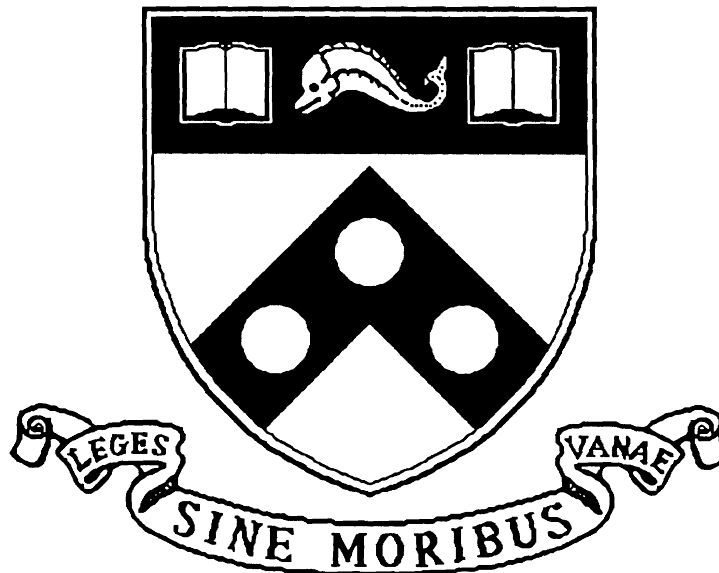
Randomized Routing and Sorting On The Reconfigurable Mesh

MS-CIS-92-36
GRASP LAB 314

Sanguthevar Rajasekaran
Theordore McKendall



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

May 1992

Permutation Routing and Sorting on the Reconfigurable Mesh

Sanguthevar Rajasekaran
Theodore McKendall

Department of Computer and Information Science
Univ. of Pennsylvania, Philadelphia, PA 19104.

Abstract In this paper we demonstrate the power of reconfiguration by presenting efficient randomized algorithms for both packet routing and sorting on a reconfigurable mesh connected computer (referred to simply as the mesh from hereon). The run times of these algorithms are better than the best achievable time bounds on a conventional mesh.

In particular, we show that permutation routing problem can be solved on a linear array of size n in $\frac{3}{4}n$ steps, whereas $n-1$ is the best possible run time without reconfiguration. We also show that permutation routing on an $n \times n$ reconfigurable mesh can be done in time $n + o(n)$ using a randomized algorithm or in time $1.25n + o(n)$ deterministically. In contrast, $2n-2$ is the diameter of a conventional mesh and hence routing and sorting will need at least $2n-2$ steps on a conventional mesh. In addition we show that the problem of sorting can be solved in randomized time $n + o(n)$. The time bounds of our randomized algorithms hold with high probability. The bisection lower bound for both sorting and routing on the mesh is $\frac{n}{2}$, and hence our algorithms have nearly optimal time bounds.

1 Introduction

A number of optimal algorithms have been proposed in the recent past for various computational problems on the reconfigurable mesh [1, 9, 10, 18, 19, 11]. In particular, constant time algorithms have been given for routing and sorting [1, 19, 11]. Even in the most powerful CRCW PRAM, we know that sorting takes $\Omega(\frac{\log n}{\log\log n})$ time given only a polynomial

number of processors. Thus the reconfigurable network seems to a be an attractive model of computing.

Past works on routing and sorting have concentrated on the case when the number of packets (or keys) is much smaller than the number of processors. Wang, Chen, and Lin [19] have presented an $O(1)$ time sorting algorithm that makes use of n^3 processors where n is the number of keys. For the same time bound, the processor bound has been reduced to n^2 by [1, 11]. The later algorithm is the best possible under some weak assumptions.

An interesting question is if it helps to have the feature of reconfiguration for problems where the number of processors is the same as the number of packets (or keys). In this paper we answer this question in the affirmative. In particular, we show that permutation routing can be completed in $\frac{3}{4}n$ routing steps on an n-node linear array. We also establish that both routing and sorting of n^2 keys can be performed in $n + o(n) + O(\frac{n}{q})$ steps on an $n \times n$ reconfigurable mesh, the queue size being $O(q)$, with very high probability. Thus this time bound will be $n + o(n)$ for instance if we pick $q = \log n$. In addition, we show how to perform routing deterministically in time $1.25n + o(n) + O(\frac{n}{q})$, corresponding to a queue size of $O(q)$. We also point out that $\frac{n}{2}$ is the bisection lower bound for routing and sorting. Therefore our algorithms are nearly optimal. In contrast, one needs at least $2n - 2$ steps for both routing and sorting on the conventional mesh since $2n - 2$ is the diameter.

Optimal algorithms have been discovered for packet routing and sorting on the conventional mesh. For instance Kunde's algorithm [5] for sorting takes $2.5n + o(n)$ steps, and Kaklamanis & Krizanc's algorithm [2] for sorting is randomized and runs in $2n + o(n)$ steps. Several optimal packet routing algorithms also exist in the literature [16, 8, 5, 13].

2 Some Preliminaries

2.1 Problem Statement

Packet routing is an important problem in parallel computing because efficient algorithms for packet routing ensure fast inter-processor communication. They also lead to efficient emulation of ideal models like PRAMs on fixed connection machines. A single step of inter-processor communication in a fixed connection network can be thought of as the following task (also called *packet routing*): each node in the network has a packet of information that has to be sent to some other node. The task is to send all the packets to their correct destinations as quickly as possible such that at the most one packet passes over any connection at any time.

A special case of the routing problem is called the *partial permutation routing.* In partial permutation routing, each node is the origin of at the most one packet and each node is the destination of no more than one packet. A packet routing algorithm is judged by its *run time*, i.e., the time taken by the last packet to reach its destination, and its *queue length*, which is defined as the maximum number of packets any node will have to store during routing.

2.2 Model Definition

In this paper we are concerned with packet routing and sorting algorithms for mesh connected computers, which are becoming increasingly popular owing to their special properties. A mesh is an $n \times n$ square grid with a processing element at each grid point. Every processor is connected to all its (four or less) neighbors via bidirectional connections. We assume the MIMD model where each processor can communicate with all its neighbors in one unit of time. This model has been widely used in current research.

In addition, the processors are connected to a reconfigurable broadcast bus. At any given time, the broadcast bus can be partitioned into subbuses. Each subbus connects a collection of successive processors. One of the processors in this collection can choose to broadcast a message which is assumed to be readable in one unit of time by all the other processors in this collection. For instance, in an $n \times n$ mesh, the different columns (or different rows) can form subbuses. Even within a column (or row) there could be many subbuses, and so on. It is up to the algorithm designer to decide what configuration of the bus should be used at each time unit. To be consistent with the MIMD model, we assume that each processor has two switches (as shown in Figure 1), one for connecting the column bus and the other for connecting the row bus. This implies for example that in one time unit independent broadcasting can be done along the rows as well as the columns.

2.3 Chernoff Bounds

Let $X = B(n,p)$ stand for the number of heads in n independent flips of a coin, the probability of a head in a single flip being p. The following three facts (known as Chernoff bounds) are now folklore:

$$\begin{aligned} \text{Prob.}[X \geq m] &\leq \left(\frac{np}{m}\right)^m e^{m-np}, \\ \text{Prob.}[X \geq (1+\epsilon)np] &\leq exp(-\epsilon^2 np/2), \text{and} \\ \text{Prob.}[X \leq (1-\epsilon)np] &\leq exp(-\epsilon^2 np/3). \end{aligned}$$

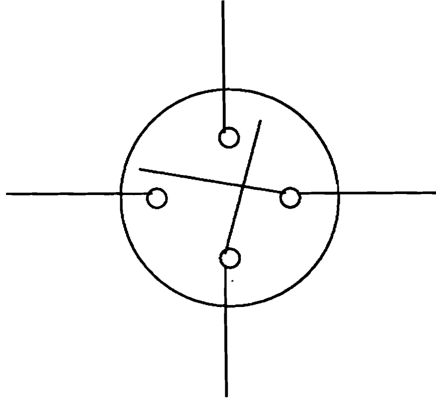


Figure 1: Independent Row and Column Switches

By *high probability*, we mean a probability of $\geq (1 - n^{-\alpha})$ for any constant $\alpha \geq 1$.

2.4 Organization of this Paper

The rest of this paper is organized as follows. In section 3 we show that permutation routing on a linear array can be accomplished within $\frac{3}{4}n$ steps on the reconfigurable model. We know that on the conventional linear array permutation routing needs at least $n - 1$ steps in the worst case. In section 3 we also identify a generic routing problem on a linear array and provide a solution, which will prove helpful in analyzing the mesh routing algorithms. In sections 4 and 5 we present our routing and sorting algorithms respectively. In section 6 we provide some concluding remarks.

3 Linear Array Routing

3.1 A $(3/4)n$ Step Routing Algorithm

In this section we will describe a $\frac{3}{4}n$ time algorithm for performing permutation routing on a linear array of n processors. Let each processor have zero or one packets of information it wishes to send to another processor in the array. Also let each processor be the destination of zero or one such packets. The goal is to send each packet to its destination within $\frac{3}{4}n$ time steps.

Partition the array into four equal sized regions A, B, C, and D, where region A is the collection of the first $\frac{n}{4}$ processors, region B is the next $\frac{n}{4}$ processors, and so on (see Figure 2).

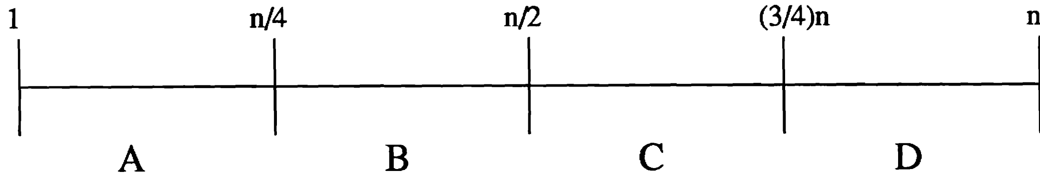


Figure 2: The Four Regions of a Linear Array

Also, let A_B denote the set of packets originating from region A with a destination in region B, A_C the set originating from A and destined for C, and so forth. We then have the following relations:

$$|A_A| + |A_B| + |A_C| + |A_D| \leq n/4,$$
$$|B_A| + |B_B| + |B_C| + |B_D| \leq n/4,$$

etc., and

$$|A_A| + |B_A| + |C_A| + |D_A| \leq n/4,$$
$$|A_B| + |B_B| + |C_B| + |D_B| \leq n/4,$$

etc.

Finally, let processors $(n/2-1)$ and $(n/2)$ each have three incremental counters b_1, b_2, b_3 and c_1, c_2, c_3 respectively, all of which are initially set to zero.

The algorithm proceeds in three phases. Phase I involves normal routing, and phases II and III involve broadcasting using the reconfigurable properties of the array. The following description gives the algorithm for all packets destined for processors in regions A and B, the other two regions being similar.

Phase I Normal routing occurs for $n/2$ time steps. During this phase, the processors use their standard neighbor connections to send packets towards their destinations.

When processor $(n/2-1)$ receives a packet it does the following:

- If the packet is a C_A packet, increment b_1 and attach b_1 onto the packet;
- If the packet is a D_B packet, increment b_2 and attach b_2 onto the packet;
- If the packet is a D_A packet, increment b_3 and attach b_3 onto the packet;
- In any case forward the packet (if needed).

At the end of this phase, any packet which originated from region A or B with a destination in region A or B will have reached it. In addition, any packet originating from region C with a destination in B will have reached it. Thus A_A, B_A, A_B, B_B, and C_B have all been properly routed.

Also note that region A now contains all of C_A, and region B now contains all of D_A and D_B, with each processor containing at most one packet which has not yet reached its destination. The counters b_1, b_2, and b_3 contain the numbers of packets in C_A, D_B, and D_A respectively.

Phase II

Step one: The processors of the array configure themselves into two broadcast busses, with all processors in regions A and B comprising one bus, and those of regions C and D the other. Processor $(n/2-1)$ broadcasts the maximum of b_1 and b_2 across the first bus.

Step two: The processors of the array configure themselves into four broadcast busses, each consisting of all of the processors of each of the four regions A, B, C, and D. At step i, $1 \leq i \leq \max\{b_1, b_2\}$, each processor j in region B does the following: If the packet at j is a D_B packet, and the rank attached onto it is i, then broadcast the packet onto the bus. Otherwise, read from the bus, and if there is a packet there which is destined for j, then store that packet. The region A processors act similarly with C_A packets.

Since this goes on for $\max\{b_1, b_2\}$ steps, by the end of this phase all packets in D_B and C_A would have been properly routed. All that remain are the D_A packets which currently rest in region B.

Phase III The processors of the array reconfigure themselves into two broadcast busses, as before, with regions A and B composing one, and C and D the other. At step i, $1 \leq i \leq b_3$, each processor j in region B does the following: If the packet at j is a D_A packet, and the rank attached onto it is i, then broadcast the packet onto the bus. Each processor k in region A does the following: read from the bus and if there is a packet there which is destined for k, then store that packet.

Since this goes on for b_3 steps, by the end of this phase all packets in D_A have been properly routed. This completes the routing of all packets destined for regions A and B.

Time Analysis: Phase I takes n/2 steps. Phase II takes $\max\{D_B, C_A\}$ steps. Phase III takes D_A steps. Since it must be that $D_A + D_B \leq n/4$ and $D_A + C_A \leq n/4$, we have that the total run time of the algorithm is $n/2 + \max\{D_B, C_A\} + D_A \leq \frac{3}{4}n$.

And so we have the following:

Theorem 3.1 *Permutation packet routing on the reconfigurable linear array can be done in* $\frac{3}{4}n$ *steps.*

3.2 A Generic Problem

The problem we consider now is this: $\mathcal{L}$ is a linear array with n nodes. There are a total of m packets in $\mathcal{L}$ whose origins and destinations could be arbitrary. Route the packets.

Theorem 3.2 *The above problem can be solved in time $m + O(\log n)$.*

Proof. The idea behind the proof is the fact that one could compute *prefix sums* in $O(\log n)$ time on an $n-$node linear array. (Given a sequence of n numbers, say, $k_1, k_2, \ldots, k_n$ the problem of *prefix sums computation* is to calculate $k_1, k_1 + k_2, \ldots, k_1 + k_2 + \ldots + k_n$).

Let k_i be the number of packets in processor i, for $i = 1, 2, \ldots, n$. We could compute the prefix sums of $k_1, k_2, \ldots, k_n$ as follows: Partition the array into two, the first part consisting of the first $\lceil \frac{n}{2} \rceil$ processors and the second part consisting of the remaining processors. 1) Recursively compute the prefix sums of the two parts; and 2) Processor $\lceil \frac{n}{2} \rceil$ broadcasts the sum of the first $\lceil \frac{n}{2} \rceil$ numbers to the whole array, so that the processors in the second part can update their sums. Clearly, this algorithm runs in time $O(\log n)$. Denote the prefix sums as $k'_1, k'_2, \ldots, k'_n$.

The above prefix sums dictate the schedule for each processor. In particular, processor 1 broadcasts its packets from time step 1 until step k'_1; processor 2 broadcasts its packets starting from time step $k'_1 + 1$ until step k'_2; and so on. Thus the total time taken by the algorithm is $m + O(\log n)$. □

4 Packet Routing on the Mesh

In this section we show how to perform permutation routing of n^2 elements on an $n \times n$ reconfigurable mesh in time $n + o(n) + O(\frac{n}{q})$, the queue size being $O(q)$. This time bound holds with high probability (abbreviated from hereon as 'w.h.p.'). $\frac{n}{2}$ is the bisection lower bound for this problem. This can be readily seen by looking at the following permutation: Exchange the $\frac{n^2}{2}$ packets in the left half of the mesh with the packets in the right half. Since these packets can cross over to the other half only via the nodes in the middle column, the lower bound follows. The same lower bound holds even in the conventional mesh [6]. We also present a deterministic algorithm whose run time is $1.25n + O(\log n) + O(\frac{n}{q})$ with a queue size of $O(q)$.

The randomized algorithm to be presented resembles the algorithm of [16]. We first describe a $2n + o(n) + O(\frac{n}{q})$ time algorithm and then show how to reduce the run time of

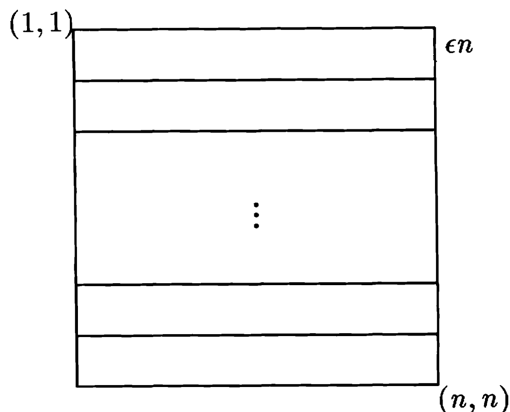


Figure 3: Partitioning of the Mesh into Slices

this algorithm to $n + o(n) + O(\frac{n}{q})$. The mesh is partitioned into horizontal slices of ϵn rows each where $\epsilon = \frac{1}{q}$ (for any $1 \leq q \leq n$) as shown in Figure 3.

The algorithm has three phases. In phase I a packet at processor (i, j), destined for processor (k, l), is routed along column j to (r, j), a processor chosen at random in the same column and slice as (i, j). In phase II the packet is sent to (r, l) along row r, and finally in phase III it is routed to its destination along column l. These three phases are assumed to be disjoint, i.e., a packet can start its phase II only after all the packets have completed their phase I, and so on.

We employ algorithm A of section 3.2 for routing in each phase.

Analysis To analyze each phase we make use of Theorem 3.2.

Phase I. Here $m = \epsilon n$ and hence phase I can be completed in $\epsilon n + O(\log n)$ steps.

Phase II. Consider an arbitrary node (i, k) in an arbitrary row i. The number of packets in this node at the end of phase I is $B(\epsilon n, \frac{1}{\epsilon n})$. The total number of nodes in this row i is $B(n\epsilon n, \frac{1}{\epsilon n})$. Using Chernoff bounds, this number is no more than $n + \sqrt{n \log n}$ w.h.p. Thus we could use Theorem 3.2 with $m = n + o(n)$. The time needed for phase II is then $n + O(\sqrt{n \log n})$ w.h.p.

Phase III. The number of packets that can be found in any column at the beginning of the third phase is clearly n (since we have a permutation routing problem). Thus applying Theorem 3.2 with $m = n$, we infer that phase III can be completed in $n + O(\log n)$ time.

Put together, the above algorithm runs in time $(2 + \epsilon)n + O(\sqrt{n \log n})$. Next we show how to reduce this run time by a factor of nearly two.

4.1 Reducing the Run Time Further

We can reduce the number of steps taken by the above algorithm by making the following modifications. Initially, each processor flips an unbiased two sided coin and colors its packet red or black depending on the result. The mesh is partitioned into both vertical and horizontal slices of ϵn columns and rows respectively.

In phase I, all the red packets choose a random node in the same column and horizontal slice as their origin and go there along the column of origin. Also in phase I, the black packets choose a random node in the same row and vertical slice as their origin and go there along the row of origin. During phase II, all red packets are routed along rows till they reach their column destination, while black packets are routed along columns till they reach their row destination. In phase III, red packets are routed along columns to their destinations, while black packets are routed along rows. This idea of coloring the packets has been used before (see e.g. [6]).

Theorem 4.1 *The above algorithm terminates in time* $n + \frac{\epsilon n}{2} + O(\sqrt{n \log n})$ *with high probability.*

Proof. The run time reduces to half because, as a result of the coloring, the number of packets that will use any row (or column) during any of the three phases now decreases nearly by a factor of two w.h.p. For instance the number of packets that will perform their phase III along any column(or row) is $B(n, \frac{1}{2})$. (Consider the packets whose destination is some column j. They could have been colored white or black with equal probability). This number is no more than $\frac{n}{2} + O(\sqrt{n \log n})$ w.h.p. (as inferred from an application of the Chernoff bounds).

Similarly, we could show that the traffic along any row (or column) is no more than $\frac{n}{2} + O(\sqrt{n \log n})$ in phase II w.h.p. Therefore phases II and III take no more than $\frac{n}{2} + O(\sqrt{n \log n})$ steps each, and phase I takes no more than $\frac{\epsilon n}{2} + O(\sqrt{n \log n})$ steps (cf. Theorem 3.2). Summing up, we conclude that the run time of the above algorithm is no more than $n + \frac{\epsilon n}{2} + O(\sqrt{n \log n})$ w.h.p. □

Queue Size Analysis

The queue size of the above algorithm in any phase is seen to be no more than the queue size at the beginning or at the end of the phase. For example at the end of phase I, the number of packets that will end up in any node is upper bounded by $B(\epsilon n, \frac{1}{\epsilon n})$. Using Chernoff bounds (equation 2), this number is $O(\log n)$ w.h.p. At the end of phase II, consider any column

slice. In the worst case, this slice can have $\frac{n}{2} + O(\sqrt{n \log n})$ packets w.h.p. Because of the randomization done in phase I, these packets will be uniformly distributed among all the ϵn nodes in the slice. This implies that the expected number of packets in any node at the end of phase II is nearly $\frac{1}{\epsilon}$. Using Chernoff bounds, the queue size at the end of phase II is $O(\frac{1}{\epsilon})$ w.h.p. (provided $\frac{1}{\epsilon}$ is $\Omega(\frac{\log n}{\log \log n})$. This is then the queue size of the whole algorithm.

Obtaining Constant Size Queues: The expected queue size of the above algorithm is nearly $\frac{1}{\epsilon}$ as was mentioned before. We can also show that $O(\frac{1}{\epsilon})$ is the queue size with high probability if $\frac{1}{\epsilon}$ is $\Omega(\frac{\log n}{\log \log n})$. But if we desire constant size queues, we have to choose ϵ to be constant fraction, in which case we could only prove an expected constant size queues (and not w.h.p.). However we could modify the above algorithm slightly to obtain high probability constant queues. This technique is due to [16]. The idea is based on the fact that the total queue size of any collection of $\log n$ successive nodes in the mesh is $O(\log n)$ w.h.p. (given that ϵ is a constant fraction). We partition the nodes in the mesh into groups of $\log n$ successive nodes each (along the rows as well as columns). At any time in the routing, packets in a group are locally distributed so as to ensure constant queue size. For instance if more than a constant number of packets want to end up in a specific node (at the end of phase I or phase II), the extra packets will be sent to other nodes in the group.

4.2 Deterministic Packet Routing

In this section we present a deterministic algorithm for permutation routing whose run time is $1.25n + O(\log n) + O(\frac{n}{q})$, the queue size being $O(q)$ (for any $1 \leq q \leq n$). We first show how to obtain a $1.5n + O(\log n) + O(\frac{n}{q})$ time algorithm. Later we will describe the modifications needed to improve the time bound.

The idea of our algorithms is to employ the 'sort and route' paradigm of Kunde [4] together with the coloring schemes proposed in [6]. Partition the mesh into submeshes of size $\frac{n}{q} \times \frac{n}{q}$, for any $1 \leq q \leq n$. In phase I sort the submeshes in time $O(\frac{n}{q})$ in column major order according to column destinations of packets. In phase II, a packet at (i, j) whose destination is (k, l) traverses along row i up to column l, and in phase III it traverses along column l up to row k. This simple algorithm can be shown to have a run time of $2n + O(\frac{n}{q})$, the queue size being $O(q)$ [4].

The above algorithm is 'uniaxial', i.e., it uses either the column edges or the row edges at any given time. One could utilize the full capacity of the MIMD mesh by sending some packets orthogonal to the directions suggested by the above algorithm (similar to what we did in section 4.1). For instance at the beginning we could color each packet (call its origin

node as (i,j)) as either red or black depending on whether $(i+j) \bmod 2$ is either 0 or 1. Route the red packets using the above algorithm. Route the black packets orthogonal to the red packets. That is, in phase I we sort the black packets in row major order according to their row destinations. In phase II, a black packet from (i,j) whose destination is (k,l) traverses along column j up to row k and in phase III it traverses along row k up to its destination.

This algorithm takes $O(\frac{n}{q})$ time in phase I. In phase II, the number of packets traversing along any row or column is exactly $\frac{n}{2}$ and hence phase II can be completed in time $\frac{n}{2}+O(\log n)$ (according to Theorem 3.2). Phase III can be completed in n steps or less (even without employing any broadcasts).

The run time of deterministic routing can further be improved to $1.25n+O(\log n)+O(\frac{n}{q})$ using the coloring scheme of Kunde and Tensi (cf. Theorem 19 in [6]). Thus we have the following

Theorem 4.2 *Permutation routing can be completed in* $1.25n+O(\log n)+O(\frac{n}{q})$ *steps on a reconfigurable mesh with a queue size of* $O(q)$.

5 Randomized Sorting

We show here that sorting of n^2 elements can be accomplished on an $n \times n$ reconfigurable mesh in $n+o(n)+O(\frac{n}{q})$ steps w.h.p., the queue size being $O(q)$ (for any $1 \le q \le n$).) Many optimal algorithms have been proposed in the literature for sorting on the conventional mesh (see e.g. [7]). Recently a $2n+o(n)$ step randomized algorithm has been discovered for sorting [2]. But $2n-2$ is a lower bound for sorting on the conventional mesh. Thus our algorithms demonstrate that a reconfigurable mesh is strictly more powerful than a conventional mesh even when the problem size and the processor size match. Our sorting algorithm makes use of random sampling and the randomized routing algorithm given in section 4. The indexing scheme assumed is the blockwise snake-like row major indexing (which is the same as the scheme assumed in [3, 5, 12, 2]). More details follow.

Summary. Random sampling has played a vital role in the design of parallel algorithms for comparison problems (including sorting and selection). Reischuk's [17] sorting algorithm is a good example. Given n keys, the idea is to: 1) randomly sample n^{ϵ} (for some constant $\epsilon < 1$) keys, 2) sort this sample (using any nonoptimal algorithm), 3)partition the input using the sorted sample as splitter keys, and 4) to sort each part separately in parallel. Similar ideas have been used in many other works as well (see e.g., [2, 3, 12]).

Let $X = k_1, k_2, \ldots, k_n$ be a given sequence of n keys and let $S = \{k'_1, k'_2, \ldots, k'_s\}$ be a random sample of s keys (in sorted order) picked from X. X is partitioned into $(s+1)$ parts defined as follows. $X_1 = \{\ell \in X \ : \ \ell \leq k'_1\}$, $X_j = \{\ell \in X \ : \ k'_{j-1} < \ell \leq k'_j\}$ for $2 \leq j \leq s$, and $X_{s+1} = \{\ell \in X \ : \ \ell > k'_s\}$. The following lemma [17, 15] probabilistically bounds the size of each of these subsets, and will prove helpful to our algorithm. (We say a function $f(n)$ is $\widetilde{O}(g(n))$ if $f(n) \leq c\alpha g(n)$ for all sufficiently large n with probability $\geq (1 - n^{-\alpha})$ for any α and some constant c).

Lemma 5.1 *The cardinality of each X_j $(1 \leq j \leq (s+1))$ is $\widetilde{O}(\frac{n}{s} \log n)$.*

Next we describe our algorithm. The mesh is partitioned into blocks of size $n^{3/4} \times n^{3/4}$. We could name the blocks with integers in the range $[1, n^{1/2}]$. This naming is done according to snake-like row major indexing, i.e., the topmost blocks in the mesh are numbered $1, 2, \ldots, n^{1/4}$ from left to right, the $n^{1/4}$ blocks immediately below are numbered $n^{1/4} + 1, n^{1/4} + 2, \ldots, 2n^{1/4}$ from right to left, and so on.

A random sample of size s (nearly equal to $n^{2/3}$) is chosen and broadcast to the whole mesh, such that each block stores a copy of all the splitter keys. We compute the partial ranks of the sample keys in each block after sorting the block. Then we perform a prefix sum operation on these partial ranks so as to obtain the global ranks of the sample keys. Let $k'_1, k'_2, \ldots, k'_s$ be the sorted order of the sample. Next we route each key to a destination that is close to its actual destination. If this key has a value that falls in between k'_i and k'_{i+1}, it is sent to a random node in block $\frac{i}{s} n^{1/2}$. The keys in each block together with the sample keys are now sorted. Using the global ranks of the sample keys we determine the rank of each key in the mesh and finally route the packets to their actual destinations.

Algorithm Sorting

Step 1: Each key includes itself as a sample key in S with probability $\frac{1}{n^{4/3}}$. The number of sample keys can be seen to be $O(n^{2/3})$ w.h.p.

Step 2: Partition the mesh into blocks of size $n^{3/4} \times n^{3/4}$. Sort each block to group and count the sample keys in this block. The number of sample keys in each block can be seen (using Chernoff bounds) to be $O(n^{1/6})$ w.h.p. Now broadcast the sample keys (using a scheme similar to the algorithm in section 3.2), so that each block will have a copy of all the sample keys of S. (Realize that it takes $O(1)$ time to broadcast a single key to the whole mesh.)

In this step sorting takes $O(n^{3/4})$ time, and the broadcast takes $O(n^{2/3})$ time.

Step 3: Now again sort each block of size $n^{3/4} \times n^{3/4}$, this time also including all the sample keys obtained in Step 2. This sorting takes $O(n^{3/4})$ time. As a by-product of this sorting we have computed the partial ranks of the sample keys in each block.

Step 4: Compute the global rank of each splitter key and broadcast this information to the whole mesh. This is done by summing up the partial ranks computed in Step 3 for each sample key. For a single key, the sum can be obtained clearly in time $O(\log n)$. Therefore this step will take a total of $O(n^{2/3} \log n)$.

Step 5: Now route each packet to a node which is close to its actual destination. In particular, we send each packet to a random node in an appropriate block depending on between which two splitter keys this key falls in (see the summary above). Using lemma 5.1, it is easy to see that the actual destination of the key can be at the most one block away from the block to where it is routed in this step w.h.p. We use the algorithm of section 4 for this routing.

Step 6: Sort the keys in each block together with the sample keys, and compute the global ranks of all the keys in the mesh. This can be done in $O(n^{3/4})$ time w.h.p.

Step 7: Finally route the packets to their actual destinations. Since each packet can be at the most one block away from its destination, this routing can be done in $O(n^{3/4})$ time w.h.p. [16].

The correctness of the above algorithm is quite clear. As a consequence of the above algorithm we have the following

Theorem 5.1 *Sorting can be performed in time $n + o(n) + O(n/q)$ w.h.p. with a queue size of $O(q)$ (for any $1 \leq q \leq n$).*

Note: The above algorithm can be considered as a reduction of sorting to permutation routing. In particular, this algorithm establishes that sorting time on a reconfigurable mesh is at the most $o(n)$ more than the time needed for packet routing.

6 Conclusions

In this paper we have addressed the problems of routing and sorting on a reconfigurable mesh when the problem size and the processor bound are the same. Even in this case we

have demonstrated that a reconfigurable mesh is more powerful than a conventional mesh model. A number of open problems remain: 1) Is $\frac{3}{4}n$ the best possible run time for routing on a linear array?; 2) Can one perform sorting and/or routing in time better than $n + o(n)$? (the lower bound is only $\frac{n}{2}$);

References

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

[2] Kaklamanis, C., and Krizanc, D., 'Optimal Sorting on Mesh Connected Processor Arrays,' to be presented in the ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, 1992.

[3] Kaklamanis, C., Krizanc, D., Narayanan, L., and Tsantilas, Th., 'Randomized Sorting and Selection on Mesh-Connected Processor Arrays,' in Proc. 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, July 1991.

[4] Kunde, M., 'Routing and Sorting on Mesh-Connected Arrays,' in Proc. 3rd Aegean Workshop on Computing. Springer-Verlag Lecture Notes in Computer Science # 319, 1988, pp. 423-433.

[5] Kunde, M., 'Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound,' IEEE Symposium on Foundations of Computer Science, 1991, pp. 141-150

[6] Kunde, M., and Tensi, T. '$(k-k)$ Routing on Multidimensional Mesh-Connected Arrays,' Journal of Parallel and Distributed Computing 11, 1991, pp. 146-155.

[7] Leighton, F.T., *Introduction to Parallel Algorithms and Architectures: Meshes, Trees, Hypercubes*, Morgan-Kaufmann Publishers, San Jose, CA, 1992.

[8] Leighton, T., Makedon, F., and Tollis, I.G. 'A $2n-2$ Step Algorithm for Routing in an $n \times n$ Array With Constant Size Queues,' Proc. ACM Symposium on Parallel Algorithms and Architectures, 1990.

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

[10] Miller, R., Prasanna-Kumar, V.K., Reisis, D., and Stout, Q.F., 'Image Computations on Reconfigurable VLSI Arrays,' in Proc. Conference on Vision and Pattern Recognition, 1988, pp. 925-930.

[11] Nakano, K., Peleg, D., and Schuster, A., 'Constant-time Sorting on a Reconfigurable Mesh,' Manuscript, 1992.

[12] Rajasekaran, S., 'k-k Routing, k-k Sorting, and Cut Through Routing on the Mesh,' Technical Report, Department of CIS, University of Pennsylvania, Philadelphia, PA 19104, October 1991.

[13] Rajasekaran, S., 'Randomized Algorithms for Packet Routing on the Mesh,' to appear in *Advances in Parallel Algorithms*, Blackwell Scientific Publications, 1992.

[14] Rajasekaran, S., and Raghavachari, M., 'Optimal Randomized Algorithms for Multi-packet and Cut Through Routing on the Mesh,' in Proc. IEEE Symposium on Parallel and Distributed Processing, Dallas, Texas, Dec. 1991.

[15] Rajasekaran, S., and Reif, J.H., 'Derivation of Randomized Sorting and Selection Algorithms,' Technical Report, Aiken Computing Lab., Harvard University, 1984.

[16] Rajasekaran, S., and Tsantilas, Th., 'Optimal Routing Algorithms for Mesh Connected Processor Arrays,' To appear in Algorithmica, 1992.

[17] Reischuk, R., 'Probabilistic Parallel Algorithms for Sorting and Selection,' SIAM Journal of Computing, 14(2), 1985, pp. 396-411.

[18] Schuster, A., 'Dynamic Reconfiguring Networks for Parallel Computers: Algorithms and Complexity Bounds,' Ph.D. Thesis, Computer Science Department, Technion-Israel Institute of Technology, August 1991.

[19] Wang, B-F., Chen, G-H., and Lin, F-C, 'Constant Time Sorting on a Processor Array with a Reconfigurable Bus System,' Information Processing Letters, 34(4), April 1990.