Advances in Switching Networks
D.-Z. Du and H.Q. Ngo (Eds.) pp. – - –

Optical and Optoelectronic Interconnection Networks

Sartaj Sahni
Dept. of Computer and Information Science and Engineering
University of Florida, Gainesville, FL 32611
E-mail: sahni@cise.ufl.edu

Contents

1 Introduction **2**

2 Static Bus **3**
2.1 A Unidirectional Bus 3
2.2 One-Dimensional Array 6
2.3 Two-Dimensional Array 7

3 Reconfigurable Bus **8**
3.1 One-Dimensional Array 8
3.2 Two-Dimensional Array 9

4 Partitioned Optical Passive Stars **10**
4.1 The POPS(d,g) Network 10
4.2 SIMD Hypercube Simulation 13
4.3 SIMD Mesh Simulation 16
4.4 Intragroup Data Permutations 17

5 OTIS **21**
5.1 OTIS Topology 21
5.2 OTIS-Mesh 22

6 Acknowledgement **23**

References

1 Introduction

Several interconnection networks that are either wholly optical or combine optical and elctronic interconnect technology have been proposed recently for use in parallel computers. The more popular of these newly proposed interconnection networks may be classified as (1) networks that employ static pipelined optical buses, (2) networks that employ dynamically reconfigurable optical buses, (3) networks that employ the passive star network, and (4) optoelectronic interconnection networks that employ the optical transpose interconnect system. In this chapter, we provide the topology and properties of these networks. We also provide a few basic algorithms for parallel computers that employ these interconnection networks.

Single-mode waveguides are unidirectional and light pulses travel down the waveguide with a highly predictable delay [20]. As a result, waveguides support pipelining (i.e., at any instance, several messages encoded as light pulses can be traveling down a waveguide, one message behind the other). This means that several processors can simultaneously write different messages to the same optical bus; all messages can flow along the bus without interference. When electronic buses are used, only one message can be present on the bus at any given time. Therefore, when two or more processors attempt to write to a shared bus in the same cycle, we get a write conflict. Several pipelined bus models for parallel computers have been proposed and studied. Sections 2 and 3 review some of the more popular bus models.

The partitioned optical passive stars network (POPS) was proposed in [3, 6, 7, 13] as an optical interconnection network for a multiprocessor computer. The POPS network uses multiple optical passive star (OPS) couplers to construct a flexible interconnection topology. Each OPS (Figure 1) coupler can receive an optical signal from any one of its source nodes and broadcast the received signal to all of its destination nodes. The time needed to perform this receive and broadcast is referred to as a *slot*.

Although a single OPS can be used to interconnect n processors (in this case the n processors are both the source and destination nodes for the OPS), the resulting multiprocessor computer has very low bandwidth—only one processor may send a message in a slot. To alleviate this bandwidth problem a $POPS(d, g)$ network partitions the n processors into g groups of size d (d is also the degree of each coupler) each (so $n = dg$), and g^2 OPS couplers are used to interconnect pairs of processor groups. Section 4 describes the POPS network and provides a few basic algorithms for parallel

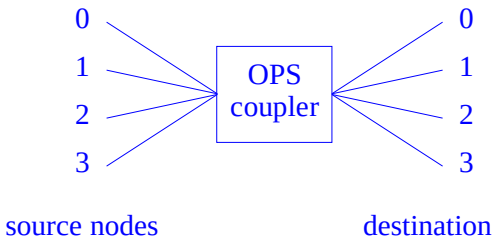

Figure 1: An optical passive star coupler with 4 source and 4 destination nodes

computers interconnected using this topology.

When designing a very large parallel computer system, processors are spread over several levels of the packaging hierarchy. For example, we might have several processors on a single chip, several chips on a wafer, several wafers on a printed circuit board, and so on. This means that, necessarily, the interprocessor distance cannot be kept small for all pairs of processors. When a connection must be run between processors on two different chips (for example), the connect distance is usually larger than when we connect two processors on the same chip. It is also known [5, 10] that optical connects provide power, speed, and crosstalk advantages over electronic interconnects when the connect distance is more than a few millimeters. Therefore, minimum overall delay is achieved when shorter interconnects are realized using electronics and longer interconnects are realized using optics. This realization leads to the concept of optoelectronic interconnection networks–interconnection networks that have a mix of optical and electronic interconnects. It is important to develop interconnection topologies that maximize the benefits of the two technologies and are manageable from the prespective of efficient algorithm design. The OTIS (optical transpose interconnection system) family of optoelectronic interconnection networks is a step in this direction.

2 Static Bus

2.1 A Unidirectional Bus

An n-processor unidirectional-bus computer is a synchronous computer in which n processors are connected to an optical bus (or waveguide). Figure 2 shows a 4-processor unidirectional-bus computer. The processors are labeled

0 through 3 and the optical bus is shown as a thick arrow. The processors are evenly spaced and the time required for an (fixed length) optical message (or signal) to travel the distance between two adjacent processors is denoted by τ. An optical bus is unidirectional–optical messages may travel in only one direction along the bus, that is, in the direction indicated by the arrow head. Each processor has a read/write connection to the optical bus, this is shown as a thin line in Figure 2.

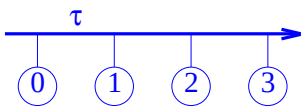

Figure 2: A unidirectional-bus computer.

If processor 0 writes a message to the bus, this message will arrive at processor 1 after τ time units and at processors 2 and 3 after 2τ and 3τ time units, respectively. If processors 0 through 3 write messages A, B, and C, respectively, to the bus at time 0, then message A arrives at processors 1, 2, and 3 at times τ, 2τ, and 3τ, respectively; message B arrives at processors 2 and 3 at time τ and 2τ; and message C arrives at processor 3 at time τ.

The *cycle time* of an n-processor unidirectional-bus computer is defined to be $n\tau$ [8]. The cycle time for the 4-processor configuration of Figure 2 is 4τ. A cycle may be regarded as composed of n slots, each of duration τ. It is generally assumed that a processor can write in only one slot in a cycle and can read from only one slot in a cycle. However, some models permit reading from several slots of a cycle. Several mechanisms have been proposed for how a processor knows which slot to write and which to read. Two of these mechanisms are:

1. *Use of a slot counter* All processors write in slot 0 of a cycle. If processor j, $j > i$, wants to read processor i's message, it starts a slot counter that is incremented by 1 every τ time units; when the counter reaches $j - i$, processor j reads from the bus. The drawback of this scheme [17] is that it requires an electronic counter that is at least as fast as the optical waveguide. This drawback is eliminated by replacing the arithmetic counter by the system clock which is advanced every τ units.

2. *Coincident Pulse Method* For this method, the optical bus is composed of three waveguides (see Figure 3). Call the original waveguide

the *message* or *data* waveguide and the new waveguides the *reference* and *select* waveguides. Delay units (essentially an optical fiber loop) are added between adjacent processors on the message and reference waveguides so that messages and reference signals (or pulses) experience a unit delay relative to select signals. Note that the time τ required for an optical signal to travel the distance between two adjacent processors is ≥ 1. Often, we assume that $\tau = 1$, and so the unit delay added by a delay unit equals one slot. If processor 0 writes a message to the message waveguide at time 0, this message gets to processor i at time $i(\tau + 1)$. To enable reading of this message by processor i, processor 0 also writes a reference signal to the reference waveguide at time 0 and a select signal to the select waveguide at time i. The reference signal arrives at processor i at the same time as does the message, that is, at time $i(\tau + 1)$. The select signal moves from one processor to the next in τ time and so takes $i\tau$ time (from initiation) to arrive at processor i. Therefore, the select signal also arrives at processor i at time $i(\tau + 1)$. When the reference and select signals arrive at a processor simultaneously (i.e., when the reference and select signals are coincident at a processor), the processor reads from the message waveguide. To enable reading of its data by all other processors, processor 0 must initiate a select signal at times 1, 2, 3, $\cdots$, $n - 1$.

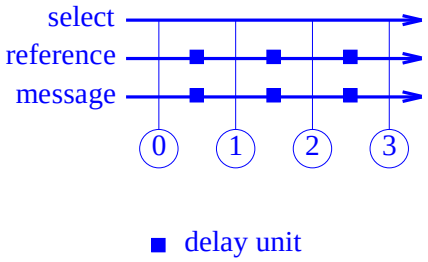

Figure 3: Reference and select buses.

As you can see, the coincident pulse method requires a clock that advances every time unit (a time unit equals the delay introduced between pairs of adjacent processors on the message and reference buses); this time unit is less than τ.

The complexity of algorithms for optical bus computers is usually measured in cycles. Although the cycle time for an n-processor bus varies with

n, for n up to a few thousand, the cycle time is no more than the time required to perform a CPU operation (such as an add or a compare) [18]. Typically, an algorithm will involve the CPU for $\Theta(1)$ steps between each bus cycle. So, the number of bus cycles times the CPU speed is a good measure of complexity for n up to a few thousand.

2.2 One-Dimensional Array

A unidirectional bus isn't of much use because there is no way for processor i to send a message to processor j when $i > j$, that is you cannot send a message to a processor on your left. This difficulty is overcome by adding an additional bus in which messages flow from right to left (Figure 4). The resulting parallel computer model is called *one-dimensional array with pipelined buses* (1D APPB) [8].

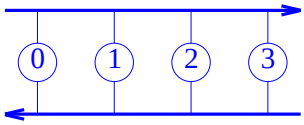

Figure 4: One-dimensional array with pipelined buses.

To see the power of a 1D APPB, suppose you want to permute the data in the n processors according to the permutation $p()$, that is processor j is to receive data from processor $p(j)$. Data from processor $p(j)$ will get to processor j in $|p(j) - j|$ slots (or $|p(j) - j|\tau$ time) provided it is written to the proper bus. So, processor j computes the number of slots $wait(j) = |p(j) - j|$ for which it must wait for its data. Following this computation, all processors write their data to the upper (i.e., the left to right) and lower buses at the start of a bus cycle. Processor j reads the desired data from the bus when its wait time is up. Amazingly, we can perform any data permutation in just one bus cycle. This is not possible using an electronic bus. For an electronic bus, define the bus cycle to be the time needed for an electronic signal written by a processor on a bus to become available at all processors on the bus. When an electronic bus is used, only one distinct data item can be transported on the bus in a bus cycle. Therefore, n cycles are required to perform a permutation.

An alternative one-dimensional model uses a folded message bus as in Figure 5 [20]. In this model, all writes are done to the upper bus segment and all reads are done from the lower bus segment. The cycle length for the

folded bus is $2n\tau$.

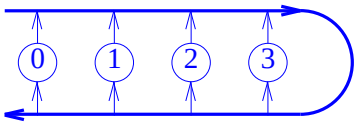

Figure 5: A folded one-dimensional bus computer.

When a folded bus is used, it takes $(2n-2-p(j)-j)\tau$ time for data to get from $p(j)$ to j. To perform the permutation $p()$, processor j computes $wait(j) = 2n-2-p(j)-j$ before the start of the bus cycle; all processors write to the upper bus segment at the start of a bus cycle; and processor j reads from the lower bus segment when its wait time is over. Again, the permutation is complete within one bus cycle.

2.3 Two-Dimensional Array

The 1D APPB model is easily generalized to higher dimensions. Figure 6 shows a two-dimensional version–the 2D APPB. The 2D APPB is quite similar to meshes with buses [26]–both have row and column buses. The essential difference is that the buses in a 2D APPB are optical whereas those in a mesh with buses are electronic; an electronic bus can carry only one distinct message at a time; whereas an optical bus can carry a distinct message in each slot.

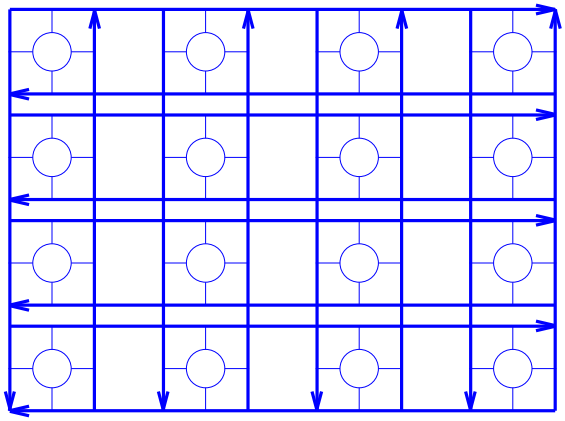

Figure 6: A 2D APPB.

A significant advantage afforded by two-dimensional arrays is the ability to build rather large computers while keeping the number of processors on an individual bus (and hence the bus cycle time) reasonable. If we limit the number of processors on a bus to a few thousand (see above), then the two-dimensional array allows us to build computers with up to a few million processors. The 2D APPB is harder to program than the 1D APPB. For example, to perform a data broadcast, we must first broadcast along a row bus (say) and then along all of the column buses. Even though an arbitrary permutation can be done in $O(1)$ bus cycles, the preprocessing needed by the permutation routing algorithm is excessive [8].

A two-dimensional array that uses folded row and column buses may also be developed.

3 Reconfigurable Bus

3.1 One-Dimensional Array

Given the success of reconfigurable architectures that employ electronic buses [14, 15], it is not surprising that reconfigurable optical bus architectures abound. In a one-dimensional reconfigurable bus, for example, processor i, $i > 0$ controls a *bus control* switch that enables it to break the optical bus at processor i. When the bus is broken at processors i_1, i_2, and i_3, $i_1 < i_2 < i_3$, for example, we get four independently operating one-dimensional bus computers. The first is comprised of processors 0 through $i_1 - 1$, the second of processors i_1 through $i_2 - 1$, the third has processors i_2 through $i_3 - 1$, and the fourth computer has processors i_3 through $n - 1$. This bus breaking into four segments is done by processors i_1, i_2, and i_3 by opening their bus control switch. Processors can open and close their switches dynamically while a program is executing. Hence, the computer may be reconfigured, as computation proceeeds, into a varying number of subcomputers.

The 1DAROB (one-dimensional array with reconfigurable optical bus) model of [18] and the LARPBS (linear array with a reconfigurable pipelined bus system) of [17] include a conditional delay unit between every pair of processors. This delay unit is on the upper segment of the select waveguide of the bus; processor i, $i > 0$, controls the delay unit to its left. When a delay unit is on, a one slot delay is introduced.

The conditional delay unit is useful in both the static and reconfigurable bus models. For example, the conditional delay unit may be used to find the

binary prefix sum in $O(1)$ bus cycles [17, 19]. Suppose that b_i is a binary value that is stored in processor i and that processor i is to compute $\sum_{j=0}^{i} b_j$, $0 \leq i < n$. Processor i, $i > 0$ turns its delay unit on (i.e., sets the unit so that the select pulse will be delayed by one slot) iff $b_i = 1$. Then, the leader (i.e., processor 0 unless the bus has been broken into subbuses) writes a select signal in either slot 0 or 1 of a bus cycle; the writing is done in slot 0 iff $b_0 = 0$. The leader also writes a reference signal in each slot. Processor i receives a reference signal in each slot beginning with slot i; it receives the select signal in slot $i + \sum_{j=0}^{i} b_j$. So, the two signals are coincident at processor i in slot $i + \sum_{j=0}^{i} b_j$. By starting a slot counter at the beginning of the bus cycle and stopping the counter when the select and reference signals are coincident at the processor, processor i can determine the value of $\sum_{j=0}^{i} b_j$. At most 2 bus cycles are needed to perform the binary prefix sum operation.

3.2 Two-Dimensional Array

The 2DAROB of [18] is a reconfigurable mesh [14, 15] in which the elctronic buses have been replaced by optical ones. Figure 7 shows a 4×4 2DAROB. Each arrangement of circular arcs denotes a switch; each line segment denotes a bidirectional optical bus segment, and there is one processor (not shown) at the center of each arrangement of circular arcs. The permissible switch settings are shown in Figure 8 Each processor can set its switch dynamically and thereby determine the bus topology. The switch settings at any time result in a set of disjoint buses. Each of these disjoint buses is required to be a unidirectional chain. The first processor on a configured bus is called the *bus leader*.

An n^2-processor 2DAROB can simulate an $n \times n$ reconfigurable mesh with a constant factor slow down [19]. Since a 2DAROB can perform a column permutation in 1 cycle, whereas an $n \times n$ reconfigurable mesh requires $O(n)$ cycles to this, a 2DAROB is more powerful (in the asymptotic sense) than a reconfigurable mesh. Some fundamental 2DAROB algorithms are developed in [21].

An alternative two-dimensional reconfigurable model, the array with synchronous switches (ASOS) was proposed in [20] (Figure 9). This model uses folded row and column buses; each processor can write only to the upper segment of its row bus; and each processor can read (concurrently) from the lower segment of its row bus and the right segment of its column bus. The shown switches can be in one of two states. In the *straight* state, mes-

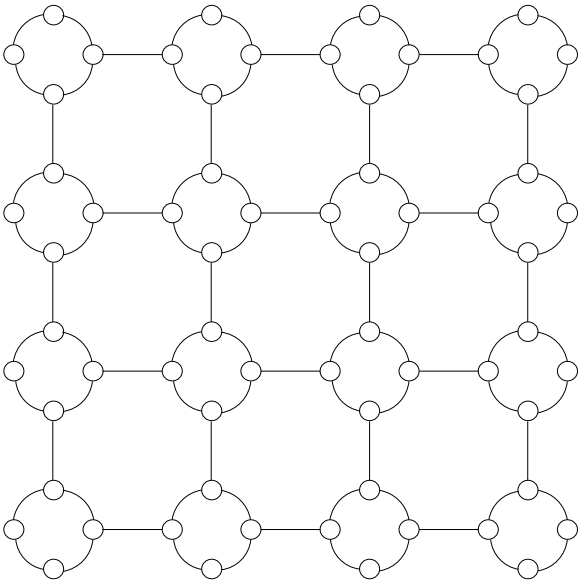

Figure 7: A 2DAROB.

sages move along row buses; and in the *cross* state, messages move from a row bus onto a column bus. Although [20] requires that all switches be set to the same state at any given time, we could permit each processor to independently control the state of its bottom left switch.

4 Partitioned Optical Passive Stars

4.1 The POPS(d,g) Network

The partitioned optical passive stars network (POPS) [3, 6, 7, 13] uses multiple optical passive star (OPS) couplers to construct a flexible interconnection topology. In a $POPS(d, g)$ network, the n processors are partitioned into g groups of size d (d is also the degree of each coupler) each (so $n = dg$), and g^2 OPS couplers are used to interconnect pairs of processor groups. Specifically the groups are numbered 0 through $g - 1$ and the source nodes for coupler $c(i, j)$ are the processors in group j; the destination nodes for this coupler are the processors in group i, $0 \leq i < g$, $0 \leq j < g$. Figure 10 shows how a $POPS(4, 2)$ network is used to connect 8 processors. Destination processor i is the same processor as source processor i, $0 \leq i < 8$. A $POPS(4, 2)$

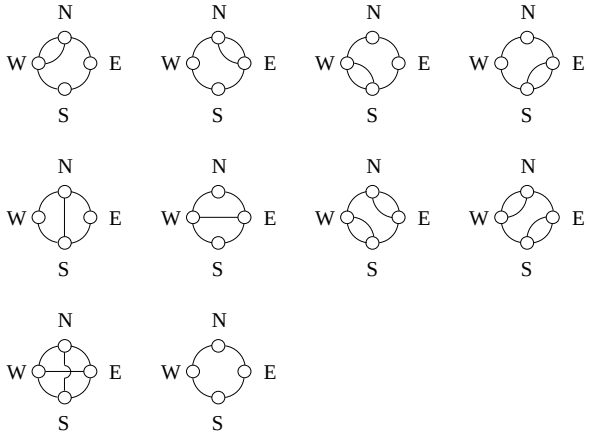

Figure 8: Permissible switch settings.

network comprises $2^2 = 4$ degree $d = 4$ OPS couplers.

When 8 processors are connected using a $POPS(8, 1)$ network only one degree 8 OPS coupler is used. A 32 processor computer may be built using any one of the following networks: $POPS(32, 1)$, $POPS(16, 2)$, $POPS(8, 4)$, $POPS(4, 8)$, $POPS(2, 16)$, and $POPS(1, 32)$. A 25 processor computer may be built using a $POPS(25, 1)$, $POPS(5, 5)$, or $POPS(1, 25)$ network. A multiprocessor computer that employs the POPS interconnection network is called a *POPS computer*.

The choice of the POPS network that is used to interconnect the processors affects both the interconnection cost as well as the bandwidth. When a $POPS(d, g)$ network is used to connect $n = dg$ processors, each processor must have g optical transmitters (one to transmit to each of the g OPSs for which it is a source node) and g receivers. The total number of transmitters and receivers is $2ng = 2n^2/d$, the number of OPSs is g^2, and each OPS has degree d. In one slot each OPS can receive a message from any one of its source nodes and broadcast this message to all of its destination nodes. In particular, in a single slot, a processor can send the same message to all of the OPSs for which it is a source node. In a single slot, a processor can receive a message from only one of the OPSs for which it is a destination node. (Melhem et al. [13] note that allowing a processor to receive different messages from different OPSs in the same slot permits faster all-to-all broadcast.)

A major advantage of the POPS network is that its diameter is 1. A

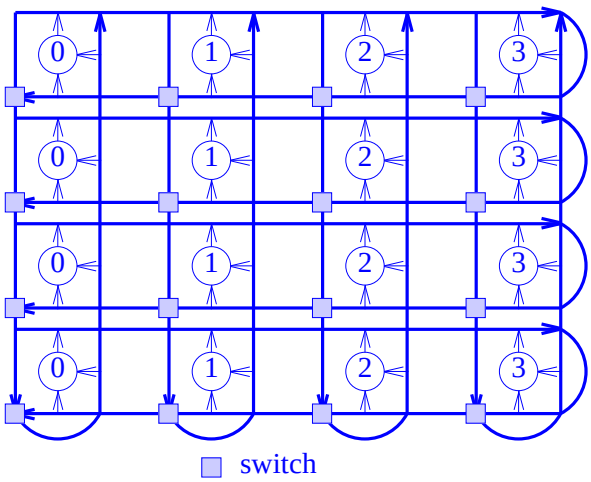

Figure 9: A 4×4 ASOS.

message can be sent from processor i to processor j, $i \neq j$, in a single slot. Let $group(i)$ be the group that processor i is in. To send a meesage to processor j, processor i first sends the message to coupler $c(group(j), i)$. This coupler broadcasts the received message to all its destination processors; that is, to all processors in $group(j)$ [7, 13]. A one-to-all broadcast can also be done in one slot [7, 13]. Suppose that processor i wishes to broadcast a message to all other processors in the system. Processor i first sends the message to all couplers $c(*, group(i))$ for which it is a source node. Next all couplers of the form $c(*, group(i))$ broadcast the received message to their destination nodes. Since processor j is a destination node of coupler $c(group(j), group(i))$, processor j, $0 \leq j < n$ receives the message broadcast by processor i. This algorithm is easily extended to perform an all-to-all broadcast in n slots (or in 1 slot when a processor can receive, in a single slot, messages from all couplers that it is a destination node of). [7, 13] also give an algorithm for all-to-all personalized communication.

[7] shows how to embed rings and tori into POPS networks. [1] shows that POPS networks may be modeled by directed stack-complete graphs with loops. This modeling is used to obtain optimal embeddings of rings and de Bruijn graphs into POPS networks. An alternative multiprocessor interconnection network using multiple OPSs is proposed in [4].

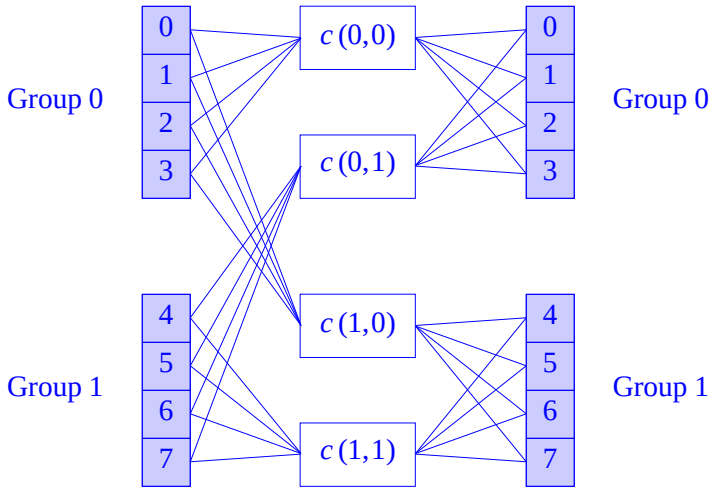

Figure 10: An 8-processor computer connected via a $POPS(4, 2)$ network

4.2 SIMD Hypercube Simulation

Let $i^{(b)}$ be the number whose binary representation differs from that of i only in bit b. The primitive communication in an $n = 2^D$ processor SIMD hypercube is for every processor i, $i \leq 0 < n$, to send data to processor $i^{(b)}$, for some fixed b, $0 \leq b < D$. This communication can be simulated by a $POPS(d, g)$ $(dg = n)$ computer using $2\lceil d/g \rceil$ moves. So every n processor SIMD hypercube algorithm can be run on an n processor $POPS(d, g)$ with a slowdown of a factor of at most $2\lceil d/g \rceil$.

Theorem 4.1 *An n processor $POPS(d, g)$ can simulate every move of an n processor SIMD hypercube using 1 slot when $d = 1$ and $2\lceil d/g \rceil$ slots when $d > 1$.*

Proof. First consider the case $d = 1$. A $POPS(1, g)$ can perform any permutation π in 1 slot using the routing

$$p(i, 1) \rightarrow c(\pi(i), i) \rightarrow p(\pi(i), 1)$$

In particular, the data routing done in 1 hypercube move can be done in 1 slot.

Next consider the case $d > 1$. Processor i of the SIMD hypercube is mapped onto processor i of the $POPS(d, g)$. Let b be the bit along which

the hypercube communication is to be done. That is, processor i of the hypercube is to send data to processor $i^{(b)}$ of the hypercube for all i. Let e and f be such that $ed + f = i$. That is $p(i)$ and $p(e, f)$ denote the same $POPS(d, g)$ processor.

First consider the case when $d = g = \sqrt{n} = 2^{D/2}$. When $b < D/2$, b is a bit of f and the communication can be done in two slots as below:

$$p(i) = p(e, f) \rightarrow c(f, e) \rightarrow p(f, e) \rightarrow c(e, f) \rightarrow p(e, f^{(b)}) = p(i^{(b)})$$

When $b \geq D/2$, b is a bit of e and the the communication is done in two slots as below:

$$p(i) = p(e, f) \rightarrow c(f, e) \rightarrow p(f, e) \rightarrow c(e^{(q)}, f) \rightarrow p(e^{(q)}, f) = p(i^{(b)})$$

The next case to consider is $d < \sqrt{n} < g$. When b is a bit of f we use the following 2 slot routing:

$$\begin{aligned} p(i) = p(e, f) &\rightarrow c(i \bmod g, e) && (1)\\ &\rightarrow p(i \bmod g, \lfloor i/g \rfloor) && (2)\\ &\rightarrow c(e, i \bmod g) && (3)\\ &\rightarrow p(e, f^{(b)}) = p(i^{(b)}) && (4) \end{aligned}$$

To establish the correctness of this routing we note that data that start in two different processors $i_1 = e_1 d + f_1$ and $i_2 = e_2 d + f_2$ use different couplers in slot 1 (i.e., for the source node to coupler routing of line 1). To see this observe that when $e_1 \neq e_2$ the slot 1 couplers differ in their second index. When $e_1 = e_2$, i_1 and i_2 differ by at most $d - 1 < g$ (and at least 1). So $i_1 \bmod g \neq i_2 \bmod g$, and the slot 1 couplers differ in their first index. Since $i = \lfloor i/g \rfloor g + i \bmod g$, data that originate in different processors are routed to different destination processors of line (2). For the slot 2 couplers (line (3)) we see that if $e_1 \neq e_2$, these couplers differ in their first index. When $e_1 = e_2$, i_1 and i_2 differ by at most $d - 1 < g$ (and at least 1). So the couplers differ in their second index.

When b is a bit of e the following 2 slot routing may be used:

$$\begin{aligned} p(i) = p(e, f) &\rightarrow c(i \bmod g, e)\\ &\rightarrow p(i \bmod g, \lfloor i/g \rfloor)\\ &\rightarrow c(e^{(b)}, i \bmod g)\\ &\rightarrow p(e^{(b)}, f) = p(i^{(b)}) \end{aligned}$$

The correctness proof for this routing is similar to that for the case when b is a bit of f. Notice that the 2 slot routing for the case $d < \sqrt{n} < g$ works when $d = g = \sqrt{n}$ also.

The final case to consider is $d > \sqrt{n} > g$. This case is handled using $\lceil d/g \rceil$ passes of the simulation strategy for the case $d < \sqrt{n} < g$. Notice that when the 2 slot routing of lines 1–4 is used up to $\lceil d/g \rceil$ source processors from the same group e attempt to route data to the same coupler in lines 1 and 3. This conflict is resolved by dividing the processors in each group into $\lceil d/g \rceil$ subgroups and doing the routing for each subgroup in a different pass. □

A SIMD hypercube algorithm may restrict a particular move along bit b to a subset of the processors. When the restriction is such that only one processor of each $POPS(d, g)$, $d > 1$, group is to send data, the simulation of Theorem 4.1 is not optimal. Suppose that a hypercube routing step is restricted in this way. When b is a bit of f, the data transfer along bit b may be accomplished using a single slot using the following routing:

$$p(i) = p(e, f) \rightarrow c(e, e) \rightarrow p(e, f^{(b)}) = p(i^{(b)})$$

Since for each $POPS(d, g)$ group e only one processor $p(e, f)$ is active, there is no coupler conflict in the above route. When b is a bit of e, the data transfer along bit b may be accomplished using a single slot as below:

$$p(i) = p(e, f) \rightarrow c(e^{(b)}, e) \rightarrow p(e^{(b)}, f) = p(i^{(b)})$$

Most SIMD hypercube data transfer steps along bit b will, however, require two or more processors of the same $POPS(d, g)$ group to transmit data. In this case the data transfer cannot be accomplished in a single slot and the 2 slot simulation of Theorem 4.1 is optimal. To see this observe that when b is a bit of f the only single slot routes for data in $p(e_1, f_1)$ and $p(e_1, f_2)$, $f_1 \neq f_2$, are:

$$p(e_1, f_1) \rightarrow c(e_1, e_1) \rightarrow p(e_1, f_1^{(b)})$$

and

$$p(e_1, f_2) \rightarrow c(e_1, e_1) \rightarrow p(e_1, f_2^{(b)})$$

These routes have a conflict because they use the same coupler. So no single slot routing is possible. When b is a bit of e, the only single slot routes for the two data are:

$$p(e_1, f_1) \rightarrow c(e_1^{(b)}, e_1) \rightarrow p(e_1^{(b)}, f_1)$$

and

$$p(e_1, f_2) \rightarrow c(e_1^{(b)}, e_1) \rightarrow p(e_1^{(b)}, f_2)$$

These routes use the same coupler and so are in conflict.

4.3 SIMD Mesh Simulation

In an $N \times N$ SIMD mesh data may be moved one processor right/left/up/down along the rows/columns of the mesh in one step. The direction of data movement in a single step is the same for all data. In a mesh with wraparound, a rightward move by 1 causes data in the rightmost processor of each row to reach the leftmost processor of the row (left, up, and down moves handle end elements similarly). A communication step of a mesh with wraparound (and hence of a mesh with no wraparound) can be simulated by a $POPS(d, g)$ ($dg = N^2$ and either d divides N or g divides N) computer using $2\lceil d/g \rceil$ slots. So every n processor SIMD mesh algorithm can be run on an n processor $POPS(d, g)$ with a slowdown of a factor of at most $2\lceil d/g \rceil$.

Theorem 4.2 *An $n = N^2$ processor $POPS(d, g)$ (d or g divides N) can simulate every move of an $N \times N$ processor SIMD mesh with wraparound using 1 slot when $d = 1$ and $2\lceil d/g \rceil$ slots when $d > 1$.*

Proof. The proof for the case $d = 1$ is the same as that for the simulation of a hypercube move. So consider the case $d > 1$. Processor (i, j) of the mesh is mapped onto processor $p(e, f)$ of the $POPS(d, g)$, where $ed + f = iN + j$. As in the case of the proof of Theorem 4.1, we consider three cases. The first case has $d = g = N$. When $d = g = N$, a rightward row move by 1 is accomplished using the 2 slot routing:

$$p(e, f) \rightarrow c(f, e) \rightarrow p(f, e) \rightarrow c(e, f) \rightarrow p(e, (f + 1) \bmod d)$$

A downward column move by 1 is done by the following 2 slot routing:

$$p(e, f) \rightarrow c(f, e) \rightarrow p(f, e) \rightarrow c((e + 1) \bmod g, f) \rightarrow p((e + 1) \bmod g, f)$$

Leftward row moves and upward column moves are done similarly.

When $d < N < g$ each row of the mesh occupies more than one group of the $POPS(d, g)$. Since d divides N, each row is housed in an integral number N/d of groups. A rightward shift by 1 along rows of the mesh requires data in processor $p(e, f)$ of the $POPS(d, g)$ to be routed to $p(e', f')$, where

$$e' = \begin{cases} e & \text{if } f \neq d - 1 \\ e + 1 & \text{if } f = d - 1 \text{ and } (e + 1) \bmod N/d \neq 0 \\ e + 1 - N/d & \text{if } f = d - 1 \text{ and } (e + 1) \bmod N/d = 0 \end{cases}$$

and

$$f' = (f + 1) \bmod d$$

Let $r = ed + f$. The following 2 slot routing moves data from $p(e, f)$ to $p(e', f')$ for all e and f.

$$\begin{aligned} p(e, f) &\rightarrow c(r \bmod g, e) && (5)\\ &\rightarrow p(r \bmod g, \lfloor r/g \rfloor) && (6)\\ &\rightarrow c(e', r \bmod g) && (7)\\ &\rightarrow p(e', f') && (8) \end{aligned}$$

The proof that different source processors use different slot 1 couplers (i.e., in line (5)) and different slot 1 destination processors (line (6)) is the same as for the corresponding lines of the proof of Theorem 4.1. Suppose that data originates in two different processors $p(e_1, f_1)$ and $p(e_2, f_2)$ in line (5). If $e_1' \neq e_2'$ there is no coupler conflict in line (7). If $e_1' = e_2'$, then $p(e_1, f_1)$ and $p(e_2, f_2)$ represent the same row of the mesh. Hence, $0 < |r_1 - r_2| < N < g$. So $r_1 \bmod g \neq r_2 \bmod g$ and there is no coupler conflict in line (7).

For a downward column move $e' = (e + N/d) \bmod g$ and $f' = f$. A downward move may also be done using the routing of lines 5–8. Leftward row moves and upward column moves are similarly dome in 2 slots per move.

As was the case in the proof of Theorem 4.1, the case $d > N > g$ is handled using $\lceil d/g \rceil$ passes of the routing of lines 5–8. □

The comments made following the proof of Theorem 4.1 regarding the optimality of the simulation apply to our simulation of a mesh also. When $d \geq 2\sqrt{n}$, a similarly optimal simulation of a 2D-mesh may be obtained using the embedding of a tori described in [2, 7].

4.4 Intragroup Data Permutations

Although [7, 13] give a data permutation algorithm for the $POPS(d, g)$ network, this algorithm is optimal only when we require that each data item be routed from its source processor to its destination processor using a single slot (or hop). Under this assumption datum initially in $p(i)$ and destined for $p(\sigma(i))$, where $\sigma(i)$ is the permutation that is to be performed and $i \neq \sigma(i)$, must be routed using the path $p(i) \rightarrow c(group(\sigma(i)), group(i)) \rightarrow p(\sigma(i))$.

When an intragroup permutation with $i \neq \sigma(i)$ for all i in the group (an example of such a permutation would be a nonzero intragroup shift), for example, is performed we get a coupler conflict—all d processors of a group need the same coupler to get their source data to the destination processors. This coupler conflict is eliminated by scheduling the use of the couplers so that no coupler has more than one source node transmitting data to the coupler. For an intragroup pemutation with $i \neq \sigma(i)$ for all i d time slots are taken to complete the permutation with each data making a single hop. We refer to the permutation algorithm of [7, 13] as the *single-hop* algorithm.

First we consider the case when a single intragroup permutation is to be performed. As noted above, when $i \neq \sigma(i)$ for all i in the group being permuted, the single hop algorithm takes d slots to perform an intragroup permutation. Without loss of generality, assume that the data in group 0 are to be permuted as per the permutation σ_0. That is, data from $p(0,i)$ is to be moved to $p(0,\sigma_0(i))$, $0 \leq i < d$. The single intragroup permutation σ_0 may be performed in $2\lceil d/g \rceil$ slots by routing g elements to their destination processors in every pair of slots. The number of slot pairs is d/g and in the qth pair, $0 \leq q < d/g$, processors $qg+i$, $0 \leq i < g$ send their data to the destination processors using two slots as follows:

1. In the first slot of a slot pair $p(0,qg+i)$ sends its data to $p(i,0)$ using the path $p(0,qg+i) \rightarrow c(i,0) \rightarrow p(i,0)$, $0 \leq i < g$.

2. In the second slot of the slot pair $p(i,0)$ sends the data it received in the first slot using the path $p(i,0) \rightarrow c(0,i) \rightarrow p(0,\sigma_0(qg+i))$, $0 \leq i < g$.

The correctness of the above algorithm is easily established. There is no advantage to having $d < \sqrt{n}$ because when $d < \sqrt{n}$, $g > \sqrt{n}$ and the intragroup permutation takes 2 slots; the same as when $d = g = \sqrt{n}$.

Although the above intragroup algorithm is simple, it is not optimal (except when $d \leq \sqrt{n}$). Figure 11 gives an optimal algorithm.

Once again, correctness is easily established. The number of slots is seen to be $\lceil (d-1)/g \rceil + 1$. Notice that each data item is routed to its destination using either one or two hops. Under the assumption that $i \neq \sigma_0(i)$ for any i in group 0, the single hop algorithm takes d slots. So the algorithm of Figure 11 provides a speedup of $d/(\lceil (d-1)/g \rceil + 1)$. When $d = g = \sqrt{n} > 1$ the speedup is $d/2$. When $g = 1$ the single hop algorithm uses the same number of slots (i.e., d) as used by the algorithm of Figure 11.

1. In slot 0 $p(0,0)$ uses the path $p(0,0) \rightarrow c(0,0) \rightarrow p(0,\sigma_0(0))$ to get its data to its destination processor. Processors $p(0,i)$, $0 < i < g$ use the paths $p(0,i) \rightarrow c(i,0) \rightarrow p(i,0)$ to get their data to intermediate processors in groups other than group 0.

2. In slot q, $q > 0$ we do the following:

   (a) $p(0,qg)$ uses the path $p(0,qg) \rightarrow c(0,0) \rightarrow p(0,\sigma_0(qg))$ to get its data to its destination processor.

   (b) Processors $p(i,0)$, $i \neq 0$, use the following paths to get the data they received in slot $q-1$ to destination processors: $p(i,0) \rightarrow c(0,i) \rightarrow p(0,\sigma_0((q-1)g+i))$.

   (c) Processors $p(0,qg+i)$, $1 \leq i < g$ and $qg+i < d$, use the following paths to get their data to intermediate processors: $p(0,qg+i) \rightarrow c(i,0) \rightarrow p(i,0)$.

Figure 11: An optimal algorithm for a single group intragroup permutation

A circular shift of the elements of group 0 by one unit takes d slots when the single hop algorithm is used and $\lceil (d-1)/g \rceil + 1$ slots when we use the algorithm of Figure 11.

Theorem 4.3 *The algorithm of Figure 11 is optimal when $i \neq \sigma_0(i)$, $0 \leq i < d$.*

Proof. Since all the data that is to be routed begins and ends in group 0, at most 1 datum can be routed to its final destination in the first slot of any algorithm. Since group 0 has g couplers, at most g different data can be routed to the group 0 processors in any slot. So every algorithm routes at most 1 datum to the final destination in the first slot and at most g in each of the remaining slots. Since d data are to be routed to their destination processors, at least $\lceil (d-1)/g \rceil + 1$ slots must be used. □

Now we consider the case when multiple intragroup permutations are to be performed. Suppose that data from $p(i,j)$ is to be routed to $p(i,\sigma_i(j))$, $0 \leq i < g$, $0 \leq j < d$. The $2\lceil d/g \rceil$-slot algorithm described above for a single permutation may be run, in parallel, on all groups. Therefore, data in all groups may be permuted in $2\lceil d/g \rceil = 2\lceil n/g^2 \rceil$ slots. We can reduce

1. Do the following for $0 \leq q < \lceil n/(g+g^2) \rceil$

2. // slot 1 of slot pair q
   Let $h = g + 1$.
   $p(i, qh)$ uses the path $p(i, qh) \rightarrow c(i, i) \rightarrow p(i, \sigma_i(qh))$ to route its data to the destination processor.
   Processors $p(i, qh+s)$, $1 \leq s < g$ use the following paths to route their data to intermediate processors (let $t = (i+s) \bmod g$): $p(i, qh+s) \rightarrow c(t, i) \rightarrow p(t, i)$ when $\sigma_t(qh) \neq i$ and $p(i, qh+s) \rightarrow c(t, i) \rightarrow p(t, t)$ when $\sigma_t(qh) = i$.

3. // slot 2 of slot pair q
   $p(i, qh+g)$ uses the path $p(i, qh+g) \rightarrow c(i, i) \rightarrow p(i, \sigma_i(qh+g))$ to route its data to the destination processors.
   Processors that received data from a different group in slot 1 of this slot pair route the received data via the coupler in the destination group to the destination processor.

Figure 12: Optimal algorithm to permute in all groups

the number of slots to $2\lceil n/(g+g^2) \rceil$ by moving batches of $g+g^2$ elements to their destination processors using 2 slots per batch. In the first slot g elements are moved to their destination processors and $g^2 - g$ elements are moved to intermediate processors. In the second slot g^2 elements are moved to their destination processors. Figure 12 gives the algorithm.

Using the algorithm of Figure 12 we can, for example, perform a circular shift by 1 of data in each group using $2\lceil n/(g+g^2) \rceil$ slots. Shifting all groups by 1 takes $d = n/g$ slots when the single hop algorithm is used.

Theorem 4.4 *The algorithm of Figure 12 is within 1 slot of being optimal when $j \neq \sigma_i(j)$, $0 \leq i < g$, $0 \leq j < d$.*

Proof. Suppose that an algorithm completes the group permutations in k slots. A total of n elements are to be moved from their initial processors to their destination processors. At most kg of these elements can be so moved using a single hop. Therefore, at least $n - kg$ elements are moved using 2 or more hops each. So the total number of data moves (in one data move a data item makes 1 hop) is $\geq 2(n - kg) + kg = 2n - kg$. In a $POPS(d, g)$ at

most g^2 data moves can be made in one slot. Therefore, $kg^2 \geq 2n - kg$. So $k \geq 2n/(g + g^2)$. Since k is an integer, $k \geq \lceil 2n/(g + g^2) \rceil$.

The number of slots used by the algorithm of Figure 12 is $2\lceil n/(g + g^2) \rceil \leq \lceil 2n/(g + g^2) \rceil + 1$. □

See [24, 25] for more algorithms that use the POPS network.

5 OTIS

5.1 OTIS Topology

The OTIS (optical transpose interconnect system) family of parallel computer models was proposed in [12, 9, 29]. In an OTIS parallel computer, the processors are divided into groups and each group of processors is realized using an electronic package (such as a high density chip or wafer). Intragroup connections are electronic and intergroup connections are realized using free space optics. Thus an OTIS system is an optoelectronic system. By contrast, all interprocessor connections in a pipelined bus system are optical.

Since optical connects provide power, speed, and crosstalk advantages over electronic interconnects when the connect distance is more than a few millimeters [5, 10], an OTIS system attempts to get the best of both worlds– electronic interconnect is used for the short-distance intragroup (or intra package) connections and optical interconnect is used for the longer-distance interpackage connections.

The bandwidth of an OTIS system is maximized and power consumption is minimized when the number of groups equals the number of processors in a group [11]. This means that an optimal N^2 processor OTIS system has N groups of N processors each. Let (G, P) denote processor P of group G (the processors in each group and the groups are numbered from 0 to $N - 1$). In an OTIS system, processor (G, P) is connected via an optical link to processor (P, G), for all G and P. If you regard (G, P) as a matrix index, then the matrix transpose operation moves element(G, P) to position (P, G), hence the name optical transpose interconnect system.

Figure 13 shows the topology of a 16-processor OTIS computer; processors are shown as small shaded boxes; the (G, P) index of each processor is given; each group of 4 processors is enclosed by a large box; and the OTIS (i.e., the optical transpose connections) are shown as bidirectional arrows.

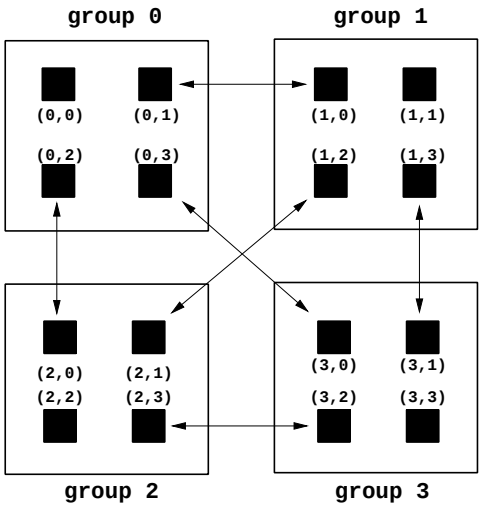

Figure 13: A 16-processor OTIS computer

For the intragroup interconnect topology, we may choose any of the electronic topologies proposed for parallel computers–mesh, hypercube, mesh of trees, etc. The selection of the intragroup topology identifies the specific OTIS model within the family of OTIS models. For example, an OTIS-mesh is an OTIS computer in which the intragroup interconnections correspond to a square mesh and in an OTIS-hypercube, the processors within each group are connected using the hypercube topology.

When analyzing the complexity of an OTIS computer we count the number of OTIS (i.e., intergroup or optical) data move steps and the number of electronic (or intragroup) data move steps.

5.2 OTIS-Mesh

Figure 14 shows a 16-processor OTIS-Mesh computer.

An OTIS-mesh can simulate each move of a four-dimensional $\sqrt{N} \times \sqrt{N} \times \sqrt{N} \times \sqrt{N}$ mesh using either one intragroup (i.e., electronic) move or one intragroup and two intergroup (i.e., OTIS) moves [29]. For the simulation, processor (i, j, k, l) of the 4D mesh is mapped on to processor (G, P) of the OTIS-Mesh, $G = i\sqrt{N} + j$ and $P = k\sqrt{N} + l$. It is easy to see that the 4D mesh moves $(i, j, k \pm 1, l)$ and $(i, j, k, l \pm 1)$ can be done with one intragroup move of the OTIS mesh. Moves of the form $(i \pm 1, j, k, l)$ and $(i, j \pm 1, k, l)$ can be done with two OTIS and one electronic move as follows. First an

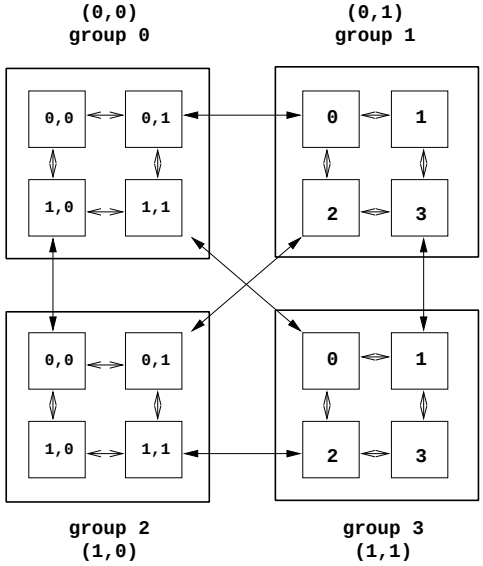

Figure 14: A 16-processor OTIS-Mesh

OTIS move is made to get data from (i, j, k, l) to (k, l, i, j); then an electronic move gets the data to $(k, l, i + 1, j)$ (say); and a final OTIS move gets it to $(i + 1, j, k, l)$.

Many OTIS-mesh and OTIS-hypercube algorithms appear in [22, 23, 16, 27, 28].

6 Acknowledgement

This work was supported, in part, by the National Science Foundation under grant CCR-9912395.

References

[1] P. Berthomé and A. Ferreira. Improved embeddings in POPS networks through stack-graph models. *Third International Workshop on Massively Parallel Processing Using Optical Interconnections*, IEEE, 130-135, 1996.

[2] P. Berthomé, J. Cohen, and A. Ferreira. Embedding tori in Partitioned Optical Passive Star networks. *Fourth International colloquium*

on Structiral Information and Communication Complexity-Sirocco'97, volume 1 of *Proceedings in Informatics*, Carleton Scientific, 40-52, 1997.

[3] D. Chiarulli, S. Levitan, R. Melhem, J. Teza, and G. Gravenstreter. Multiprocessor interconnection networks using partitioned optical passive star (POPS) topologies and distributed control. *First International Workshop on Massively Parallel Processing Using Optical Interconnections*, IEEE, 70-80, 1994.

[4] D. Coudert, A. Ferreira, and X. Muñoz. Multiprocesor architectures using multi-hop multi-ops lightwave networks and distributed control *12th International Parallel processing Symposium and 9th Symposium on Parallel and Distributed Processing*, IEEE, 151-155, 1998.

[5] M. Feldman, S. Esener, C. Guest, and S. Lee, Comparison between electrical and free-space optical interconnects based on power and speed considerations, *Applied Optics*, 27, 9, 1988.

[6] G. Graventreter, R. Melhem, D. Chiarulli, S. Levitan, and J. Teza. The partitioned optical passive stars (POPS) topology. *9th International Parallel processing Symposium*, IEEE, 4-10, 1995.

[7] G. Graventreter and R. Melhem. Realizing common communication patterns in partitioned optical passive star (POPS) networks. *IEEE Transactions on Computers*, 998-1013, 1998.

[8] Z. Guo, R. Melhem, R. Hall, D. Chiarulli, and S. Levitan, Pipelined communications in optically interconnected arrays, *JPDC*, 12, 3, 1991, 269-282.

[9] W. Hendrick, O. Kibar, P. Marchand, C. Fan, D. Blerkom, F. McCormick, I. Cokgor, M. Hansen, and S. Esener, Modeling and optimization of the optical transpose interconnection system in Optoelectronic Technology Center, Program Review, Cornell University, Sept. 1995.

[10] F. Kiamilev, P. Marchand, A. Krishnamoorthy, S. Esener, and S. Lee, Performance comparison between optoelectronic and vlsi multistage interconnection networks, *Journal of Lightwave Technology*, 9, 12, 1991.

[11] A. Krishnamoorthy, P. Marchand, F. Kiamilev, and S. Esener, Grain-size considerations for optoelectronic multistage interconnection networks, *Applied Optics*, 31, 26, 1992.

[12] G. Marsden, P. Marchand, P. Harvey, and S. Esener, Optical transpose interconnection system architectures, *Optics Letters*, 18, 13, 1993, 1083–1085.

[13] R. Melhem, G. Graventreter, D. Chiarulli, and S. Levitan. The communication capabilities of partitioned optical passive star networks. In *Parallel computing using optical interconnections*, K. Li, Y. Pan, and S. Zheng, Editors, Kluwer Academic Publishers, 77-98, 1998.

[14] R. Miller, V. K. Prasanna Kumar, D. Reisis, and Q. Stout, Efficient parallel algorithms for intermediate level vision analysis on the reconfigurable mesh, in *Parallel Architectures and Algorithms for Image Understanding*, V. K. Prasanna Kumar, ed., Academic Press, 1991, 185-207.

[15] R. Miller, V. K. Prasanna Kumar, D. Reisis, and Q. Stout, Meshes with reconfigurable buses, *IEEE Transactions on Computers*, 42, 1993, 678-692.

[16] A. Osterloh, Sorting on the OTIS-Mesh, *Proceedings 14th Annual International Parallel & Distributed processing Symposium*, IEEE Computer Society, 2000, 269-274.

[17] Y. Pan, Basic data movement operations on the LARPBS model, in *Parallel Computing Using Optical Interconnections*, K. Li, Y. Pan, and S. Zheng, editors, Kluwer Academic Publishers, Boston, 1998, 227-247.

[18] S. Pavel and S. Akl, Matrix operations using arrays with reconfigurable optical buses, *Journal of Parallel Algorithms and Applications*, 8, 1996, 223-242.

[19] S. Pavel and S. Akl, On the power of arrays with optical pipelined buses, *Proceedings 1996 International Conference on Parallel and Distributed Processing Techniques and Applications*, 1996, 1443-1454.

[20] C. Qiao and R. Melhem, Time-division optical communications in multiprocessor arrays, *IEEE Transactions on Computers*, 42, 5, 1993, 577-590.

[21] S. Rajasekeran and S. Sahni, Fundamental algorithms for the array with reconfigurable optical buses, in *Parallel Computing Using Optical Interconnections*, Kluwer, 1998, 185-204. Ed. Li and Zhenq.

[22] S. Sahni and C. Wang, BPC permutations on the OTIS-hypercube optoelectronic computer, *Informatica*, 22, 1998, 263-269.

[23] S. Sahni and Chih-Fang Wang, BPC permutations on the OTIS-mesh optoelectronic computer, *IEEE Conference on Massively Parallel Programming with Optical Interconnect*, 1997, 130-135.

[24] S. Sahni, Matrix Multiplication And Data Routing Using A Partitioned Optical Passive Stars Network, *IEEE Transactions on Parallel and Distributed Systems*, 11, 7, 2000, 720-728.

[25] S. Sahni, The Partitioned Optical Passive Stars Network: Simulations And Fundamental Operations, *IEEE Transactions on Parallel and Distributed Systems*, 11, 7, 2000, 739-748.

[26] Q. Stout, Mesh connected computers with broadcasting, *IEEE Transactions on Computers*, 1983, 826-830.

[27] Chih-fang Wang and S. Sahni, OTIS optoelectronic computers, in *Parallel Computing Using Optical Interconnections*, Kluwer, 1998, 99-116. Ed. Li and Zhenq.

[28] C. Wang and S. Sahni, Basic operations on the OTIS-Mesh optoelectronic computer, *IEEE Trans. on Parallel and Distributed Systems*, 9, 12, 1998, 1226-1236.

[29] F. Zane, P. Marchand, R. Paturi, and S. Esener, Scalable network architectures using the optical transpose interconnection system (OTIS), *Proceedings of the second International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'96)*, 1996, 114–121.