Sorting, Selection and Routing on the Array with Reconfigurable Optical Buses

S. Rajasekaran[1] **and S. Sahni**[2]

Department of CIS, University of Florida

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

1 Introduction

An Array with Reconfigurable Optical Buses (AROB) [20] 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.

A 4×4 reconfigurable mesh is shown in figure 1. The switch in each processor can be used to connect together subsets of the four bus segments connected to the processor. Reconfigurable meshes that use electronic buses have been studied extensively. Various models such as the RN [4], RMESH [10], PARBUS [12], M_r [21], RMBM [26], and DMBC [25] have been proposed and studied.

Reconfigurable meshes with optical buses have been less extensively studied. In the AROB model of [20], the allowable switch settings of the processors are the same as those in the RN model of [4]. These are shown in figure 2. A bus link connects two adjacent

[1]Research of this author is supported, in part, by an NSF Grant CCR-9596065

[2]Research of this author is supported, in part, by the Army Research Office under grant DAA H04-95-1-0111

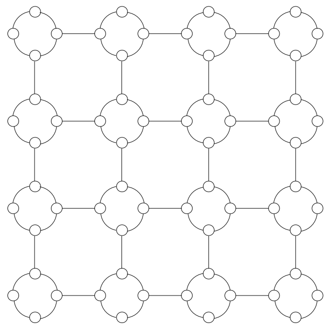

Figure 1: A 4×4 Reconfigurable Mesh

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 τ and each slot can carry a different optical signal. τ is the time needed for an optical pulse to move down one bus link. Pavel and Akl [20] have argued 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.

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

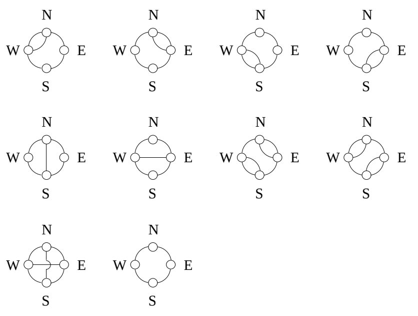

Figure 2: Possible Switch Connections

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 [20].

In Section 2 we provide some preliminaries related to randomized algorithms and survey known results in the area of AROBs. Sections 3, 4, and 5 are devoted to the problems of sorting, selection and packet routing, respectively. In Section 6 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 $\widetilde{O}(f(n))$ 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 α, c being a constant. We could also define $\widetilde{\Theta}(.)$, $\widetilde{o}(.)$, 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 [6]. 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 $B(n, p)$, is the number of successes in n independent Bernoulli trials.

Let X be a binomial random variable whose distribution is $B(n, p)$. If m is any integer $> np$, then the following are true:

$$\begin{aligned} Prob.[X > m] &\leq \left(\frac{np}{m}\right)^m e^{m-np}; \\ Prob.[X > (1+\delta)np] &\leq e^{-\delta^2 np/3}; \text{ and} \\ Prob.[X < (1-\delta)np] &\leq e^{-\delta^2 np/2} \end{aligned}$$

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 sorting is to rearrange them in nondecreasing order.

The problem of selection is to identify the ith 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 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 [16]. 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* [27].

2.3 Previous Results and Extensions

In [20], the AROB model has been defined. Similar models have been employed before as well (see e.g., [19]). A related model known as the *Optical Communication Parallel Computer (OCPC)* has also been defined in the literature (see e.g., [1], [8], [27], [9]). 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 [20], 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, [19] considers the problem of selection on a mesh with optical buses.

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 [20].*

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 [20]:

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 [20]. 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 [18] to compute the prefix sums of an integer sequence. The maximum bus length employed by this algorithm is $3\sqrt{n}$.

We next show that one can reduce the bus length for the prefix sums problem to $\sqrt{n}$ (which is a factor of 3 improvement over [20]'s algorithm). Furthermore, our algorithm is simpler.

Lemma 2.4 *Prefix sums of bits on a $\sqrt{n} \times \sqrt{n}$ AROB can be computed in $O(1)$ cycles, keeping the maximum bus length as $\sqrt{n}$.*

Proof. We proceed as in [20]. Say we are interested in computing the prefix sums in row major order. Using the algorithm of lemma 2.1, we can compute the prefix sums along each row in $O(1)$ time. At the end of this step, each processor in the last column has the sum of all 1's in the corresponding row. Now the original problem of computing prefix sums reduces to computing prefix sums of $\sqrt{n}$ numbers where each number is at most $\sqrt{n}$ (i.e. each number is an $O(\log n)$-bit number). Next we describe how to compute the prefix sums of $\sqrt{n}$ $O(\log n)$-bit numbers in $O(1)$ time on a $\sqrt{n} \times \sqrt{n}$ AROB. (In [20], this is done using the algorithm of [18].)

Step 1: Group the numbers with $\log n$ numbers in each group. Allocating a subarray of size $\log n \times \log n$, compute the sum of numbers in each group. For each group, first reduce the problem to that of adding $\log n$ $O(\log \log n)$-bit numbers (by summing the corresponding bits using lemma 2.1) and then apply lemma 2.2. Maximum bus length is $O(\log n)$.

Step 2: Compute prefix sums of the $\frac{\sqrt{n}}{\log n}$ group sums. This can be done using lemma 2.2 as follows. For each prefix sum (there are $\frac{\sqrt{n}}{\log n}$ of them) allocate a subarray of size $\log n \times \frac{\sqrt{n}}{\log n}$. Broadcast the appropriate numbers to each subarray and within each subarray add the numbers using lemma 2.2. Here also, the maximum bus length is $\sqrt{n}$.

Step 3: Compute prefix sums local to each group of $\log n$ numbers. This step is similar to step 2. Each group will get a subarray of size $\log n \times \log^2 n$. Maximum bus length is $O(\log n)$.

Clearly, the above algorithm runs in $O(1)$ time. $\Box$

Lemma 2.5 *In a LAROB of size n any permutation can be routed in $O(1)$ cycles [20].*

The time it takes for a packet to move from one processor to the next is assumed to be τ. 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 τ 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:

Lemma 2.6 *Let $\mathcal{L}$ 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.*

Proof. Let c (resp. d) be an upper bound on the number of packets destined for (originating from) any node. It suffices to consider the case $d = 1$, since that algorithm can be repeated d times to take care of the general case. There will be $2c$ runs of the algorithm given above for lemma 2.5. The above algorithm has the property that the message read by any processor q is the last message written in time slot q. If more than one processors wrote in time slot q, they can determine if their message was read by q or not by reversing the routing process. This way, in every two executions of the above routing algorithm, a processor receives one message destined for it. $\square$

A further extension of the above ideas leads to the following [20]:

Lemma 2.7 *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 $\lceil \frac{k}{\sqrt{n}} \rceil$ 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 [19]:

Lemma 2.8 *The selection problem (from out of n elements) can be solved in an expected time of $O(\frac{n}{p} \log n)$ 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 ith 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., [12],[17]):

Lemma 2.9 *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 $\widetilde{O}(\log n)$ time [1]. Also, [27] and [8] have presented efficient algorithms for h-relations. An algorithm for arbitrary h-relations with a run time of $\widetilde{O}(h + \log\log n)$ has been given by Goldberg, Jerrum, Leighton, and Rao [9].

2.4 New Results

In this paper we present a sorting algorithm that can sort n general keys in $O(1)$ time on an AROB of size $n^\epsilon \times n$ for any constant $\epsilon > 0$. We also point out that this algorithm is optimal. We also present a sorting algorithm that can sort n k-bit numbers in $O(k)$ time on a LAROB of size n. Notice that such an algorithm cannot be devised even on the CRCW PRAM.

Our selection algorithm is randomized and can perform selection from out of n elements in $\widetilde{O}(1)$ time using an n-node LAROB. This algorithm is clearly optimal. In contrast, the selection algorithm of Pan [19] 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 $\widetilde{O}(h)$ time on a LAROB. In contrast, the best known algorithm for the OCPC model has a run time of $\widetilde{O}(h + \log\log n)$ [9] 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 $\widetilde{O}(h + \log\log n)$. On the other hand the best known algorithm for the same problem on the OCPC model has a run time of $\widetilde{O}(h + \frac{\log n}{\log\log n})$ [8]. Our deterministic algorithm for h-relations runs in time $O(h \log n)$ on a $\sqrt{n} \times \sqrt{n}$ AROB as well as on an n-node LAROB.

3 Sorting on the AROB

In this section we present optimal algorithms for sorting both general and integer keys on the two dimensional AROB. The general sorting algorithm sorts n numbers in an AROB of size $n^\epsilon \times n$ for any constant $\epsilon > 0$. The run time is $O(1)$ cycles and hence the algorithm is optimal in view of the following lower bound:

Lemma 3.1 *Sorting of n numbers needs $\Omega\left(\frac{\log n}{\log(1+\frac{p}{n})}\right)$ time using p parallel comparison tree processors [5].*

This lower bound implies that if sorting of n numbers has to be done in $O(1)$ time, then there must be $\Omega(n^{1+\epsilon})$ processors, for some constant $\epsilon > 0$. In [5], the lower bound has been proven for the parallel comparison tree model of Valiant. Since a parallel comparison tree can simulate an AROB step per step, the same lower bound applies to the AROB as well.

Our integer sorting algorithm runs on a LAROB of size n and can sort n k-bit numbers in $O(k)$ time. Notice that even on the CRCW PRAM model, such an algorithm cannot be devised in view of the lower bound result of Beame and Hastad [3].

3.1 General Sorting

Let $k_1, k_2, \ldots, k_n$ be the n given numbers. Think of these numbers of as forming a matrix M with $r = n^{2/3}$ rows and $s = n^{1/3}$ columns. We employ the column sort algorithm of Leighton [14]. There are 7 steps in the algorithm:

Algorithm **Sort**

1. Sort the columns of M in increasing order;
2. Transpose the matrix preserving the dimension as $r \times s$. In particular, pick the elements in column major order and fill the rows in row major order;
3. Sort each column in increasing order;
4. Rearrange the numbers applying the reverse of the permutation employed in step 2;
5. Sort the columns in a way that adjacent columns are sorted in reverse order;
6. Apply two steps of odd-even transposition sort to the rows. Specifically, in the first step perform a comparison-exchange between processors $2i+1$ and $2i$, for $i = 0, 1, \ldots$ and in the second step perform a comparison-exchange between processors $2i$ and $2i+1$, for $i = 1, 2, \ldots$; and
7. Sort each column in increasing order. At the end of this step, it can be shown that, the numbers will be sorted in column major order.

Implementation on the AROB. The n given numbers will be stored in the first row of the $n^\epsilon \times n$ AROB one key per processor. At any given time each key will know which row and which column of the matrix M it belongs to. Whenever we need to sort the columns, we will make sure that the numbers belonging to the same column will be found in successive processors.

On a LAROB of size n note that any permutation can be performed in $O(1)$ time. This means that steps 2 and 4 can be performed in $O(1)$ time. Step 6 can be performed in $O(1)$ time as well as follows: Rearrange the numbers such that elements in the same row are in successive processors and apply two steps of the odd-even transposition sort. After this, move the keys to where they came from.

Next we describe how we implement steps 1, 3, 5, and 7. We first assume that we have an AROB of size $n^{2/3} \times n$. Later we will indicate how to reduce the size to $n^{\epsilon} \times n$ for any $\epsilon > 0$.

Partition the AROB into $n^{1/3}$ parts each of size $n^{2/3} \times n^{2/3}$, each part corresponding to a column of M. Rearrange the n given numbers such that the first column of M is in the first $n^{2/3}$ processors of row 1; the second column is in the next $n^{2/3}$ processors of the first row; and so on. Now sort the numbers in each part (i.e., each column of M) using lemma 2.9. This can be done in $O(1)$ time. This implies that steps 1, 3, 5, and 7 of column-sort can be performed in $O(1)$ time. Therefore it follows that n numbers can be sorted in $O(1)$ time on an AROB of size $n^{2/3} \times n$.

We can reduce the size of the AROB to $n^{4/9} \times n$ as follows: We still use Leighton's sort with $r = n^{2/3}$ and $s = n^{1/3}$. In steps 1, 3, 5, and 7, each part of $n^{2/3}$ numbers will be sorted using an AROB of size $n^{4/9} \times n^{2/3}$. This is done using the AROB algorithm above.

In a similar way we can reduce the size to $n^{8/27} \times n$, $n^{16/81} \times n$, and so on. Thus we get the following theorem:

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

3.2 Integer Sorting

In this subsection we present an algorithm for sorting n k-bit numbers in $O(1)$ time on a LAROB with n processors. This algorithm makes use of the idea of *radix sorting* and lemmas

2.1 and 2.5.

Radix Sorting. The idea is captured by the following lemma:

Lemma 3.2 *If n numbers in the range $[0, R]$ can be stable sorted using P processors in time T, then we can also stable sort n numbers in the range $[0, R^c]$ in $O(T)$ time using P processors, c being any constant.*

A sorting algorithm is said to be *stable* if equal keys remain in the same relative order in the output as they were in the input.

The algorithm proceeds as follows: There are k stages. In stage i we sort the numbers with respect to their ith LSBs. To be more specific, in the first stage we sort the numbers with respect to their LSBs. In the next stage, we apply a sort in the resultant sequence with respect to the next LSBs, and so on. Thus there will be k stages in the algorithm.

Each stage can be performed in $O(1)$ time as follows: Notice that each stage is nothing but sorting n 1-bit numbers. Perform a prefix sums operation for the zeros in the input. Do the same for the 1's in the input. Using these two sums, each processor can determine the position of its data in the sorted list. Permute the data to complete the sort for the stage. Since the prefix sums as well as the permutation take $O(1)$ time each, each stage takes $O(1)$ time as well (c.f. lemmas 2.1 and 2.5.)

Thus we have proven the following:

Theorem 3.2 *A LAROB with n processing elements can sort n k-bit numbers in $O(k)$ cycles.*

Realize that no PRAM algorithm can achieve the above performance, since the lower bound theorem of [3] implies that sorting of n bits on the CRCW PRAM will need $\Omega(\frac{\log n}{\log\log n})$ time, given only a polynomial number of processors.

4 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 ith smallest of the n numbers. An elegant linear time sequential algorithm is known for selection (see e.g., [11]). Floyd and Rivest have given a simple linear time randomized algorithm for sequential selection [7].

Optimal parallel algorithms are also known for selection on various models of computing (see e.g., [22]). 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 $\widetilde{O}(1)$ time 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 $q = o(n)$; 2) Choose two elements ℓ_1 and ℓ_2 from the sample whose ranks in S are $i\frac{q}{n} - \delta$ and $i\frac{q}{n} + \delta$ for some appropriate δ. 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_i is the rank of k'_i in Y, the following lemma provides a high probability confidence interval for r_i. (The rank of any element k in Y is one plus the number of elements $< k$ in Y.)

Lemma 4.1 *For every α, Prob.* $\left(|r_i - i\frac{n}{s}| > \sqrt{3\alpha}\frac{n}{\sqrt{s}}\sqrt{\log n}\right) < n^{-\alpha}$.

A proof of the above lemma can be found in [23]. 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.

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 $\frac{1}{N^{0.6}}$. There will be $\widetilde{\Theta}(N^{0.4})$ keys in the sample.

    2. Concentrate the sample keys in the first row. This is done using lemma 2.7. 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 $\left\lceil \frac{iq}{n} \right\rceil - d\sqrt{q \log n}$ and $\left\lceil \frac{iq}{n} \right\rceil + d\sqrt{q \log n}$, 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 $\widetilde{O}(N^{0.8}\sqrt{\log n})$.

    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 $del \geq i$ 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 ith smallest key from out of the remaining keys.

Theorem 4.1 *The above algorithm runs in $\widetilde{O}(1)$ 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 $B(N, \frac{1}{N^{0.6}})$. Using Chernoff bounds, this number is $\widetilde{\Theta}(N^{0.4})$. Step 2 takes $O(1)$ time (c.f. lemma 2.7). 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 $\widetilde{O}(1)$. 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 $\widetilde{O}(N^{0.8}\sqrt{\log N})$ as per lemma 4.1. This in turn means that the number of alive keys at the end of the first i iterations is $\widetilde{O}(n^{0.8^i}(\log n)^3)$. Thus with high probability there will be only ≤ 4 iterations of the *repeat* loop. $\square$

5 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 $\widetilde{O}(\log n)$ time [1]; 2) Valiant extended their algorithm to show that any h-relation can be achieved in time $\widetilde{O}(h + \log n)$ [27]; 3) Geréb-Graus and Tsantilas have given a simple algorithm that can route any h-relation in time $\widetilde{O}(h + \log n \log\log n)$ [8]; 4) A more complicated algorithm with a run time of $\widetilde{O}(h + \log\log n)$ has been given by Goldberg, Jerrum, Leighton, and Rao [9].

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 π at the same time, none of them reaches π. On the other hand, under the same scenario, one of the messages will reach π 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 5.1 *Any h-relation can be routed on an n-processor LAROB in $\widetilde{O}(h + \log\log n)$ 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 $\widetilde{O}(\log n)$ 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 $\widetilde{O}(h)$ time algorithm for h-relations on an n-node LAROB; 2) An $\widetilde{O}(h + \log\log n)$ time algorithm for a $\sqrt{n} \times \sqrt{n}$ AROB that can route arbitrary h-relations; and 3) An $O(h \log n)$ time deterministic algorithm for any h-relations on a LAROB as well as a 2D AROB.

5.1 Routing on a LAROB

Let $\mathcal{L}$ be a LAROB with n processors. We are interested in routing an arbitrary h-relation. Here we present an algorithm that runs in time $\widetilde{O}(h)$. Notice that a special case where $h = O(1)$ has already been considered in lemma 2.6.

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 5.2 *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.*

Proof. We give the proof for a LAROB. The same proof can be extended to a 2D AROB also. Let the processors order their packets from 1 to k. Perform a prefix sums computation for the first packets of all the processors (using lemma 2.1) and compute a unique address for each such packet. Route the first packets. Prefix takes $O(1)$ cycles and so does the routing. Likewise process the second packets, the third packets, and so on. One should make sure, for example, that if q is the number of first packets, then, the second packets will be routed to nodes starting from $q + 1$. Total time is clearly $O(k)$. $\Box$

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

Lemma 5.3 *Problem 2 can be solved on an n-node LAROB in $O(k\ell)$ cycles.*

Proof. The packets are routed in cycles. In any cycle, a processor will choose one of its remaining packets (if any) and try to send it. It may not be successful in one attempt. If it succeeds, it takes up the next packet; otherwise it will try to send the same packet. A packet will not reach its destination only if there is a conflict. Thus a packet can meet with failure in at most $k - 1$ cycles. This in turn means that every processor will be able to transmit all of its ℓ packets in ℓk cycles or less. $\Box$

Now we state and prove our main theorem:

Theorem 5.1 *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 $\widetilde{O}(\log h)$ stages of routing. After $\widetilde{O}(\log h)$ stages of routing, we also prove that, there will be $\widetilde{O}(n)$ packets left. They can be routed by load balancing followed by an application of lemma 2.6. More details follow:

Algorithm **Route**

```
$i = 1$; $k_i = h$; $N_i = nh$;
*repeat*

    *for* $j = 1$ *to* $1.5k_i$ *do*

        **Step 1**. Each processor $\pi$ 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 $\frac{1}{q}$ that any packet from $\pi$ will be sent, $q$ being
        the number of remaining packets. The packet sent may or may not
        successfully reach its destination (depending on conflicts).

        **Step 2**. Reverse the direction of the above routing to inform the
        processors as to whether or not their packets have been successfully
        sent.

    Perform load balancing such that every processor gets very nearly the same num-
    ber of remaining packets; compute $k_{i+1}$ and $N_{i+1}$; $i = i + 1$;
```

until $N_i \leq cn$ (c being a constant).

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 $\leq \frac{h}{2^i}$ and also that k_{i+1} is no more than $\frac{h}{2^i}$. The base case is easy.

Assume the hypothesis for all stages up to i. We shall prove it for stage $i+1$. Let π be any processor. If at any time during stage i the number (call it m) of packets yet to be routed to π 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 π will be sent is $\geq 1 - \frac{1}{\sqrt{e}}$. This can be proved as follows: Let the number of packets left at processor q and destined for π 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 $\geq \frac{x_q}{k_i}$. Therefore, probability that none of these x_q packets will be sent is $\leq (1 - \frac{x_q}{k_i}) \leq e^{-x_q/k_i}$, using the fact that $(1 - \frac{1}{y})^y \leq \frac{1}{e}$ for any y. And hence, probability that none of the m packets destined for π 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 π will be sent is $\geq 1 - e^{-m/k_i} \geq 1 - \frac{1}{\sqrt{e}}$.

The fact that at least one of these m packets has been sent ensures that one packet will successfully reach π. Call this event (of sending at least one packet destined for π) a *success.*

Therefore, the expected number of successes in $1.5k_i$ steps of routing is $= 1.5k_i(1 - \frac{1}{\sqrt{e}})$. This number is $\geq \frac{k_i}{2}$ with probability $\geq (1 - \frac{1}{k_i^4})$ (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 $\leq \frac{n}{k_i^4}$. Expected total number of such packets is $\leq \frac{n}{k_i^3}$. We could also show that the number of such packets is $\widetilde{O}(\frac{n}{k_i^3})$.

In summary, for almost all the processors, the number of remaining packets destined for any processor at the end of stage i is $\leq \frac{k_i}{2}$. Since the total number of packets that can not be delivered (as expected) is $\widetilde{O}(\frac{n}{k_i^3})$, 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 $\widetilde{O}(\log h)$

stages. As we have seen, the number of packets that cannot be routed in stage i is $\widetilde{O}(\frac{n}{k_i^3})$. The total of this over all the stages is $\sum_{i=1}^{d\log h} \widetilde{O}(\frac{n}{k_i^3}) = \sum_{i=1}^{d\log h} \widetilde{O}\left(\frac{n}{(h/2^i)^3}\right) = \widetilde{O}(n)$.

Thus it follows that after $\widetilde{O}(\log h)$ stages, the number of packets remaining to be routed is $\widetilde{O}(n)$. They can be routed using a load balancing (in time $O(1)$) followed by an application of lemma 5.3. 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 5.2). Therefore, the total time spent in all the stages is $\sum_{i=1}^{d\log h} O(k_i) = \sum_{i=1}^{d\log h} \widetilde{O}(\frac{h}{2^i}) = \widetilde{O}(h)$. $\square$

5.2 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 5.4 *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 2.5). 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. $\Box$

An (on-line) algorithm that will realize any h-relation in $\widetilde{O}(h + \frac{\log n}{\log\log n})$ time has been given in [24] for the OCPC model. It is not clear if the algorithm of [9] can be extended to get a run time of $\widetilde{O}(h + \log\log n)$ 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 $\widetilde{O}(h + \log\log n)$. Before considering the general case, we look at the case of $h = 1$.

Lemma 5.5 *$O(1)$-relations can be routed in $\widetilde{O}(\log\log n)$ cycles on a 2D AROB.*

Proof. Rao and Tsantilas [24] 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 5.4 and can be run on a 2D AROB with no change. $\Box$

Now we are ready to prove the main result:

Theorem 5.2 *Any h-relation can be routed in $\widetilde{O}(h+\log\log n)$ 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 2.5. 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 2.5. 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 2.5.

(b) The load balancing step is done using lemma 5.2 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 [28].

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 $\widetilde{O}(n)$. These packets can be balanced at the end (in time $O(h)$) and routed in $\widetilde{O}(h + \log\log n)$ cycles (c.f. lemma 5.5). The total number of cycles spent in routing in the adapted algorithm **Route** is $\widetilde{O}(h + \log\log n)$. So the overall number of cycles is $\widetilde{O}(h + \log\log n)$. □

Deterministic Routing. We can also perform deterministic routing in an efficient manner on the AROB:

Lemma 5.6 *Any partial permutation can be routed in $O(\log n)$ time on a $\sqrt{n} \times \sqrt{n}$ AROB.*

Proof. Sort the packets into ascending order of destinations. For this, the destinations are mapped into a single number using the row major mapping scheme. The sorted packets are in processors $1, 2, \ldots$ (in row major order). This sort can be accomplished in $O(\log n)$ time using a binary radix sort and two applications of lemma 5.2 to accomplish the sort on each bit (notice that the load balancing scheme of lemma 5.2 is equivalent to a stable sort of bits). Following the sort, no two packets in the same column have the same row as their destination (as there are $\sqrt{n} - 1$ packets between them). So, we may use lemma 2.5 to route packets in each column to their destination rows. Following this, no two packets in the same row have the same column as their destination. So lemma 2.5 may be used again to route packets in the same row to their destination columns. The work done following the sort takes $O(1)$ cycles. So, the overall number of cycles is $O(\log n)$. □

An extension of the above result can also be proven:

Lemma 5.7 *Any h-relation can be routed in $O(h \log n)$ cycles on a 2D AROB of size $\sqrt{n} \times \sqrt{n}$.*

Proof. Sort the packets into nondescending order of destination. Following the sort, the packets are in the first few processors (in row major order), h packets to a processor. This is accomplished using a binary radix sort on the row major index of the packet's destination processor.

When sorting on bit k of this index, we first concentrate the packets with bit k equal to 0, h packets to a processor and then concentrate those with bit k equal to 1. The process for each bit value is similar. Consider the case of packets with bit k equal to 0. Call these packets *selected packets.* A processor may have up to h selected packets.

The selected packets in each processor are combined to form a 'superpacket' of size at most h. The superpackets are compacted into processors $1, 2, \ldots$ (in row major order) using the 2D compaction algorithm of [20]. Since the superpacket size is $O(h)$, this takes $O(h)$ time. The superpackets are now decomposed into the original packets. The original packets are to be further compacted so that we have h packets to a processor. Each packet in a processor is assigned a level number corresponding to its order in the processor. Level numbers are in the range 1 to h.

Prefix sums for the level i packets, $1 \leq i \leq h$ are computed using the 2D prefix sum algorithm given in section 2. The rank $r(i,j)$ of a level i packet in processor j is $\sum_{k=1}^{h} ps(k, j-1) + (i-1)$, where $ps(k, j-1)$ is the prefix sum of the level k packet in processor $j-1$. The processor $P(i,j)$ to which this packet is to be routed is $\lfloor r(i,j)/h \rfloor$. Furthermore, this packet will be the $round(i,j) = r(i,j) \text{mod } h + 1$-th packet in this processor. Since the number of packets in each row is at most $h\sqrt{n}$, no two packets (i,j) and (k,l), where j and l are processors in the same row, have $column(P(i,j)) = column(P(k,l))$ and $(row(P(i,j)) \neq row(P(k,l))$ or $round(i,j) = round(k,l))$. As a result, the compaction may be completed as below:

Step 1: Perform h rounds of row permutation routing on each row. In round k, packets (i,j) with $round(i,j) = k$ are routed to the processor in column $column(P(i,j))$.

Step 2: Perform h rounds of column permutation routing. In round k, packets (i,j) with $round(i,j) = k$ are routed to the processor in row $row(P(i,j))$.

The radix sort described above takes $O(h \log n)$ time. To complete the h-relation we perform h rounds of column and row permutations. In round i, the level i packets in each column are first routed to the correct row using a column permutation. There can be no collision as for a collision the number of packets destined to the same row needs to be $> h\sqrt{n}$. Next, the level i packets are routed to the correct column using row permutations. Again, collisions are not possible. □

6 Conclusions

In this paper we have presented efficient algorithms for sorting, selection and packet routing on the AROB. We have considered both integer sorting and general sorting problems.

Our general sorting and selection algorithms are optimal. So is the h-relations algorithm for the LAROB. An interesting open problem is if there exists an $O(h)$ routing algorithm (deterministic or randomized) for the 2D AROB. Also, can our integer sorting algorithm be improved?

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] M. Baumslag and F. Annexstein, A Unified Framework for Off-line Permutation Routing in Parallel Networks, Mathematical Systems Theory, 24, 1991, pp. 233-251.

[3] P. Beame and J. Hastad, Optimal Bounds for Decision Problems on the CRCW PRAM, Journal of the ACM, 36(3), 1989, pp. 643-670.

[4] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, The Power of Reconfiguration, Journal of Parallel and Distributed Computing, 1991, pp. 139-153.

[5] R. Bopanna, A Lower Bound for Sorting on the Parallel Comparison Tree, Information Processing Letters, 1989.

[6] 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.

[7] R.W. Floyd, and R.L. Rivest, Expected Time Bounds for Selection, Communications of the ACM, Vol. 18, No.3, 1975, pp. 165-172.

[8] M. Geréb-Graus and T. Tsantilas, Efficient Optical Communication in Parallel Computers, Symposium on Parallel Algorithms and Architectures, 1992, pp. 41-48.

[9] L. Goldberg, M. Jerrum, T. Leighton, and S. Rao, A Doubly-Logarithmic Communication Algorithm for the Completely Connected Optical Communication Parallel Computer, Proc. Symposium on Parallel Algorithms and Architectures, 1993.

[10] E. Hao, P.D. McKenzie and Q.F. Stout, Selection on the Reconfigurable Mesh, Proc. Frontiers of Massively Parallel Computation, 1992.

[11] E. Horowitz and S. Sahni, *Fundamentals of Computer Algorithms*, Computer Science Press, 1978.

[12] J. Jang and V.K. Prasanna, An Optimal Sorting Algorithm on Reconfigurable Mesh, Proc. International Parallel Processing Symposium, 1992, pp. 130-137.

[13] J. Jenq and S. Sahni, Reconfigurable Mesh Algorithms for Image Shrinking, Expanding, Clustering, and Template Matching, Proc. International Parallel Processing Symposium, 1991, pp. 208-215.

[14] T. Leighton, Tight Bounds on the Complexity of Parallel Sorting, IEEE Transactions on Computers, C-34(4), 1985, pp. 344-354.

[15] R. Miller, V.K. Prasanna-Kumar, D. Reisis and Q.F. Stout, Meshes with Reconfigurable Buses, in Proc. 5th MIT Conference on Advanced Research in VLSI, 1988, pp. 163-178.

[16] 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.

[17] M. Nigam and S. Sahni, Sorting n Numbers on $n \times n$ Reconfigurable Meshes with Buses, Proc. International Parallel Processing Symposium, 1993, pp. 174-181.

[18] S. Olariu, J.L. Schwing and J. Zhang, Integer Problems on Reconfigurable Meshes, with Applications, Proc. 1991 Allerton Conference, 4, 1991, pp. 821-830.

[19] Y. Pan, Order Statistics on Optically Interconnected Multiprocessor Systems, Proc. First International Workshop on Massively Parallel Processing Using Optical Interconnections, 1994, pp. 162-169.

[20] S. Pavel and S.G. Akl, Matrix Operations using Arrays with Reconfigurable Optical Buses, manuscript, 1995.

[21] S. Rajasekaran, Meshes with Fixed and Reconfigurable Buses: Packet Routing, Sorting and Selection, Proc. First Annual European Symposium on Algorithms, Springer-Verlag Lecture Notes in Computer Science 726, 1993, pp. 309-320.

[22] S. Rajasekaran, Sorting and Selection on Interconnection Networks, to appear in Proc. *DIMACS Workshop on Interconnection Networks and Mapping and Scheduling Parallel Computation*, 1995.

[23] 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.

[24] S. Rao and T. Tsantilas, Optical Interprocessor Communication Protocols, Proc. Workshop on Massively Parallel Processing Using Optical Interconnections, 1994, pp. 266-274.

[25] S. Sahni, Data Manipulation on the Distributed Memory Bus Computer, to appear in Parallel Processing Letters, 1995.

[26] R.K. Thiruchelvan, J.L. Trahan, and R. Vaidyanathan, Sorting on Reconfigurable Multiple Bus Machines, Proc. International Parallel Processing Symposium, 1994.

[27] L.G. Valiant, General Purpose Parallel Architectures, in Handbook of Theoretical Computer Science: Vol. A (J. van Leeuwen, ed.), North Holland, 1990.

[28] L.G. Valiant and G.J. Brebner, Universal Schemes for Parallel Communication, Proc. 13th Annual ACM Symposium on Theory of Computing, 1981, pp. 263-277.