Computing on the Array with Reconfigurable Optical Buses
S. Rajasekaran and S. Sahni
University of Florida, Gainesville

Abstract. In this paper we present efficient algorithms for selection and packet routing on the AROB (Array with Reconfigurable Optical Buses) model.

1 Introduction
An Array with Reconfigurable Optical Buses (AROB) [18] is essentially an $m \times n$ reconfigurable mesh in which the buses are implemented using optical technology. This model has attracted the attention of many researchers in the recent past owing to its promise in superior practical performance.

In a reconfigurable mesh, the switch in each processor can be used to connect together subsets of the four bus segments connected to the processor. (A processor has bus segments that can be connected to its four neighbors.) Reconfigurable meshes that use electronic buses have been studied extensively. Various models such as the RN [3], RMESH [8], PARBUS [10], $Mr$ [19], RMBM [25], and DMBC [24] have been proposed and studied.

Reconfigurable meshes with optical buses have been less extensively studied. In the AROB model of [18, 22], the allowable switch settings of the processors are the same as those in the RN model of [3]. A bus link connects two adjacent processors $x$ and $y$ and has two associated wave guides. One of the wave guides permits an optical signal to travel from $x$ to $y$ and the other permits signal movement from $y$ to $x$. By setting processor switches, bus links are connected together to form disjoint buses. On each bus, we need to specify which orientation of the waveguide on each link of the bus is to be used. The resulting directed graph that represents the bus should be a directed chain. The root of this chain is the bus ‘leader’. The length of a bus is the number of links on the chain representing that bus. The position of any processor on a bus is its distance from the bus leader. The time needed to transmit a message on a bus is referred to as one cycle. A cycle is divided into slots of duration $\tau$ and each slot can carry a different optical signal. $\tau$ is the time needed for an optical pulse to move down one bus link. Pavel and Akl [18] have argued that for reasonable size meshes (say up to $1000 \times 1000$), 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.

To assist a processor in determining which slot to use, each processor has a slot counter. These counters may be started at the beginning of a cycle. The bus leader initiates a light pulse at this time (i.e., it writes a one to the bus). The counter at each processor stops when the light pulse reaches that processor. This special timing mechanism does not require any bus read operation. The terminal counter value is the distance of the processor from the bus leader. Note that because 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. An additional AROB feature that facilitates the development of algorithms is the delay unit at each processor. This permits a processor to introduce a one time slot delay in the light pulses passing through it.

A linear AROB (LAROB) is a $1 \times n$ AROB [18, 22].

In Section 2 we provide some preliminaries related to randomized algorithms and survey known results in the area of AROBs. Sections 3 and 4 are devoted to the problems of selection and packet routing, respectively. In Section 5 we provide our conclusions and list some open problems.

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

2.1 Randomized Algorithms
We say a randomized algorithm uses $\tilde{O}(f(n))$ amount of any resource (like time, space, etc.) if the amount of resource used is no more than $c\alpha f(n)$ with probability $\geq (1-n^{-\alpha})$ for any $\alpha, c$ 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 $\alpha \geq 1$.

Chernoff Bounds [4]. These bounds can be used to closely approximate the tail ends of a binomial distribution.

A Bernoulli trial has two outcomes namely success and failure, the probability of success being $p$. A binomial distribution with parameters $n$ and $p$, denoted as
Let $X$ be a binomial random variable whose distribution is $B(n, p)$. If $m$ is any integer $> np$, then the following are true:

\[
\text{Prob.}[X > m] \leq \left( \frac{np}{m} \right)^m e^{m-np};
\]

\[
\text{Prob.}[X > (1 + \delta)np] \leq e^{-\delta^2 np/3}; \quad \text{and}
\]

\[
\text{Prob.}[X < (1 - \delta)np] \leq e^{-\delta^2 np/2}
\]

for any $0 < \delta < 1$.

### 2.2 Problem Definitions

Given a sequence of numbers, say, $k_1, k_2, \ldots, k_n$, the problem of selection is to identify the $i$th smallest number from out of the $n$ 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 making sure that at most one packet crosses any edge at any time. Packet routing is equivalent to the random access write operation first defined by Nassimi and Sahni [14]. 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 problem of partial permutation routing is the task of routing where at most one packet originates from any node and at most one packet is destined for any node. Any routing problem where at most $h$ packets originate from any node and at most $h$ packets are destined for any node will be called $h-h$ routing or $h$-relations [26].

### 2.3 Previous Results

In [18], the AROB model has been defined. Similar models have been employed before as well (see e.g., [17]). A related model known as the Optical Communication Parallel Computer (OCPC) has also been defined in the literature (see e.g., [1], [6], [26], [7]). In an OCPC any processor can communicate with any other processor in one unit of time, provided there are no conflicts. If more than one processors try to send a message to the same processor, no message reaches the intended destination.

In [18], algorithms for such problems as prefix computation, routing on a linear array, matrix multiplication, etc. have been given for the AROB. On the other hand, [17] considers the problem of selection on a mesh with optical buses. Sorting and routing problems have been investigated in [22].

**Lemma 2.1** Consider an $n$ processor LAROB. If each processor has a bit, then the prefix sums of these bits can be computed in $O(1)$ cycles [18].

The idea behind the above the algorithm is 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. All processors start their counters at the start of the cycle and also set their delay units to introduce a one slot delay in case the processor’s bit is one. A processor’s counter is turned off when the light pulse initiated by processor 1 reaches it. By using the terminal counter value, its data bit and its distance from processor 1, each processor can compute its prefix sum value.

The above algorithm can be extended to show the following [18]:

**Lemma 2.2** The addition of $n \log n$-bit numbers can be performed in $O(1)$ cycles on a log $n \times n$ AROB.

A constant time algorithm for prefix sums on a 2D AROB can be found in [18]. In particular, they show:

**Lemma 2.3** 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.

This algorithm uses the algorithm of [16] to compute the prefix sums of an integer sequence. The maximum bus length employed by this algorithm is $3\sqrt{n}$. Rajasekaran and Sahni [22] reduce this bus length to $\sqrt{n}$.

**Lemma 2.4** In a LAROB of size $n$ any permutation can be routed in $O(1)$ cycles [18].

The time it takes for a packet to move from one processor to the next is assumed to be $\tau$. Consider any permutation to be routed. Let the processors be numbered $1, 2, \ldots, n$ starting from left. There is a time slot assigned to each processor for reading from (and writing into) the bus. Let the reading time slot for processor $i$ be $2i$. Processor 1 creates a ‘time slot’ for each packet that moves one edge per $\tau$ time. The time slots created will be in the order of the processors, i.e., the first time slot is meant for processor 1, time slot 2 is meant for processor 2, and so on. If processor $p$ has a message for processor $q$, $p$ will write this message at
time $t + (p + q)\tau$, where $t$ is the start time. Clearly, this algorithm terminates in 2 cycles (or $2n\tau$ time).

The above lemma can be strengthened as follows [22]:

**Lemma 2.5** Let $L$ be a LAROB of size $n$. 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.

A further extension of the above ideas leads to the following [18]:

**Lemma 2.6** Say there are $k$ elements arbitrarily distributed (at most one per processor) in a two dimensional AROB of size $\sqrt{n} \times \sqrt{n}$. We would like to ‘compact’ them in the first $\lceil \frac{\sqrt{n}}{h} \rceil$ rows. This problem can be solved in $O(1)$ cycles.

The algorithm for the above problem figures out a unique address for each element and then routes the elements using greedy paths. There is no possibility of a collision.

The following lemma pertains to selection on a LAROB and is due to [17]:

**Lemma 2.7** The selection problem (from out of $n$ elements) can be solved in an expected time of $O(\frac{1}{h} \log n)$ cycles on a LAROB of size $p$. The worst case run time is $O(n)$ cycles.

The basic idea behind the above algorithm is to pick a random element as the ‘pivot’ element and partition the input into two around the pivot; Decide which partition the $i$th element is in; Throw away the irrelevant partition and perform an appropriate selection in the relevant partition.

Many $O(1)$ time algorithms are known for sorting on the reconfigurable mesh (e.g., [10],[15]):

**Lemma 2.8** Sorting of $n$ numbers can be performed in $O(1)$ time on a reconfigurable mesh of size $n \times n$.

The same algorithm runs on the AROB preserving the run time.

**Routing on the OCPC.** Several packet routing algorithms for the OCPC model can be found in the literature. Anderson and Miller have shown that a special case of log $n$-relations on an $n$-node OCPC can be routed in $O(\log n)$ time [1]. Also, [26] and [6] have presented efficient algorithms for $h$-relations. An algorithm for arbitrary $h$-relations with a run time of $O(h + \log \log n)$ has been given by Goldberg, Jerrum, Leighton, and Rao [7].

### 2.4 New Results

In this paper we present a selection algorithm that is randomized and can perform selection from out of $n$ elements in $O(1)$ time using an $n$-node LAROB. This algorithm is clearly optimal. In contrast, the selection algorithm of Pan [17] has an expected run time of $O(\log n)$ and a worst case run time of $O(n)$.

We consider several variants of the packet routing problem in this paper. One of the main theorems we prove shows that any $h$-relation can be routed in $O(h)$ time on a LAROB. In contrast, the best known algorithm for the OCPC model has a run time of $O(h + \log \log n)$ [7] and is much more complicated than our algorithm. Clearly, our algorithm is optimal. We also present an algorithm for $h$-relation routing on a $\sqrt{n} \times \sqrt{n}$ AROB with a run time of $O(h + \log \log n)$. On the other hand the best known algorithm for the same problem on the OCPC model has a run time of $O(h + \log \frac{n}{\log \log n})$ [6].

### 3 Selection

Given a sequence of $n$ numbers $k_1, k_2, \ldots, k_n$ 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., [9]). Floyd and Rivest have given a simple linear time randomized algorithm for sequential selection [5].

Optimal parallel algorithms are also known for selection on various models of computing (see e.g., [20]). Most of the parallel selection algorithms (both deterministic and randomized) make use of the technique of sampling.

In this section we show that selection from out of $n$ numbers can be done in $O(1)$ time on an AROB of size $\sqrt{n} \times \sqrt{n}$. The same algorithm runs in $O(1)$ time on an $n$-node LAROB. There is a number input at each processor. The basic idea is the following: 1) Pick a random sample $S$ of size $q = o(n)$; 2) Choose two elements $\ell_1$ and $\ell_2$ from the sample whose ranks in $S$ are $i - \delta$ and $i + \delta$ for some appropriate $\delta$. One can show that these elements ‘bracket’ the element to be selected with high probability; 3) Eliminate all keys whose values are outside the range $[\ell_1, \ell_2]$; 4) Perform an appropriate selection from out of the remaining keys.

**A Sampling Lemma.** Let $Y$ be a sequence of $n$ numbers from a linear order and let $S = \{k_1, k_2, \ldots, k_s\}$ be a random sample from $Y$. Also let $k'_1, k'_2, \ldots, k'_s$ be the sorted order of this sample. If $r_1$ is the rank of $k'_1$ in $Y$, the following lemma provides a high probability confidence interval for $r_1$. (The rank of any element $k$
Lemma 3.1 For every $\alpha$, \[
\left| r_i - \frac{i}{2} \right| > \sqrt{3\alpha} \sqrt{\frac{1}{n} \log n} < n^{-\alpha}.
\]

A proof of the above lemma can be found in [21]. This lemma can be used to analyze many of the selection and sorting algorithms based on random sampling. Our selection algorithm makes use of random sampling.

The following theorem due to [22] will be employed in our algorithm:

**Theorem 3.1** We can sort $n$ numbers in $O(1)$ cycles using an AROB of size $n^2 \times n$, where $\epsilon$ is any constant $> 0$.

Now we are ready to present our selection algorithm. To begin with, the number of alive keys, $N$, is the same as $n$.

**Algorithm Select**

repeat

1. Each alive key decides to include itself in the sample $S$ with probability $\frac{1}{N^{\epsilon}}$. There will be $\Theta(N^{0.4})$ keys in the sample.

2. Concentrate the sample keys in the first row. This is done using lemma 2.6. With high probability only the first row will be nonempty.

3. Sort the numbers in the first row using Theorem 3.1. Processors which do not get a sample key will have a key valued $\infty$. Let $q$ be the number of keys in the sample. Choose keys $\ell_1$ and $\ell_2$ from $S$ with ranks $\left\lfloor \frac{i q}{n} \right\rfloor - d\sqrt{q \log n}$ and $\left\lfloor \frac{i q}{n} \right\rfloor + d\sqrt{q \log n}$, respectively, $d$ being a constant $> \sqrt{3}\alpha$. This takes $O(1)$ time.

4. Eliminate keys that fall outside the range $[\ell_1, \ell_2]$. Count the number, $s$, of surviving keys. It can be shown that this number is $O(N^{0.8} \sqrt{\log n})$.

5. Count the number $del$ of keys deleted that are $< \ell_1$. If the key to be selected is not one of the remaining keys (i.e., if $del > i$ or $i > del + s$), start all over again (i.e., go to step 1 with $N = n$). Set $i = i - del$ and $N = s$.

until $N \leq \sqrt{n}$

6) Concentrate the surviving keys in the first row just like in step 2. Sort the first row and output the $i$th smallest key from out of the remaining keys.

**Theorem 3.2** The above algorithm runs in $\tilde{O}(1)$ time on a $\sqrt{n} \times \sqrt{n}$ AROB.

**Proof.** Step 1 takes $O(1)$ time. The number of keys in the sample $S$ is $B(N, \frac{2}{N^\epsilon})$. Using Chernoff bounds, this number is $\Theta(N^{0.4})$. Step 2 takes $O(1)$ time (c.f. lemma 2.6). Step 3 runs in $O(1)$ time in accordance with Theorem 3.1. Counting in steps 4 and 5 can be performed in time $O(1)$ as well (cf. lemma 2.2). In step 6, concentration and sorting take $O(1)$ time each.

Now it suffices to show that the number of times the repeat loop will be executed is $\tilde{O}(1)$. If $N$ is the number of alive keys at the beginning of any iteration, then the number of alive keys at the end of this iteration is $\tilde{O}(N^{0.8} \sqrt{\log N})$ as per lemma 3.1. This in turn means that the number of alive keys at the end of the first $i$ iterations is $O(n^{0.8}(\log n)^i)$. Thus with high probability there will be only $\leq 4$ iterations of the repeat loop. $\square$

## 4 Packet Routing

Packet routing is a fundamental problem of parallel computing since algorithms for packet routing can be used as mechanisms for interprocessor communication. In this section we present efficient algorithms for packet routing on the AROB.

For the OCPC model several routing algorithms are known: 1) Anderson and Miller have shown that a special case of log $n$-relations on an $n$-node OCPC can be routed in $\tilde{O}(\log n)$ time [1]; 2) Valiant extended their algorithm to show that any $h$-relation can be achieved in time $\tilde{O}(h + \log n)$ [26]; 3) Gereb-Graus and Tsantilas have given a simple algorithm that can route any $h$-relation in time $\tilde{O}(h + \log n \log \log n)$ [6]; 4) A more complicated algorithm with a run time of $\tilde{O}(h + \log \log n)$ has been given by Goldberg, Jerrum, Leighton, and Rao [7].

There is a crucial difference between the OCPC model and the AROB model. On the OCPC model if more than one messages are sent to some processor $\pi$ at the same time, none of them reaches $\pi$. On the other hand, under the same scenario, one of the messages will reach $\pi$ on the AROB model. Also, operations such as prefix sums (limited to integers of certain magnitude) and compaction can be performed in $O(1)$ time on the AROB model and not on the OCPC model.
Realize that a single step of an $n$-node OCPC can be simulated on a LAROB of size $n$ in $O(1)$ cycles. Therefore, all the OCPC packet routing algorithms mentioned above can be used on the LAROB without any change in the asymptotic run times:

**Lemma 4.1** Any $h$-relation can be routed on an $n$-processor LAROB in $\tilde{O}(h + \log \log n)$ cycles.

An interesting question will be if there exists a better algorithm for the LAROB. Consider the simple problem where there are two processors (on an $n$-node machine) that want to send a message to the same processor. If both of them attempt to transmit at the same time, then no message will ever reach the destination on the OCPC. A randomized strategy could break the deadlock. For example, each processor can attempt a transmission with some probability less than one (say $1/2$). Such an algorithm can be seen to terminate in $O(\log n)$ time. The same problem can be solved in three cycles on the LAROB.

In this section we demonstrate the power of LAROB with efficient algorithms for $h$-relations. We also consider the same problem on a two dimensional AROB. In particular, we present the following main results: 1) An $\tilde{O}(h)$ time algorithm for $h$-relations on an $n$-node LAROB; and 2) An $\tilde{O}(h + \log \log n)$ time algorithm for a $\sqrt{n} \times \sqrt{n}$ AROB that can route arbitrary $h$-relations.

### 4.1 Routing on a LAROB

Let $L$ be a LAROB with $n$ processors. We are interested in routing an arbitrary $h$-relation. Here we present an algorithm that runs in time $\tilde{O}(h)$. Notice that a special case where $h = O(1)$ has already been considered in Lemma 2.5.

We look at some special cases of routing before dealing with the general case.

**Problem 1.** In an $n$-node network there are at most $k$ packets at any node. Let $N$ be the total number of packets. The problem is to do a load balancing, i.e., to rearrange the packets such that each node has at most $\lceil \frac{N}{n} \rceil$ packets.

**Lemma 4.2** Problem 1 can be solved in $O(k)$ cycles on an $n$-node LAROB. The same problem can also be solved in $O(k)$ cycles on a $\sqrt{n} \times \sqrt{n}$ AROB. [22]

**Problem 2.** There are at most $\ell$ packets originating from any node of a network $N$. Also, at most $k$ packets are destined for any node. Route the packets.

**Lemma 4.3** Problem 2 can be solved on an $n$-node LAROB in $O(k\ell)$ cycles [22].

Now we state and prove our main theorem:

**Theorem 4.1** Any $h$-relation can be routed in $\tilde{O}(h)$ time on an $n$-node LAROB.

**Proof.** We present a randomized algorithm. The idea is for every processor to randomly choose one of its packets and send it. At any time step, there is some constant probability that a processor will succeed. The algorithm can be thought of as running in stages. At the end of every stage we perform load balancing. 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 $N_1 \leq nh$. We’ll 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, we also prove that, there will be $\tilde{O}(n)$ packets left. They can be routed by load balancing followed by an application of Lemma 2.5. More details follow:

**Algorithm Route**

\[
\begin{align*}
   i &= 1; k_1 = h; N_1 = nh; \\
   \text{repeat} & \\
   & \quad \text{for } j = 1 \text{ to } 1.5k_i, \text{ do} \\
   & \quad \text{Step 1. Each processor } \pi \text{ does the following: It picks one of the remaining packets uniformly at random and sends it in the bus. Thus there is a probability of } \frac{q}{\ell} \text{ that any packet from } \pi \text{ will be sent, } q \text{ being the number of remaining packets. The packet sent may or may not successfully reach its destination (depending on conflicts).} \\
   & \quad \text{Step 2. Reverse the direction of the above routing to inform the processors as to whether or not their packets have been successfully sent.} \\
   & \quad \text{Perform load balancing such that every processor gets very nearly the same number of remaining packets; compute } k_{i+1} \text{ and } N_{i+1}; \\
   & \quad i = i + 1; \\
   & \quad \text{until } N_i \leq cn \text{ (c being a constant).} \\
\end{align*}
\]

**Analysis.** We’ll prove by induction that at the beginning of stage $i + 1$, for almost all the processors, the
number of remaining packets destined for any processor is $\leq \frac{k}{2}$ and also that $k_{i+1}$ is no more than $\frac{k_i}{2}$. The base case is easy.

Assume the hypothesis for all stages up to $i$. We shall prove it for stage $i + 1$. Let $\pi$ be any processor. If at any time during stage $i$ the number (call it $m$) of packets yet to be routed to $\pi$ falls below $\frac{k_i}{2}$, we are fine. Thus assume that $m$ is at most $k_i$ and at least $\frac{k_i}{2}$.

At any time during stage $i$, the probability that at least one of the packets destined for $\pi$ will be sent is $\geq 1 - \frac{1}{\sqrt{k_i}}$. This can be proved as follows: Let the number of packets considered at processor $q$ and destined for $\pi$ be $x_q$, $q = 1, 2, \ldots, n$. Clearly, $\sum_{q=1}^{n} x_q = m$. Probability that one of the $x_q$ packets at $q$ will be sent is $\geq \frac{k_i}{2}$. Therefore, probability that none of these $x_q$ packets will be sent is $\leq (1 - \frac{1}{\sqrt{k_i}})^m \leq e^{-x_q/k_i}$, using the fact that $(1 - \frac{1}{\sqrt{k_i}})^y \leq \frac{1}{2}$ for any $y$. And hence, probability that none of the $m$ packets destined for $\pi$ will be sent is $\leq e^{-(x_1 + x_2 + \ldots + x_n)/k_i} = e^{-m/k_i}$. Thus, probability that at lease one packet destined for $\pi$ will be sent is $\geq 1 - e^{-m/k_i} \geq 1 - \frac{1}{\sqrt{k_i}}$.

The fact that at least one of these $m$ packets has been sent ensures that one packet will successfully reach $\pi$. Call this event (of sending at least one packet destined for $\pi$) a success. Therefore, the expected number of successes in $1.5k_i$ steps of routing is $= 1.5k_i (1 - \frac{1}{\sqrt{k_i}})$. This number is $\geq \frac{k_i}{2}$, with probability $\geq (1 - \frac{1}{\sqrt{k_i}})$ (using Chernoff bounds). Another way of looking at this is that the expected number of processors that have not received $\geq \frac{k_i}{2}$ packets in this stage is $\leq \frac{n}{k_i}$. Expected total number of such packets is $\leq \frac{n}{k_i}$. We could also show that the number of such packets is $\tilde{O}(\frac{n}{k_i})$.

In summary, for almost all the processors, the number of remaining packets destined for any processor at the end of stage $i$ is $\leq \frac{k_i}{2}$, and since the total number of packets that can not be delivered (as expected) is $\tilde{O}(\frac{n}{k_i})$, after load balancing, $k_{i+1}$ will be $\leq \frac{k_i}{2}$.

Since the value of $k_i$ decreases by a factor of 2 in every stage, there will be $\tilde{O}(\log h)$ stages. As we have seen, the number of packets that cannot be routed in stage $i$ is $\tilde{O}(\frac{n}{k_i})$. The total of this over all the stages is $\sum_{i=1}^{d \log h} \frac{n}{(n/2)^{\log h}} = \tilde{O}(n)$.

Thus it follows that after $\tilde{O}(\log h)$ stages, the number of packets remaining to be routed is $\tilde{O}(n)$. They can be routed using a load balancing (in time $O(1)$) followed by an application of lemma 4.3. The additional time needed is $O(h)$.

The amount of time spent in each stage can be computed as the sum of routing time and load balancing time. Routing takes $O(k_i)$ time and load balancing also takes $O(k_i)$ time (c.f. lemma 4.2). Therefore, the total time spent in all the stages is $\sum_{i=1}^{d \log h} O(k_i) = \sum_{i=1}^{d \log h} \tilde{O}(\frac{n}{k_i}) = \tilde{O}(h)$. □

4.2 Routing on a Two Dimensional AROB

Now we consider the problem of routing $h$-relations on a two dimensional AROB of size $\sqrt{n} \times \sqrt{n}$. When it comes to off-line routing (in this, the $h$-relation is known in advance and we may precompute functions that can be used for free when realizing the $h$-relation), an optimal algorithm is immediate from Hall’s theorem (that says that any $h$-relation can be decomposed into $h$ permutations) and the algorithm of Baumslag and Annexstein [2]:

Lemma 4.4 Off-line routing of $h$-relations can be performed in $O(h)$ cycles on a $\sqrt{n} \times \sqrt{n}$ AROB.

Proof.
The off-line permutation routing algorithm of [2] has three phases: **Phase 1:** Perform column permutations; **Phase 2:** Perform row permutations; **Phase 3:** Perform column permutations. Since each row and column of a 2D AROB is a LAROB with $\sqrt{n}$ processors, each of the above phases can be done in $O(1)$ cycles (see lemma 2.4). Hence, if the specific permutation needed for each phase is precomputed (actually, only the phase 1 permutation needs to be precomputed [2]), the permutation can be realized in $O(1)$ cycles. Since each $h$-relation can be decomposed into $h$ permutations, the time needed to realize the $h$-relation is $O(h)$. This does not include the time to decompose the $h$-relation or that needed to precompute the $h$ phase 1 permutations.

The column permutations of phase 1 are computed using a bipartite graph construction and Hall’s theorem for complete matchings. These column permutations ensure that following phase 1 no two packets in the same row have the same destination column. As a result, routing packets to their destination columns can be done using row permutations (phase 2). Following phase 2, all packets are in their destination columns. Hence no two packets in the same column can have the same row as their destination. The column permutations of phase 3 therefore suffice to complete the packet routing. □

An (on-line) algorithm that will realize any $h$-relation in $\tilde{O}(h + \frac{\log n}{\log \log n})$ time has been given in [23] for the OCPC model. It is not clear if the algorithm of
can be extended to get a runtime of $\tilde{O}(h + \log \log n)$ for the 2D OCPC.

We show in this section that an arbitrary $h$-relation can be routed on a $\sqrt{n} \times \sqrt{n}$ AROB in time $\tilde{O}(h + \log \log n)$. Before considering the general case, we look at the case of $h = 1$.

**Lemma 4.5** $O(1)$-relations can be routed in $\tilde{O}(\log \log n)$ cycles on a 2D AROB.

**Proof.** Rao and Tsantilas [23] provide a randomized routing algorithm with this complexity for the 2D OCPC model. This algorithm is a randomized version of the off-line algorithm of lemma 4.4 and can be run on a 2D AROB with no change. $\square$

Now we are ready to prove the main result:

**Theorem 4.2** Any $h$-relation can be routed in $\tilde{O}(h + \log \log n)$ cycles on a $\sqrt{n} \times \sqrt{n}$ AROB.

**Proof.** The algorithm to be used is similar to that of Route given for the LAROB. The changes are:

(a) In step 1, ‘sends it on the bus’ is replaced by:

**Step 1.1.** Each processor selects a random row index. The processors in each column attempt to route their selected packets to the randomly selected rows using the algorithm of lemma 2.4. Since the routes in each column do not necessarily define a permutation, only some of the packets get through.

**Step 1.2.** The packets that get through in step 1.1 are routed along rows to their destination columns using the permutation routing scheme of lemma 2.4. Again, since the destination columns in a row may not form a partial permutation, some packets may not get through.

**Step 1.3.** The destination rows of the packets in each column that got through in step 1.2 form a partial permutation. The partial permutation in each column may be realized using the method of lemma 2.4.

(b) The load balancing step is done using lemma 4.2 as applied to a 2D AROB.

We note that steps 1.1, 1.2 and 1.3 are similar to the three phase algorithm of [27].

The analysis of the algorithm is similar to that of algorithm Route. The value of $k_i$ decreases by a factor of 2 per iteration of the for loop with high probability. Also, the total number of packets that remain following the adapted algorithm Route is $\tilde{O}(n)$. These packets can be balanced at the end (in time $O(h)$) and routed in $\tilde{O}(h + \log \log n)$ cycles (c.f. lemma 4.5). The total number of cycles spent in routing in the adapted algorithm Route is $\tilde{O}(h + \log \log n)$. So the overall number of cycles is $\tilde{O}(h + \log \log n)$. $\square$

## 5 Conclusions

In this paper we have presented efficient algorithms for selection and packet routing on the AROB. Our $h$-relations algorithm for the LAROB is optimal. An interesting open problem is if there exists an $O(h)$ routing algorithm (deterministic or randomized) for the 2D AROB.

## Acknowledgments

Research of Rajasekaran is supported, in part, by an NSF Grant CCR-9596065. Research of Sahni is supported, in part, by the Army Research Office under grant DAA H04-95-1-0111.

## References


