = T_p/T_D$.

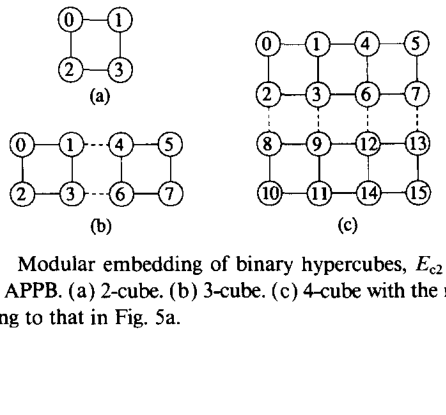


FIG. 11. Modular embedding of binary hypercubes, E_{c2}, in the two-dimensional APPB. (a) 2-cube. (b) 3-cube. (c) 4-cube with the node labeling corresponding to that in Fig. 5a.

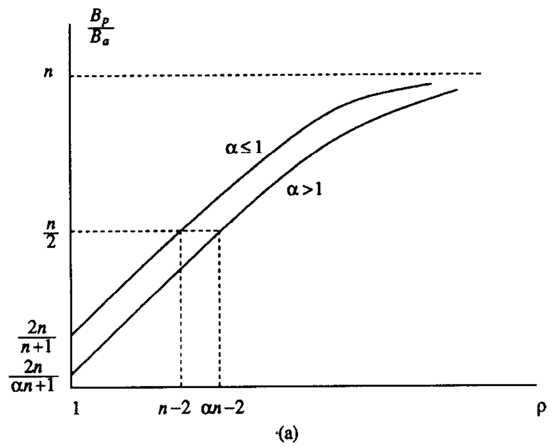
For the pipelined linear APPB, the optical distance, D_0, between two consecutive processors should be larger than the message length bwc_g (see Eq. (1) in Section 2). In other words, if $D \geqslant bwc_g$, we set $D_0 = D$; otherwise D_0 should be made equal to bwc_g (for example, by coiling an optical fiber) so that each processor can inject a message into the bus without collision. Thus, the signal propagation time, T_{D_0}, between two consecutive processors 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

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

and thus,

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

In Fig. 12a a parametric plot showing the relation between B_p/B_a and ρ is given in terms of n for $\alpha \leqslant 1$ and $\alpha > 1$. The curve for $\alpha \leqslant 1$ corresponds to the case where the message length is less than or equal to the physical separation between processors, while the curve for $\alpha > 1$ reflects the case where message length is longer than the physical separation between processors, and thus the optical path has been extended to accommodate the entire message. By taking the limit of Eq. (3) as $\rho \rightarrow \infty$, it is clear that, for fixed α and large ρ, the ratio B_p/B_a approaches n. Also, when $\rho = 1$ and $\alpha \leqslant 1$, we obtain $B_p/B_a \approx 2$. In Fig. 12b we plot B_p/B_a versus ρ for a fixed-size array with $n = 64$ and for several values of α. These plots show that the pipelined bus is more effective for larger values of ρ and smaller values of α.



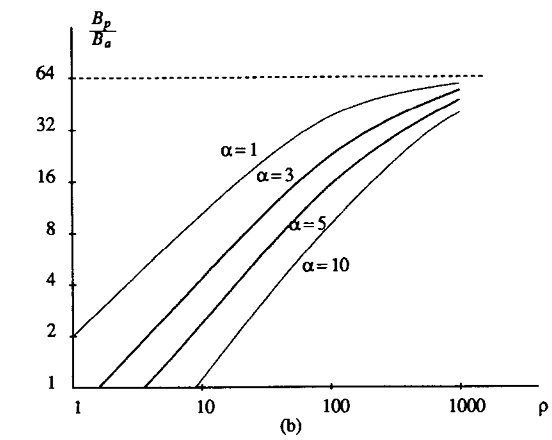


FIG. 12. The ratio, B_p/B_a, of the bandwidth of a pipelined bus to that of a linear array with nearest-neighbor connections as a function of ρ, α, and n. (a) A parametric curve. (b) For a fixed-size system with $n = 64$.

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, and therefore T_D, is relatively 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 1 (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/s, 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 buses are used to reduce b.

Next we compare the bandwidth of a pipelined bus with that of an exclusive access bus. Given that the bandwidth of an exclusive access bus 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 1, the pipelined bus can accommodate n messages in the same cycle time as the exclusive access bus. For larger α, 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.

The above analysis is independent of the media used for communication. If optical pipelined buses are to be compared with electronic buses, 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. Thus, message pipelining using electro-optical technology offers a potential for substantially enhancing bandwidth utilization. Further, pipelining techniques will be of increasing effectiveness because this technology offers the capability of generating very short pulses [12, 33], thus reducing w and decreasing α.

6. CONCLUDING REMARKS

We have presented efficient communication architectures which exploit the optical signal's properties of unidirectional propagation and predictable path delays in order to pipeline messages on optical buses. As shown in Section 5, 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 [36]. Thus pipelined buses 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.

We proposed efficient approaches to fundamental message routings including one-to-one, broadcast, semigroup communications, and permutations for the APPB architectures. Such efficient accomplishment of these commonly used message routing patterns can significantly improve the efficiency of many parallel algorithms. We presented here efficient embeddings of the binary trees and hypercube networks. Embeddings for other well-known interconnection networks, including pyramids, shuffle-exchange networks, X-binary-trees [9], and X-quad-trees, have also been obtained [14, 16]. Such efficient embeddings of these well-known communication structures allow all algorithms designed for these structures to be efficiently executed on the APPB architectures. They also allow an APPB to be logically reconfigured as an architecture which is more suitable for a given computation task.

We have not considered in this paper several issues that are relevant to the implementation of the proposed architectures. Such issues include the synchronization of the processors 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 regard to the reliability, scale, and device technology which is appropriate for computing applications. Some of these issues have been presented in [7, 25, 31].

In our experimental work [6, 8, 21] we are investigating the practical limits to these technological concerns. We have shown that three factors, threshold power margin, synchronization error, and coupling ratio, determine the system scale. On the basis of current and near-term technology, our experiments show that synchronization error does not contribute significantly to the bounds of system size. Rather, power distribution effects dominate. Preliminary investigations show that by using off-the-shelf optical components we can currently build linear buses operating at 300 MHz and containing about 100 processors. Using more sophisticated electro-optics (gallium arsenide, custom couplers, and dual level bus structures) we believe that 10-GHz buses of over 400 processors are feasible. Further, we believe that near-term technologies such as fiber amplifiers as well as alternate bus structures will alleviate the power distribution problem.

REFERENCES

1. Bailey, D., and Cuny, J. An efficient embedding of large trees in processor grids. *Proc. 1986 International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, 1986, pp. 819–822.
2. Batcher, K. E. Design of a massively parallel processor. *IEEE Trans. Comput.* **C-29,** 9 (1980), 836–840.
3. Bokhari, S. H. Finding maximum on an array processor with a global bus. *IEEE Trans. Comput.* **C-32,** 2 (1984), 133–139.
4. Chen, Y. C., Chen, W. T., Chen, G. H., and Sheu, J. P. Designing efficient parallel algorithms on mesh-connected computers with multiple broadcasting. *IEEE Trans. Parallel Distrib. Systems* **1,** 2 (1990), 241–245.
5. Chiarulli, D. M., Melhem, R. G., and Levitan, S. P. Using coincident optical pulses for parallel memory addressing. *IEEE Comput.,* (Dec. 1987), 48–57.
6. Chiarulli, D. M., Levitan, S. P., and Melhem, R. G. Self routing interconnection structures using coincident pulse techniques. *SPIE Proc. International Symposium on Advances in Interconnects and Packaging,* Boston, MA, 1990, Vol. 1390.
7. Chiarulli, D. M., Levitan, S. P., and Melhem, R. G. Optical bus control for distributed multiprocessors. *J. Parallel Distrib. Comput.* **10** (1990), 45–54.
8. Chiarulli, D. M., Levitan, S. P., and Melhem, R. G. Demonstration of an all optical addressing circuit. *Proc. OSA Topical Meeting on Optical Comput.,* Salt Lake City, UT, 1991, pp. 235–238.
9. Despain, A. M., and Patterson, D. A. X-tree: A tree structured multiprocessor computer architecture. *Proc. 5th International Symposium on Computer Architecture,* 1978, pp. 144–151.
10. Duff, M. J. B., Watson, D. M., Fountain, T. J., and Shaw, G. K. A cellular logic array for image processing. *Pattern Recognition* **5** (1973), 229–237.
11. Duff, M. J. B., and Fountain, T. J. *Cellular Logic Image Processing.* Academic Press, New York, 1986.
12. Fujimoto, J., Weiner, A., and Ippen, E. Generation and measurement of optical pulses as short as 16 fs. *Appl. Phys. Lett.* **44** (1984), 832–834.
13. Gordon, D., Koren, I., and Silberman, G. Embedding tree structures in VLSI hexagonal arrays. *IEEE Trans. Comput.* **C-33,** 1 (1984), 104–107.
14. Guo, Z. Array processors with pipelined busses and their implication in optically and electronically interconnected multiprocessor architectures. Ph.D. thesis, Department of Electrical Engineering, University of Pittsburgh, 1991.
15. Guo, Z., Melhem, R. G., Hall, R. W., Chiarulli, D. M., and Levitan, S. P. Array processors with pipelined optical busses. *Proc. 3rd Symposium on Frontiers of Massively Parallel Computation,* 1990, pp. 333–342.
16. Guo, Z., and Melhem, R. G. Embedding pyramids in array processors with pipelined busses. *Proc. International Conference on Application Specific Array Processors,* 1990, pp. 665–676.
17. Hunt, D. J. The ICL DAP and its application to image processing. In Duff, M. J. B., and Levialdi, S. (Eds.). *Languages and Architectures for Image Processing.* Academic Press, San Diego, CA, 1981.
18. Jrad, A. M., and Hall, R. W. The OFC enhanced mesh architecture: A performance study. *Proc. 1987 Workshop on Computer Architecture for Pattern Analysis and Machine Intelligence,* 1987, pp. 184–191.

19. Jrad, A. M., and Hall, R. W. Orthogonal fast channels: An enhanced mesh architecture. *Proc. 1987 International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, 1987, pp. 828-831.
20. Kawasaki, B. S., Hill, K. O., and Lamont, R. G. Biconical-taper single-mode fiber coupler. *Opt. Lett.* **6,** 7 (1981), 327-328.
21. Levitan, S. P., Chiarulli, D. M., and Melhem, R. G. Coincident pulse techniques for multiprocessor interconnection structures. *Appl. Opt.* **29,** 14 (1990), 2024-2033.
22. Melhem, R. G., Chiarulli, D. M., and Levitan, S. P. Space multiplexing of waveguides in optically interconnected multiprocessor systems. *Comput. J.* **32,** 4 (1989), 362-369.
23. Miller, R., and Stout, Q. F. Mesh computer algorithms for computational geometry. *IEEE Trans. Comput.* **C-38,** 3 (1989), 321-340.
24. Misra, M., and Prasanna-Kumar, V. K. Efficient VLSI implementation of iterative solutions to sparse linear systems. Tech. Rep. IRIS 246, University of Southern California, 1988.
25. Nassehi, M., Tobagi, F., and Marhic, M. Fiber optic configurations for local area networks. *IEEE J. Selected Areas Commun.* **SAC-3,** 6 (1985), 941-949.
26. Nassimi, D., and Sahni, S. Data broadcasting in SIMD computers. *IEEE Trans. Comput.* **C-30,** 5 (1981), 101-107.
27. Nath, D., Maheshwari, S. N., and Bhatt, P. C. P. Efficient VLSI networks for parallel processing on orthogonal trees. *IEEE Trans. Comput.* **C-32,** 6 (1983), 569-581.
28. Prasanna-Kumar, V. K., and Eshaghian, M. M. Parallel geometric algorithms for digitized pictures on mesh of trees. *Proc. 1986 International Conference on Parallel Processing.* IEEE Computer Society, Silver Spring, MD, 1986, pp. 270-273.
29. Prasanna-Kumar, V. K., and Raghavendra, C. S. Array processor with multiple broadcasting. *J. Parallel Distrib. Comput.* **4** (1987), 173-190.
30. Prasanna-Kumar, V. K., and Reisis, D. Image computations on meshes with multiple broadcast. *IEEE Trans. Pattern Anal. Mach. Intell.* **PAMI-11,** 11 (1989), 1194-1202.
31. Prucnal, P., Blumenthal, D., and Perrier, P. Self routing photonic switching demonstration with optical control. *Opt. Engrg.* **26,** 5 (1987), 473-477.
32. Raghavendra, C. S., and Prasanna-Kumar, V. K. Permutations on Illiac-IV type networks. *IEEE Trans. Comput.* **C-37,** 7 (1986), 662-669.
33. Shank, C. The role of ultrafast optical pulses in high speed electronics. In Morou, G., Bloom, D., and Lee, C. (Eds.). *Picosecond Electronics and Opto-Electronics.* Springer-Verlag, New York, 1985.
34. Singh, A. Near optimal embedding of binary tree architecture in VLSI. *Proc. 8th Symposium on Distributed Computing Systems,* 1988, pp. 86-93.
35. Stout, Q. F. Mesh connected computers with broadcasting. *IEEE Trans. Comput.* **C-32,** 9 (1983), 826-830.
36. Tanenbaum, A. S. *Computer Networks.* Prentice-Hall, Englewood Cliffs, NJ, 1981.
37. Thompson, C. D., and Kung, H. T. Sorting on a mesh-connected parallel computer. *Commun. ACM* **20,** 4 (1977), 263-271.
38. Tobagi, F., Borgonovo, F., and Fratta, L. Expressnet: A high-performance integrated-services local area network. *IEEE J. Selected Areas Commun.* **SAC-1,** 5 (1983), 898-912.
39. Ullman, J. D. *Computational Aspects of VLSI.* Computer Science Press, Rockville, MD, 1984.
40. Whalen, M. S., and Wood, T. H. Effectively nonreciprocal evanescent-wave optical-fibre directional coupler. *Electron. Lett.* **21,** 5 (1985), 175-176.

Received April 6, 1990; accepted January 11, 1991

ZICHENG GUO is finishing his Ph.D. in the Department of Electrical Engineering at the University of Pittsburgh. His current research interests include parallel computer architectures and algorithms, optical communications in multiprocessor networks, and image computation and pattern recognition.

RAMI G. MELHEM is an associate professor of computer science at the University of Pittsburgh. He received a B.E. in electrical engineering from Cairo University, Egypt, in 1976, an M.S. in mathematics/computer science from the University of Pittsburgh in 1981, and a Ph.D. in computer science from the University of Pittsburgh in December 1983. He has been an assistant professor of computer science at Purdue University from 1984 to 1986 and at the University of Pittsburgh from 1986 to 1989. His research interests include optical computing, parallel systems, fault-tolerant systems, and the application of large computational arrays to scientific problems.

RICHARD W. HALL received the B.S.E. degree in electrical engineering from The Evening College of the Johns Hopkins University in 1969 as part of the Westinghouse-Johns Hopkins Awards Program and the M.S. and Ph.D. degrees in electrical engineering from Northwestern University in 1971 and 1975, respectively. He joined the Department of Electrical Engineering at the University of Pittsburgh in 1975 and is currently an associate professor in that department. His current research interests are in the study of parallel algorithms and architectures for visual information processing.

DONALD M. CHIARULLI is an assistant professor of computer science at the University of Pittsburgh. He received a B.S. degree in physics from Louisiana State University in 1976, an M.S. degree in computer science from Virginia Polytechnic Institute in 1979, and a Ph.D. in computer science from Louisiana State University in 1986. From 1979 to 1983, he was President of Datanet Services Inc., a consulting and software development firm. While at Louisiana State he was responsible for the design and construction of The Factoring Machine, a reconfigurable VLIW machine for factoring large numbers. Dr. Chiarulli's current research interests include hybrid optical/electronic computer architecture, optical interconnects, VLSI design, and parallel computation. He is a member of the IEEE Computer Society, ACM, SPIE, and the Optical Society of America.

STEVEN P. LEVITAN is the Wellington C. Carl Assistant Professor of Electrical Engineering at the University of Pittsburgh. He received the B.S. degree from Case Western Reserve University (1972) and his M.S. (1979) and Ph.D. (1984) degrees, both in computer science, from the University of Massachusetts, Amherst. He worked for Xylogic Systems, designing hardware for computerized text processing systems, and for Digital Equipment Corp. on the Silicon Synthesis project. He was an assistant professor from 1984 to 1986 in the Electrical and Computer Engineering Department at the University of Massachusetts. In 1987 he joined the electrical engineering faculty at the University of Pittsburgh. Dr. Levitan's research interests include computer-aided design for VLSI, parallel computer architecture, parallel algorithm design, and VLSI design. He is a member of the IEEE Computer Society, ACM, SPIE, and OSA.