

BASIC ALGORITHMS ON PARALLEL OPTICAL MODELS OF COMPUTING

SANGUTHEVAR RAJASEKARAN*

Abstract. In this paper we identify some of the optical models of computing that have been proposed and survey algorithms for fundamental problems. Problems considered include prefix computation, packet routing, selection, sorting, and matrix operations.

Key words. Optical computing, Randomized algorithms, Parallel computing, Matrix operations, Prefix, Sorting, Selection.

1. Introduction. Parallel machines based on optical technology offer superior power and speed. In the past several years many optical computing models have been proposed and studied. In this paper we identify three models that have been widely investigated. They are the Optical Communication Parallel Computer (OCPC), the Array with Reconfigurable Optical Buses (AROB), and the Optical Transpose Interconnection System (OTIS). We provide a survey of fundamental algorithms available for these models.

We begin by giving an introduction to the models of computing. For any of these models there are two versions, namely 1D and 2D. The 1D version has a $1 \times n$ configuration, whereas the 2D version has a $\sqrt{n} \times \sqrt{n}$ configuration.

1.1. Arrays with Reconfigurable Optical Buses. An Array with Reconfigurable Optical Buses (AROB) [24, 18] is an $m \times n$ reconfigurable mesh [11] wherein the buses are implemented using optical technology. This model has been extensively studied.

Figure 1 shows a 4×4 reconfigurable mesh. Local switches in each processor can be used to connect together subsets of the four bus segments connected to the processor.

Several closely related variants of the reconfigurable mesh with optical buses have been proposed in the literature. The variant used in this paper was proposed by Pavel and Akl [24, 18]. In this model, the switch settings of the processors permitted are the same as those in the RN model of [2]. Figure 2 shows these settings. Any bus link connects two adjacent processors. There are two associated wave guides, one for left to right transmission and the other for right to left transmission. Local bus links are set so as to form disjoint buses. The length of a bus is the number of links on that bus. The time needed to transmit a message on a bus is called a cycle.

*Department of Computer and Information Science and Engg., University of Florida, Gainesville, Florida 32611. Email: raj@cise.ufl.edu. This work is supported, in part, by an NSF Grant CCR-9596065 and an EPA Grant R 825 293-01-0.

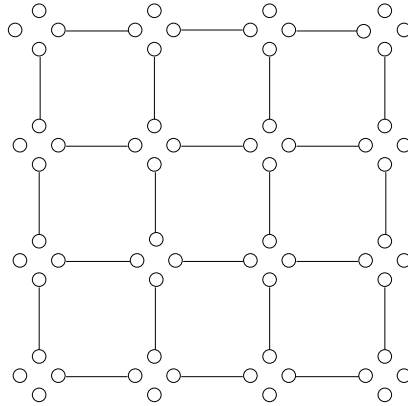
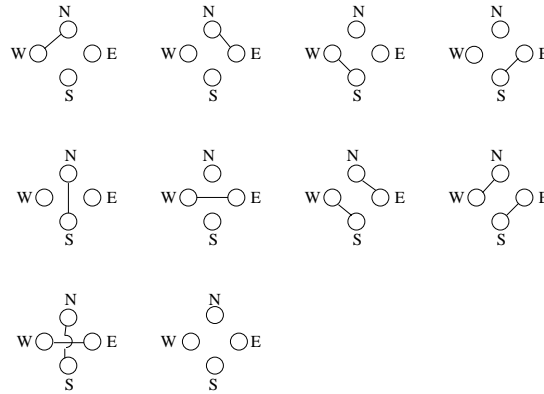
FIG. 1. A 4×4 Reconfigurable Mesh

FIG. 2. Possible Switch Connections

A cycle consists of *slots* of duration τ and each slot carries a different optical signal. The time needed for an optical pulse to move down a bus link is τ . To be more precise, τ time is enough to send a b -bit message where each bit is a light pulse with a w second duration, for suitable values of $b \geq 1$ and w . Pavel and Akl [24] argue that for reasonable size meshes (say up to 1000×1000), the number of slots in a cycle may be assumed to be n for an $n \times n$ mesh. Further, the duration of a cycle may be assumed constant and comparable to the time for a CPU operation.

In an AROB each processor has an associated slot counter. These counters may be started at the beginning of a cycle. For every processor, a slot in the cycle is allotted for reading and writing. This processor can write to or read from the bus only in the allotted slots. If more than one message gets written in a slot, the last written message remains in the slot. Since a processor can read/write from/to its bus during only one

slot of a cycle, it cannot poll the up to n light pulses moving through it in one cycle. Another AROB feature that facilitates the development of algorithms is the delay unit at each processor. A processor can introduce a one time slot delay in the light pulses passing through it.

1.2. The Optical Transpose Interconnection System (OTIS).

This model of computing has been proposed in [9, 17, 38]. In this model, processors are partitioned into groups where each group is realized as chips with electronic interprocessor connections. Connections among the groups are realized using free space optical links. The advantage of this opto-electronic architecture lies in the fact that free space optical links provide superior speed and power when the connect distance is more than a few millimeters.

Processors in any group can be organized as a mesh, a hypercube, or any other network. Accordingly the OTIS-Mesh, the OTIS hypercube, etc. will arise. Let j be any processor in group i . In the OTIS model, this processor is connected to processor i of group j . Figure 3 shows an OTIS-Mesh where there are four groups and each group has 4 processors.

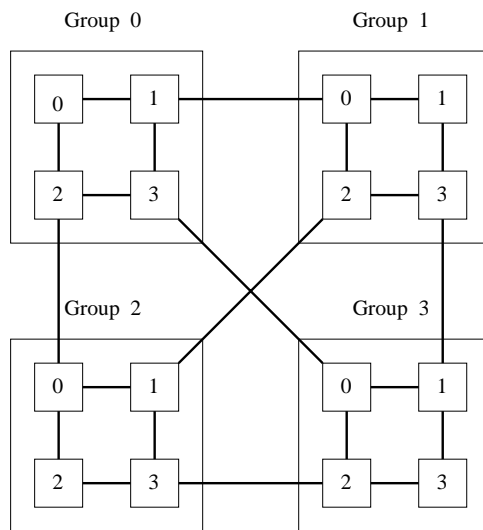


FIG. 3. A 4×4 OTIS-Mesh

An $n \times n$ OTIS-Mesh has n groups where each group has n processors organized as a $\sqrt{n} \times \sqrt{n}$ mesh. The diameter of this mesh has been shown to be $4\sqrt{n} - 3$ [33]. The Mesh architecture could either be SIMD or MIMD. Accordingly, two variants of the OTIS-Mesh can be conceived of.

1.3. The Optical Communication Parallel Computer. In an OCPC any processor can communicate with any other processor in one step provided there are no conflicts. If more than one processor sends a message to the same processor π at the same time, then π does not receive any meaningful message. Each step of an EREW PRAM that has n processors and n common memory cells can be simulated by an OCPC with $2n$ processors in $O(1)$ time as follows. Allocate n of the OCPC processors to simulate the PRAM processors. The other n processors are allocated to the common memory cells one processor per cell. Clearly, each read or write step of the PRAM now can be simulated in $O(1)$ communication steps on the OCPC. In general we can state the following Lemma.

LEMMA 1.1. *Each step of an N -processor EREW PRAM that uses M common memory can be simulated in $O(1)$ steps on an OCPC with $\max\{N, M\}$ processors.*

2. Preliminaries. In this section we provide some preliminary facts and results that will be employed in the paper.

2.1. Randomized Algorithms. A randomized algorithm is said to use $\tilde{O}(f(n))$ amount of any resource (like time, space, etc.) if the amount of resource used is no more than $cf(n)$ with probability $\geq (1 - n^{-\alpha})$ for any $\alpha, c > 0$ being a constant. We could also define $\tilde{\Theta}(\cdot), \tilde{o}(\cdot)$, etc. in a similar manner. By high probability we mean a probability of $\geq (1 - n^{-\alpha})$ for any constant $\alpha \geq 1$.

2.2. Problems Definition. Given a sequence of numbers, say, k_1, k_2, \dots, k_n , the problem of sorting is to rearrange them in nondecreasing order.

The problem of selection is to identify the i th smallest number from out of n given numbers (where i is an input and $1 \leq i \leq n$).

In any fixed connection network, a single step of interprocessor communication can be thought of as a packet routing task. The problem of routing can be stated as follows: There is a packet of information at each node that is destined for some other node. Send all the packets to their correct destinations as quickly as possible such that no more than one packet crosses any edge at any time. The *run time* of any packet routing algorithm is defined to be the time taken by the last packet to reach its destination. The *queue size* is the maximum number of packets that any processor will have to store during the algorithm.

The restriction of routing where at most one packet originates from any node and at most one packet is destined for any node is called *partial permutation routing*. If at most h packets originate from any node and at most h packets are destined for any node, then we have a *h -relations routing* or a *$h-h$ routing* problem [36, 27].

Matrix operations considered in this paper include multiplication and inverse.

3. Prefix Computation.

3.1. AROB. LEMMA 3.1. *Let \mathcal{L} be a $1 \times n$ -processor AROB. If each processor of \mathcal{L} has a bit, then the prefix sums of these bits can be computed in $O(1)$ cycles [24].*

Proof. The algorithm works as follows: Processor 1 initiates a light pulse in time slot one of a cycle if its bit is zero and in slot two otherwise. Each processor starts its counter at the start of the cycle and also sets its delay unit to introduce a one slot delay if its bit is one. When the light pulse initiated by processor 1 reaches any processor its counter is turned off. A processor can compute its prefix sum value from the terminal counter value, its data bit, and its distance from processor 1. \square

The above algorithm can be extended to show the following [24]:

LEMMA 3.2. *The addition of $n \log n$ -bit numbers can be performed in $O(1)$ cycles on a $\log n \times n$ AROB.*

Using Lemmas 3.1 and 3.2 we can get an $O(1)$ -cycle algorithm for prefix sums of bits on a $\sqrt{n} \times \sqrt{n}$ AROB as follows. Assume that we are interested in computing prefix sums in row major order. Compute prefix sums along each row in $O(1)$ cycles using Lemma 3.1. After this, each processor in the last column has the corresponding row sum. Now compute prefix sums on this column in $O(1)$ time using Lemma 3.2. In an additional $O(1)$ cycles, we can update all the prefixes.

LEMMA 3.3. *If there is a bit at each node of a $\sqrt{n} \times \sqrt{n}$ AROB, we can compute the prefix sums of these bits in $O(1)$ cycles.*

The maximum bus length employed by [24]'s algorithm is $3\sqrt{n}$. The bus length is reduced to $\sqrt{n} + o(\sqrt{n})$ in [30].

3.2. OTIS-Mesh. Consider an $n \times n$ OTIS-Mesh. There are n groups of processors where each group is a $\sqrt{n} \times \sqrt{n}$ Mesh. Let the processors in any group be indexed in snake-like row-major order. There is an element to begin with at each processor. Let the prefix indexing be the lexicographical ordering of $(group, processor)$.

Prefix computation on the OTIS-Mesh can be done in $6\sqrt{n} + O(1)$ time as follows. In $2\sqrt{n} - 2$ time, each group performs a local prefix computation. As a result processor n in every group has the sum of all the elements in the group. The sum of group i is sent to processor i of group n , for $1 \leq i \leq n$. This can be done in one time unit using the transpose OTIS connections. Now group n performs a prefix computation on the sums in $2\sqrt{n} - 2$ time. The result is moved right by one position in one time unit. These prefixes are sent out using the transpose OTIS connections. After this, processor n in group i will have the sum of all the elements in groups 1 through $i - 1$ (for $2 \leq i \leq n$). Finally, a broadcasting local to groups is done in order to update the prefixes.

LEMMA 3.4. *We can do prefix computation on an $n \times n$ OTIS-Mesh in $6\sqrt{n} + O(1)$ time.*

3.3. OCPC. An OCPC with n processors can simulate each step of an EREW PRAM with n processors and $O(n)$ common memory cells in $O(1)$ time (c.f. Lemma 1.1). This observation implies:

LEMMA 3.5. *Prefix computation on a sequence of length n can be computed in $O(\log n)$ time on an n -processor OCPC.*

4. Packet Routing.

4.1. AROB. LEMMA 4.1. *In an AROB of size $1 \times n$ any permutation can be routed in $O(1)$ cycles [24].*

Proof. Let τ be the time taken by a packet to move from one processor to the next. The flow of packets from left to right is not affected by the flow of packets from right to left since there are two waveguides. Consider any permutation to be routed. Let the processors be numbered $1, 2, \dots, n$ starting from left. Each processor has a time slot assigned for reading from (and writing into) the bus. The reading time slot for processor i is $2i$. Processor 1 creates a ‘time slot’ for each packet that moves one edge per τ time. The first time slot is meant for processor 1, time slot 2 is meant for processor 2, and so on. Let the destination of the message from p be q . Processor p writes this message at time $t + (p + q)\tau$, where t is the start time. This algorithm takes 2 cycles (or $2n\tau$ time). \square

The preceding lemma can be strengthened easily as follows:

LEMMA 4.2. *Let \mathcal{L} be an AROB of size $1 \times n$. Consider a routing problem where $O(1)$ packets originate from any node and $O(1)$ packets are destined for any node. This problem can also be solved in $O(1)$ cycles.*

LEMMA 4.3. *There are k elements arbitrarily distributed in a $\sqrt{n} \times \sqrt{n}$ AROB, with ≤ 1 element per processor. The problem is to collect them in the first $\lceil \frac{k}{\sqrt{n}} \rceil$ rows. This problem can be solved in $O(1)$ cycles.*

Proof. Figure out a unique address for each element (using the prefix algorithm of Lemma 3.3) and then route the elements using greedy paths. There is no possibility of a collision. \square

There is an $\tilde{O}(\log \log n)$ time algorithm available for routing any partial permutation on a 2D OCPC. The same algorithm can be simulated on a 2D AROB to get the same run time.

LEMMA 4.4. *Any partial permutation can be routed in $\tilde{O}(\log \log n)$ time on a $\sqrt{n} \times \sqrt{n}$ AROB.*

Now we look at the problem of h -relations routing. This problem has been studied extensively on the conventional mesh as well as the OCPC. Rajasekaran and Sahni have given a randomized algorithm for this problem that runs in $\tilde{O}(h)$ cycles on a 1D AROB. The best known h -relations routing algorithm on the OCPC has a run time of $\tilde{O}(h + \log \log n)$ [8].

There are two crucial differences between the OCPC and AROB. In the case of an OCPC if two or more processors try to send a message to the same processor at the same time, then no message reaches the destination. On the other hand in an AROB, under the same scenario, an arbitrary packet reaches the destination. Also, prefix sums of bits can be computed

in $O(1)$ cycles on the AROB model and there is no such algorithm for the OCPC.

In [30]'s algorithm, every processor randomly chooses one of its packets and sends it in the bus. Let π be any such chosen packet. If there are collisions (i.e., if there are other packets in the bus with the same destination as π 's), π may not reach its destination. At any time step, there is some constant probability that a processor will succeed (i.e., the packet chosen by it will reach its destination). The algorithm runs in *stages*. At the end of every stage a *load balancing* operation is performed. The load balancing operation distributes unsuccessful packets equally among all the processors. If the maximum number of packets in any processor is k , then the load balancing operation takes only $O(k)$ time.

Let k_i be the maximum number of packets in any node at the beginning of stage i , with $k_1 = h$. Also let N_i be the number of packets that have not yet been routed at the beginning of stage i , where $N_1 \leq nh$. They show that after every stage of routing, the value of k_i decreases by a constant factor with high probability. Thus there will be $\tilde{O}(\log h)$ stages of routing. After $\tilde{O}(\log h)$ stages of routing, there will be $\tilde{O}(n)$ packets left. These packets can be routed by load balancing followed by an application of Lemma 4.2. The amount of time spent in stage i is $O(k_i)$ cycles. Thus the total run time is $\tilde{O}(h)$ cycles.

LEMMA 4.5. *There is a randomized algorithm for routing h -relations that takes $\tilde{O}(h)$ cycles on a $\sqrt{n} \times \sqrt{n}$ AROB.*

They [30] extend their algorithm to a $\sqrt{n} \times \sqrt{n}$ AROB. The resultant algorithm runs in $\tilde{O}(h + \log \log n)$ cycles. This result has to be contrasted with the run time of $\tilde{O}(h + \log n / \log \log n)$ that the algorithm of [32] takes for the same problem on the OCPC model.

Let π be any packet. It chooses a random node in its row of origin and tries to go there in the first phase. In phase two it traverse along the current column up to the destination row. In phase three it traverses along the current row up to its destination column. Clearly, π can participate in the second phase only if it were successful in the first phase, and so on. Call these three phases a *stage* of the algorithm. Note that a phase takes three cycles. The OCPC algorithm of [32] consists of a sequence of such stages. The algorithm of [30] also consists of stages except that they perform a load balancing operation after every stage.

Here also the value of k_i decreases by a constant factor after every stage, with high probability. It takes $\tilde{O}(\log \log n)$ time to process the $\tilde{O}(n)$ packets that will be left at the end.

LEMMA 4.6. *Any h -relation can be routed in $\tilde{O}(h + \log \log n)$ cycles on a $\sqrt{n} \times \sqrt{n}$ AROB.*

The following Lemma is also proven in [30].

LEMMA 4.7. *Any partial permutation can be routed deterministically in $O(\log n)$ cycles on a $\sqrt{n} \times \sqrt{n}$ AROB. Also, any h -relation can be deter-*

ministically routed in $O(h \log n)$ cycles on a 2D AROB of size $\sqrt{n} \times \sqrt{n}$.

4.2. OTIS. Rajasekaran and Sahni [29] have presented a randomized algorithm that can route any partial permutation in $4\sqrt{n} + \tilde{o}(\sqrt{n})$ time on an $n \times n$ OTIS-Mesh. The algorithm is an adaptation of [31]’s algorithm. In the algorithm of [31], a packet chooses a random node within a restricted space around its origin and goes there. From then on it greedily progresses toward its destination traveling the dimensions one at a time.

A similar algorithm applies to the OTIS-Mesh also. If the OTIS-Mesh is thought of as a 4D Mesh (with dimensions u, v, w , and x), and [31]’s algorithm is applied, then a packet may in the worst case have to traverse close to \sqrt{n} distance in each of the four dimensions. But a data movement along the u and v dimensions will need three steps [33]. This in turn means that the packet has to spend at least $8\sqrt{n}$ time. Traversals along the u and v dimensions can be converted into traversals along the w and x dimensions using the transpose OTIS connections. This technique is called *swapping dimensions*. There are four phases in the algorithm of [29]. In the first phase each packet traverses to a random node (within some restricted space) and in the next three phases it traverses along the w, x dimensions; transpose OTIS connection; and the w, x dimensions. The first phase takes $\tilde{o}(\sqrt{n})$ time. Phases 2 and 4 take $2\sqrt{n} + \tilde{o}(\sqrt{n})$ time each. Phase 3 needs $O(1)$ time.

LEMMA 4.8. *We can route any partial permutation on an $n \times n$ OTIS-Mesh in $4\sqrt{n} + \tilde{o}(\sqrt{n})$ time, the queue sizes being $\tilde{O}(1)$.*

The problem of h -relations routing can be solved by applying the algorithm given in [27].

LEMMA 4.9. *Any h -relation can be routed on an $n \times n$ OTIS-Mesh in $\tilde{O}(h\sqrt{n})$ time, the queue sizes being $\tilde{O}(h)$.*

4.3. OCPC. On a 1D OCPC, any partial permutation can be clearly routed in one unit of time. Rao and Tsantilas [32] have given a partial permutation routing algorithm for a $\sqrt{n} \times \sqrt{n}$ OCPC that runs in time $\tilde{O}(\log \log n)$. If π is any packet whose origin is (i, j) and whose destination is (k, l) , π chooses a random node (i, r) in its row of origin and tries to go there. If the transmission is successful, (i, j) will get an acknowledgement. If π reached (i, r) successfully, it will be sent to (k, r) . If this transmission is successful, π will finally reach (k, l) . Note that in the third phase there is no chance of collisions. The time complexity is proven by showing that if there are $\alpha\sqrt{n}$ packets at any given time in any row, then after a constant amount of time the row will have no more than $\alpha^2\sqrt{n}$ packets, with high probability.

LEMMA 4.10. *Any partial permutation can be routed in $\tilde{O}(\log \log n)$ time on a $\sqrt{n} \times \sqrt{n}$ OCPC.*

Several h -relations routing algorithms have been proposed in the literature. Anderson and Miller have shown that $\log n$ -relations on an n -node 1D OCPC can be routed in $\tilde{O}(\log n)$ time [1]. Followed by this, Valiant gave

an algorithm that works for any h -relation with a run time of $\tilde{O}(h + \log n)$. Later, Geréb-Graus and Tsantilas [7] came up with a simple algorithm that has a run time $\tilde{O}(h + \log n \log \log n)$. An algorithm for arbitrary h -relations with a run time of $\tilde{O}(h + \log \log n)$ has been given by Goldberg, Jerrum, Leighton, and Rao [8].

Geréb-Graus and Tsantilas' algorithm works as follows. There are *phases* in the algorithm. At the start, the algorithm has to solve a h -relations problem. After one phase, with high probability, only a $\frac{h}{2}$ -relation will remain to be routed. If there is a k -relation at the beginning of a phase, the time spent in this phase is $\max\{k, \log n\}$. A step in a phase consists of every processor picking randomly one of its remaining packets and attempting to send it. If there are r packets in a processor, it attempts to send its packet only with probability $\frac{r}{k}$.

Goldberg, Jerrum, Leighton, and Rao's algorithm is more complex. There are four stages in the algorithm each taking $\tilde{O}(h + \log \log n)$ time. 1) In every step of the first stage each processor picks one of its remaining packets and tries to send it. After the first stage there are only $\tilde{O}\left(\frac{n}{h \log \log n}\right)$ unsent packets. 2) In the second stage a load balancing operation is performed so that each processor ends up with at most one packet. Group the processors so that each group has $\log^c n$ processors, for some constant c . 3) In the third stage packets are sent to some processors in their target groups. 4) Finally, the packets are routed within their groups.

LEMMA 4.11. *On an n -node 1D OCPC any h -relation can be routed within $\tilde{O}(h + \log \log n)$ communication steps.*

The following Lemma is also in [32].

LEMMA 4.12.

An arbitrary h -relation can be realized in $\tilde{O}\left(h + \frac{\log n}{\log \log n}\right)$ communication steps on a $\sqrt{n} \times \sqrt{n}$ OCPC.

5. Selection. Given a sequence of n numbers k_1, k_2, \dots, k_n and an $i \leq n$, the problem of selection is to identify the i th smallest of the n numbers. An elegant linear time sequential algorithm is known for selection (see e.g., [10]). A simple linear time randomized algorithm for sequential selection has been given by Floyd and Rivest [6].

Optimal parallel algorithms are also known for selection on various models of computing (see e.g., [26]). Most of the parallel selection algorithms (both deterministic and randomized) make use of the technique of sampling. In this section we describe selection algorithms that have been proposed for optical computers.

5.1. AROB. Pan [23] has given a selection algorithm for the 1D AROB.

LEMMA 5.1. *Given an input of size n , the problem of selection can be solved in an expected time of $O\left(\frac{n}{p} \log n\right)$ cycles on an AROB of size $1 \times p$.*

The worst case run time is $O(n)$ cycles.

Proof. This algorithm picks a random element as the *splitter* element and partitions the input into two. The first part has all the elements less than or equal to the splitter and the second part has the rest of the elements. Then the algorithm decides which part the i th element is in. The irrelevant part is discarded. An appropriate selection is done recursively in the relevant part. It is easy to see that the expected number of remaining elements decreases by a constant factor after every stage of partitioning. This in turn means that there are only an expected $O(\log n)$ stages of partitioning.

Each stage itself can be implemented in $O(1)$ cycles using a prefix computation (c.f. Lemma 3.1) followed by a permutation routing (c.f. Lemma 4.1) each of which takes $O(1)$ cycles. \square

Many $O(1)$ time algorithms are known for sorting on the reconfigurable mesh (e.g., [11], [21], [16]). Rajasekaran and Sahni [30] show that selection from out of n numbers can be done in $\tilde{O}(1)$ cycles on an AROB of size $\sqrt{n} \times \sqrt{n}$. There is a number input at each processor. The basic idea is the following: 1) Pick a random sample S of size $s = o(n)$; 2) Choose two elements ℓ_1 and ℓ_2 from the sample whose ranks in S are $i_n^s - \delta$ and $i_n^s + \delta$ for some appropriate δ . One can show that these elements ‘bracket’ the element to be selected with high probability; 3) Eliminate all the input keys whose values are outside the range $[\ell_1, \ell_2]$; 4) Perform an appropriate selection from out of the remaining keys.

Each processor can independently decide if its key will be in the sample or not. Thus picking the sample can be done in $O(1)$ time. The number of sample keys picked is $\tilde{O}(n^{0.4})$. They are collected and sorted in $O(1)$ cycles. Once ℓ_1 and ℓ_2 are identified, they can be broadcast to the whole mesh in $O(1)$ cycles. After elimination in Step 3, only $\tilde{O}(n^{0.8}\sqrt{\log n})$ keys remain. Steps 1 to 3 are repeated $\tilde{O}(1)$ times so that only $\leq \sqrt{n}$ keys remain. At this time the remaining keys are collected and sorted in $O(1)$ cycles.

LEMMA 5.2. *Selection from out of n elements can be done in $\tilde{O}(1)$ cycles on a $\sqrt{n} \times \sqrt{n}$ AROB.*

5.2. OTIS. Sahni and Wang’s deterministic algorithm on an $n \times n$ SIMD OTIS-Mesh [34] has a run time of $22\sqrt{n} + o(\sqrt{n})$ which is the same as the time their sorting algorithm takes.

The algorithm of Rajasekaran and Sahni [29] for selection on the OTIS-Mesh is randomized and the underlying idea is the same as the one used for the AROB. A random sample is picked to identify two elements that will bracket the element to be selected with high probability. Some input keys that can not possibly be the element to be selected are eliminated. An appropriate selection is done in the set of remaining keys. Though this general idea has been applied over a variety of parallel models, the challenge is in coming up with the best implementation of this idea on the model under concern. In general this may not be a trivial task.

One of the problems on the OTIS-Mesh is to devise an efficient way of collecting the sample keys and sorting them. The sample keys are collected in the ‘center’ of the OTIS-Mesh. The bracketing elements in the sample ℓ_1 and ℓ_2 can be broadcast to the whole OTIS-Mesh in $2\sqrt{n}$ time, where the OTIS-Mesh is of size $n \times n$. It is shown in [29] that sampling collection problem can be solved in $2\sqrt{n} + \tilde{o}(\sqrt{n})$ time. This involves making use of the transpose OTIS connections in an interesting manner.

Another important feature of this algorithm is that the number of sample keys picked is $\tilde{\Theta}(n^{1.47})$. This in turn means that after one stage of elimination, the number of remaining keys will be $\tilde{O}(n^{1.27})$. They show that these many keys can be collected and sorted in the ‘center’ of the OTIS-Mesh in $2\sqrt{n} + \tilde{o}(\sqrt{n})$ time. In order to achieve this some local randomization is applied.

LEMMA 5.3. *Selection from out of n elements can be performed in $6\sqrt{n} + \tilde{o}(\sqrt{n})$ time on an $n \times n$ OTIS-Mesh.*

5.3. OCPC. Note that the input size here is n and hence $\geq n$ common memory cells will be used by any PRAM algorithm. If we desire to apply Lemma 1.1, then at least n OCPC processors will be needed. Thus we could simulate any of the EREW PRAM sorting algorithms on the OCPC in order to perform selection. For example Cole’s parallel merge sort [5] runs in $O(\log n)$ time using n processors.

LEMMA 5.4. *Selection from out of n elements can be performed in $O(\log n)$ time on an n -processor 1D OCPC.*

6. Sorting.

6.1. AROB. Sorting of n numbers on a p -processor parallel comparison tree will need $\Omega\left(\frac{\log n}{\log(1+\frac{p}{n})}\right)$ time [3]. The same lower bound holds on the AROB as well. Therefore, if sorting has to be done in $O(1)$ time, $\Omega(n^{1+\epsilon})$ processors will be needed, for some constant $\epsilon > 0$. Rajasekaran and Sahni [30] have presented such an optimal algorithm.

LEMMA 6.1. *If there is one key per processor in the first row of an $n^\epsilon \times n$ AROB, then sorting of these n keys can be done in $O(1)$ time.*

They employ Leighton’s column sort [14]. A summary of this algorithm follows. Think of the n numbers as forming a matrix M with $r = n^{2/3}$ rows and $s = n^{1/3}$ columns. There are seven steps. 1) Sort the columns of M in increasing order. 2) Transpose M preserving the dimension as $r \times s$. 3) Sort the columns in increasing order. 4) Apply the reverse of Step 2’s permutation to M . 5) Sort columns so that adjacent columns are sorted in reverse order. 6) Apply two steps of the odd-even transposition sort to the rows. 7) Sort each column in increasing order.

Any permutation can be done in $O(1)$ time. Thus Steps 2 and 4 take $O(1)$ time. In order to perform sorting in Steps 1, 3, 5, and 7 they extend the constant time sorting algorithms that have been proposed for the (conventional) reconfigurable meshes.

6.2. OTIS. Sahni and Wang [34] have given a deterministic algorithm for sorting on the SIMD OTIS-Mesh that has a run time of $22\sqrt{n} + o(\sqrt{n})$.

LEMMA 6.2. *Sorting and hence selection on an $n \times n$ SIMD OTIS-Mesh can be performed in $22\sqrt{n} + o(\sqrt{n})$ time.*

An $8\sqrt{n} + \tilde{o}(\sqrt{n})$ algorithm has been given in [29] for sorting on an $n \times n$ OTIS-Mesh. This algorithm uses random sampling just like randomized sorting algorithms available for various other parallel models. The idea is to pick a random sample S with $s = o(n)$ keys, sort the sample, partition the input using the sample keys as *splitter* keys, and to sort the individual parts.

Let X be any set of n keys and let S be a random sample with $|S| = s$. Let the sequence of sorted sample keys be $\ell_1, \ell_2, \dots, \ell_s$. Define $X_1 = \{x \in X | x \leq \ell_1\}$. Let $X_i = \{x \in X | \ell_{i-1} < x \leq \ell_i\}$, for $i = 2, 3, \dots, s$ and $X_{s+1} = \{x \in X | x > \ell_s\}$. It can be shown that the size of each of these X_i 's is $\tilde{O}(\frac{n}{s} \log n)$, for $1 \leq i \leq s + 1$.

The algorithm of [29] picks a random sample of size $\tilde{O}(n^{1.34})$, collects these keys near the ‘center’ of the OTIS-Mesh, and sorts them in $2\sqrt{n} + \tilde{o}(\sqrt{n})$ time. The sample keys are broadcast to the whole mesh in $2\sqrt{n} + \tilde{o}(\sqrt{n})$ time. Each input key, after having looked at the random sample, figures out an approximate destination and goes there. This routing is done using Lemma 4.8 in $4\sqrt{n} + \tilde{o}(\sqrt{n})$. By the time the keys reach their approximate destinations, their exact ranks in the input will be known and also they won't be far from their actual destinations. It takes only an additional $\tilde{o}(\sqrt{n})$ time to finish off routing.

An important feature of this algorithm is that steps that involve the movement of the sample keys is overlapped with the steps that involve the movement of input keys. When edge conflicts arise, priority is given the sample keys. Since the number of sample keys encountered by any input key along any dimension is not large, the additional delays that the input keys suffer is not significant.

LEMMA 6.3. *Sorting on an $n \times n$ OTIS-Mesh takes $8\sqrt{n} + \tilde{o}(\sqrt{n})$ time.*

6.3. OCPC. Lemma 1.1 and Cole's parallel mergesort can be used to derive the following result.

LEMMA 6.4. *Sorting can be accomplished in $O(\log n)$ time on an n -processor OCPC.*

7. Matrix Operations.

7.1. AROB. The results discussed in this section are due to Pavel and Akl [24].

The first algorithm makes use of the fact that prefix sums can be computed in $O(1)$ cycles provided the numbers have $O(\log n)$ bits each (c.f. Lemma 3.2).

LEMMA 7.1. *Two $n \times n$ matrices can be multiplied in $O(r)$ cycles on an $n \times n \log n / r \times n$ AROB, for any $1 \leq r \leq \log n$, provided the numbers have $O(\log n)$ bits each.*

They also consider the problem of multiplying sparse matrices where each row (or column) has at most r nonzero elements. In particular they show:

LEMMA 7.2. *If A is a matrix with at most r nonzero elements per row and B is a matrix with at most r nonzero elements per column, then AB can be computed in $O(\log r)$ cycles on an $n \times n \times r$ AROB, in $O(r \log r)$ cycles on an $n \times n$ AROB, or in $O(1)$ cycles on an $n \times n \log n \times r$ AROB, where $r \leq \frac{n}{\log n}$.*

In [24] an algorithm is also given for Gaussian elimination.

7.2. OTIS. Consider an $n \times n$ OTIS-Mesh. Let A and B be two given matrices of size $n \times n$ each that we are interested in multiplying. Let the first row of A or B be stored in the OTIS-mesh as follows. The first \sqrt{n} elements are in the first row of group 1, the next \sqrt{n} elements are in the first row of group 2, and so on. Similarly store the rest of the rows.

Now one could simulate the standard ‘systolic’ algorithm for matrix multiplication on the conventional Mesh (see e.g., [13]) and get the following Lemma.

LEMMA 7.3. *Two $n \times n$ matrices can be multiplied in $O(n)$ time on an $n \times n$ OTIS-Mesh.*

7.3. OCPC. The trivial algorithm for parallel matrix multiplication on the EREW PRAM can be simulated on the OCPC as follows.

LEMMA 7.4. *Two $n \times n$ matrices can be multiplied in $O(\log n)$ time using n^3 OCPC processors.*

Also sparse matrix multiplication algorithms known for the EREW PRAM can be transported to the OCPC.

8. Conclusions. In this paper we have considered three parallel models based on optical technology. Algorithms that have been proposed on these models for some fundamental problems have been surveyed.

REFERENCES

- [1] R. J. ANDERSON AND G. L. MILLER, *Optical Communication for Pointer Based Algorithms*, Technical Report CRI-88-14, Computer Science Department, University of Southern California, 1988.
- [2] Y. BEN-ASHER, D. PELEG, R. RAMASWAMI, AND A. SCHUSTER, *The Power of Reconfiguration*, *Journal of Parallel and Distributed Computing* (1991), pp. 139–153.
- [3] R. BOPANA, *The Average-Case Parallel Complexity of Sorting*, *Information Processing Letters* **33** (1989), pp. 145–146.
- [4] H. CHERNOFF, *A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations*, *Annals of Mathematical Statistics* **2** (1952), pp. 241–256.

- [5] R. COLE, *Parallel Merge Sort*, SIAM Journal on Computing **17** (1988), pp. 770–785.
- [6] R. W. FLOYD AND R. L. RIVEST, *Expected Time Bounds for Selection*, Communications of the ACM **18(3)** (1975), pp. 165–172.
- [7] M. GERÉB-GRAUS AND T. TSANTILAS, *Efficient Optical Communication in Parallel Computers*, in Proc. ACM Symposium on Parallel Algorithms and Architectures, 1992, pp. 41–48.
- [8] L. GOLDBERG, M. JERRUM, T. LEIGHTON, AND S. RAO, *A Doubly-Logarithmic Communication Algorithm for the Completely Connected Optical Communication Parallel Computer*, in Proc. ACM Symposium on Parallel Algorithms and Architectures, 1993, pp. 300–309.
- [9] W. HENDRICK, O. KIBAR, P. MARCHAND, C. FAN, D. V. 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, September 1995.
- [10] E. HOROWITZ AND S. SAHNI AND S. RAJASEKARAN, *Computer Algorithms*, Computer Science Press, 1998.
- [11] J. JANG AND V. K. PRASANNA, *An Optimal Sorting Algorithm on Reconfigurable Mesh*, in Proc. International Parallel Processing Symposium, 1992, pp. 130–137.
- [12] J. JENQ AND S. SAHNI, *Reconfigurable Mesh Algorithms for Image Shrinking, Expanding, Clustering, and Template Matching*, in Proc. International Parallel Processing Symposium, 1991, pp. 208–215.
- [13] T. LEIGHTON, *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubs*, Morgan-Kaufmann Publishers, 1992.
- [14] T. LEIGHTON, *Tight Bounds on the Complexity of Parallel Sorting*, IEEE Transactions on Computers **C-34(4)** (1985), pp. 344–354.
- [15] R. LIN, S. OLARIU, *Reconfigurable Buses with Shift Switching: Concepts and Applications*, IEEE Transactions on Parallel and Distributed Systems **6(1)** (1995), pp. 93–102.
- [16] R. LIN, S. OLARIU, J. L. SCHWINE, AND J. ZHANG, *Sorting in $O(1)$ Time on a Reconfigurable Mesh of Size $N \times N$* , in Proc. European Workshop on Parallel Computing, 1992, pp. 16–27.
- [17] G. C. MARSDEN, P. J. MARCHAND, P. HARVEY, AND S. C. ESENER, *Optical Transpose Interconnection System Architectures*, Optic Letters **18(3)** (1993), pp. 1083–1085, July 1993.
- [18] R. G. MELHEM, D. M. CHIARULLI, AND S. P. LEVITAN, *Space Multiplexing of Waveguides in Optically Interconnected Multiprocessor Systems*, Computer Journal **32(4)** (1989), pp. 362–369.
- [19] R. MILLER, V. K. PRASANNA-KUMAR, D. REISIS AND Q. F. STOUT, *Meshes with Reconfigurable Buses*, IEEE Transactions on Computers **42** (1993), pp. 678–692.
- [20] D. NASSIMI AND S. SAHNI, *A Self-Routing Benes Network and Parallel Permutation Algorithms*, IEEE Transactions on Computers **C-30(5)** (1981), pp. 332–340.
- [21] M. NIGAM AND S. SAHNI, *Sorting n Numbers on $n \times n$ Reconfigurable Meshes with Buses*, in Proc. International Parallel Processing Symposium, 1993, pp. 174–181.
- [22] S. OLARIU, J. L. SCHWINE AND J. ZHANG, *Integer Problems on Reconfigurable Meshes, with Applications*, in Proc. 1991 Allerton Conference **4** (1991), pp. 821–830.
- [23] Y. PAN, *Order Statistics on Optically Interconnected Multiprocessor Systems*, in Proc. First International Workshop on Massively Parallel Processing Using Optical Interconnections, 1994, pp. 162–169.
- [24] S. PAVEL AND S. G. AKL, *Matrix Operations using Arrays with Reconfigurable Optical Buses*, manuscript, 1995.
- [25] S. RAJASEKARAN, *Meshes with Fixed and Reconfigurable Buses: Packet Rout-*

- ing, *Sorting and Selection*, in Proc. First Annual European Symposium on Algorithms, Springer-Verlag Lecture Notes in Computer Science 726, 1993, pp. 309–320. Also see IEEE Transactions on Computers **45(5)** (1996), pp. 529–539.
- [26] S. RAJASEKARAN, *Sorting and Selection on Interconnection Networks*, in DIMACS Series in Discrete Mathematics and Theoretical Computer Science **21** (1995), pp. 275–296.
 - [27] S. RAJASEKARAN, *k – k Routing, k – k Sorting, and Cut Through Routing on the Mesh*, Journal of Algorithms **19** (1995), pp. 361–382.
 - [28] S. RAJASEKARAN AND J.H. REIF, *Derivation of Randomized Sorting and Selection Algorithms*, in Parallel Algorithm Derivation and Program Transformation, Edited by R. Paige, J.H. Reif, and R. Wachter, Kluwer Academic Publishers, 1993, pp. 187–205.
 - [29] S. RAJASEKARAN AND S. SAHNI, *Randomized Routing, Selection, and Sorting on the OTIS-Mesh*, manuscript, 1997.
 - [30] S. RAJASEKARAN AND S. SAHNI, *Sorting, Selection, and Routing on the Array with Reconfigurable Optical Buses*, IEEE Transactions on Parallel and Distributed Systems **8(11)** (1997).
 - [31] S. RAJASEKARAN AND TH. TSANTILAS, *Optimal Routing Algorithms for Mesh Connected Processor Arrays*, Algorithmica **8** (1992), pp. 21–38.
 - [32] S. RAO AND T. TSANTILAS, *Optical Interprocessor Communication Protocols*, in Proc. Workshop on Massively Parallel Processing Using Optical Interconnections, 1994, pp. 266–274.
 - [33] S. SAHNI AND C. WANG, *BPC Permutations on the OTIS-Mesh Optoelectronic Computer*, in Proc. The Fourth International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'97), 1997, pp. 130–135.
 - [34] S. SAHNI AND C. WANG, *Basic Algorithms on the OTIS-Mesh Optoelectronic Computer*, Manuscript, 1997.
 - [35] R. K. THIRUCHELVAN, J. L. TRAHAN, AND R. VAIDYANATHAN, *On the Power of Segmenting and Fusing Buses*, in Proc. International Parallel Processing Symposium, 1993, pp. 79–83.
 - [36] L. G. VALIANT, *General Purpose Parallel Architectures*, in Handbook of Theoretical Computer Science: Vol. A (J. van Leeuwen, ed.), North Holland, 1990.
 - [37] L. G. VALIANT AND G. J. BREBNER, *Universal Schemes for Parallel Communication*, in Proc. 13th Annual ACM Symposium on Theory of Computing, 1981, pp. 263–277.
 - [38] F. ZANE, P. MARCHAND, R. PATURI, S. ESENER, *Scalable Network Architectures Using the Optical Transpose Interconnection System (OTIS)*, in Proc. Third International Conference on Massively Parallel Processing Using Optical Interconnections (MPPOI'96), 1996, pp. 114–121.