Locality-based control algorithms for reconfigurable optical interconnection networks

Donald M. Chiarulli, Steven P. Levitan, Rami G. Melhem, and Chunming Qiao

Hybrid optoelectronic computing structures are required for providing the information processing capabilities for the next generation of computing and communications systems. Reconfigurable optoelectronic interconnection networks are networks constructed of optical waveguides in which messages are switched or routed by means of optoelectronic devices. For these networks, the dichotomy between the bandwidth of the optical channels that carry messages and the performance of the electronic controllers and decoders that determine the routing and destination of those messages is a significant bottleneck. We introduce a class of routing algorithms for reconfigurable networks that is designed to bridge this gap in optical versus electronic performance. The algorithms are based on a new control paradigm that exploits the locality in multiprocessor communication streams to reduce the control latency inherent in reconfigurable interconnection structures. In addition, we show that this problem maps directly to the problem of page replacement in a virtual-memory hierarchy. Thus our solution is well suited to networks for multiprocessor applications.

1. Introduction

Research on electronic reconfigurable interconnection networks spans almost two decades of computer engineering literature. In the optical domain, reconfigurable systems that use spatial light modulators as the switching fabric in free-space designs have been reported in Refs. 4–7. Guided-wave reconfigurable systems have been studied with space-division, time-division, and wavelength-division switching systems, as well as various hybrids. Our previous work has focused on optical busses for multiprocessor applications.

Busses, however, are only the simplest of shared-resource interconnection networks. In busses, arbitration of access to the resource and the addressing of messages can be done independently by the use of simple hardware control. For more complex shared-resource networks, addressing and control issues cannot be solved independently. In these networks, addressing and control are implemented jointly as message routing. Unlike addressing, routing considers not just the destination of a message, but also its path and the network resources needed for that path. Routing must both arbitrate and allocate the resources necessary to provide that path. This can be implemented by a variety of techniques. However, any routing technique represents a trade-off between explicit addressing and global allocation. In most systems, explicit addressing dominates this trade-off. In other words, message sources makes requests to the control hardware that arbitrates resources to create the message path.

Here we present an alternative routing technique in which the control system dynamically allocates resources based on global knowledge of the message traffic. Thus, rather than message sources presenting addresses to the network, the network establishes a set of paths and presents these to the sources and the receivers. As most interconnection networks cannot provide all possible paths simultaneously, the issue in these routing strategies is to allocate the network resources such that they consistently meet the needs of the current message traffic. By use of the locality that is inherent in the message traffic, a single control operation may allocate a network resource for use by a sequence of messages between a common source and destination. Thus the latency of making the control decision is amortized over the

D. M. Chiarulli, S. P. Levitan, and R. G. Melhem are with the University of Pittsburgh, Pittsburgh, Pennsylvania 15260; D. M. Chiarulli and R. G. Melhem are with the Department of Computer Science and Engineering; and C. Qiao is with the Department of Electrical Engineering, State University of New York at Buffalo, Buffalo, New York 14260.

Received 9 June 1993; revised manuscript received 30 September 1993.

0003-6935/94/081528-10$06.00/0.
© 1994 Optical Society of America.
entire sequence. We call this technique state-sequence routing.

In Section 2 we present an abstract model of an interconnection network that uses state-sequence routing. In Section 3 we show that such a network can be controlled in a manner analogous to virtual-memory allocation computer memory management systems. Next, to demonstrate that the technique is independent of the switching domain, we show in Section 4 how state-sequence routing is applicable to a variety of reconfigurable optical interconnection networks including a time-division-switched (TDS) linear bus, a space-division-switched (SDS) multistage network, and a wavelength-division-switched (WDS) star network. Section 5 presents two classes of control algorithms for these networks. The first class uses static allocation techniques and is based on a fixed computational structure derived from compilation analysis. The second class uses dynamic techniques that can respond to varying message traffic. Section 6 looks at the impact of locality of the message traffic on system performance by the use of an analytic model, and Section 7 presents results from an application running in a simulated network environment. Section 8 presents our conclusions and directions for future research.

2. Network Model

Our model of a communication network is shown in Fig. 1. Let INET in this figure be an $n \times n$ interconnection network that connects a set $I$ of $n$ input ports to a set $O$ of $n$ output ports, and let $p_{io} = (i, o) \in I \times O$ denote the path between a specific input port $i \in I$ and a specific output port $o \in O$. We assume that the INET may establish any of the possible $N = n^2$ paths but that it cannot establish them simultaneously. Thus an INET may be a bus, a multistage interconnection network (MIN), a WDS star, or any other type of interconnection network. Within an INET, we define a mapping $M$ to be a set of paths that can be established at the same time without conflicts in the INET. For each mapping $M$ there is a corresponding state $S$ that represents the configuration of the network (i.e., switch settings, detector tunings) corresponding to that mapping.

As the establishment of two paths at the same time in an INET may cause conflicts, not every set of paths is a mapping. We refer to the establishment of all the paths in a mapping as the realization of the mapping. Given a set of paths $P \subseteq I \times O$, it may not be possible to realize all paths in $P$ at the same time without conflicts. However, $P$ can be partitioned into several mappings: $P = M_1 \cup M_2 \cup \ldots \cup M_t$. Each mapping $M_i$, $i = 1, 2, \ldots, t$, may be realized in sequence. Given that each mapping has a corresponding state, the set of paths $P$ may be implemented as an ordered sequence of states $(S_1, S_2, \ldots, S_t)$, where $t$ is the length of the sequence.

Returning now to Fig. 1, we find that the state generator block is responsible for generating the current state sequence. In each step, the state of the network is output to the switches and is also communicated to each of the transmitting nodes. Thus a transmitting node waits for the network state corresponding to a mapping that contains the required path. When such a state is detected, the node transmits its message. If no such mapping exists within the current state sequence, the control algorithm must transform the state sequence to include a mapping that supports the requested path. The control algorithm that determines these transformations runs in the state transformer.

It is this transformation mechanism that distinguishes our model from a multiplexed network in which all possible connections are provided in turn. We avoid the excessive latency of full multiplexing by using a sequence length that is substantially shorter than that required for providing full interconnection. The current sequence of configurations is sufficient only to support all the current traffic in an interconnection network. Because control is based on the transformation of this sequence, we can use the principle of locality to decouple the performance of the transformation algorithm from the bandwidth of the message traffic. By exploiting locality, we can reduce message routing to a problem of providing a repeated sequence of configurations to the network.

Control becomes a problem of transforming that sequence to track the changes in the locality of message traffic.

3. Routing as Selection in a Virtual-Connection Space

The principle of locality is a well-established paradigm that characterizes the way that processors communicate with memory units in computer systems. This principle is the basis for virtual-memory systems in which a large memory space is provided by a much smaller amount of physical memory. In our network model, the same principles can be applied to permit an interconnection network of lower bandwidth to provide the bandwidth requirements of a fully interconnected system.
We call the set of all possible paths a virtual-connection space and note that the set of paths provided by state-sequence routing is a changing subset of those paths. We can then draw a direct analogy between the routing problem for reconfigurable interconnection networks that use state-sequence routing and the paging problem in virtual-memory systems. The correspondence between the two is summarized in Table 1. As shown in the table, an individual path between a source and a destination in a fully connected network can be viewed as the analog of an individual memory location in a complete virtual-address space. Physical memory is shared and reused in units of a page, that is, a block of memory locations. The corresponding unit of sharing for the interconnection network is a mapping, a single network configuration. Physical memory is a collection of pages that supports a subset of the virtual-address space corresponding to the current memory traffic. Similarly, in a network, the configuration sequence supports all paths in the current message traffic. In computer systems, the functionality of the full virtual-memory address space is supported by the management of pages by moving them in and out of physical memory. In the network, the functionality of a fully interconnected network is supported by adding or deleting mappings from the state sequence.

The analogy also extends to addressing. In memories, a k-bit virtual address defines an address space of $2^k = N$ addressable locations. Paging divides this address space into $p$ pages, each of size $s$ locations such that $N = s \times p$. In communication networks, a full interconnection network for $n$ sources and $n$ destinations provides $n \times n = N$ interconnection paths, i.e., $N$ unique source–destination pairs. Although a typical switched interconnection network may establish any of these $N$ paths, it is capable only of connecting a subset of $m$ paths simultaneously in a particular configuration. For enumerating all possible paths, a sequence of $c$ different configurations is required such that $N = m \times c$. Just as at any given time, a subset of the $p$ pages resides in physical memory to satisfy the current set of memory requests, a communication network needs only to sequence through a subset of configurations to satisfy the current requests for paths.

Virtual-memory systems work because the principle of locality implies that if a working set of pages is made available to a group of programs, that set of pages will change slowly over time. Changes in the working set of pages are costly but need be made only occasionally relative to the total volume of memory traffic. The analogous principle in a network that uses state-sequence routing allows us to reduce control operations to an occasional transformation with frequency that is substantially less than the message rate.

In Section 5 we discuss the details of both static and dynamic (locality-based) control algorithms. First we present some example networks to show that the model presented above is generally applicable to reconfigurable optical networks.

4. Example Networks

In this section we demonstrate the application of state-sequence routing to networks that are based on three different implementation technologies: a TDS linear bus, a SDS multistage network, and a WDS star network. The topologies of these networks are shown in Fig. 2.

Figure 2(a) is a linear bus interconnection of four transmitters and four receivers. This bus implements time-division switching as follows. Synchronized by a global clock, a time-division multiplexed pipeline of four messages is originated simultaneously, one message from each transmitter, $O_1 \cdots O_4$. In order for the pipeline to be formed without collisions, the length of fiber between each transmitter is large enough that an entire message can be stored in its length. In this version of the bus we have opted for a completely passive interconnect. Therefore we must assume that each receiver is capable of distinguishing between individual messages within the pipeline. In other words, each receiver can be programmed to accept the $i$th message, including null messages, in the pipeline and to ignore all others. If this capability is not available in the receivers, we can easily support the same functionality by replacing the passive receiver splitters with active switches. Busses of this type have been studied in detail by the authors in Ref. 17.

For this network, a path between sender $O_i$ and receiver $I_j$ is formed when $I_j$ is programmed to select the $i$th message from a pipeline. A mapping is formed by the set of paths supported in a pipeline. The current programming of all receivers is the state. The state generator in this example has two responsibilities: first, to transmit the programming for the current state to each receiver, and second, to inform each transmitter which receiver has been programmed to receive its message. If more than one mapping is required, the state generator would sequence through a set of mappings, informing the transmitters and receivers of each state in the sequence.

| Table 1. Summary of Virtual Memory versus Switched Interconnection Analogy |
|-----------------------------|-----------------------------|
| **Entity**                  | **In Virtual Memory**       | **In Communications Network** |
| Addressing space            | Virtual address             | All $n \times n$ connections in a fully connected network |
| Degree of sharing           | Physical memory size        | The current configuration sequence length |
| Unit of sharing             | Page                       | A single network configuration |
| Addressable unit            | Memory location             | A particular path from a source to a destination |

1530 APPLIED OPTICS / Vol. 33, No. 8 / 10 March 1994
Figure 2(b) is a MIN. In this network, a unique path can be established between any input port and any output port. Along each path, there are \( \log n \) switches, one at each stage. In order to establish a path, each switch along the path must be set properly to either a straight or a cross state. Other switches in the network can be in either state without affecting the path. Therefore in any mapping there will be exactly \( n \) paths.

Given a mapping, it is straightforward to find the state \( S \) that realizes the mapping. Once the set of mappings is determined, the state generator outputs the sequence of states, which establishes the mapping for all current paths. As described above, each transmitter monitors the output of the state generator and waits for a mapping that contains the requested path. Unlike the linear bus case, in which only a limited set of shared resources were available, supporting a new
path in the MIN may not require replacing an existing path or extending the sequence. Instead it may be possible to transform one or more mappings in the current sequence to form a mapping in which the new request is realizable.

Finally, Fig. 2(c) is a wavelength-division-multiplexed star network. A variety of interconnection protocols exist for these nets. For this example, we assume the most general case in which any of I transmitters may communicate with any of O receivers on any of W wavelengths. In other words, transmitters and receivers are independently tuned. Because, in general, \( |W| \leq \min(|I|, |O|) \), we further assume that any transmitter or receiver may be effectively turned off such that it will neither transmit nor receive. Under these assumptions, a path consists of a triple \((i, o, w) \in I \times O \times W\), and a mapping consists of any realizable set of paths such that each path communicates on a different wavelength. The state \( S \) that corresponds to each mapping is therefore a collection of transmitter and receiver tunings. In this case the state generator outputs a sequence of mappings by assigning a wavelength for each path in the mapping. Each source node monitors the state for an assignment of a wavelength to its transmitter. When such a wavelength is assigned, the message is sent by using the current mapping. Receivers must also monitor the state for wavelength assignments. In protocols that require the identification of the sender, they must also monitor the state to discover the transmitting node to which that wavelength was assigned. This is the general case; a number of simplifications are also possible. For example, either the transmitters (or receivers) could be assigned fixed wavelengths, and only the receivers (or transmitters) could be tuned by the state generator.

In this section we have demonstrated that the basic concept of stage-sequence routing is consistent across the domains of time, space, and wavelength switching. Specifically, paths, mappings, and states can be easily identified independently of the nature of the INET. Built on this basic idea is the concept that mappings can be sequenced and the current sequence matched to actual traffic. We now turn to the discussion of these control algorithms.

5. Control Algorithms

In this section we present two classes of control algorithms, static and dynamic. The static algorithms represent the simple case in which the communication pattern of the application is known before the execution of the program and does not change during its execution. Only a single state sequence is necessary, and no transformations of the sequence are required. In terms of the virtual-memory analogy, this would correspond to a single program being loaded into main memory and running to completion without changing its memory configuration.

However, when the communication graph of the application cannot be determined in advance, dynamic algorithms are required. For these applications, the sequence must be transformed on demand during execution. Dynamic algorithms can be further classified as using either fixed- or variable-length sequences and as using either preemptive or nonpreemptive transformations. We present examples of both static and dynamic control algorithms. Our virtual-memory analogy in a multiprocessing environment is most closely modeled by the dynamic control algorithms that use fixed-length sequences and preemptive transformations.

A. Static Allocation Algorithms

Given a specific application (a parallel algorithm), the communication requirements of that application can often be estimated and modeled by a bipartite graph, which we call a connection request (CR) graph. A directed edge from a source processor \( o \) to a destination processor \( i \) in a CR indicates a request for the establishment of a path \( p_{io} \) in the MIN. We use the same notation for a path to denote an edge and use the terms edge and path interchangeably. The maximum number of edges going out from a node or coming into a node in a CR graph is called the degree of the graph.

Given a CR graph, we call a mapping sequence \((M_1, M_2, \ldots, M_t)\) a minimal connection (MC) sequence for the CR graph if every edge \((i, o) \in CR\) is established in some mapping and if no two mappings in the sequence can be merged together. We call a sequence for a CR graph optimal if it is a MC sequence for the graph and has the shortest length among all other MC sequences for the same graph. Note that if \( t \) is equal to the degree of the graph, then the MC configuration is optimal.

When the communication structure of an application is regular, finding a sequence for its CR graph is often called embedding. Note that the ability to embed regular communication structures efficiently is important as there are many existing applications designed for them. The sequence length \( t \) is a measure of the efficiency of the embedding. Table 2 shows results that have been proven\(^1\)\(^3\)\(^9\) for several structures embedded in a TDS bus and a SDS MIN.

For nonregular CR graphs, a MC sequence can always be obtained by selecting a subset of mappings from a complete connection (CC) sequence. Assuming that each mapping is a set of \( n \) paths, any CC sequence will consist of \( n \) mappings. The problem with this selection algorithm is that it restricts itself to the particular set of \( n \) mappings in a chosen CC sequence. However, there may be as many as \( n! \) mappings possible in the network.

| Table 2. Sequence Lengths for Various Embeddings of \( n \) Nodes |
|-----------------|--------------|-----------------|
| Structure       | TDS Bus      | SDS MIN         |
| Binary tree     | 3            | 4               |
| Binary hypercube| \( \log n \)  | \( \log n \)    |
| Fully connected | \( n \)       | \( n \)         |
| Bidirectional ring | 2           | 2               |
| Mesh            | 4            | 4               |
| Cubo-connected cycle | 3         |                 |
Alternatively, given a CR graph with a set of edges $E$, a MC configuration can be obtained by composing mappings based on the set $E$, which may be different from any mapping in a chosen CC configuration. More specifically, the composition algorithm composes each mapping in a greedy fashion.

Under low or medium load conditions, the composition algorithm improves over the selection algorithm, as expected. However, when the load is extremely high, the sequence length of a MC configuration could exceed $n$. That is, when this algorithm is used, sequences under high load conditions may be worse than when a CC configuration is used. To improve the selection algorithm under low or medium load conditions and yet bound the sequence length of any MC configuration by $n$ under high load conditions, we may use the selection algorithm first to determine a set of up to $n$ mappings needed, then examine each mapping to see if it can be deleted from the configuration by migrating paths established in it to other mappings in the configuration. Simulation results show that this last algorithm outperforms both the selection and the composition algorithms on a SDS MIN.

### B. Dynamic Allocation Algorithms

Static control algorithms work only if all paths are required to be established from the beginning of the execution of an application. Unfortunately, most applications generate requests that cannot be determined until run time. Also, even if a CR graph that contains all edges needed during the execution can be constructed a priori, it may be inefficient to perform static control based on such a graph. Some paths are used for only a certain duration of time and are wasted for the remaining time during execution. It may be possible to reduce the total execution time of such an application by dynamically modifying the mapping sequence at run time. In this section we present two algorithms that support dynamic allocation.

#### Nonpreemptive Variable-Sequence-Length Control

This algorithm uses an explicit request–release protocol, assumes that the sequence length as well as its contents may vary dynamically, and does not use preemption. That is, once a path is established in the sequence, that path will not be removed from the sequence until it is released by the requesting processor. As with static allocation, we assume an INET that connects $n$ transmitters and $n$ receivers, in which each mapping may consist of up to $n$ paths. To support the protocol we further assume that each processor can independently communicate request–release messages to the state transformer.

The request–release algorithms for state transformations under these assumptions are as follows:

For each path request not contained in the current sequence:

**Case 1: Incremental Inclusion.** An existing mapping that is capable of realizing both its active paths and the new path is transformed such that it includes the new path. (This is the most desirable option.)

**Case 2: Migration and Inclusion.** An existing path can migrate from one mapping to another in order to create a mapping into which the new path can be incrementally included. (This is part of a more general problem of sequence compression that is also associated with the handling of release messages.)

**Case 3: Sequence Extension.** The sequence length is extended by 1 to provide space for a mapping that contains the requested path.

The performance of insertion for case 1 is limited by the need to search the existing mapping sequence. However, if a path were already in the sequence, the worst-case performance for transmitting the mapping to the INET is also proportional to the sequence length. For WDS, TDS, and nonblocking SDS networks, the selection of an insertion point need only take into account the destination of active paths in a test for realizability. Even for networks with internal contention, realizability can still be implemented by a lookup operation. In either case, we assume that the test for realizability can work in time equal to a sequence step. Insertion and migration, case 2, which results in more efficient sequences, and sequence extension, case 3, are the most efficient to implement. However, both require the implementation of compression, which is described in the deletion cases.

For each request to release a path:

The mapping that currently establishes the path may be deleted if

**Case 1: Sequence Contraction.** The mapping contains no other active paths and is therefore deleted from the sequence.

**Case 2: Sequence Compression.** All other active paths in the mapping containing the deleted path can migrate to other mappings, thus permitting the mapping to be deleted from the sequence.

**Otherwise:** The sequence remains unchanged.

If the sequence length is allowed to grow during insertions, simple sequence contraction, case 1, operations will not be sufficient to maintain efficiency. Paths that could potentially be serviced in a single mapping will naturally fragment into several. Compaction is necessary for maintaining an efficient sequence length. This requires multiple scans of the current sequence and a global analysis of its contents. However, compaction can be done simultaneously with other insertion and deletion cases, and its performance can be decoupled from the performance of the network. For example, when case 3, for insertion, is used, a mapping can be immediately added to the end of the sequence to service a pending request. Simultaneously to that service, the off-line compaction algorithm may search for a mapping transformation.
to include the path in other mappings for subsequent requests.

Finally, one reasonable trade-off is to buffer runtime requests and to execute a static reconfiguration algorithm periodically. At each selected instance, a snapshot of the CR graph would be constructed based on all current paths that need to be established. This approach may also be useful as an off-line compression algorithm that constructs a complete optimal sequence for all current paths and uses it to replace the current sequence periodically.

Preemptive Fixed-Sequence-Length Control

In this section we present a transformation algorithm suitable to the most general case of network control, for example, heterogeneous program environments in which the CR graphs cannot be predicted in advance or environments in which an explicit request-release protocol cannot be supported. In such systems, the interconnection network is invisible to both systems and application software. This system is the direct analog to the virtual-memory page replacement model discussion above.

We use a system model similar to the dynamic allocation algorithm described above. Once again, the INET connects $n$ transmitters and $n$ receivers with each mapping capable of establishing up to $n$ paths. Unlike the previous model, there is no explicit request–release protocol, and the sequence length $t$ is fixed at some value $t \leq n$. Once a path is established in the sequence, it remains in place until forcibly removed by replacement with a new path. As in the above models, each transmitting node holds a message until the state of the INET represents a mapping that contains the required path. In the absence of explicit request–release signals, it is the responsibility of the state transformer to guarantee that the state generator will output such a state in a finite amount of time.

The state-transformation algorithm must satisfy two constraints. On the one hand, new paths must be implemented in minimum time. On the other hand, with preemption, optimal performance requires that the state preempted by any transformation have the smallest impact on the current traffic. In order to satisfy both of these constraints, the control algorithm is implemented as a set-associative replacement system. In set-associative replacement, the set of all sources from which messages may originate is partitioned equally by the length of the sequence. Messages from a particular source preempt only in the sequence slot that corresponds to that source’s partition. This mechanism satisfies the first constraint of fast response to path requests. To minimize the impact of preemptions, the partitions themselves are periodically reorganized to balance the activity across the entire sequence.

It is clear from the above discussion of dynamic algorithms that two types of message-handling operations can be identified: those that require a transformation in the state sequence and those that do not.

These operations have significantly different latency. The locality in the message traffic determines the proportion of these two kinds of operation and thus determines the efficiency of the control system. In Section 6 we analytically show how this has a substantial impact on overall system performance.

6. Impact on System Performance

The relationship of control latency to the overall throughput of high-speed networks can be seen from the following argument. Each message traverses the network with latency composed of several components: the message transmission time, the physical delay of the network, and the message control time. Only the message transmission time (i.e., the bit rate of the message) is directly tied to the bandwidth of the network. The physical delay of the network is limited by the speed of light in the waveguide and the physical length of the path. We cannot reduce this time. The control time is a function of the speed of the control hardware and the complexity of the control task. For high-speed optoelectronic networks, the control time dominates the latency of the system. This is true even for systems in which each of the steps in pipelined. Therefore we must reduce the control time to utilize the high bandwidth of these networks fully.

More formally, for any path through a reconfigurable communications network, the end-to-end latency $t_{path}$ can be characterized as

$$t_{path} = t_{control} + t_{launch} + t_{fly}.$$ (1)

In other words, latency is the sum of the time necessary for establishing a path, launching a message into the channel, and propagating that message for the length of the channel. For circuit-switched systems, each of these terms are single values. For packet-switched or multihop systems, the end-to-end latency is simply the sum over each hop. Thus, if all three of the control, launch, and propagation phases of message transmission operate sequentially, the maximum message throughput $T_{path}$ is given by

$$T_{path} = 1/t_{path} = 1/(t_{control} + t_{launch} + t_{fly}).$$ (2)

In high-bandwidth networks these operations are commonly overlapped in a pipeline fashion, and thus the maximum throughput is

$$T_{path} = 1/\max(t_{control}, t_{launch}, t_{fly}).$$ (3)

In other words, the inverse of the longest of the pipeline stages. If the transmission media are capable of holding multiple messages (i.e., message pipelining in a fiber), then we need consider only the maximum of launch time and control time:

$$T_{path} = 1/\max(t_{control}, t_{launch}).$$ (4)

Consider an arbitrary reconfigurable network that uses a centralized controller to arbitrate access to $m$ channels. From Eq. (4) we know that the through-
put of any end-to-end connection is inversely proportional to the maximum of the control and the launch times. As the overall network is capable of launching \( m \) packets simultaneously, the network throughput is

\[
T_{\text{network}} = \frac{m}{\max\{t_{\text{control}}, t_{\text{launch}}\}}. \tag{5}
\]

A network is considered to be saturated when the rate of packets entering the network is equal to \( T_{\text{network}} \).

Equations (1)-(5) establish the upper limit on throughput for all networks, whether optical or electronic, by using distributed or centralized control. For both electronic and optical networks, control time is by far the dominating term. Clearly, as \( m/t_{\text{launch}} \) is directly dependent on the channel bandwidth of the network, Eq. (5) tells us that increasing the network bandwidth alone will not lead to an increase in throughput. Only by reducing the control time can we effectively increase the capacity of a fixed-size network. Increasing the size of the network may not be a solution either because most control algorithms increase in complexity with network size.

To see how control time and locality interact, we use the following simplified model of message traffic. We divide the message traffic into two classes of messages, those messages that follow the same path as their predecessors and those that require the establishment of new paths. The former are local messages and the latter are nonlocal messages. Assuming that \( P_{\text{local}} \) is the probability that a connection request does not require the establishment of a new path (local messages), then \( 1 - P_{\text{local}} \) is the probability that a connection request does require the establishment of a new path (nonlocal messages). Hence the effective rate by which the controller can service connection requests is

\[
t_{\text{control}} = t_{\text{local}} P_{\text{local}} + t_{\text{nonlocal}} (1 - P_{\text{local}}), \tag{6}
\]

where \( t_{\text{local}} \) and \( t_{\text{nonlocal}} \) represent the control time for local and nonlocal messages, respectively. For example, in dynamic state-sequence-based control algorithms, nonlocal control involves a sequence transformation time, whereas local control time is, on the average, half of the sequence period. Hence the maximum bandwidth of the controller and thus the maximum throughput of the network are

\[
T_{\text{network}} = \frac{1}{\max\{t_{\text{launch}}, \lfloor t_{\text{local}} P_{\text{local}} + t_{\text{nonlocal}} (1 - P_{\text{local}}) \rfloor\}}. \tag{7}
\]

In other words, higher throughputs are obtained in message traffic in which \( P_{\text{local}} \) is larger, that is, message traffic with a high degree of locality. By using \( t_{\text{launch}} = 32/\text{bandwidth} \) to simulate 32-bit words, Fig. 3 shows the relationship of bandwidth and locality to throughput for a network with \( t_{\text{local}} = 10 \) ns, \( t_{\text{nonlocal}} = 100 \) ns, 1 MHz \( \leq \text{bandwidth} \leq 1000 \) MHz, and \( 0 \leq P_{\text{local}} \leq 0.8 \). In this example, the local transfer wait time of 10 ns represents one half of the sequence period, and the nonlocal transfer time of 100 ns is a conservative estimate of the time for transformation.

In reality, this probability, \( P_{\text{local}} \), depends on three factors: the actual message locality, the relative time between successive local messages, and (for preemptive algorithms) the probability that a sequence slot will be preempted. The first factor is a property of the application, as is the second, which is additionally dependent on process granularity and the relative bandwidth of the processors and the network. The third factor is a function of the network traffic, the network topology, the sequence length, and the global sequence-transformation algorithm.

A full analytical model that considers all these factors is extremely complex. Therefore in Section 7 we present simulation data in which we establish the relationship between \( P_{\text{local}} \) and the sequence length for a specific network and parallel application.

7. Simulation Results

In this section we examine the factors that determine \( P_{\text{local}} \) in more detail and present the results of two multiprocessor simulation studies that show the relationship between \( P_{\text{local}} \) and the sequence length. Our multiprocessor model is an eight-processor system, with memory organized into shared and local partitions. Each processor may access variables in a local memory as well as variables in any of eight shared-memory partitions. All accesses to shared memory must traverse the interconnection network. In the first case, the network is an \( 8 \times 8 \) crossbar, and in the second case it is an \( 8 \times 8 \) multistage banyan network.

In Section 6 we identified the three factors on which \( P_{\text{local}} \) depends. The first, actual program locality, is obviously a function of the application and the specific design of the program code. For our purposes, actual program locality is the probability that two successive shared-memory accesses from one process access will refer to memory locations in a single shared partition. The second and third factors, the relative time between such accesses and the probability of path preemption, are related to the amount of contention.

In the first simulation, by using a crossbar network, we do not need to make a distinction between contention for network resources and contention for a single memory partition. For a system so con-
contention will occur when two processors attempt to access the same shared-memory partition in the same cycle. For state-sequence routing, such a mapping of input to output ports cannot happen. However, contention occurs when two successive accesses from one processor are interposed by an access to the same partition from another processor in the same position of the state sequence. For the second simulation, in which the MIN network is used, the identical contention problems arise with regard to common destination ports. However, we must additionally consider contention in cases in which shared-network resources are also interposed, causing blocking conditions on paths that do not share the same destination.

Figure 4 shows the results of the simulations for an eight-processor parallel difference equation computation resulting from the discretization of the Laplace equation on a unit square domain with Dirichlet boundary conditions. Each processor was allocated an equal partition from a uniform mesh on which the discretization was done. The x axis in this plot is the sequence length. The y axis is \(P_{\text{local}}\) calculated as the percentage of the total transactions through the network for which no sequence transformation was necessary. The dotted curve shows the results for the crossbar network simulation. The solid curve shows the results for the MIN simulation.

Of particular interest are the cases for a sequence length of 1 and a sequence length of 8. At a sequence length of 1, the result is the actual locality in the memory access pattern. Looking at the crossbar case, we see that approximately 30% of the shared-memory accesses are successive accesses between one processor and the same memory partition. In the MIN case, approximately 24% of the memory accesses are successive uses of nonblocking paths. At the other extreme, in which the sequence length is equal to 8, we have, in both cases, the equivalent of a completely time-multiplexed system. Thus each processor has a slot reserved for access to each memory partition, and no contention occurs. The small difference from 100% occurs because the sequences are initially random, and a few transformations are required at startup.

Two significant conclusions can be drawn from these results. First, the similar shape of both curves suggests that the advantages of state-sequence routing are achievable for networks both with and without internal resource contention. As our ultimate goal is to trade bandwidth for network complexity, this is an important result.

Second, the shape of the curves suggests that we can make this trade-off with a limited latency penalty. The large increases in \(P_{\text{local}}\) over short sequence lengths suggests that the overhead inherent in state-sequence routing, namely, the sequence wait time, can be kept to a minimum.

8. Conclusions and Future Research

We have introduced a paradigm for routing and control in high-bandwidth optoelectronic networks that alleviates the bottleneck of control throughput on the capacity of these networks. By exploiting the locality in message traffic, we have shown that control bandwidth need only operate at performance levels necessary for supporting the changes in the traffic rather than the rate of individual messages.

In our discussion of latency we have dealt with only end-to-end transmission times and have assumed that, in a comparable network, messages would be transmitted without wait times introduced by contention within the network. This is obviously not the case in systems with moderate to heavy traffic loads. The issue that remains for further study is to compare these wait times for specific network and traffic patterns with the corresponding wait times at the transmitter that are experienced in state-sequence routing as the source waits for its turn to transmit.

The research of C. Qiao is supported in part by U.S. Air Force Office of Scientific Research contract F49620-93-1-0023.

References

8. J. Goodman, “Switching in an optical interconnect environ-


