Models and Algorithms for Optical and Optoelectronic Parallel Computers*

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

February 26, 2001

ABSTRACT

This paper briefly reviews some of the more popular parallel-computer models–pipelined optical bus, optical transpose interconnect system (OTIS), and partitioned optical passive stars (POPS) network–that employ optical interconnect. The interconnect topology and some simple algorithms for each model are also described.

1. Introduction

The emerging feasibility of optical interconnects in very large parallel computers introduces new challenges in the development of efficient parallel algorithms. Single-mode waveguides are unidirectional and light pulses travel down the waveguide with a highly predictable delay [19]. 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. We review some of the more popular models here.

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 [4, 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

*This work was supported, in part, by the Army Research Office under grant DAA H04-95-1-0111. A preliminary version of this paper was presented at the 1999 International Symposium on Parallel Architectures, Algorithms, and Networks (I-SPAN'99) held in Perth/Fremantle, Australia

realization leads to the concept of optoelectronic computers–computers which have a mix of optical and electronic interconnects (or more generally, components). 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 family of optoelectronic computers is a step in this direction.

2. Static Bus Models

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 1 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 1.

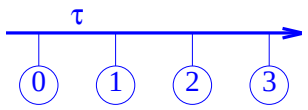

Figure 1: A unidirectional-bus computer.

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 1 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 [16] 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 2). Call the original waveguide the *message* or *data* waveguide and the new waveguides the *reference* and *select* waveguides. Delay units (es-

sentially 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, $\cdots$, $n - 1$.

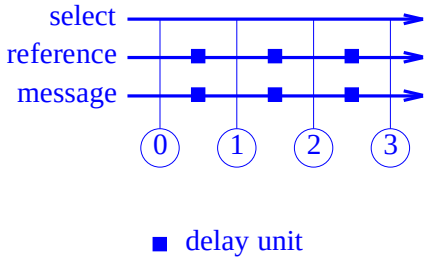

Figure 2: Reference and select buses.

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) [17]. 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 3). The resulting parallel computer model is called *one-dimensional array with pipelined buses* (1D APPB) [8].

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 $|p(j) - j|\tau$

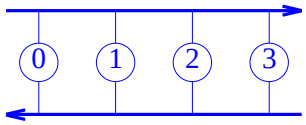

Figure 3: One-dimensional array with pipelined buses.

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 4 [19]. 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 $2n\tau$.

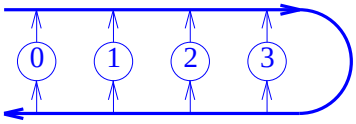

Figure 4: A folded one-dimensional bus computer.

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 5 shows a two-dimensional version–the 2D APPB. The 2D APPB is quite similar to meshes with buses [25]–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.

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].

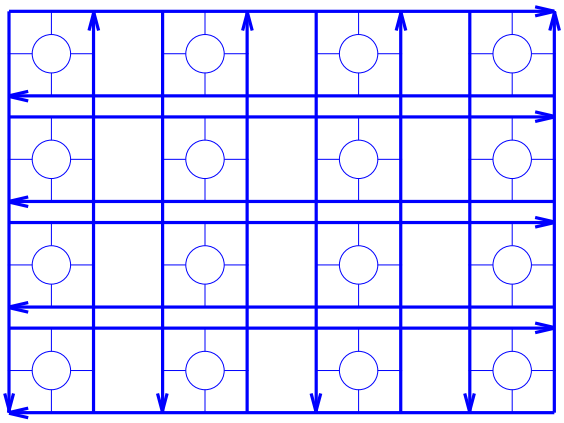

Figure 5: A 2D APPB.

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

3. Reconfigurable Bus Models

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 Pavel et al.[17] and the LARPBS (linear array with a reconfigurable pipelined bus system) of Pan[16] 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 [16, 18]. 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[17] is a reconfigurable mesh [14, 15] in which the elctronic buses have been replaced by optical ones. Figure 6 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 7 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*.

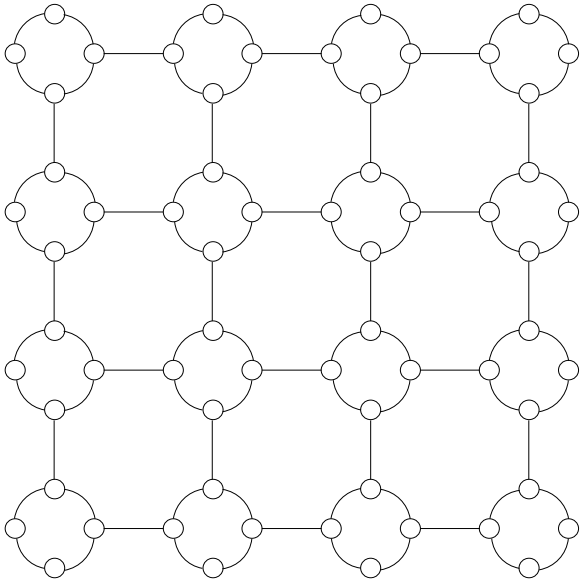

Figure 6: A 2DAROB.

An n^2-processor 2DAROB can simulate an $n \times n$ reconfigurable mesh with a constant factor slow down [18]. 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 by Rajasekeran and Sahni [20].

An alternative two-dimensional reconfigurable model, the array with synchronous switches (ASOS) was proposed by Qiao and Melhem[19] (Figure 8). 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, messages move along row buses; and in the *cross* state, messages move from a row bus onto a column bus. Although the model of Qiao and Melhem[19] requires that

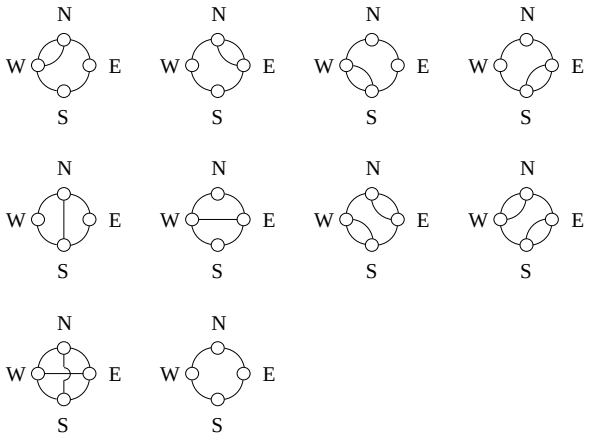

Figure 7: Permissible switch settings.

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.

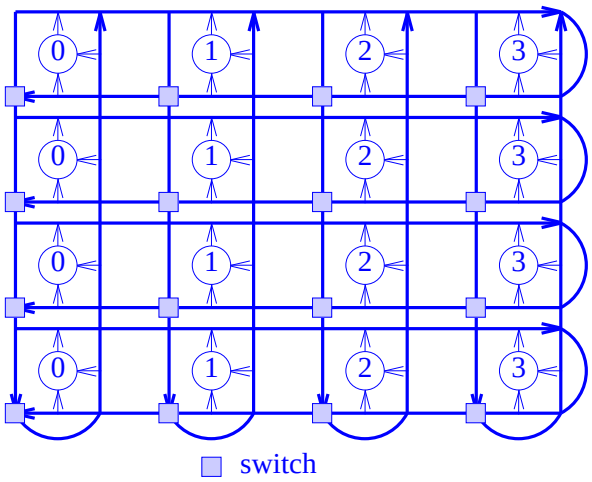

Figure 8: A 4×4 ASOS.

4. OTIS Models

4.1. OTIS Topology

In the OTIS (optical transpose interconnect system) family of parallel computer models[12, 9, 28], 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 [4, 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 9 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.

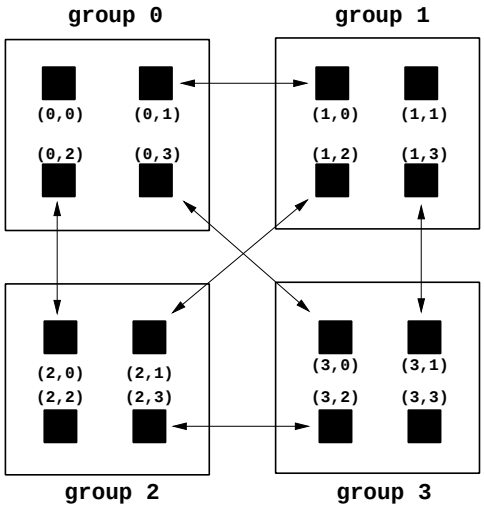

Figure 9: A 16-processor OTIS computer

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.

4.2. OTIS-Mesh

Figure 10 shows a 16-processor OTIS-Mesh computer.

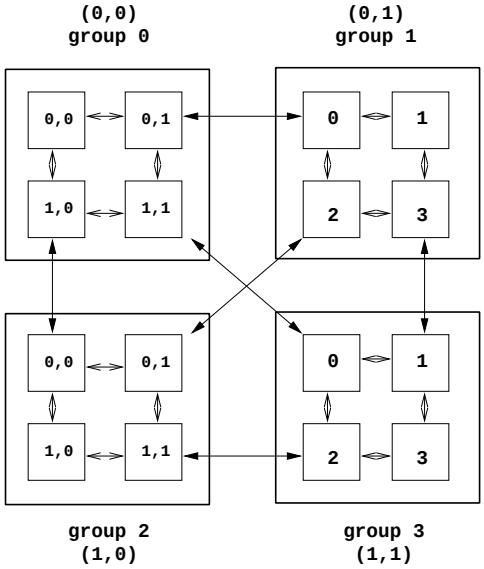

Figure 10: A 16-processor OTIS-Mesh

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 [28]. 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 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 have been developed[21, 22, 26, 27].

5. POPS Network

The partitioned optical passive stars (POPS) network[2, 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 11) 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

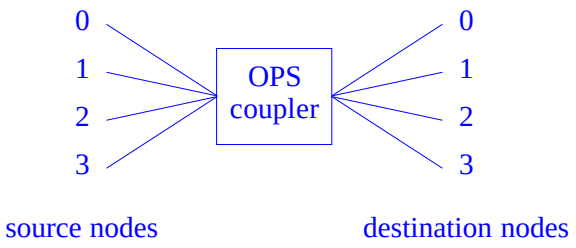

Figure 11: An optical passive star coupler with 4 source and 4 destination nodes

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 12 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 $POPS(4,2)$ network comprises $2^2 = 4$ degree $d = 4$ OPS couplers.

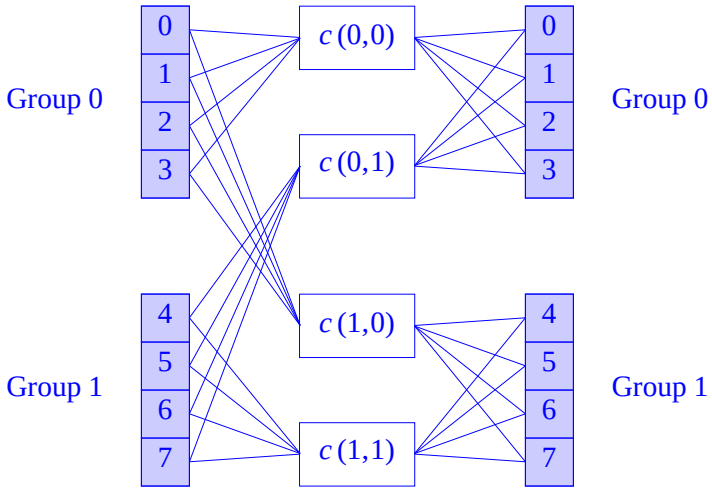

Figure 12: An 8-processor computer connected via a $POPS(4,2)$ network

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 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 procesors 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). Melhem et al.[7, 13] also give an algorithm for all-to-all personalized communication.

Graventreter et al.[7] show how to embed rings and torii into POPS networks. Berthomé et al.[1] show 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 has been proposed by Couderta et al.[3].

To get a feel for how we might program a POPS computer for an application, we consider the matrix multiplication problem. We make the following assumptions:

1. Two $N \times N$ matrices A and B are to be multiplied using a $POPS(d, g)$ computer with $n = dg = N^2$ processors.

2. A and B are mapped on to the computer one element per processor in row-major order. That is $A(i, j)$ and $B(i, j)$ are initially in processor $iN + j$, $0 \leq i < N$, $0 \leq j < N$. The result matrix C is to be similarly mapped on to the $n = N^2$ processors.

3. When $d \leq N$, d divides N, and when $d > N$, N divides d. This assumption ensures that a row of the matrix uses an integral number of processor groups or that a group contains an integral number of rows.

We use the notation $group(i, j)$ to refer to the processor group that contains elements $A(i, j)$ and $B(i, j)$ and that will eventually contain $C(i, j)$. $processor(i, j)$ refers to the processor that contains these three matrix elements. We also introduce a second numbering scheme for the processors of a $POPS(d, g)$ computer. Recall that in the first numbering scheme the processors are numbered 0 through $n - 1$ with processor i being the i mod d processor in group $\lfloor i/d \rfloor$. The second numbering scheme is a two-dimensional scheme. $p(i, j)$ refers to processor j of group i. So processors $p(i, j)$ and $p(id + j)$ are the same.

The matrix multiplication is done using the algorithm of Figure 13. The algorithm is readily seen to be correct. It is also simpler than the algorithm used by Dekel et al.[5] to multiply $N \times N$ matrices on mesh and hypercube computers that have $n = N^2$ processors.

```
C(i,j) = 0,  0 ≤ i < N,  0 ≤ j < N;
for (k = 0; k < N; k++)
{
    broadcast A(i,k) from processor(i,k) to processor(i,*), 0 ≤ i < N;
    broadcast B(k,j) from processor(k,j) to processor(*,j), 0 ≤ j < N;
    C(i,j) += A(i,k) * B(k,j),  0 ≤ i < N,  0 ≤ j < N;
}
```

Figure 13: Parallel matrix multiplication algorithm

The CPU time is $N + 1$ units because it takes 1 unit to initialize all $C(i, j)$ values to 0 and another 1 unit in each iteration of the `for` loop to increment $C(i, j)$ by $A(i, k)*B(k, j)$ (note that the N^2 elements of C are initialized in parallel and also incremented in parallel). The CPU time may be reduced to N units by initializing $C(i, j) = A(i, 0) * B(0, j)$. This requires a slight restructuring of the algorithm and does not affect the time spent broadcasting the As and Bs.

The specifics of how the A and B values are broadcast and the complexity analysis for the broadcasts depends on whether $d = \sqrt{n} = N$, $d < N$, or $d > N$.

The simplest case to consider is when each group holds exactly one row of each of the matrices A, B, and C. In this case $processor(i, j) = p(i, j)$.

The $A(i, k)$s, $0 \leq i < N$ are broadcast in one slot. First $processor(i, k)$ sends $A(i, k)$ to coupler $c(i, i)$. Next coupler $c(i, i)$ broadcasts $A(i, k)$ to $p(i, q) = processor(i, q)$, $0 \leq q < N$. $processor(i, q)$, $q \neq k$ receives the broadcast $A(i, k)$.

The broadcast of the $B(k, j)$s, $0 \leq j < N$ is done in two slots as follows. In the first slot $processor(k, j) = p(k, j)$ sends $B(k, j)$ to coupler $c(j, k)$, $j \neq k$. Next coupler $c(j, k)$ broadcasts the received $B(k, j)$ to $processor(j, *)$. The broadcast $B(k, j)$ is received by only one of the destination nodes of coupler $c(k, j)$; that is, by $p(j, k)$. In slot 2 $p(j, k)$, $0 \leq j < N$ sends the $B(k, j)$ value it received in slot 1 (if $j \neq k$) or its initial $B(k, j)$ value (in case $j = k$) to coupler $c(*, j)$ (i.e., to all couplers that it is a source node of). Next couplers $c(*, j)$ broadcast the received $B(k, j)$ value to their destination nodes and $p(*, j)$ receives $B(k, j)$.

An examination of how the $A(i, k)$ and $B(k, j)$ values are broadcast reveals that slot 1 of the $B(k, j)$ broadcast can be done at the same time as the $A(i, k)$ broadcast. This is because the $A(i, k)$ broadcast uses $p(*, k)$ as source nodes, couplers $c(i, i)$ for all i, and all processors other than $p(*, k)$ as destination nodes. Slot 1 of the $B(k, j)$ broadcast does not use any of these source nodes, couplers, or destination nodes. So there is no conflict in performing the slot 1 data routes of the $A(i, k)$ and $B(k, j)$ broadcasts in the same physical slot. Therefore each iteration of the matrix multiplication algorithm of Figure 13 takes only two slots. The total data routing time is $2N$ slots.

The cases $d < N$, or $d > N$ are considered by Sahni[23]. Sahni[23, 24] develops algorithms for basic arithmetic operations and data routing on a $POPS(d, g)$ computer.

6. 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. 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.
3. 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.
4. 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.
5. E. Dekel, D. Nassimi, and S. Sahni, Parallel matrix and graph algorithms. *SIAM Jr. on Comp.*, 10, 4, 657-675, 1981.
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, "Effi cient parallel algorithms for intermediate level vision analysis on the reconfi gurable 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 reconfi gurable buses," *IEEE Transactions on Computers*, 42, 1993, 678-692.
16. 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.
17. S. Pavel and S. Akl, "Matrix operations using arrays with reconfi gurable optical buses," *Journal of Parallel Algorithms and Applications*, 8, 1996, 223-242.
18. 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.
19. C. Qiao and R. Melhem, "Time-division optical communications in multiprocessor arrays," *IEEE Transactions on Computers*, 42, 5, 1993, 577-590.

20. 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.

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

22. 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.

23. S. Sahni, Matrix multiplication and data routing using a partitioned optical passive stars network. Submitted, 1999.

24. S. Sahni, The Partitioned Optical Passive Stars Network: Simulations And Fundamental Operations. Submitted, 1999.

25. Q. Stout, "Mesh connected computers with broadcasting," *IEEE Transactions on Computers*, 1983, 826-830.

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

27. 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.

28. 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.