Space Multiplexing of Waveguides in Optically Interconnected Multiprocessor Systems*

R. G. MELHEM‡, D. M. CHIARULLI‡ AND S. P. LEVITAN§

‡ *Department of Computer Science, the University of Pittsburgh, Pittsburgh, PA 15260, USA*

§ *Department of Electrical Engineering, the University of Pittsburgh, Pittsburgh, PA 15260, USA*

Optical waveguides allow for enhanced bandwidth, loosened loading constraints and large physical distribution of computing resources. Moreover, optics enjoys a unique property that is not shared with electronics, namely the unidirectional propagation of signals. It is this property that is exploited in this paper to increase the effective bandwidth of optical buses. Specifically, a space-multiplexing technique for pipelining messages on optical buses is introduced and analysed. It is shown that pipelined buses support arbitrary routeing permutations in synchronous systems with only linear hardware complexity. Further, a bus arbitration protocol which extends the technique to asynchronous systems is presented. The pipelining of control and data signals represents a significant departure from the conventional exclusive access discipline which characterises bus-interconnected multiprocessors. By relaxing the exclusive access requirement, space multiplexing can support the design of large-scale, distributed, tightly coupled multiprocessor systems.

Received October 1988, revised December 1988

1. INTRODUCTION

There are three fundamental constraints which bound bus interconnections in electronic systems: limited bandwidth, capacitive loading, and cross-talk caused by mutual inductance. Optical systems provide both an opportunity and a challenge to redesign our traditional multiprocessor solutions free of these limitations. Although direct technology substitution may alleviate, to some extent, the communications bottleneck in computer systems, there are obvious limitations to such substitution. For example, any interface between electronics and optics lowers the speed at that interface to the speed of electronics. Even though optical pulses as short as a few femto-seconds may be generated and detected,[4,12] such short pulses may not be used to transmit data on an optical bus, since no existing electronic circuit at the transmitting or the receiving end of the bus can match that speed. In other words, the speed of electronics puts bounds on the transmission speed of optical buses.

Another limitation concerns the end-to-end propagation time of long buses. Due to the absence of self-inductance or -capacitance, long optical buses may be constructed without the need for signal repeaters. Several designs of optical communication networks have already been constructed taking advantage of this property.[1,9,11] However, the control overhead associated with the networking environment is relatively high and cannot support efficient distributed multiprocessing. The major requirement for multiprocessing is that many short messages are transmitted with low overhead. Specifically, the assumption of exclusive access to the bus resource limits throughput to a function of the end-to-end transmission time for the signals on the bus, irrespective of the length of the messages. End-to-end transmission times for optical signals are not inherently shorter than for electronics.

A unique property of optics provides an alternative to exclusive bus 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. Thus, for directional propagation in pipelines, electronic systems introduce directional amplifiers between adjacent bus sources. These amplifiers cause unpredictable delays, which make any large-scale distributed implementation impractical. On the other hand, optical channels are inherently directional and have predictable delay per unit length. This allows a pipeline of signals to be created by the synchronised directional coupling of each signal at specified locations along the channel. This property, which has been used to parallelise access to shared memory[2] and to minimise the control overhead in networking environments,[16] is applied in this paper to optimise the use of optical buses in multiprocessor systems.

We present a technique for space multiplexing of optical channels in distributed tightly coupled multiprocessors. In the next section we introduce the concept of pipelined optical buses as applied to synchronous processor arrays. We show that by pipelining messages on a single bus we may realise arbitrary routeing permutations. We provide a detailed example of embedding tree interconnections in linear arrays of processors. In Section 3 we continue in the more general framework of asynchronous multiprocessors. The primary concern in such an environment is the realisation of a distributed arbitration mechanism for pipelined buses. We introduce such a mechanism in Section 3.2, and study its performance in Section 3.3. In Section 3.4 we further analyse the technique in the context of large shared-memory multiprocessor systems.

2. SPACE MULTIPLEXING OF WAVEGUIDES IN SYNCHRONOUS PROCESSOR ARRAYS

Multistage networks have been studied extensively[13] as a means of making processor–processor and processor–memory interconnections. In the context of synchronous processor arrays, however, an $n \times n$ multistage network with $\log n$ stages may not realise arbitrary permutations in

* This work is, in part, supported under the Air Force Office of Scientific Research contract AFOSR-88-0198.

† To whom correspondence should be addressed.

a single pass. For example, it has been shown in Ref. 17 that the Omega network[8] needs at least three passes in order to realise arbitrary permutations, and that, alternatively, a network with three log$n$ stages may be used. The hardware complexity of such multistage networks is clearly of the order of $O(n \log n)$. In this section we shall demonstrate that a single optical bus, which has $O(n)$ hardware complexity, may be used to realise arbitrary routeing permutations, as well as many-to-one and one-to-many routeings.

2.1 Pipelining messages on optical buses

Consider a linear array of n nodes connected by a single optical bus (waveguide) that is also connected to a host as shown in Fig. 1. Each node may inject optical signals into the waveguide through a directional coupler. The injected signals propagate from left to right and may be read by any subsequent node on the waveguide. As would be the case in electronics, the bus of Fig. 1 may be used as a medium for broadcasting messages from the host to the nodes. However, because of the directionality of signal propagation, the same bus may also be used to transmit messages from node 1 to node 2, from node 2 to node 3 and, in general, from node i to node $i+1$, $i = 1, \ldots, n-1$, simultaneously.

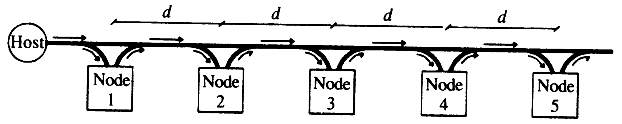

Figure 1. An optically interconnected linear array.

Assuming that each message transmitted between any two nodes consists of b bits, and that each bit is represented by a light pulse of duration w (sec), the concurrent transmission of the $n-1$ messages described above may be accomplished if the following criteria are met.

(1) All transmissions are synchronised to start simultaneously. This may be enforced by the use of a synchronisation signal that arrives at all the nodes at the beginning of each transmission cycle.

(2) The length of the optical path on the waveguide between any two consecutive nodes (d in Fig. 1) is larger than bwc_g, where c_g is the velocity of light in the waveguide. For example, if $c_g = 2 \times 10^8$ m/sec, and 10-bit messages are transmitted at 10 GHz, d should be larger than 20 cm. This minimal optical path length can be reduced by decreasing the message length via parallel transmission on multiple waveguides, by decreasing c_g via the use of waveguides with higher refractive indices, or by shortening the pulse width. Here we note that d is the optical path length between any two nodes, which is not necessarily equal to their physical separation. Also, d does not have to be the same for every pair of adjacent nodes.

Conditions 1 and 2 guarantee that the signals corresponding to two different messages do not physically overlap at any point on the waveguide (hence the term 'space multiplexing'). If the first of the b bits in each message is a start-of-message indicator that is always set to one, a receiving node may pull-off the $b-1$ bits following the start-of-message bit, and ignore any following signal up to the initiation of the next transmission. With this scheme every node may send a message to its right neighbour, simultaneously, on the same waveguide. Note, however, that the initiation of consecutive transmission should be separated by at least nd/c_g sec, where n is the number of nodes in the array. Clearly, the size of the array should be such that the value of nd/c_g is compatible with the computation speed in the nodes. For example, if $d = 10$ cm and $n = 50$ nodes, transmission may be initiated every 25 nsec.

2.2 Realisation of arbitrary permutations

Nearest-neighbour connections are not the only point-to-point communications that may be supported by the single waveguide of Fig. 1. In fact, using space multiplexing, point-to-point messages between any pairs of nodes may be transmitted simultaneously as long as their paths do not intersect in space and time. To be more specific, let $m_{i,dest(i)}$ be a message that is sent from node i to node $dest(i)$, and let $M = \{m_{i,dest(i)};\ 1 \leqslant i < n\}$ be a set of such messages for some one-to-one function $dest$. If conditions 1 and 2 above are satisfied and $dest$ is a strictly increasing function, that is $dest(i) > i$, then all the messages in M may be transmitted on the waveguide simultaneously without causing any signal overlap.

Let $S = \{i: m_{i,dest(i)} \in M\}$ and $D = \{j: m_{i,j} \in M\}$ be the sets of source and destination nodes, respectively, for messages in M. If all the nodes in S initiate transition simultaneously, then a node j in D, where $j = dest(i)$ for some i in S, may have to skip a few messages before reading the message $m_{i,j}$ intended for it. Specifically, j has to skip a number of messages equal to the number of nodes between i and j that are in S. That is:

$$skip(j) = \sum_{l=i+1}^{j-1} \phi(l) \qquad (1)$$

where

$$\phi(l) = \begin{cases} 1 & \text{if } l \in S \\ 0 & \text{otherwise} \end{cases}$$

Hence, by using a single waveguide, it is possible to send messages from any node i to any destination $dest(i) > i$. In order to support communications from i to some destination $dest(i) < i$ a second waveguide may be used as shown in Fig. 2. Clearly, the two directional waveguides in Fig. 2 may support the simultaneous transmission of any set of messages M for any permutation function $dest$. Specifically, M can be partitioned into two set M_1 and M_2 such that M_1 contains the messages with $dest(i) > i$, and M_2 contains the messages with $dest(i) < i$. Messages in M_1 are transmitted on the left-to-right waveguide and messages in M_2 are transmitted on the right-to-left waveguide. The sets of source nodes, S_1, S_2, and destination nodes, D_1, D_2 may be defined for M_1 and M_2, respectively, and at each destination node $j \in D_1 \cup D_2$, the number of messages that

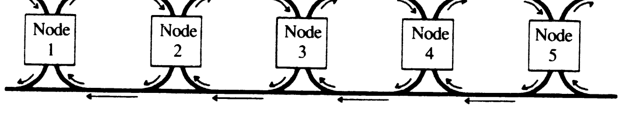

Figure 2. A dual waveguide system.

have to be skipped on a particular waveguide before reading $m_{i,dest(i)}$ may be determined by an expression similar to (1).

The dual waveguide system of Fig. 2 may be viewed as a communication network that may realise any permutation. Given a specific permutation *dest*, a single register, SKIP, may be used at each node j to store information about message reception; the sign of SKIP may indicate whether j is in D_1 or D_2, and its magnitude may indicate the number of messages to be skipped before reading the appropriate message. With this, setting and changing of the interconnection patterns may be accomplished at run-time by programming the values of the SKIP registers at the nodes. Compared to cross-bar switches or multistage interconnection networks, our system uses less hardware (linear with n), eliminates switch delays, and may be re-configured by programming registers that are local to the processors.

The same communication capabilities of the dual waveguide system may be obtained in the folded waveguide system of Fig. 3. In that system each node writes its message on track 1 of the waveguide and senses any signal on the waveguide through a photo-detector coupled to track 2. At the reception of the synchronisation signal, each node puts its message (b bits) on track 1 of the waveguide. The n messages form a train which travels on track 2, thus allowing each node to read the message that is destined to it. As in the single-waveguide system, a register SKIP may be used at each node to indicate the number of messages to be skipped before reading the appropriate message. In this case,

$$skip(j) = \sum_{l=i+1}^{n} \phi(l),$$

where $\phi(l)$ is as defined in (1). Note that the folded waveguide system uses less couplers and photodetectors than the dual-waveguide system at the expense of doubling the optical length of the bus. A new round of communication may be initiated on the folded waveguide every $2nd/c_g$ secs.

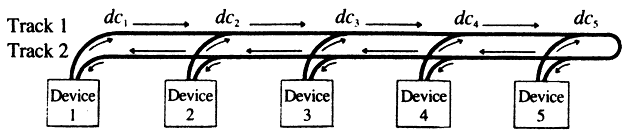

Figure 3. A folded waveguide system.

In addition to allowing arbitrary permutations, a pipelined optical bus may allow many-to-one communications if the appropriate number of SKIP registers are provided. For example, if two registers are provided in each node, each node may receive up to two messages in each bus cycle. Moreover, if the length of the optical path d between two adjacent nodes is increased to $Kbwc_g$, where, as before, bwc_g is the length of a single message, then each node may send up to K messages in each bus cycle, and thus one-to-many communications may also be allowed. This flexibility may be applied to realise several logical interconnections. We demonstrate this capability by an example.

2.3 Tree-interconnections on pipelined buses: an example

Consider a multiprocessor system which consists of $n = 2^L - 1$ processors logically connected in a complete binary tree structure. If a breadth-first numbering is used to identify the processors, the tree connection implies that a processor j should be connected to its parent, processor $\lfloor j/2 \rfloor$, and to its children, processors $2j$ and $2j+1$ (see Fig. 4(a) and Fig. 5(a) for examples). If the n processors are connected by a dual-waveguide system similar to the one shown in Fig. 2, the left-to-right waveguide may support messages from parent processors to children processors, and the right-to-left waveguide may support messages from children processors to parent processors.

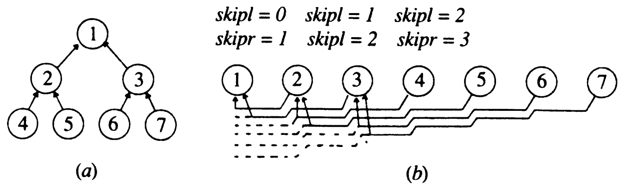

Figure 4. Simultaneous transmission from children processors to parent processors.

First, we illustrate the many-to-one communication capability by assuming that each processor is to send a message to its parent. Clearly, a processor $j, j < 2^{L-1}$, will receive one message from each of its two children, and hence two skip registers are needed at each node. Let $skipl(j)$ and $skipr(j)$ be the number of messages that a node j has to skip before reading the messages addressed to it by its left child (processor $2j$) and right child (processor $2j+1$), respectively. Each of the $2j-j-1$ processors between processor $2j$ and processor j sends a message to its parent, and hence processor $2j$ will have to skip $j-1$ messages before reading the message of its left child. That is

$$skipl(j) = j-1 \quad j = 1, \ldots, 2^{L-1}-1$$

Similarly, we find that

$$skipr(j) = j \quad j = 1, \ldots, 2^{L-1}-1$$

The values of *skipl* and *skipr* are shown in Fig. 4(b) for the case $L = 3$.

In order to illustrate the one-to-many communication capability, we assume that each processor (except the leaf processors) is to send out two messages, the first to its left child and the second to its right child. If each processor writes its two messages consecutively on the right-to-left waveguide, and the length of the optical path between any two processors is larger than $2bwc_g$, where b is the number of bits per message, the messages will not overlap on the waveguide. Let $skip(j)$ be the number of messages that a processor j has to skip before reading the message sent to it by its parent, namely processor $\lfloor j/2 \rfloor$, where $\lfloor j/2 \rfloor$ is the largest integer smaller than $j/2$.

If j is even, then there are $j/2-1$ processors between processor j and its parent, $j/2$. Denote by S_j the set containing these processors. Now, if j is not a leaf processor each processor in S_j will write two messages on the waveguide, and j will have to skip these messages before reading its message. That is, $skip(j) = 2(j/2-1) = j-2$. However, if j is a leaf processor $j-2^{L-1}$ of the processors in S_j are also leaf processors that will not write any messages on the waveguide. In this case, j will have to skip only $2(j/2-1-j+2^{L-1}) = 2^L-j-2$ messages before reading its message.

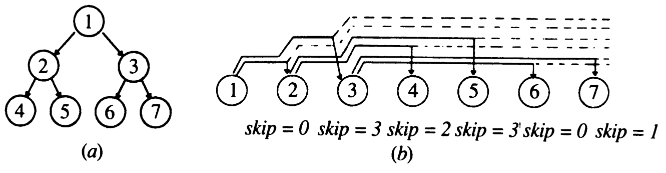

Figure 5. Simultaneous transmission from parent processors to children processors.

If j is odd, its parent is $(j-1)/2$ and the set S_j contains $(j-1)/2$ processors. Again, if j is not a leaf, each processor in S_j will write two messages on the waveguide. In addition to skipping these messages, processor j will have to skip one more message because its parent, $(j-1)/2$, writes the message destined to its left child, $j-1$, before the one destined to its right child j. Hence, in this case, $skip(j) = j$. However, if j is a leaf processor the 2^L-j-2 leaf processors in S_j will not write any message on the waveguide resulting in $skip(j) - j$. In summary, we have

$$skip(j) = \begin{cases} j-2 & \text{if } j \text{ is even and } j \text{ is not a leaf processor} \\ j & \text{if } j \text{ is odd and } j \text{ is not a leaf processor} \\ 2^L-j-2 & \text{if } j \text{ is even and } j \text{ is a leaf processor} \\ 2^L-j & \text{if } j \text{ is odd and } j \text{ is a leaf processor} \end{cases}$$

The values of *skip* are shown in Fig. 5(*b*) for a tree with three levels.

Finally, it should be noted that an analysis similar to the one presented above may be applied if a folded waveguide system is used. Also, pipelined buses may be used to realise other logical interconnection structures such as the barrel switch networks, the shuffle/exchange networks and the hypercube network.[6]

3. MESSAGE PIPELINING IN ASYNCHRONOUS MULTIPROCESSORS

In the previous section, space multiplexing was applied to computational models which assume a fixed communication pattern at each cycle. Systolic arrays and SIMD multiprocessors are clear examples of such models. However, the same dual-waveguide and folded-waveguide configurations shown in Figs 2 and 3 may also be used to multiplex messages between arbitrary sources and destinations. Clearly, this may be accomplished if each message includes the address of its destination, and if individual messages are framed by appropriate delimiters. As described earlier, each of the n devices may put its message on the bus during the same bus cycle, provided that the bits of different messages do not overlap. That is, provided that all devices start the transmission simultaneously and that the length d, of the optical path between any two devices satisfies $d \geqslant d_{\min} = wb_{\max} c_g$, where $b_{\max}$ is the number of bits in the longest message (address + data + delimiters). For a given $d_{\min}$, this condition represents an upper limit on the length of the messages that may be transmitted on the bus.

For asynchronous MIMD multiprocessor systems, it may not be possible to synchronise the simultaneous transmission of messages to the accuracy required in optical systems, especially if the transmitting devices are physically separated, as is the case in distributed multiprocessor systems. In order to resolve this problem,

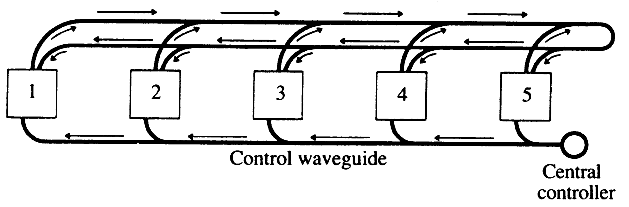

Figure 6. Synchronising transmission in physically distributed systems.

a *grant* signal may be issued by a central controller and propagated through the processor array on a separate waveguide. For example, in the folded-waveguide configuration, the grant signal may be propagated in a direction opposite to that of track 1 of the message waveguide (see Fig. 6). The arrival of the grant signal at device i initiates the transmission of the message from that device. Because the *grant* signal and the signal on track 1 of the message waveguide propagate in opposite directions, the bus cycle becomes $2nd/c_g$ and the condition on the inter-device optical path length d becomes $d \geqslant 0.5\, wb_{\max} c_g$.

So far we have described a multiplexing method in which a train of n slots, one for the message transmitted by each device, is pipelined on the waveguide. This is quite acceptable provided that the messages fit in the designated slots, and provided that a central controller assumes the charge of issuing the synchronisation signal or the grant signal. In distributed asynchronous multiprocessor systems these conditions are not generally satisfied, and different techniques must be applied to arbitrate the access to the bus. A distributed arbitration technique is described in the next section.

3.1 Distributed control of pipelined optical buses

In addition to pipelining the data signals on the bus, the unidirectional propagation of optics may be applied to pipelining the control signals on the bus. The arbitration mechanism described in this section relies on the directional propagation of signals on a control waveguide that is folded into three tracks as shown in Fig. 7. This three-track waveguide is similar to the one used in Refs. 15 and 16 to pipeline modulated data signals, except that it is used exclusively for control signals. The mechanism results in a batch priority queue protocol.[5] In brief, all devices that request the bus while the bus is idle form a batch, and requests in the batch are serviced in linear priority until all requests are satisfied. Requests that arrive while a certain batch is being serviced have to wait until all the requests in that batch are serviced and then form a new batch.

Normally, the control waveguide is at logic zero (no light). A device which wants to use the bus may assert a *request* signal on track 1 of the control waveguide only if a logic zero is read on the *ack* input which is coupled to track 3 of the control waveguide. If *ack* is high, this is considered as an indication that a batch has already been formed, and hence the device has to wait until that batch is serviced before it may assert its request. As will be explained later, this will be indicated by a high-to-low transition on *ack*.

Clearly, the end of a batch formation period is signalled at each device by a rising edge on *ack* (low-to-

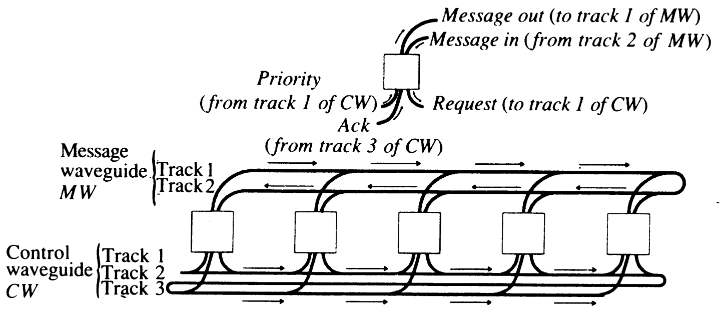
Message out (to track 1 of MW)
Message in (from track 2 of MW)
Priority (from track 1 of CW)
Request (to track 1 of CW)
Ack (from track 3 of CW)
Message waveguide MW
Track 1
Track 2
Control waveguide CW
Track 1
Track 2
Track 3

Figure 7. Distributed control of pipelined optical buses.

high transition). This edge, which travels towards the right on track 3 of the control waveguide, is the feedback of a signal that is generated on track 1 of that waveguide by some device asserting a request signal.

A device is included in the current batch as soon as it asserts *request*. However, it is not granted control of the bus until it reads both a one on *ack* (end of batch formation) and a zero on the *priority* input, which taps track 1 of the control waveguide. With this scheme, after the formation of a batch, the devices within the batch will be granted control of the bus in the order at which they are connected to the control waveguide. Hence, in Fig. 7, the leftmost device requesting the bus is granted the bus first. Here we note that the only function of track 2 of the control waveguide is to cause the feedback signal on track 3 to travel in the same direction as the request signal on track 1. As proved in Ref. 3, this ensures that the arbitration mechanism handles simultaneous requests correctly.

After a device i is granted the bus ($ack = 1$ and $priority = 0$), it sends its message on the message waveguide, and after sending the last bit of the message it relinquishes the bus by lowering the *request* line. This will cause a low-going edge to travel on track 1 of the control waveguide. The next requesting device, say j, will receive that edge at the same time when the last bit of the message sent by device i passes through the coupler dc_j. In other words, when device j is granted control of the bus upon the high-to-low transition on *priority*, it may immediately start to transmit its own message on the message waveguide. The two messages from device i and device j, say m_i and m_j, will thus be pipelined on the message waveguide. The spacing (separation) between m_i and m_j depends on the time required for processing the control signal within the controller of device j.

Since electronic delays cannot be predicted to the accuracy of optics, it is impossible to eliminate completely the separation between messages in the pipeline. However, such separation can be reduced if the *request* signal of device i is lowered before the transmission of the last bit of m_i by a time period $\tau_{e,min}$ equal to the minimum electronic delay expected in a controller. The actual separation between messages will therefore vary within the specified tolerances of a small number of gates in the control circuit. Specifically, if $\tau_e = \tau_{e,min} + \Delta_e$ is the actual time consumed by device j to process the control signal, then m_i and m_j will be separated on the waveguide by a distance of $\Delta_e c_g$. This, of course, assumes that the time to transmit m_i is larger than $\tau_{e,min}$. If this is not the case, then the separation between m_i and m_j becomes $(\tau_e - b_i w) c_g$, where b_i is the number of bits in m_i and $1/w$ is the baud rate used for transmission. In order to simplify future analysis, we assume that the separation between messages is $\tau_e c_g$, where τ_e is a delay due to few electronic gates (for a precise estimation of τ_e we refer to Ref. 3).

With the above scheme, all the messages transmitted by devices in a particular batch will be pipelined into a train that will travel on track 2 of the message waveguide and thus will be seen by every device. It is the responsibility of each device to recognise its address within each message and to read the messages that are addressed to it.

When the rightmost device in the batch finishes sending it message and lowers its *request* line, the corresponding low-going edge will travel on the control waveguide all the way to track 3 of that waveguide. The detection of this high-to-low transition on the *ack* input of a device is interpreted by that device as a signal of completion of the current batch, and thus as a permission to assert *request* if the device was waiting to request the bus.

3.2 Control overhead

In the above bus arbitration scheme there is an overhead associated with forming a batch and with signalling the termination of a batch. For a given batch we use the term 'batch initiation' to refer to the instant of time at which the first device in the batch asserts its request signal. We also use the term 'batch termination' to refer either to the instant of time at which all devices are notified that the service of the current batch has been completed, or to the initiation of the next batch, whichever is first. Note that in the case of high bus contention the bus is never idle, and thus batch termination is always due to the initiation of the next batch. Now, we may define *batch formation* overhead as the time interval between batch initiation and the instant when the highest priority device in the batch starts the transmission of its message. Similarly, we may define *batch release* overhead as the time interval between the instant at which the lowest priority device in the batch finishes the transmission of its message and batch termination.

In order to estimate the overhead associated with batch control in a high-contention environment, we assume that device r is the lowest-priority (rightmost device) in some batch and that device f is the leftmost device that is waiting for the termination of that batch to request the bus. In this case, by tracing the control signals on the control waveguide it is easy to see that the *batch release* overhead is equal to $2\tau_{1,n} - \tau_{f,r}$, where $\tau_{i,j}$ is

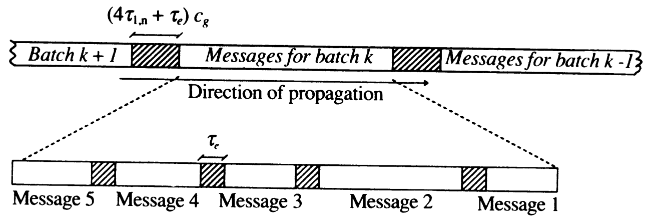

Figure 8. A snapshot of the pipelined messages on track 2 of the message waveguide.

the signal propagation delay between devices i and j, and thus $\tau_{1,n}$ is the bus end-to-end propagation delay. Given that f is the highest-priority device in the next batch, the *batch formation* overhead for that batch is equal to $2\tau_{1,n}+\tau'_e$, where $2\tau_{1,n}$ is the time for the request signal asserted by f on track 1 of the control waveguide to wrap around and reach the *ack* input of f, and τ'_e is the logic delay from the time the *ack* input of f goes high to the time f starts writing its message on the bus. This logic delay is of the same order as the logic delay τ_e discussed earlier, and it will be assumed, for simplicity, that $\tau'_e = \tau_e$.

Hence, the time interval between the instant at which r finished the transmission of its message until the instant at which f starts its transmission is $4\tau_{1,n}-\tau_{f,r}+\tau_e$. This interval creates a spatial separation on the message waveguide between the two trains of messages corresponding to the two batches. Specifically, if, as in Fig. 7, the direction of propagation on the message waveguide is identical to that of tracks 1 and 3 of the control waveguide, then, as shown in Fig. 8, the spatial separation between the two trains of messages is equal to $(4\tau_{1,n}+\tau_e)\,c_g$.

3.3 Performance analysis

An important measure of performance of any bus control protocol is its efficiency, defined as the ratio $\eta = T_d/(T_c+T_d)$, where T_d is the time spent for transmitting data on the bus and T_c is the time spent in controlling the bus. This ratio is particularly important if the bus is subject to high contention. That is, if the bus is never idle; an assumption that we will maintain throughout this section.

The optical bus protocol described in the previous section utilises the property of undirectional propagation of optical signals in two different ways. First, by pipelining control signals, the control overhead of $4\tau_{1,n}+\tau_e$ is only paid for each batch rather than for each message. Secondly, due to the pipelining of data signals, each device holds the bus only while writing a message on the bus and there is no need to wait for the signals to reach their destination after each transmission. Assuming that N_{av}, $1 \leqslant N_{av} \leqslant n$ is the average number of devices included in each batch, and that b_{av} is the average number of bits in a message, then, over a time span required to process a given number of batches, the efficiency of the pipelined bus protocol is

$$\eta_{\text{pipe}} = \frac{N_{av}\beta_{av}}{4\tau_{1,n}+\tau_e+N_{av}(\beta_{av}+\tau_e)}, \qquad (2)$$

where $\beta_{av} = b_{av}\,w$ is the average time length of a message. If the same number of messages are transmitted on an electronic or an optical bus, using a protocol which does not exploit the unidirectional propagation of signals, the upper bound on the efficiency is

$$\eta_{\text{non-pipe}} = \frac{N_{ab}\beta_{av}}{N_{av}(2\tau_{1,n}+2\tau_e+\beta_{av})}, \qquad (3)$$

where it is assumed that the minimum arbitration time for each message is $\tau_{1,n}+2\tau_e$, for some logic delay τ_e. It is also assumed that the physical location of the receiver is not known to the sender, and hence each sender should not relinquish the bus unless it is certain that the message reached the receiver. This requires, at least, an average time of $\tau_{1,n}$ per message. Note that the efficiency of non-pipelined buses does not reach the bound (3) in practice.

By comparing (2) and (3), it becomes clear that the efficiency of pipelined buses is always larger than that of non-pipelined buses as long as $N_{av} > 1$ – that is, as long as each batch contains more than one request. Moreover, pipelining the bus becomes more advantageous when the arbitration overhead is dominated by the end-to-end propagation delay – that is, when $\tau_{1,n} \gg \tau_e$. For example, if sub-nanosecond ECL electronics is used for the control logic and the bus is longer than a few metres, then $\tau_{1,n} \gg \tau_e$, and the effect of τ_e in (2) and (3) becomes very small. In this case, η_{pipe} and $\eta_{\text{non-pipe}}$ may be approximated to

$$\eta_{\text{pipe}} = \frac{N_{av}}{2\rho+N_{av}} \qquad (4)$$

and

$$\eta_{\text{non-pipe}} = \frac{1}{(2\rho+1)}, \qquad (5)$$

where $\rho = \tau_{1,n}/\beta_{av}$ is the ratio of bus length to message length. In Fig. 9(*a*), Equations (4) and (5) are plotted for a bus connecting $n = 64$ processors, assuming that $N_{av} = n/4$. The plot shows that pipelined buses may be efficiently used even for values of ρ larger than unity. For such ρ, the length of the messages is smaller than the end-to-end delay and the overhead associated with non-pipelined buses becomes prohibitive. In Fig. 9(*b*), we fix ρ at $\rho = 2$ and show the effect of N_{av} on the efficiency of pipelined buses. As expected, the efficiency of the pipelined protocol increases when the bus becomes busier and the batches become larger.

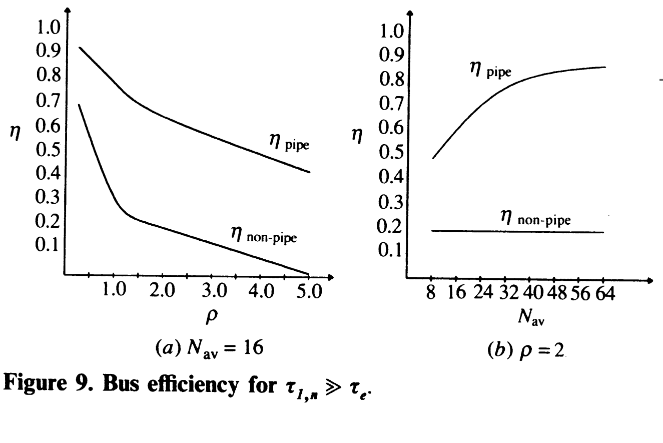

Figure 9. Bus efficiency for $\tau_{1,n} \gg \tau_e$.

Pipelining the bus reduces the control overhead, even for short buses in which the logic delay τ_e may not be neglected. In order to illustrate this, we consider Equations (2) and (3) with $n = 64$ and $N_{av} = n/2$ and we plot, in Fig. 10, the bus efficiency against the ratio $(\sigma = \beta_{av}/\tau_e)$ for different values of ρ. If the same electronic technology is used for both the controller circuit and the transmitter circuits, it is reasonable to assume that the width of the pulses used for transmission is approximately equal to one logic level delay in the controller circuit. With this assumption, $\sigma = b_{av}/g$, where b_{av} is the average number of bits in a message and g is the number of logic levels in the controller circuit (in the controller design described in Ref. 3, $g = 5$). We have chosen to use σ instead of β_{av} in Fig. 10 because σ is a measure of the length, in bits, of the messages transmitted on the bus, and is independent of the electronic technology used in the system.

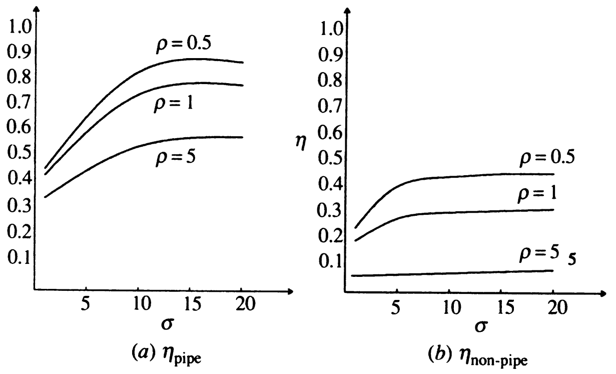

Figure 10. Bus efficiency as a function of σ.

3.4 Application to distributed common bus multiprocessors

The results of the previous section indicate that optical buses may be used for the construction of distributed multiprocessor systems with relatively large physical separations. Specifically, pipelining the signals on the bus provides high message throughput and thus large bandwidth, even for relatively short messages. In order to illustrate this point, we consider a 64-multiprocessor system with $N_{av} = n/2 = 32$, $b_{av} = 32$, $g = 5$, and we assume a 1 nsec delay per logic level. For such a system, $\sigma = 6.4$ and $\rho = 0.104L$, where L is the length of the bus in metres. By substituting these values in (3) we find that, in order to maintain a bus efficiency of more than 0.5, the physical length of the bus should be less than 3.4 m. From (2), this same efficiency may be obtained for pipelined buses of length up to 64 m. The ability of pipelined optical buses to support distributed multiprocessors increases as the number of processors increases. For example, with $N_{av} = 128$, $\sigma = 6.4$, $\rho = 0.104L$ and a 50% bus efficiency, pipelined optical buses of length up to 259 m may be used. This clearly allows for the construction of distributed multiprocessor systems across buildings in university campuses or industrial sites.

Pipelining the messages on the bus, however, does not shorten the delay required for delivering any particular message. This delay remains proportional to the length of the path travelled by that message. For shared-memory multiprocessor systems such delay is especially critical, since it represents a performance bottleneck. In the remaining part of this section we shall analyse in some detail the memory-response time in shared-memory systems interconnected by pipelined optical buses.

Consider a system of n processors, each connected to a local memory such that the n local memories form a global memory space that is addressable by any processor in the system. The processors are connected by an optical bus that is used to transmit memory requests to non-local memories. Also, memory modules receiving read requests from remote processors use the bus to send back the content of the requested memory locations. For simplicity, it will be assumed that messages are of some fixed length, say b bits. This is a reasonable assumption, since only three types of message are needed for 'read requests', 'write requests' and 'returned data', and since separate message waveguides may be used for address, data and processor identification. If, as before, the bus cycle time, τ_{cycle}, is defined as the time for servicing the bus requests in one batch plus the associated control overhead, then

$$\tau_{cycle} = 4\tau_{1,n} + \tau_e + N_{av}(\beta + \tau_e), \qquad (6)$$

where $\beta = bw$.

The memory-response time for a read operation, denoted by τ_{fetch}, is the time elapsed between the instant when a processor issues a memory read request and the instant it receives the addressed data. For a non-local memory read, the value of τ_{fetch} depends on the location of the processor relative to the addressed memory module, and on the status of the bus at the time of the request. In order to compute the worst response time, we consider a processor P, which issues a read request to a module M just after the formation of a batch. In this case, P has to wait for one complete bus cycle before it can even request control of the bus. Thus the read request will be transmitted on the bus during the following bus cycle (call this cycle 2). If P is the lowest-priority device on the second batch, M will receive the request at the end of cycle 2 and may not have time to fetch the data before the formation of the third batch. Thus the data will be sent back to P during cycle 4, and if M has a low priority the data will reach P at the end of cycle 4. In other words, τ_{fetch} may be as high as $4\tau_{cycle}$.

The above analysis assumes that memory contention within M may be resolved in a τ_{cycle} time. That is, if a given batch contains k read requests, $k \leqslant n$, for locations in M, then all the k requests will be fetched during cycle 3, stored in a queue, and sent back during cycle 4. If this is not possible some requests will not be ready during cycle 4, and the response time for these requests will be greater than $4\tau_{cycle}$. This is a memory-contention problem, which is typical for all shared-memory multiprocessor systems. In our system such contention will only delay memory access to M, but unlike multistage interconnection networks will not affect request to other memory modules. Specifically, for multistage networks, this situation, referred to as a hot spot[10], creates a saturation tree which affects all memory requests, including those that do not address M.[7]

By carefully following the path of the different control signals in the pipelined bus, it becomes clear that, even if P is the lowest-priority device and the read request to M is the last message in batch 2, there is a period of $2\tau_{1,n}$ between the instant when M receives the read request and the instant when M receives the rising edge on *ack* indicating the formation of batch 3. Hence, if M can fetch the required location in a time less than $2\tau_{1,n}$ the requested data may be sent back on the bus during cycle 3. For relatively long buses the above condition is usually satisfied and thus the upper bound on τ_{fetch} is $3\tau_{\text{cycle}}$. The lowest bound on τ_{fetch} is obtained when P is the lowest-priority device and M is the highest-priority device. In this case it takes at least $\tau_{1,n}$ time units for the request of P to reach M. The next batch, which includes the reply of M, requires $2\tau_{1,n}$ time units for its formation, and an additional $\tau_{1,n}$ is needed for the data to travel back from M to P. In other words,

$$4\tau_{1,n} \leqslant \tau_{\text{fetch}} \leqslant 3\tau_{\text{cycle}}. \quad (7)$$

For a distributed 64-processor system connected by a 100 m-long bus, if $N_{\text{av}} = 32$, $b = 32$, $w = 1$ nsec and $\tau_e = 5$ nsec, we have $4\tau_{1,n} = 1.32$ μsec and $N_{\text{av}}(\beta + \tau_e) = 1.18$ μsec. Hence, from (6) and (7), the memory-access time for non-local memory is between 1.32 and 7.5 μsec. This is comparable to the memory-access time in multiprocessor systems that use multistage interconnection networks.[14] However, the use of optical buses allows for a separation of 100 m between the processors in the system. In addition, a bus structure has the obvious advantage of using hardware that is linear with the number of processors.

4. CONCLUDING REMARKS

In this paper we have presented a technique for space multiplexing of signals on optical waveguides. This technique uses the unique transmission characteristics of optical signals to construct message pipelines in the communication channels of multiprocessors. We have shown that it is possible to realise a variety of interconnection structures for both synchronous and asynchronous systems. The achievable performance is comparable to that of multistage networks and the hardware complexity is linear with the number of processors. Further, the use of optics allows for the system to be distributed over distances that are larger than those feasible in electronics.

Thus we have shown that is is possible to implement large-scale distributed and tightly coupled multiprocessors. We foresee these systems as an alternative to multistage interconnection networks for implementing the next generation of supercomputers.

REFERENCES

1. R. Allan, Low-loss tapping opens doors to optical network buses. *Electronic Design* (Oct. 1984).
2. D. Chiarulli, R. Melhem and S. Levitan, Using coincident optical pulses for parallel memory addressing. *IEEE Computer*, **20** (12), 48–58 (1987).
3. D. Chiarulli, S. Levitan and R. Melhem, *Asynchronous control of optical buses in closely coupled distributed systems.* Technical Report 88–2, Department of Computer Science, the University of Pittsburgh (1988).
4. J. Fujimoto, A. Weiner and E. Ippen, Generation and measurement of optical pulses as short as 16 fs. *Applied Physics Letters*, **44** (9) (1984).
5. D. Gustavson, Introduction to the fastbus. *Microprocessors and microsystems.* **10**, (2), 77–85 (1986).
6. K. Hwang and F. Briggs. *Computer Architecture and Parallel Processing*, McGraw-Hill, Maidenhead (1984).
7. M. Kumar and G. Pfister, The onset of hot spot contention. *Proc. Int. Conf. Parallel Processing*, pp. 28–34 (1986).
8. D. Lawrie, Access and alignment of data in an array processor. *IEEE Transactions on Computers*, **C-24** (12), 1145–1155 (1975).
9. M. Nassehi, F. Tobagi and M. Marhic, Fiber optic configurations for local area networks. *IEEE Journal on Selected Areas in Communications*, **SAC-3** (6), 941–949 (1985).
10. G. Pfister and V. Norton, Hot spot contention and combining in multistage interconnection networks. *Proc. Int. Conf. Parallel Processing*, pp. 790–797 (1985).
11. R. Schmidt, E. Rawson, R. Norton, S. Jackson and M. Bailey, FIBERNET II: a fiber optic ethernet. *IEEE Journal on Selected Areas in Communications*, **SAC-1** (5) 702–710 (1983).
12. C. Shank, The role of ultrafast optical pulses in high speed electronics. In *Picosecond Electronics and Opto-electronics*, edited G. Morou, D. Bloom and C. Lee. Springer, Heidelberg (1985).
13. H. J. Siegel. *Interconnection Networks for Large-Scale Parallel Processing: Theory and Case Studies.* Lexington Books, Lexington, Mass (1984).
14. R. Thomas, R. Gurwitz, J. Goodhue and D. Allen, *Butterfly Parallel Processor Overview.* BBN Report no. 6148. Bolt Beranek and Newman, Cambridge, Mass. (1986).
15. F. Tobagi and M. Fine, Performance of unidirectional broadcast local area networks: Expressnet and Fastnet. *IEEE Journal on Selected Areas in Communications*, **SAC-1** (5), 913–926 (1983).
16. F. Tobagi, F. Borgonovo and L. Fratta, Expressnet: a high-performance integrated-services local area network. *IEEE Journal on Selected Areas in Communications*, **SAC-1**, (5) 898–912 (1983).
17. C. Wu and T. Feng, Universality of the shuffle exchange network. *IEEE Trans. on Computers*, **C-30** (5), 324–331 (1981).