



University of Pennsylvania
ScholarlyCommons

Technical Reports (CIS)

Department of Computer & Information Science

July 1992

Mesh Connected Computers With Multiple Fixed Buses: Packet Routing, Sorting and Selection

Sanguthevar Rajasekaran
University of Pennsylvania

Follow this and additional works at: https://repository.upenn.edu/cis_reports

Recommended Citation

Sanguthevar Rajasekaran, "Mesh Connected Computers With Multiple Fixed Buses: Packet Routing, Sorting and Selection", . July 1992.

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-56.

This paper is posted at ScholarlyCommons. https://repository.upenn.edu/cis_reports/498
For more information, please contact repository@pobox.upenn.edu.

Mesh Connected Computers With Multiple Fixed Buses: Packet Routing, Sorting and Selection

Abstract

Mesh connected computers have become attractive models of computing because of their varied special features. In this paper we consider two variations of the mesh model: 1) a mesh with fixed buses, and 2) a mesh with reconfigurable buses. Both these models have been the subject matter of extensive previous research. We solve numerous important problems related to packet routing, sorting, and selection on these models. In particular, we provide lower bounds and very nearly matching upper bounds for the following problems on both these models: 1) Routing on a linear array; and 2) k - k routing, k - k sorting, and cut through routing on a 2D mesh for any $k \geq 12$. We provide an improved algorithm for 1-1 routing and a matching sorting algorithm. In addition we present greedy algorithms for 1-1 routing, k - k routing, cut through routing, and k - k sorting that are better on average and supply matching lower bounds. We also show that sorting can be performed in logarithmic time on a mesh with fixed buses. As a consequence we present an optimal randomized selection algorithm. In addition we provide a selection algorithm for the mesh with reconfigurable buses whose time bound is significantly better than the existing ones. Our algorithms have considerably better time bounds than many existing best known algorithms.

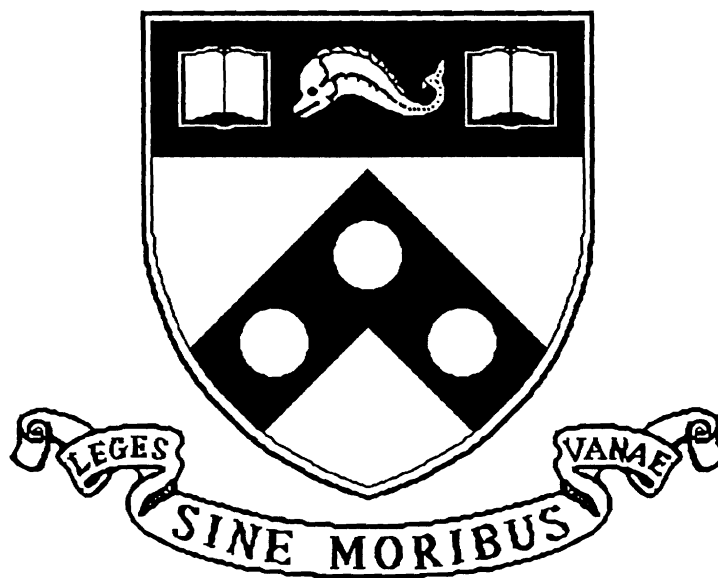
Comments

University of Pennsylvania Department of Computer and Information Science Technical Report No. MS-CIS-92-56.

**Mesh Connected Computers with Multiple Fixed
Buses:
Packet Routing, Sorting, and Selection**

**MS-CIS-92-56
GRASP LAB 323**

Sanguthevar Rajasekaran



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

July 1992

Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing, Sorting, and Selection*

Sanguthevar Rajasekaran

Department of CIS
University of Pennsylvania
Philadelphia, PA 19104.

Abstract Mesh connected computers have become attractive models of computing because of their varied special features. In this paper we consider two variations of the mesh model: 1) a mesh with fixed buses, and 2) a mesh with reconfigurable buses. Both these models have been the subject matter of extensive previous research. We solve numerous important problems related to packet routing, sorting, and selection on these models. In particular, we provide lower bounds and very nearly matching upper bounds for the following problems on both these models: 1) Routing on a linear array; and 2) $k - k$ routing, $k - k$ sorting, and cut through routing on a 2D mesh for any $k \geq 12$. We provide an improved algorithm for $1 - 1$ routing and a matching sorting algorithm. In addition we present greedy algorithms for $1 - 1$ routing, $k - k$ routing, cut through routing, and $k - k$ sorting that are better on average and supply matching lower bounds. We also show that sorting can be performed in logarithmic time on a mesh with fixed buses. As a consequence we present an optimal randomized selection algorithm. In addition we provide a selection algorithm for the mesh with reconfigurable buses whose time bound is significantly better than the existing ones. Our algorithms have considerably better time bounds than many existing best known algorithms.

*This research was supported in part by an NSF Research Initiation Award CCR-92-09260 and an ARO grant DAAL03-89-C-0031.

1 Introduction

1.1 Packet Routing

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 through any wire 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 1) its *run time*, i.e., the time taken by the last packet to reach its destination, and 2) its *queue length*, which is defined as the maximum number of packets any node will have to store during routing. Contentions for edges can be resolved using a *priority scheme*. Furthest destination first, furthest origin first, etc. are examples of priority schemes. We assume that a packet not only contains the message (from one processor to another) but also the origin and destination information of this packet. An algorithm for packet routing is specified by 1) the path to be taken by each packet, and 2) a priority scheme.

1.2 Different Models of Packet Routing and $k - k$ Sorting

How large a packet is (when compared with the channel width of the communication links) will determine whether a single packet can be sent along a wire in one unit of time. If a packet is very large it may have to be split into pieces and sent piece by piece. On this criterion many models of routing can be derived. A packet can be assumed to be either atomic (this model is known as the *store and forward model*), or much larger than the channel width of communication links (thus necessitating splitting).

In the later, if each packet is broken up into k pieces (also called *flits*), where k depends on the width of the channel, the routing problem can be studied under two different approaches. We can consider the k flits to be k distinct packets, which are routed independently. This is known as the *multipacket routing approach* [20]. Each flit will contain information about its origin and destination. The problem of $k - k$ routing is one where $\leq k$ packets originate from any node and $\leq k$ packets are destined for any node under the multipacket model.

Alternatively, one can consider the k flits to form a *snake*. All flits follow the first one, known as the head, to the destination. A snake may never be broken, i.e., at any given time, consecutive flits of a snake are at the same or adjacent processors. Only the head has to contain the origin and destination addresses. This model is called the *cut through routing with partial cuts* or simply the *cut through routing* [28].

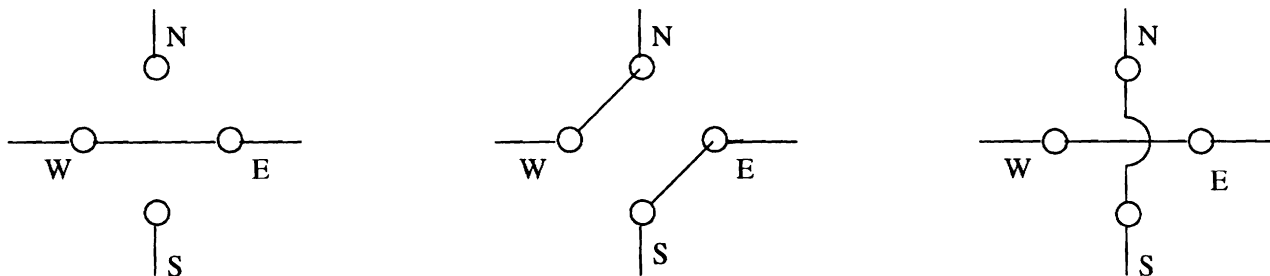


Figure 1: Some Possible Connection Patterns

The problem of $k - k$ sorting on any fixed connection machine is the problem of sorting where exactly k packets are input at any node. Given a set of N numbers and an $i \leq N$, the problem of selection is to find the i th smallest element from out of the N given numbers.

1.3 Definition of Models

The fixed connection machines assumed in this paper are: 1) the mesh connected computer with fixed buses (denoted as M_f), and 2) the mesh with reconfigurable buses (denoted as M_r). The basic topology of a two dimensional mesh is an $n \times n$ square grid with one processor per grid point. Except for processors at the boundary, every other processor is connected to its neighbors to the *left*, *right*, *above*, and *below* through bidirectional links. The instruction stream assumed is MIMD. This in particular means that each node can send and receive a packet (or a flit) from all its (four or less) neighbors in one unit of time.

In M_f we assume that each row and each column has been augmented with a broadcast bus. Only one packet can be broadcast along any bus at any time, and this message can be read by all the processors in the row or column associated with this bus in the same time unit.

In the model M_r , processors are connected to a reconfigurable broadcast bus. At any given time, the broadcast bus can be partitioned (i.e., reconfigured) dynamically into subbuses with the help of locally controllable switches. 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. Each processor has 4 I/O ports. In PARBUS model, any combination of 4 port connections is permitted for each processor [10]. (See Figure 1).

In fact, this is the model we assume in the paper. However, the algorithms given can be adopted to other variation of M_r , in which case the stated time bounds will also change. This model has been assumed in previous works as well (see e.g., [10, 9]).

Both M_r and M_f are becoming popular models of computing because of the absence of diameter consideration and because of the commercial implementations [2, 29, 52, 27].

Mesh connected computers have drawn the attention of computer scientists in recent times because of their many special properties. Some of the special features of meshes are: 1) they have a simple interconnection pattern, 2) many problems have data which map naturally onto them, and 3) they are linear-scalable.

1.4 Previous and New Results

Numerous papers have been written on routing and sorting on the conventional mesh (see e.g., [53, 55, 47, 43, 44, 19, 20, 21, 24, 22, 28, 39, 40, 38, 37, 36, 16, 15, 14]). An excellent reference for algorithms on the conventional mesh is Leighton [23].

Meshes with multiple buses have been studied by various researchers in the past (see e.g., [49, 50, 3, 18, 51, 4, 25, 26, 17]). An equally impressive amount of work has been done on the mesh with reconfigurable buses as well (see e.g., [2, 30, 31, 48, 57, 32]).

Mesh with fixed buses

Leung and Shende [25, 26] have shown that on a linear array with a bus, permutation routing needs $\frac{2n}{3}$ steps in the worst case and presented a $\frac{2n}{3}$ step algorithm as well. They also proved that on a two dimensional mesh M_f , permutation routing needs at least $\frac{2n}{3}$ steps in the worst case and can be performed within $n + 4\frac{n}{q} + o(n)$ time, the queue length being $2q$, for any $1 \leq q \leq n$.

In this paper we prove a lower bound for routing on a linear array for any input (not necessarily the worst case input), and present an algorithm whose run time matches this lower bound (up to a $o(n)$ lower order term). For the problem of permutation routing on a 2D mesh M_f , we present a very simple randomized algorithm with a run time of $n + \frac{n}{2q} + O(\sqrt{n \log n})$ and a queue length of $q + o(q)$ for any $1 \leq q \leq n$.

Next we consider the problems of $k - k$ routing, $k - k$ sorting, and cut through routing. We show that $\frac{kn}{3}$ is a lower bound for the worst case run time of any algorithm for these problems on a 2D mesh. We also give randomized algorithms for $k - k$ routing and $k - k$ sorting which run in $\frac{kn}{3} + o(kn)$ steps with high probability, and a randomized algorithm for cut through routing that runs in $\frac{kn}{3} + 1.5n + o(kn)$ steps with high probability, for any $k \geq 12$. All these algorithms have a queue length of $k + o(k)$ with high probability and extend to higher dimensional meshes as well.

We also present greedy algorithms for $1 - 1$ routing, $k - k$ routing, cut through routing, and $k - k$ sorting which are better on the average. We prove a matching lower bound for the expected run time of any algorithm for random $k - k$ routing, $k - k$ sorting, and cut through routing problems. These results highlight a crucial difference between a conventional mesh and a mesh with buses. For instance, in a conventional 2D mesh, the worst case run time for $1 - 1$ routing is $2n - 2$ and one can show that $2n - o(n)$ is a lower bound for the expected routing time of a random routing

problem. Remarkably, in the case of M_f , the worst case lower bound and the average case lower bound differ widely.

In addition we show that sorting of n keys can be performed on an $n \times n \times n$ mesh or on an $n^2 \times n^2$ mesh M_f in $O(\log n)$ time. As a consequence we show that selection on an $n \times n$ mesh M_f (from out of n^2 elements) can be accomplished within $O(n^{1/3})$ time using a randomized algorithm. The best known previous algorithm has a run time of $O(n^{1/3}(\log n)^{2/3})$ [18]. In [18, 51], a lower bound of $\Omega(n^{1/3})$ is proven for selection and related problems and hence our algorithm is asymptotically optimal. Our selection algorithm also runs in $O(n^{1/4})$ time on an $n^{5/4} \times n^{3/4}$ mesh, which is optimal and is an improvement over the run time of $O(n^{1/4} \log n)$ that the algorithm of [4] has.

Reconfigurable Mesh

A number of interesting results have been obtained on the reconfigurable mesh. Some of them are: 1) Prefix and other problems [30]; 2) Constant time sorting and routing algorithms [2, 32, 57]; 3) Random Access Read and Random Access Write [30]; 4) Histogramming and Related Problems [12, 13, 10]; 5) Permutation Routing and Sorting [38, 16]; 6) Selection [7, 9, 6]. In this paper we show that $k - k$ routing, $k - k$ sorting, and cut through routing need $\frac{kn}{2}$ steps on M_r and show that there exist algorithms for these three problems with matching time bounds, for any $k \geq 8$. More interestingly, we provide algorithms for these three problems that are optimal on the average. In particular these algorithms run in time $\frac{kn}{4} + o(kn)$ for $k - k$ routing and $k - k$ sorting and in $\frac{kn}{4} + 1.5n + o(kn)$ time for cut through routing, for any $k \geq 8$. We prove a lower bound of $\frac{kn}{4}$ for these three problems.

The problem of selection has been well studied on M_r . ElGindy and Wegrowicz [7] presented an $O(\log^2 n)$ time deterministic algorithm; Doctor and Krizanc [6] have recently presented three algorithms. Specifically, their algorithms can 1) select in $O(b \log^* n)$ time if the numbers are distinct and of length at the most b bits; or 2) select in $O(\log n)$ expected time assuming that each input permutation is equally likely; or 3) select in randomized $O(\log^2 n)$ time (with no assumptions). Very recently Hao, MacKenzie, and Stout [9] have given three algorithms as well: 1) An $O((b/\log b) \max\{\log^* n - \log^* b, 1\})$ time algorithm for selecting among b -bit words; 2) An algorithm that runs in $O(\log^* n)$ time assuming that each input permutation is equally likely (However no details of the algorithm appear in the proceedings); and 3) an $O(\log n)$ time algorithm without making any assumptions. In this paper we show that selection can be done using a randomized algorithm that runs in 1) $O(\log^* n)$ expected time assuming that each input permutation is equally likely (This is an independent work); or 2) $O(\log^* n \log \log n)$ time (with no assumptions).

Table I summarizes best known results for packet routing, sorting and selection on M_f and M_r . In this table W.C. stands for worst case, Av. stands for average case, and q stands for queue size. $\lceil \sqrt{\cdot} \rceil$ denotes new results presented in this paper. Also, γ stands for $1.5n$. Define $\log^{(1)} x$ as $\log x$, and $\log^{(i)} x$ as $\log(\log^{(i-1)} x)$ (for any integer $i \geq 2$). Then, $\log^* x$ is nothing but the smallest i such

that $\log^{(i)} x \leq 1$.

PROBLEM	M_f (W.C.)	M_r (W.C.)	M_f (Av.)	M_r (Av.)
Permutation Routing	$n + O(\frac{n}{q}) + o(n)$ [26]	$n + O(\frac{n}{q}) + o(n)$ [38]	$n + o(n)$ [✓]	$n + o(n)$ [✓]
$k - k$ Routing	$\frac{kn}{3} + o(kn)$ [✓]	$\frac{kn}{2} + o(kn)$ [✓]	$\frac{kn}{6} + o(kn)$ [✓]	$\frac{kn}{4} + o(kn)$ [✓]
$k - k$ Sorting	$\frac{kn}{3} + o(kn)$ [✓]	$\frac{kn}{2} + o(kn)$ [✓]	$\frac{kn}{6} + o(kn)$ [✓]	$\frac{kn}{4} + o(kn)$ [✓]
Cut Thro' Routing	$\frac{kn}{3} + \gamma + o(kn)$ [✓]	$\frac{kn}{2} + \gamma + o(kn)$ [✓]	$\frac{kn}{6} + n + o(kn)$ [✓]	$\frac{kn}{4} + n + o(kn)$ [✓]
Selection	$O(n^{1/3})$ [✓]	$O(\log^* n \log \log n)$ [✓]	$O(n^{1/3})$ [✓]	$O(\log^* n)$ [9] [✓]

Table I: Contributions of this paper.

1.5 Some Definitions

We say a randomized algorithm uses $\tilde{O}(g(n))$ amount of any resource (like time, space, etc.) if there exists a constant c such that the amount of resource used is no more than $c\alpha g(n)$ with probability $\geq 1 - n^{-\alpha}$ on any input of length n and for any α . Similar definitions apply to $\tilde{o}(g(n))$ and other such ‘asymptotic’ functions.

By *high probability* we mean a probability of $\geq 1 - n^{-\alpha}$ for any fixed $\alpha \geq 1$ (n being the input size of the problem at hand).

Let $B(n, p)$ denote a binomial random variable with parameters n and p , and let ‘w.h.p.’ stand for ‘with high probability’.

1.6 Chernoff Bounds

One of the most frequently used facts in analyzing randomized algorithms is *Chernoff bounds*. These bounds provide close approximations to the probabilities in the tail ends of a binomial distribution. Let X stand for the number of heads in n independent flips of a coin, the probability of a head in a single flip being p . X is also known to have a binomial distribution $B(n, p)$. The following three facts (known as Chernoff bounds) are now folklore (and were discovered by Chernoff [5] and Angluin & Valiant [1]):

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

$$\text{Prob.}[X \geq (1 + \epsilon)np] \leq \exp(-\epsilon^2 np/2), \text{ and}$$

$$\text{Prob.}[X \leq (1 - \epsilon)np] \leq \exp(-\epsilon^2 np/3),$$

for any $0 < \epsilon < 1$, and $m > np$.

2 Locally Optimal Routing on a Linear Array M_f

Leung and Shende [25, 26] have shown that routing on an n -node linear array with a fixed bus needs at least $\frac{2n}{3}$ steps in the worst case. They also matched this lower bound with an algorithm that runs in $\frac{2n}{3}$ steps in the worst case. Thus as far as the worst case input is concerned, the permutation routing problem on a linear array has been solved.

An interesting question is: ‘Can we perform optimal routing on a linear array for **any** input?’ In the case of a conventional array, the maximum distance any packet will have to travel is clearly a lower bound for routing on any input and this lower bound can be matched with an algorithm as well [44]. In the case of M_f it is not even clear what a lower bound will be for any input. In this section we prove such a lower bound and present an algorithm that matches this lower bound (up to a $o(n)$ lower order term).

A Lower Bound: Let \mathcal{L} be an n -node linear array with a fixed bus and π be a permutation to be routed. If the number of packets that have to travel a distance of d or more is n_d , then $\min\{d, n_d\}$ is a lower bound for the routing time of π (for each $1 \leq d \leq n$).

Proof: From among the packets that have to travel a distance of d or more, if there exists at least one packet that never uses the bus, a lower bound for routing is d . On the other hand, if all of these packets use the bus one time or the other, clearly, n_d steps will be needed. \square

The above observation yields the following

Lemma 2.1 *Routing a permutation π needs at least $\max_d \{\min\{d, n_d\}\}$ steps on a linear array with a fixed bus.*

Our algorithm for routing uses a subroutine for calculating prefix sums:

Computing Prefix Sums: Given a sequence of n numbers, say, k_1, k_2, \dots, k_n the problem of *prefix sums computation* is to calculate $k_1, k_1 + k_2, \dots, k_1 + k_2 + \dots + k_n$. Assume that k_i is in processor i of a linear array M_f . The following Lemma is proven in [3, 49, 50]:

Lemma 2.2 *Prefix sums of n numbers can be computed on an n -node linear array M_f in $O(\sqrt{n})$ time steps. $\Omega(\sqrt{n})$ time is needed for computing prefix sums.*

Locally Optimal Routing: The idea of this algorithm is to exploit Lemmas 2.1 and 2.2. We first compute $d' = \max_d \{\min\{d, n_d\}\}$. We route all the packets that have to travel a distance of d' or more using the bus, and the other packets are routed using the edge connections under the furthest destination first priority scheme.

We claim that d' can be computed in $O(\sqrt{n} \log n)$ time. Observe that 1) For a given d , we can compute n_d in $O(\sqrt{n})$ time using Lemma 2.2; and 2) As d increases n_d remains nonincreasing (it might decrease more often). Thus we could determine d' using a binary search on the values of d .

Once we determine d' , we can perform routing in time $\max\{d', n_{d'}\}$. Thus we arrive at

Theorem 2.1 *There exists an algorithm for routing on a linear array that is optimal for every input up to a $o(n)$ lower order term.*

Observation. In the case of a conventional linear array, if L is the maximum distance any packet has to travel, we can perform routing in L steps, provided we know the value of L . But for the above algorithm, no such information about the input is needed.

Routing on a linear array M_r : Rajasekaran and McKendall [38] have presented an algorithm for routing a permutation that runs in $\frac{3}{4}n$ steps. The obvious lower bound is $\frac{n}{2}$. The problem of optimal routing still remains open.

3 Routing on a 2D Mesh M_f

In this section we present a very simple randomized algorithm that has a run time slightly better than that of [26]’s. This is only a minor result in this paper. Leung and Shende’s algorithm [25, 26] is deterministic and runs in $n + 4\frac{n}{q} + o(n)$ time, the queue size being $2q$, for any $1 \leq q \leq n$. This time bound will be $n + o(n)$, for instance, if we pick $q = \log n$. They also proved a lower bound of $\frac{2n}{3}$ steps for routing. We believe that n is a lower bound in the worst case for the partial permutation routing time. We are currently working on this problem. The algorithm presented in this section is very simple and has a run time of $n + \frac{n}{2q} + \tilde{O}(\sqrt{n \log n})$, the queue length being $q + \tilde{o}(q)$ (for any $1 \leq q \leq n$).

The algorithm is similar to an algorithm given in [38]. Color each packet as red or blue initially (each color being equally likely). We describe the algorithm used by the red packets now: Partition the mesh into horizontal slices of $\frac{n}{q}$ rows each. There are three phases in the algorithm. In phase I a red packet goes to a random node along its column of origin within the same slice of origin. In phases II and III it greedily travels to its destination, first traversing along the row and then along the column. Blue packets execute a symmetric algorithm. That is, the mesh is partitioned into vertical slices of $\frac{n}{q}$ columns each, and so on.

In this algorithm in phase I only edge connections are used. In phases II and III only buses are used to transport packets.

Lemma 3.1 *The above algorithm runs in time $n + \frac{n}{2q} + \tilde{O}(\sqrt{n \log n})$, the queue length being $q + \tilde{o}(q)$, for any $1 \leq q \leq n$.*

Proof Easy and omitted. \square

Permutation Routing on M_r : In [38], a randomized algorithm is given whose run time for routing any permutation is $n + O(\frac{n}{q}) + \tilde{o}(n)$, where q is the queue size. They also present a deterministic algorithm with a run time of $1.25n + O(\frac{n}{q}) + o(n)$. The lower bound of $\frac{n}{2}$ steps holds for routing on a 2D mesh also. Here again finding an optimal routing algorithm is open.

4 $k - k$ Routing and Cut Through Routing on M_f

In this section we prove a lower bound of $\frac{kn}{3}$ for $k - k$ routing and cut through routing on a two dimensional mesh with fixed buses and match this lower bound with an algorithm whose run time is $\frac{kn}{3} + \tilde{o}(kn)$ for $k - k$ routing and is $\frac{kn}{3} + 1.5n + \tilde{o}(kn)$ for cut through routing, whenever $k \geq 12$. The algorithm we use resembles the one given by Rajasekaran in [36] but there are many crucial differences. The lower bound of $\frac{kn}{3}$ applies to $k - k$ sorting as well.

The Lower Bound: Consider a permutation in which we need to interchange packets in the left half of the mesh with packets in the right half. There are $\frac{kn^2}{2}$ packets in each half. The interchange can occur only via row edges through nodes in column $n/2$ or through the row buses. There are a total of n row buses and a total of n nodes in column $n/2$. Realize that the edge connections are bidirectional whereas only one packet can be broadcast along any bus at any time. Let the number of packets that cross column $n/2$ in either direction using edge connections be N_1 , and let the number of packets that cross this column using broadcast buses be N_2 . Clearly, $N_1 + N_2 = kn^2$. The time needed for crossing is $\geq \max\{\frac{N_1}{2n}, \frac{N_2}{n}\}$. The minimum of this quantity is $\frac{kn}{3}$. This leads to the following

Lemma 4.1 *$k - k$ routing, cut through routing, and $k - k$ sorting on an $n \times n$ mesh M_f need $\geq \frac{kn}{3}$ steps each.*

4.1 Routing on a Linear Array

The algorithm for $k - k$ routing on a 2D mesh consists of 3 phases where each phase corresponds to routing along a linear array. Here we state and prove a Lemma that will prove useful in analyzing all the three phases of the mesh algorithm.

Problem 1. There are a total of $\epsilon kn + o(kn)$ packets in an n -node linear array (for some constant $\epsilon \leq 1$), such that the number of packets originating from or the number of packets destined for any successive i nodes is $\leq \epsilon ki + o(kn)$ (for any $1 \leq i \leq n$). Route the packets.

Lemma 4.2 *Problem 1 can be solved in time $\frac{\epsilon kn}{3} + \tilde{o}(kn)$ under the model M_f if $k \geq \frac{3}{\epsilon}$.*

Proof. Let the nodes of the array be named $1, 2, \dots, n$. Certain packets (call them *special packets*) will be routed using the bus, whereas the other packets will be routed using edge connections under the furthest destination first priority scheme. Whether or not a packet is special is decided by a coin flip. A packet can become special with probability $\frac{1}{3}$.

Using Chernoff bounds, the number of special packets is $\frac{\epsilon kn}{3} + \tilde{o}(kn)$. We could perform a prefix sums computation in $o(n)$ time and arrive at a schedule for these special packets. Thus the special packets can be routed within the stated time bound.

Observe that the number of non special packets originating from or destined for any successive i nodes is $\frac{2}{3}\epsilon ki + \tilde{o}(kn)$ (for any $1 \leq i \leq n$). Let $\beta = \frac{2}{3}\epsilon$.

Consider only non special packets whose destinations are to the right of their origins. Let i be an arbitrary node. It suffices to show that any packet that ever wants to cross node i from left to right will do so within $\frac{\beta kn}{2} + \tilde{o}(kn)$ steps. Ignore the presence of packets that do not have to cross node i .

The number of packets that want to cross node i from left to right is $\min\{\beta ki, \beta k(n-i)\} + \tilde{o}(kn)$. The maximum of this number over all i 's is $\frac{\beta kn}{2} + \tilde{o}(kn)$. It immediately follows that the non special packets will be done in $\frac{\beta kn}{2} + \frac{n}{2} + \tilde{o}(kn)$ steps. But our claim is slightly stronger. If node i is busy transmitting a flit at every time unit, the result follows. There may be instances when i may be idle. We could prove the claim using the involved proof given in [16]. But we give a very simple proof.

Consider only non special packets whose destinations are to the right of their origins. Let i be an arbitrary node. The proof makes use of the free sequence idea introduced in [43]. The number of non special packets with a destination to the right of i is $\beta k(n-i) + \tilde{o}(kn)$. Let $\ell_1, \ell_2, \dots, \ell_n$ be the number of such packets, at the beginning, at nodes $1, 2, \dots, n$, respectively. Let m be such that $\ell_m > 1$ and $\ell_{m'} \leq 1$ for $m < m' \leq n$. Call the sequence $\ell_{m+1}, \dots, \ell_n$ the *free sequence*. Notice that a packet in the free sequence will not be delayed by any other packet in the future. It is easy to see that at least one new packet joins the free sequence at every time step. Moreover, if r is the leftmost node in which a packet (that has to cross i) can be found, clearly, this left boundary moves roughly one position to the right every βk steps. More precisely, the left boundary moves $\geq t$ positions to the right every $\beta kt + \tilde{o}(kn)$ steps (for any $1 \leq t \leq n$).

Case 1: $i \leq \frac{n}{2}$: Even if all the packets that originate to the left of i want to cross i , they can do so within $\beta ki + \tilde{o}(kn)$ time, since there are only these many packets to the left of i and the left boundary will move one step roughly every βk steps. (See the 2 statements immediately above Case 1.)

Case 2: $i > \frac{n}{2}$: Here the claim is that all the packets that have to cross i will do so within $\beta k(n-i) + 2i - n + \tilde{o}(kn)$ steps. Clearly, after $\beta k(n-i) + \tilde{o}(kn)$ steps, all the non special packets that have to cross i will be in the free sequence. Also, the left boundary (i.e., the leftmost node in which a packet (that has to cross i) can be found) moves $\geq t$ positions to the right every $\beta kt + \tilde{o}(kn)$ steps (for any $1 \leq t \leq n$). Therefore, after $\beta k(n-i) + \tilde{o}(kn)$ steps, the maximum distance any packet (that has to cross i) has to travel, in order to cross i is $i - (n-i)$. Thus the claim of Case 2 follows.

Case 1 and Case 2 immediately imply that all the non special packets will reach their destinations within $\frac{\beta kn}{2} + \tilde{o}(kn)$ steps. \square

Corollary 4.1 $k - k$ routing (for any $k \geq 2$) on a (conventional) linear array can be performed in $\frac{kn}{2}$ steps using the furthest destination first priority scheme.

4.2 Routing on an $n \times n$ Mesh

Next we show that $k-k$ routing can be completed using a randomized algorithm in time $\frac{kn}{3} + \tilde{o}(kn)$. This algorithm has three phases. This three phase algorithm will be employed later on in many other contexts as well. Call this algorithm *Algorithm B*.

Algorithm B

To start with each packet is colored red or blue using an unbiased coin. The algorithm used by red packets is described next. Blue packets execute a symmetric algorithm using, at any given time, the dimension unused by the red packets. Let q be any red packet whose origin is (i, j) and whose destination is (r, s) .

Phase I: q chooses a random node in the column of its origin (each such node being equally likely). If (i', j) was the node chosen, it traverses along column j up to this node.

Phase II: q travels along row i' up to column s .

Phase III: The packet q reaches its destination traversing along column s .

A blue packet in phase I chooses a random node in the **row** of its origin and goes there along the row. In phase II it traverses along the current column to the row of its destination and in phase III it travels along the current row to its destination. Because of the MIMD model assumed in this paper, and because all the three phases are disjoint there will not be any conflict between blue and red packets.

Our algorithm for $k-k$ routing is Algorithm B with some slight modifications. We only describe the algorithm for red packets. Blue packets execute a symmetric algorithm. Routing of packets in phase II is done using the algorithm of section 4.1. The algorithm for routing in phases I and III is slightly different. We describe the algorithm for phase I and the same is used in phase III as well.

Algorithm for phase I: Consider the task of routing along an arbitrary column. Let \mathcal{A} and \mathcal{C} be the regions of the first $\frac{n}{\sqrt{12}}$ nodes and the last $\frac{n}{\sqrt{12}}$ nodes of this n -node linear array, respectively. Let \mathcal{B} be the region of the rest of the nodes in the array. Any packet that originates from \mathcal{A} whose destination is in \mathcal{C} and any packet that originates from \mathcal{C} with a destination in \mathcal{A} will be routed using the bus. Scheduling for the bus is done using a prefix sums operation. The rest of the packets are routed using the edge connections employing the furthest destination first priority scheme.

Theorem 4.1 *The above algorithm for $k-k$ routing runs in time $\frac{kn}{3} + \tilde{o}(kn)$, the queue length being $k + \tilde{o}(k)$, for any $k \geq 12$.*

Proof. Both the number of blue packets and the number of red packets is $B(kn^2, 1/2)$. Thus w.h.p. these two numbers will be nearly the same. Further, the number of blue (red) packets that will

participate in row routing of phases I and III (phase II) is $B(kn, 1/2)$ each. Also, the number of red (blue) packets that participate in column routing of phases I and III (phase II) is $B(kn, 1/2)$ each. Thus using Lemma 4.2 (with $\epsilon = 1/2$) we can show that phase II can be completed in $\frac{kn}{6} + \tilde{o}(kn)$ steps, for any $k \geq 6$.

We claim that phase I and phase III can be completed within $\frac{kn}{12} + \tilde{o}(kn)$ steps each, for any $k \geq 12$. We only provide the analysis for phase I, since phase III is just the reverse of phase I.

Analysis of phase I: Consider only blue packets and an arbitrary row. If i is any node in this row, it suffices to show that all the packets that ever want to cross node i from left to right will do so within $\frac{kn}{12} + \tilde{o}(kn)$ steps. In the following case analysis we only obtain an upper bound for the number of packets that want to cross i . But i may not be busy transmitting a packet at every time unit. We could employ the proof technique of section 4.1 to show that the given time is enough even after accounting for possible idle times of i . Observe that the number of packets that originate from region \mathcal{A} with a destination in region \mathcal{C} is $\frac{kn}{24} + \tilde{o}(kn)$.

Case 1. $i \leq \frac{n}{\sqrt{12}}$: The number of packets that have to cross node i from left to right is $\leq \frac{ki}{2} \left(\frac{n-i-n/\sqrt{12}}{n} \right) + \tilde{o}(kn)$. This number is $\leq \frac{kn}{24}(\sqrt{12}-2) + \tilde{o}(kn) = \frac{(\sqrt{3}-1)kn}{12} + \tilde{o}(kn)$, for any i in region \mathcal{A} .

Case 2. $\frac{n}{\sqrt{12}} < i < n - \frac{n}{\sqrt{12}}$: In this case the number of packets that want to cross i from left to right is $\leq \frac{ki}{2} \left(\frac{n-i}{n} \right) - \frac{kn}{24} + \tilde{o}(kn)$. This number is no more than $\frac{kn}{12} + \tilde{o}(kn)$ for any i in region \mathcal{B} .

Case 3. $i > n - \frac{n}{\sqrt{12}}$: Number of packets that have to cross i is $\leq \frac{k(i-n/\sqrt{12})}{2} \left(\frac{n-i}{n} \right) + \tilde{o}(kn)$, which is $\leq \frac{(\sqrt{3}-1)kn}{12} + \tilde{o}(kn)$ for any i in region \mathcal{C} .

Thus phase I (and phase III) can be completed within $\frac{kn}{12} + \tilde{o}(kn)$ steps.

Queue size analysis. The total queue length of any successive $\log n$ nodes is $\tilde{O}(k \log n)$ (because the expected queue length at any single node is k implying that the expected queue length in $\log n$ successive nodes is $k \log n$; now apply Chernoff bounds). One could employ the technique of Rajasekaran and Tsantilas [43] to distribute packets locally such that the number of packets stored in any node is $\tilde{O}(k)$. The queue length can further be shown to be $k + \tilde{o}(k)$ using the same trick. \square

Corollary 4.2 *If $k = O(n^\nu)$ for some constant $\nu < 1$, the queue length of the above algorithm is only $k + \tilde{O}(1)$.*

The following Theorem pertains to $k-k$ routing on r -dimensional meshes.

Theorem 4.2 *$k-k$ routing on an r -dimensional mesh can be performed within $\frac{kn}{3} + \tilde{O}(krn^{(r-1)/r})$ steps, the queue length being $k + \tilde{o}(k)$, as long as $k \geq 12$. If $k = O(n^\nu)$, the queue length is only $k + \tilde{O}(1)$.*

Similar Theorems can be proven for cut through routing as well. The proofs of the following Theorems are omitted:

Theorem 4.3 *Cut through routing can be completed in time $\frac{kn}{3} + \frac{3}{2}n + \tilde{o}(kn)$, the queue length being $k + \tilde{o}(k)$ for any $k \geq 12$.*

Theorem 4.4 *Cut through routing on an r -dimensional mesh M_f can be performed in $\frac{kn}{3} + (r+1)\frac{n}{2} + \tilde{O}(krn^{(r-1)/r})$ steps, the queue length being $k + \tilde{o}(k)$, for any $k \geq 12$. If $k = O(n^\nu)$ for some constant $\nu < 1$, the queue length is only $k + \tilde{O}(1)$.*

5 $k - k$ Routing and Cut Through Routing on M_r

In this section we consider the problems of $k - k$ routing and cut through routing on M_r . It turns out that $\frac{kn}{2}$ is a lower bound for these problems on M_r . Kaufmann, Rajasekaran, and Sibeyn [16] show that $k - k$ routing and cut through routing can be performed in $\frac{kn}{2} + \tilde{o}(kn)$ and $\frac{kn}{2} + 1.5n + \tilde{o}(kn)$ steps respectively.

The algorithm used in [16] is nothing but Algorithm B. We could make use of the same algorithm to route on M_r also. In particular, we won't be making use of the reconfiguration facility at all—we don't need to. The lower bound of $\frac{kn}{2}$ also can be proven easily. Consider the task of exchanging the $\frac{kn}{2}$ packets in the left half of the mesh M_r with packets in the right half. All the packets (there are kn of them) will have to cross column $\frac{n}{2}$. The time for crossing is reduced only if the horizontal bidirectional edges incident on the nodes of column $\frac{n}{2}$ are used. Even then only two packets can cross through any node at any time. Thus the lower bound follows. Therefore we have the following

Theorem 5.1 *$\frac{kn}{2}$ is a lower bound for $k - k$ routing, $k - k$ sorting, and cut through routing on the mesh M_r . We can perform $k - k$ routing and cut through routing in time $\frac{kn}{2} + \tilde{o}(kn)$ and $\frac{kn}{2} + 1.5n + \tilde{o}(kn)$, respectively, the queue length being $k + \tilde{o}(k)$, for any $k \geq 8$.*

Theorem 5.2 *$k - k$ routing on an r -dimensional mesh M_r can be performed within $\frac{kn}{2} + \tilde{O}(krn^{(r-1)/r})$ steps, the queue length being $k + \tilde{o}(k)$, for any $k \geq 8$. If $k = O(n^\nu)$, the queue length is only $k + \tilde{O}(1)$. Cut through routing on an r -dimensional mesh can be performed in $\frac{kn}{2} + (r+1)\frac{n}{2} + \tilde{O}(krn^{(r-1)/r})$ steps, the queue length being $k + \tilde{o}(k)$, if $k \geq 8$.*

6 $k - k$ Sorting

6.1 Sorting on M_f

We show here that sorting of kn^2 elements can be accomplished on an $n \times n$ mesh M_f with fixed buses in time that is only $o(kn)$ more than the time needed for $k - k$ routing w.h.p.

Many optimal algorithms have been proposed in the literature for $1 - 1$ sorting on the conventional mesh (see e.g. [23]). A $2n + o(n)$ step randomized algorithm has been discovered for sorting by Kaklamanis and Krizanc [14]. But $2n - 2$ is a lower bound for sorting on the conventional mesh. Recently Rajasekaran and McKendall [38] have presented an $n + o(n)$ randomized algorithm for routing on a reconfigurable mesh, where it was shown that sorting can be reduced to routing easily if there exists a mechanism for broadcasting. Using this reduction, Krizanc, Rajasekaran, and Shende [17] have given an algorithm for M_f that runs in time $n + 4\frac{n}{q} + \tilde{o}(n)$, the queue size being $2q$. In this section also we adopt this reduction.

Summary. Random sampling has played a vital role in the design of parallel algorithms for comparison problems (including sorting and selection). Reischuk's [46] sorting algorithm and the FLASHSORT of Reif and Valiant [45] are good examples. 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., [46, 45, 15, 14, 36, 38, 17, 42]).

Let $X = k_1, k_2, \dots, k_n$ be a given sequence of n keys and let $S = \{k'_1, k'_2, \dots, k'_s\}$ 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 [46, 41] probabilistically bounds the size of each of these subsets, and will prove helpful to our algorithm.

Lemma 6.1 *The cardinality of each X_j ($1 \leq j \leq (s + 1)$) is $\tilde{O}(\frac{n}{s} \log n)$.*

Next we describe our algorithm and prove its time bound. This algorithm is similar to the one given in [38]. We only provide a brief summary of the algorithm. More details can be found in [38] or [15]. The mesh is partitioned into blocks of size $n^{4/5} \times n^{4/5}$.

- i) A random sample of size very nearly $kn^{3/5}$ is chosen and broadcast to the whole mesh, such that each block stores a copy of all these splitter (i.e., sample) keys.
- ii) We compute the partial ranks of the sample keys in each block after sorting the block.
- iii) Then we perform a prefix sums operation on these partial ranks so as to obtain the global ranks of the sample keys.
- iv) Now we route each packet to an approximate destination that is a random node in an appropriate block of size $n^{4/5} \times n^{4/5}$. This approximate destination is very close to its actual destination and depends on the two splitter keys between which it falls. In particular, the approximate destination of any packet will be at the most a block away from its actual destination w.h.p.

- v) Next we sort the individual blocks and compute the rank of each key in the mesh.
- vi) Finally we route the packets to their actual destinations.

Analysis. The key to the analysis is the observation that the global ranks of the sample keys can be computed in $o(kn)$ steps. This observation was first made in [38] in connection with sorting on a reconfigurable mesh.

Step i) takes $O(kn^{3/5})$ steps, since a single key can be broadcast to the whole mesh in 2 steps using the buses. Step ii) involves sorting blocks of size $n^{4/5} \times n^{4/5}$ (together with the sample keys) and can be completed in $O(kn^{4/5})$ time using any standard sorting algorithm. In step iii), the global rank of a single key can be computed in time $O(n^{1/5})$. This can be done for instance by concentrating all the partial ranks of this key in a region of size $n^{1/5} \times n^{1/5}$. Thus the global ranks of all the keys can be determined in time $O(n^{1/5} \times kn^{3/5}) = O(kn^{4/5})$.

In step iv), routing takes $\frac{kn}{3} + \tilde{o}(kn)$ steps using the algorithm of section 4.2. Sorting in step v) takes $O(kn^{4/5})$ time. Step vi) also can be finished in time $O(kn^{4/5})$ because the actual destination of any key can be at the most one block away from where it is after step iv) (c.f. Lemma 6.1).

Thus we have the following

Theorem 6.1 *$k - k$ sorting on an $n \times n$ mesh with buses can be performed in $\frac{kn}{3} + \tilde{o}(kn)$ steps, the queue length being $k + \tilde{o}(k)$, for any $k \geq 12$.*

The above algorithm together with the routing algorithm given in section 3 yield the following theorem (which is a slight improvement over the algorithm of [17]).

Theorem 6.2 *$1 - 1$ sorting on the mesh M_f can be completed in time $n + \frac{n}{2q} + \tilde{o}(n)$, the queue size being $q + \tilde{o}(q)$.*

6.2 Sorting on M_r

The following Lemma due to Miller, Prasanna, Reisis, and Stout [30] deals with the problem of prefix computation and will be used in our algorithm.

Lemma 6.2 *Prefix computation on an $n \times n$ mesh M_r (with one item per node) can be performed in $O(\log n)$ time.*

We prove now that $k - k$ sorting on M_r can be done in $\frac{kn}{2} + \tilde{o}(kn)$ steps. The idea is use to make use of the randomized algorithm of section 6.1. Step i) now also takes $O(kn^{3/5})$ time, since a single key can be broadcast in 2 steps. Step ii) takes $O(kn^{4/5})$ time. Step iii) can be completed in $O(n^{3/5} \log n)$ time, since the global rank of a key can be computed in $O(\log n)$ time using a prefix computation (c.f. Lemma 6.2). Routing in step iv) takes time $\frac{kn}{2} + \tilde{o}(kn)$ (c.f. Theorem 5.1). Steps v) and vi) can be performed in a total of $O(kn^{4/5})$ time. Thus we get

Theorem 6.3 $k - k$ sorting on M_r can be completed in $\frac{kn}{2} + \tilde{o}(kn)$ time, the queue size being $k + \tilde{o}(k)$, for any $k \geq 8$.

On the conventional mesh, there exists a randomized algorithm for $k - k$ sorting that runs in $\frac{kn}{2} + 2n + \tilde{o}(kn)$ time [36].

7 Algorithms with Better Average Performance

In this section we present algorithms for routing and sorting that perform better on average than the worst case behavior of algorithms presented in previous sections. The average case behavior assumed here is that each packet is destined for a random location (this notion being the same as the one assumed in [22]). Leighton [22] has shown that the greedy algorithm for $1 - 1$ routing on the conventional mesh indeed runs in time $2n - \tilde{o}(n)$, the queue size at any node being no more than 4 plus the number of packets destined for this node. (The greedy algorithm referred to here is: a packet originating from (i, j) with a destination at (r, s) is sent along row i up to column s , and then along column s up to row r . Also, the high probability involved in the definition of $\tilde{o}()$ here is over the space of all possible inputs.)

In a conventional mesh, it is easy to see that if a single packet originates from each node and if this packet is destined for a random node, then there will be at least one packet that has to travel a distance of $\geq 2n - o(n)$ with high probability. Thus $2n - o(n)$ is a lower bound even on average (compared with the $2n - 2$ lower bound for the worst case $1 - 1$ permutation routing time).

However, on a mesh with fixed buses, there seems to be a clear separation of the average case and the worst case. For instance, on a linear array $1 - 1$ routing needs $\frac{2n}{3}$ steps in the worst case, whereas in this section we show that on average it only takes $\frac{(3-\sqrt{5})n}{2} \approx .382n$ steps. We also prove similar results for routing on a $2D$ mesh, $k - k$ routing, $k - k$ sorting, and cut through routing.

The following Lemmas are folklore (see e.g., [33, 23]) and will prove helpful in our analysis: Let z_1, z_2, \dots, z_m be 0,1 valued independent random variables such that $\text{Prob.}[z_j = 1] = p_j$ for $1 \leq j \leq m$. Let $S^m = \sum_{j=1}^m z_j$ and the expectation of S^m be $\mu = E[S^m] = \sum_{j=1}^m p_j$. We are interested in the probability that S^m is above or below its expectation. The following Lemma bounds the probability that S^m is below its mean.

Lemma 7.1 For $0 \leq T < \mu$, $\text{Prob.}[S^m < T] \leq e^{-(\mu-T)^2/(2\mu)}$.

The next Lemma bounds the probability that S^m is above its mean.

Lemma 7.2 For $\mu < T \leq 2\mu$, $\text{Prob.}[S^m \geq T] \leq e^{-(T-\mu)^2/(3\mu)}$.

7.1 The Case of a Linear Array M_f

Problem 2. Let \mathcal{L} be a linear array with n nodes numbered $1, 2, \dots, n$. There is a packet at each node to start with. The destination of each packet could be any of the n nodes all with equal probability. Route the packets.

Lemma 7.3 *Problem 2 can be solved in time $\frac{(3-\sqrt{5})}{2}n + \tilde{o}(n)$ steps.*

Proof. Make use of the optimal algorithm given in section 2. The claim is that the algorithm will terminate within the specified time.

For some fixed d , $1 \leq d \leq n$, let \mathcal{A} stand for the first d nodes, \mathcal{B} stand for the next $(n - 2d)$ nodes, and \mathcal{C} stand for the last d nodes of \mathcal{L} .

From among the packets originating from region \mathcal{A} , the expected number of packets that have to travel a distance of d or more is $\frac{(n-d)+(n-d-1)+\dots+(n-2d+1)}{n} = d - \frac{1.5d^2}{n} + \frac{d}{2n}$. From among the packets originating from region \mathcal{B} , the expected number of packets that will travel a distance of d or more is $\frac{(n-2d)(n-2d+1)}{n}$ which simplifies to $n - 4d + 4\frac{d^2}{n} + 1 - \frac{2d}{n}$. Also, the expected number of packets that have to travel d or more distance from region \mathcal{C} is $\frac{(n-d)+(n-d-1)+\dots+(n-2d+1)}{n} = d - \frac{1.5d^2}{n} + \frac{d}{2n}$.

Summing, the total expected number of packets that will travel a distance of d or more is $= E_d = \frac{d^2+n^2-2dn+n-d}{n} = \frac{(n-d)(n-d+1)}{n}$. Using Lemma 7.2, the actual number of packets is only $\tilde{o}(n)$ more than the expectation. The algorithm will run for $d' + \tilde{o}(n)$ time where d' is such that $E_{d'} \approx d'$. d' can be seen to be no more than $\frac{(3-\sqrt{5})n}{2} + O(1)$. \square

7.2 The Case of a 2D Mesh M_f

Problem 3. There is a packet to start with at each node of an $n \times n$ mesh M_f . The destination of each packet is a random node in the mesh, each such node being equally likely. Route the packets.

Lemma 7.4 *Problem 3 can be solved in time $2(2 - \sqrt{3})n + \tilde{o}(n) \approx 0.536n + \tilde{o}(n)$ using the greedy algorithm. The queue size at each node is no more than 2 plus the number of packets destined for that node.*

Proof. The greedy algorithm referred to is the following: Initially each packet is colored red or blue each color being equally likely. A red packet travels along the row of its origin up to its destination column in phase I. In phase II this red packet travels along its destination column to its actual destination. A blue packet executes a symmetric algorithm, i.e., it travels along the column of its origin in phase I and in phase II it travels along its destination row.

Assume that the two phases of the greedy algorithm are disjoint. Then, each phase is nothing but routing along a linear array. The number of packets originating from any i successive nodes can be seen to be $\frac{i}{2} + \tilde{o}(n)$. This fact, together with a computation similar to that given in the proof of Lemma 7.3, implies that the expected number of packets that have to travel a distance of

d or more in any of the phases is $= E_d = \frac{(n-d)(n-d+1)}{2n} + \tilde{o}(n)$. Thus, a single phase will terminate in time $d' + \tilde{o}(n)$ where d' is such that $E_{d'} \approx d'$. One can see that d' is nearly $= (2 - \sqrt{3})n + \tilde{o}(n)$.

The proof of the queue size is cumbersome. However, we could modify the greedy algorithm using the trick described in [44]. The idea is based on the fact that the expected queue size at the end of phase I at any node is 1. This in particular means that the total queue size in any successive n^ϵ nodes (for some constant $\epsilon < 1$) is $n^\epsilon + \tilde{O}(\sqrt{n^\epsilon \log n})$. Thus we could group the processors in each row into groups of n^ϵ processors each, and locally distribute packets destined for each group. The extra queue size then will not exceed 2 w.h.p. \square

Observations. Notice that the expected run time of the greedy algorithm is less than the lower bound for the worst case input. This is also true for routing on a linear array. Thus in the model M_f , there is a clear separation between the average case and worst case routing.

7.3 A Lower Bound

As far as routing on a linear array is concerned, $\frac{(3-\sqrt{5})}{2}n - \tilde{o}(n)$ can be seen to be a lower bound as well for the expected running time. (This is just an application of Lemma 7.1 together with the expected computation done in the proof of Lemma 7.3)

One could prove a similar lower bound on the expected run time of routing on a 2D mesh using the greedy algorithm. A tedious but routine calculation shows that the expected number of packets that have to travel a distance of d or more in a 2D mesh is $= E_d \approx n^2 \left[1 - 2\frac{d^2}{n^2} + \frac{4}{3}\frac{d^3}{n^3} - \frac{1}{6}\frac{d^4}{n^4} \right]$. There are a total of $2n$ buses in the mesh. Therefore, a lower bound for routing is $d' - \tilde{o}(n)$ where d' is such that $E_{d'} \approx 2nd'$. Such a d' can be seen to be $\approx 0.386n$. But this lower bound is not tight since a packet may have to take two buses before it reaches its destination. But the above proof uses only the fact that some packets will have to take a bus at least once.

7.4 Random Routing on M_r

On an $n \times n$ mesh M_r , if at the most one packet originates from any node and the destination of each packet is random, we could make use of the last two phases of Algorithm B. There will be only $\frac{n}{2} + \tilde{o}(n)$ packets that perform phase II (phase III) along any row (column). Therefore each phase can be completed in $\frac{n}{2} + \tilde{o}(n)$ time; we just perform a prefix computation followed by a broadcast of packets one at a time. Therefore we obtain

Theorem 7.1 *Random routing on M_r can be done in $n + \tilde{o}(n)$ time, the queue size being $\tilde{O}(1)$.*

7.5 $k - k$ Routing, $k - k$ Sorting, and Cut Through Routing on M_f

In this section we show that the greedy algorithm for $k - k$ routing indeed has a run time of $\frac{kn}{6} + \tilde{o}(kn)$ on a random input. The queue size can also shown to be $k + \tilde{o}(k)$ using the ideas given in section 7.2

The greedy algorithm referred to here is also the last two phases of Algorithm B. Scheduling for the bus in both the phases is done exactly as in phase I of algorithm in section 4.2.

Theorem 7.2 *The above greedy algorithm terminates in an expected $\frac{kn}{6} + \tilde{o}(kn)$ steps and the queue length can be adjusted to be $k + \tilde{o}(k)$, for any $k \geq 12$.*

Proof. Phase II and Phase III of Algorithm B (as applied to the random case) can be analyzed along the same lines as that of phase I in section 4.2. This is because on a random input, routing of phase II as well phase III correspond very nearly to the problem of randomizing a linear array (as in phase I of section 4.2), for a very large fraction of all possible inputs. The queue size can also be analyzed using the trick employed in section 7.2. \square

Similarly we can also prove the following

Theorem 7.3 *Cut through routing takes an expected $\frac{kn}{6} + n + \tilde{o}(kn)$ steps using the greedy algorithm, the queue length being $k + \tilde{o}(k)$, for any $k \geq 12$.*

Theorem 7.4 *$k - k$ sorting can be performed in an expected $\frac{kn}{6} + \tilde{o}(kn)$ steps on a random input, for any $k \geq 12$.*

A Lower Bound. Consider a random $k - k$ routing problem. The expected number of packets that have to cross from the left half of the mesh to the right half as well as the number of packets that have to cross from the right half to the left half is clearly $\frac{kn^2}{4}$. Using Lemma 7.1, the number of packets that have to cross from the left half to the right half is at least $\frac{kn^2}{4} - \tilde{o}(kn^2)$. These packets can cross only using the row buses or via the nodes in column $\frac{n}{2}$ using edge connections. Let the number of packets that cross column $\frac{n}{2}$ in either direction using edge connections be N_1 and let the number of packets that cross this column using broadcast buses be N_2 . Notice also that $N_1 + N_2 \geq \frac{kn^2}{4} - \tilde{o}(kn^2)$. The time needed for crossing is at least $\max\left\{\frac{N_1}{2n}, \frac{N_2}{n}\right\}$. The minimum of this quantity is $\frac{kn}{6} - \tilde{o}(kn)$. This lower bound establishes that the greedy algorithms for $k - k$ routing, $k - k$ sorting, and cut through routing are indeed very nearly optimal on average.

7.6 $k - k$ Routing, $k - k$ Sorting and Cut Through Routing on M_r

Consider the problem of $k - k$ routing on M_r . At the most k packets originate from any node and the destination of any packet is random. For this problem it turns out that the greedy algorithm runs in time $\frac{kn}{4} + \tilde{o}(kn)$, for any $k \geq 8$. $\frac{kn}{4} - \tilde{o}(kn)$ is a lower bound for routing as well.

The lower bound can easily be seen from a count of how many packets will have to cross from the left half of the mesh to the right half and vice-versa. Expected number of packets that will cross from one side to the other is $\frac{kn}{4}$. Applying Lemma 7.1, this number is $\geq \frac{kn}{4} - \tilde{o}(kn)$. These packets can only cross using the nodes in column $\frac{n}{2}$ and hence the lower bound follows.

We make use of the last two phases of Algorithm B. Packets are routed using the furthest destination first priority scheme. We don't even have to use the reconfiguration facility.

Consider the red packets and phase II. There are $\frac{kn}{2} + \tilde{o}(kn)$ red packets being routed along any row. If i is any node in an arbitrary row, the number of packets that will cross i from left to right is $\frac{ki}{2} \frac{n-i}{n} + \tilde{o}(kn)$. The maximum of this over all i 's is $\frac{kn}{8} + \tilde{o}(kn)$. Using an argument similar to the one used in the proof of Lemma 4.2, we conclude that phase I terminates in $\frac{kn}{8} + \tilde{o}(kn)$ time. Similar analysis can be done for phase II and red packets yielding the following

Theorem 7.5 *Random $k - k$ routing on M_r can be completed in $\frac{kn}{4} + \tilde{o}(kn)$ time, the queue size being $k + \tilde{o}(k)$, for any $k \geq 8$.*

Observation Realize that in the above algorithm we haven't made use of the reconfiguration facility at all. Therefore, the above Theorem holds for the conventional mesh as well.

The following Theorem is now easy:

Theorem 7.6 *$k - k$ sorting and cut through routing on M_r can be realized in time $\frac{kn}{4} + \tilde{o}(kn)$ and $\frac{kn}{4} + n + \tilde{o}(kn)$, respectively assuming random inputs, for any $k \geq 8$.*

8 A Logarithmic Time Sorting Algorithm for M_f

In this section we show that sorting of n keys can be performed on an $n \times n \times n$ mesh or on an $n^2 \times n^2$ mesh with fixed buses can be performed in $O(\log n)$ time. The algorithm for sorting is based on a subroutine for adding n numbers in $O(\log n)$ time. The idea is to compute the rank of each key and route the key whose rank is i to node i (for $i = 1, 2, \dots, n$). We provide details below: (Some notations: Let $(*, *, *)$ stand for the whole mesh, $(i, *, *)$ stand for the 2D submesh in which the first coordinate is i . Similar notations apply to all the other 2D submeshes. Let $(*, j, k)$ stand for the one dimensional submesh (also called a 'row' or a 'column') in which the second and third coordinates are j and k respectively. Similar notations apply to all other 1D submeshes.)

To obtain the sum of n numbers using an $n \times n$ mesh: Consider n numbers b_1, b_2, \dots, b_n that we want to compute the sum of. Let the nodes in the 2D mesh be named (i, j) , $1 \leq i, j \leq n$. b_i is input at node $(i, 1)$ for $1 \leq i \leq n$. The following Lemma is due to Kumar and Raghavendra [18]:

Lemma 8.1 *The sum of n numbers can be computed in $O(\log n)$ time using an $n \times n$ mesh M_f .*

Computing Ranks using an $n \times n \times n$ mesh: Let k_1, k_2, \dots, k_n be the given n keys. We use a submesh of size $n \times n$ to compute the rank of each key. Each node in the mesh is named with a triple (i, j, ℓ) , $(1 \leq i, j, \ell \leq n)$. Assume that the input is given in the processors $(i, 1, 1)$, $1 \leq i \leq n$ one key per processor. The algorithm has the following steps:

- **Step 1.** Broadcast k_i along the row $(i, 1, *)$ (for $i = 1, 2, \dots, n$). This takes one unit of time. Now each 2D mesh of the form $(*, *, \ell)$ (for $\ell = 1, 2, \dots, n$) has a copy of the input.
- **Step 2.** Submesh $(*, *, \ell)$ computes the rank of k_ℓ (for $\ell = 1, 2, \dots, n$) in $O(\log n)$ time as follows: Broadcast k_ℓ to the row $(*, 1, \ell)$. Processors in this row compare k_ℓ with every key in the input. In particular, processor $(i, 1, \ell)$ computes $b_{i,\ell} = \text{'Is } k_i \leq k_\ell?'$. Processors in the submesh $(*, *, \ell)$ add up the n bits $b_{1,\ell}, b_{2,\ell}, \dots, b_{n,\ell}$, to obtain the rank of k_ℓ (using Lemma 8.1).

Routing: After the above rank computation, the rank of key k_ℓ is available in processor $(1, 1, \ell)$ (for $\ell = 1, 2, \dots, n$). We make use of the 2D submesh $(*, 1, *)$ to route the packets in $O(1)$ time as follows: Broadcast packet k_ℓ along row $(*, 1, \ell)$ (for $\ell = 1, 2, \dots, n$). Now broadcast the key whose rank is i along the row $(i, 1, *)$ (for $i = 1, 2, \dots, n$). After this broadcast the n numbers are available in the row $(*, 1, 1)$ in sorted order.

Thus we have the following

Theorem 8.1 *Sorting of n keys can be performed on an $n \times n \times n$ mesh with fixed buses in time $O(\log n)$, the queue size being 2.*

Along the same lines we can prove the following

Theorem 8.2 *Sorting of n keys can be performed on an $n^2 \times n^2$ mesh M_f in $O(\log n)$ time, the queue size being 2.*

9 An Optimal Randomized Selection Algorithm on M_f

In this section we show that selection on an $n \times n$ mesh M_f can be performed within $\tilde{O}(n^{1/3})$ steps. The problem of selection is: Given n^2 elements from a linear order (one element per node of the mesh), and an integer $i \leq n^2$, find the i th smallest element. The best known previous algorithm is due to Kumar and Raghavendra [18] and it runs in $O(n^{1/3}(\log n)^{2/3})$ time. In [18, 51], a lower bound of $\Omega(n^{1/3})$ is proven for selection and related problems and hence our selection algorithm is optimal. Our algorithm also runs in an optimal $\tilde{O}(n^{1/4})$ time on an $n^{5/4} \times n^{3/4}$ mesh M_f . In contrast, the best known previous selection algorithm on an $n^{5/4} \times n^{3/4}$ mesh had a run time of $O(n^{1/4} \log n)$ [4].

Randomized selection algorithms have a rich history. Floyd and Rivest [8] presented an optimal randomized sequential algorithm for selection. Followed by this work, Reischuk [46] and Vishkin [56] showed how to perform selection in parallel on various models. Recently, Rajasekaran [35] gave an optimal selection algorithm for the hypercube. More recently, Kaklamanis, et. al. [15] have presented an efficient selection algorithm for the MIMD mesh that runs in $1.22n + \tilde{o}(n)$ steps.

All these randomized algorithms have the following general scheme: 1) Sample $o(n)$ keys from the input; 2) Sort the sample and identify two keys in the sample (call these ℓ_1 and ℓ_2) such that the element to be selected has a value in between ℓ_1 and ℓ_2 w.h.p.; 3) Eliminate all the keys from the input whose values fall outside the range $[\ell_1, \ell_2]$; and 4) Finally perform an appropriate selection in the set of remaining keys.

We also use a variation of this scheme in our algorithm. But the implementation is very different from previous approaches (such as [35, 15]). The following Lemma from Kumar and Raghavendra [18] will be helpful in our algorithm:

Lemma 9.1 *Prefix sums computation of n^2 elements on an $n \times n$ mesh M_f can be performed within $O(n^{1/3})$ time (in row major or snake-like row major ordering).*

Let $S = \{k_1, k_2, \dots, k_s\}$ be a random sample from a set X of cardinality N . Let ‘select(X, i)’ stand for the i th smallest element of X for any set X and any integer i . Also let k'_1, k'_2, \dots, k'_s be the sorted order of the sample S . If r_i is the rank of k'_i in X and if $|S| = s$, the following Lemma [41] provides a high probability confidence interval for r_i .

Lemma 9.2 *For every α , Prob. $\left(|r_i - i \frac{N}{s}| > c\alpha \frac{N}{\sqrt{s}} \sqrt{\log N}\right) < N^{-\alpha}$ for some constant c .*

INPUT: n^2 elements (one element per node), and an i ($1 \leq i \leq n^2$).

OUTPUT: The i th smallest element from out of the n^2 input elements.

The Algorithm

In the following algorithm each element (or key) is *alive* to start with.

repeat forever

- **Step 1.** Count the number of *alive* keys using the prefix sums algorithm. Let N be this number. If N is $\leq n^{1/3}$ then *quit* and go to Step 7;
- **Step 2.** Each *alive* element includes itself in a sample S with probability $\frac{n^{1/3}}{N}$. The total number of keys in the sample will be $\tilde{O}(n^{1/3})$;
- **Step 3.** Concentrate the sample keys in a square submesh and sort this submesh. Let ℓ_1 be select($S, i \frac{s}{N} - \delta$) and let ℓ_2 be select($S, i \frac{s}{N} + \delta$), where $\delta = d\sqrt{s \log N}$ for some constant d ($> c\alpha$) to be fixed;
- **Step 4.** Broadcast ℓ_1 and ℓ_2 to the whole mesh;
- **Step 5.** Count the number of *alive* keys $< \ell_1$ (call this number N_1); Count the number of *alive* keys $> \ell_2$ (call this number N_2); If i is not in the interval $(N_1, N - N_2]$, go to Step 2 else let $i := i - N_1$;

- **Step 6.** Any *alive* key whose value does not fall in the interval $[\ell_1, \ell_2]$ *dies*;

end repeat

Step 7

Concentrate the *alive* keys in a square submesh and sort it; Output the i th smallest key from this set.

Theorem 9.1 *The above selection algorithm runs in $\tilde{O}(n^{1/3})$ time.*

Proof. We first show that the *repeat* loop is executed no more than 11 times w.h.p. Followed by this, we show that each of the seven steps in the algorithm runs in $\tilde{O}(n^{1/3})$ time each.

An application of Lemma 9.2 implies that if d is chosen to be large enough ($> c\alpha$), the i th smallest element will lie between ℓ_1 and ℓ_2 w.h.p. Also, the number of keys *alive* after j runs of the *repeat* loop is $\tilde{O}\left(\frac{n^2}{(\sqrt{n^{1/3}})^j}(\sqrt{\log n})^j\right)$. After 11 runs, this number is $\tilde{O}(n^{1/6}(\sqrt{\log n})^{11}) = \tilde{O}(n^{1/3})$.

Step 1 takes $O(n^{1/3})$ time since it involves just a prefix sums computation. Step 2, Step 4, and Step 6 take $O(1)$ time each. In Step 3, concentration of keys can be done by broadcasting each sample key. Realize that a single packet can be broadcast to the whole mesh in 2 steps, and that a schedule for the sample keys can be determined with a prefix sums computation. Also, sorting can be done in $O(\log n)$ time (c.f. Theorem 8.2). Thus this step runs in $\tilde{O}(n^{1/3})$ time. Step 5 involves two prefix sums computations and hence can be finished within $O(n^{1/3})$ time. Finally, Step 7 is similar to Step 3. \square

In an analogous way the following Theorem can also be proven:

Theorem 9.2 *Selection on an $n^{5/4} \times n^{3/4}$ mesh M_f can be performed in an optimal $\tilde{O}(n^{1/4})$ time.*

10 Selection on M_r

The problem of selection on M_r has been studied by many researchers [7, 6, 9]. ElGindy and Wegrowicz have presented an $O(\log^2 n)$ time algorithm. Doctor and Krizanc's algorithms can select in 1) $O(b \log^* n)$ time, given that the numbers are at the most b bits long; or 2) $O(\log n)$ expected time assuming that each input permutation is equally likely; or 3) Randomized $O(\log^2 n)$ time with no assumptions. Hao, MacKenzie, and Stout [9] show that selection can be done in: 1) $O((b/\log b) \max\{\log^* n - \log^* b, 1\})$ time given that the numbers are b -bits long; or 2) $O(\log^* n)$ time assuming a uniform distribution on all possible inputs; or 3) $O(\log n)$ time with no assumptions. In this section we show that selection on M_r can be performed in: 1) $O(\log^* n)$ expected time assuming that each input permutation is equally likely (Though the same result is presented in [9], no details of the algorithm have been given in the proceedings; our work is independent); and 2) Randomized $O(\log^* n \log \log n)$ time, with no assumptions.

10.1 Some Basics

The following Lemmas will be employed in our selection algorithm:

Lemma 10.1 *Jang, Park, and Prasanna [10]: If each node in an $n \times n$ mesh M_r has a bit, the number of 1's can be computed in $O(\log^* n)$ time.*

Lemma 10.2 *Jang and Prasanna [11]: For any $1 \leq r \leq n$, elements in the first r rows of an $n \times n$ mesh M_r can be sorted in $O(r)$ time.*

Problem 4. Consider an $n \times n$ mesh M_r . Say there are ℓ_i elements arbitrarily distributed in row i , for $1 \leq i \leq n$. Let $\ell = \max\{\ell_1, \ell_2, \dots, \ell_n\}$. For each i , concentrate the elements of row i in the first ℓ_i columns of row i .

Lemma 10.3 *Problem 4 can be solved in time $O(\ell)$ if ℓ is given.*

Proof. It is easy to solve Problem 4 in $O(\log n)$ time using Lemma 6.2. In order to solve this problem in $O(\ell)$ time we use the following algorithm: There are ℓ rounds in the algorithm (for $t = 1, 2, \dots, \ell$).

for $t := 1$ to ℓ do

(Computation is local to each row i , $1 \leq i \leq n$ *)*

Step 1. If node j in row i has an element, then it sends a 1 to its right; at the same time it opens its switch so that any message from left is blocked. If node j has no element, it simply closes its switch so that any message from left is simply forwarded to the right.

Step 2. If a node has an element and if it receives a 1 from left it simply accepts failure in this round. The node with an element which does not receive a 1 from left broadcasts its packet so that it can be concentrated in column t of row i . Realize that there can be only one such node that gets to broadcast in any round. The node that gets to broadcast will not participate in any future rounds, whereas every other node with an element will participate in the next round.

Clearly, the above algorithm runs correctly in $O(\ell)$ time. The above algorithm, though very simple, brings out the power of reconfiguration. \square

Lemma 10.4 *Problem 4 can be solved in $O(\ell + \log \ell \log^* n)$ time if ℓ is unknown. Within the same time, we will also be able to estimate ℓ to within a factor of 2.*

Proof. The idea is to make use of the same algorithm with a slight modification. We make use of the ‘doubling’ trick. We guess a value of 2 for ℓ and run two rounds of the above algorithm. At the end of two rounds we make use of Lemma 10.1 to check if all the elements have been concentrated. This checking takes $O(\log^* n)$ time.

Even if there is a single packet that has not been concentrated, we increase the guess for ℓ to 4 and perform 2 more rounds of the algorithm, and so on. Clearly, the number of rounds made is $< 2\ell$ and the number of checkings done is $O(\log \ell)$. Thus the claim follows. The second part of the Lemma is obvious. Call this algorithm as Algorithm C. \square .

Corollary 10.1 *Say there are only ℓ elements in an $n \times n$ mesh M_r . Then we could compute the prefix sums of these elements in time $O(\ell)$ if ℓ is given, or in time $O(\ell + \log^* n \log \ell)$ if ℓ is not known.*

Lemma 10.5 *If i is any row and j is any column of an $n \times n$ mesh M_r , and if each node of row i has an element, then, we can copy the elements of row i into column j in constant time.*

Proof. Broadcast the elements of row i along the columns so that elements of row i appear along the diagonal. Now perform another broadcast of the diagonal elements along the rows. \square

10.2 The Algorithm for the Uniform Case

The selection algorithm to be described assumes that each input permutation is equally likely and is similar to the one given in section 9. But there are many important differences. More details follow:

Each element (or key) is *alive* to start with.

repeat forever

- **Step 1.** Count the number of *alive* keys using the algorithm of Lemma 10.1. Let N be this number. If N is $\leq n^{1/3}$ then *quit* and go to Step 7;
- **Step 2.** Each *alive* element includes itself in a sample S with probability $\frac{n^{1/3}}{N}$. The total number of keys in the sample will be $\tilde{O}(n^{1/3})$;
- **Step 3.** Concentrate the sample keys in row i tightly to the left (employing Lemma 10.4), for each $1 \leq i \leq n$ in parallel. Let ℓ be the upper bound obtained for the maximum number of sample keys in any row. Sort the first ℓ columns using the algorithm of Lemma 10.2. Let ℓ_1 be $\text{select}(S, i \frac{s}{N} - \delta)$ and let ℓ_2 be $\text{select}(S, i \frac{s}{N} + \delta)$, where $\delta = d\sqrt{s \log N}$ for some constant $d (> c\alpha)$ to be fixed;
- **Step 4.** Broadcast ℓ_1 and ℓ_2 to the whole mesh;

- **Step 5.** Count the number of *alive* keys $< \ell_1$ (call this number N_1); Count the number of *alive* keys $> \ell_2$ (call this number N_2); If i is not in the interval $(N_1, N - N_2]$, go to Step 2 else let $i := i - N_1$;
- **Step 6.** Any *alive* key whose value does not fall in the interval $[\ell_1, \ell_2]$ *dies*;

end repeat

Step 7

Concentrate the *alive* keys in any row tightly to the left (c.f. Lemma 10.4); If ℓ is an estimate on the maximum number of keys in any row, sort the first ℓ columns using Lemma 10.2. Output the i th smallest key from this set.

Theorem 10.1 *The above algorithm selects in $O(\log^* n)$ expected time assuming that each input permutation is equally likely.*

Proof. Like in the proof of Theorem 9.1, the *repeat* loop is executed no more than 11 times w.h.p. The crucial fact is that each of the above seven steps can be performed in $\tilde{O}(\log^* n)$ time. This follows from the fact that the value of ℓ in any iteration is $\tilde{O}(1)$. Notice that in any iteration, there are only $\tilde{O}(n^{1/3})$ sample keys and these sample keys will be uniformly distributed among all the n rows. Expected number of packets in any row will be $\Theta(\frac{n^{1/3}}{n})$, immediately implying that the number of sample keys in any row is $\tilde{O}(1)$.

Steps 2, 4, and 6 take $O(1)$ time each. Counting in steps 1 and 5 takes $O(\log^* n)$ time (c.f. Lemma 10.1). Given that ℓ is $\tilde{O}(1)$, steps 3 and 7 take $\tilde{O}(\log^* n)$ time each (c.f. Lemma 10.4). Thus the theorem follows. \square

10.3 A Selection Algorithm for the General Case

The algorithm to be used for the general case is the same as the one given in section 10.2, with some crucial modifications. In the average case, in any iteration, the alive keys will be uniformly distributed among the nodes of the mesh. Thus an expected $O(1)$ number of iterations (of the *repeat* loop) sufficed. The same need not hold in general. For instance, the alive keys after the first iteration may appear concentrated in a small region of the mesh (e.g., in a $\sqrt{N} \times \sqrt{N}$ submesh). The same might be the case after every iteration. Thus it seems $\Omega(\log \log n)$ iterations will be needed. Also, concentrating the sample keys (in order to identify ℓ_1 and ℓ_2) now becomes more complicated, for the same reason namely, these sample keys may not appear uniformly distributed among the nodes. Next we present the algorithm:

Each element (or key) is *alive* to start with.

repeat forever

- **Step 1.** Count the number of *alive* keys using the algorithm of Lemma 10.1. Let N be this number. If N is $\leq \log \log n$ then *quit* and go to Step 7;
- **Step 2.** Each *alive* element includes itself in a sample S with probability $\frac{N^{1/6}}{N}$. The total number of keys in the sample will be $\tilde{O}(N^{1/6})$;
- **Step 3.**

3.1 Concentrate the sample keys as follows: Some of the keys will be concentrated along the rows and the others will be concentrated along the columns. This is done by simultaneously concentrating the keys along the rows as well as along the columns. I.e., perform one round of Algorithm C concentrating along the rows, followed by one round of Algorithm C concentrating along the columns, and so on. If a node succeeds in concentrating its element along the row (say), this element will be eliminated from future consideration. In particular, in the next run, this node will behave as though it does not have any element. Concentration stops when each sample key has been concentrated either along the row or along the column. Let r and c stand for the maximum number of rows and columns, respectively, used for concentrating the sample keys. Let $\ell = \max\{r, c\}$. (We'll show that ℓ is $\tilde{O}(1)$.)

3.2 Now copy the ℓ or less rows of sample keys into columns (using the algorithm of Lemma 10.5). This copying is done one row at a time.

3.3 Sort the $\leq 2\ell$ columns of sample keys using Lemma 10.2. Let ℓ_1 be $\text{select}(S, i\frac{s}{N} - \delta)$ and let ℓ_2 be $\text{select}(S, i\frac{s}{N} + \delta)$, where $\delta = d\sqrt{s \log N}$ for some constant d ($> c\alpha$) to be fixed;

- **Step 4.** Broadcast ℓ_1 and ℓ_2 to the whole mesh;
- **Step 5.** Count the number of *alive* keys $< \ell_1$ (call this number N_1); Count the number of *alive* keys $> \ell_2$ (call this number N_2); If i is not in the interval $(N_1, N - N_2]$, go to Step 2 else let $i := i - N_1$;
- **Step 6.** Any *alive* key whose value does not fall in the interval $[\ell_1, \ell_2]$ *dies*;

end repeat

Step 7

Concentrate the *alive* keys in the first row. This can be done for instance by broadcasting one key at a time. Realize that if there are only ℓ elements in the mesh, we can perform

a prefix computation (c.f. Corollary 10.1) to arrive at a schedule for the broadcasts in $O(\ell)$ time. Output the i th smallest key from this set.

Theorem 10.2 *The above algorithm runs in time $\tilde{O}(\log^* n \log \log n)$.*

Proof. It suffices to show that the value of ℓ in Step 3.1 is $\tilde{O}(1)$. The rest of the steps can be analyzed as before.

Let α be the probability parameter (i.e., the algorithm will terminate in the specified amount of time with probability $\geq 1 - n^{-\alpha}$). If there are N alive keys at the beginning of some iteration of the *repeat* loop, then each alive key will be included in the sample with probability $\frac{N^{1/6}}{N}$.

How many rows will have more than 12 sample keys? Realize that if there are p_i alive keys in some row i , the expected number of sample keys from this row will be $p_i N^{-5/6}$. Classify a row as either *dense* or *sparse*, depending on whether it has $> N^{4/6}$ active elements or $\leq N^{4/6}$ active elements, respectively. For any sparse row, applying Chernoff bounds (equation 1), the number of sample keys in this row can not be more than 6β with probability $\geq (1 - N^{-\beta})$, for any $\beta \geq 1$. Since there are at the most N rows in the mesh, the expected number of sparse rows that have > 12 sample keys is $\leq N^{-1}$. This in turn means that the number of sparse rows with > 12 sample keys is $\tilde{O}(1)$. That is, every other row with > 12 elements has to be dense.

Sample keys in the sparse rows (except for $\tilde{O}(1)$ of them) will potentially get concentrated along the rows. Also notice that there can be at the most $N^{2/6}$ dense rows. Even if these dense rows and the sparse rows with > 12 sample keys are such that each column (when restricted to these rows) is completely full with active elements, the number of sample keys in each column can only be $\tilde{O}(1)$ and hence these sample keys will get concentrated along the columns.

In the above algorithm, if N is the number of alive keys at the beginning of any iteration, then at the end of this iteration the number of alive keys is no more than $N^{11/12}$ w.h.p. Here, the high probability is with respect to the current size of the problem, i.e., N . Therefore we conclude that the expected number of iterations of the *repeat* loop is $O(\log \log n)$. We can also show that the number of iterations is $\tilde{O}(\log \log n)$. Moreover, the above analysis shows that each iteration takes $\tilde{O}(\log^* n)$ time. \square

A Note on Optimality: It is easy to see that a single step of computation on the mesh M_r can be simulated in $O(1)$ time on the Parallel Comparison Tree (PCT) model of Valiant [54]. The effect of reconfiguration can be achieved for free on the PCT, since the later charges only for the comparisons performed. Thus it will follow that selection needs $\Omega(\log \log n)$ time on the mesh M_r using any deterministic comparison based algorithm. (The same fact is mentioned in [9] as well). We believe that $\Omega(\log \log n)$ is a lower bound for selection on M_r even using a randomized comparison algorithm. This is an interesting open problem.

11 Conclusions

In this paper we have addressed numerous important problems related to packet routing, sorting, and selection on a mesh with fixed buses and on a mesh with reconfigurable buses. Many existing best known results have been improved. Some remaining open problems are: 1) Is n a lower bound for the worst case partial permutation routing time on a 2D mesh M_r or M_f ?; 2) Can the randomized algorithms given in this paper be matched with deterministic algorithms?; 3) Can sorting be performed in time asymptotically less than $\log n$ on a fixed dimensional mesh M_f with a polynomial number of processors?

Acknowledgements

I am grateful to Sunil Shende for introducing me to this area, and to Doctor and Krizanc for supplying me with a copy of their manuscript [6]. I also would like to thank Prsanna and Schuster for providing me with a number of relevant articles.

References

- [1] D. Angluin and L.G. Valiant, Fast Probabilistic Algorithms for Hamiltonian Paths and Matchings, *Journal of Computer and Systems Science*, 18, 1979, pp. 155-193.
- [2] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, The Power of Reconfiguration, *Journal of Parallel and Distributed Computing*, 1991, pp. 139-153.
- [3] S.H. Bokhari, Finding Maximum on an array processor with a global bus, *IEEE Trans. Computers* 33, 1984, pp. 133-139.
- [4] Y-C. Chen, W-T. Chen, and G-H. Chen, Efficient Median Finding and Its Application to Two-Variable Linear Programming on Mesh-Connected Computers with Multiple Broadcasting, *Journal of Parallel and Distributed Computing* 15, 1992, pp. 79-84.
- [5] H. Chernoff, A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations, *Annals of Mathematical Statistics* 23, 1952, pp. 493-507.
- [6] D.P. Doctor and D. Krizanc, Three Algorithms for Selection on the Reconfigurable Mesh, Manuscript, 1992.
- [7] H. ElGindy and P. Wegrowicz, Selection on the Reconfigurable Mesh, *Proc. International Conference on Parallel Processing*, 1991, pp. 26-33.
- [8] R.W. Floyd and R.L. Rivest, Expected Time Bounds for Selection, *Communications of the ACM*, vol. 18, no.3, 1975, pp. 165-172.

- [9] E. Hao, P.D. McKenzie and Q.F. Stout, Selection on the Reconfigurable Mesh, Proc. Frontiers of Massively Parallel Computation, 1992.
- [10] J. Jang, H. Park, and V.K. Prasanna, A Fast Algorithm for Computing Histograms on a Reconfigurable Mesh, Proc. Frontiers of Massively Parallel Computing, 1992, pp. 244-251.
- [11] J. Jang and V.K. Prasanna, An Optimal Sorting Algorithm on Reconfigurable Mesh, Proc. International Parallel Processing Symposium, 1992, pp. 130-137.
- [12] J. Jenq and S. Sahni, Reconfigurable Mesh Algorithms for Image Shrinking, Expanding, Clustering, and Template Matching, Proc. International Parallel Processing Symposium, 1991, pp. 208-215.
- [13] J. Jenq and S. Sahni, Histogramming on a Reconfigurable Mesh Computer, Proc. International Parallel Processing Symposium, 1992, pp. 425-432.
- [14] C. Kaklamanis and D. Krizanc, Optimal Sorting on Mesh Connected Processor Arrays, in Proc. ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, 1992.
- [15] C. Kaklamanis, D. Krizanc, L. Narayanan, and Th. Tsantilas, Randomized Sorting and Selection on Mesh Connected Processor Arrays, Proc. ACM Symposium on Parallel Algorithms and Architectures, 1991.
- [16] M. Kaufmann, S. Rajasekaran, and J. Sibeyn, Matching the Bisection Bound for Routing and Sorting on the Mesh, in Proc. ACM Symposium on Parallel Algorithms and Architectures, San Diego, CA, 1992.
- [17] D. Krizanc, S. Rajasekaran, and S. Shende, A Comparison of Meshes with Static Buses and Unidirectional Wrap-Arounds, Manuscript, 1992.
- [18] V.K.P. Kumar and C.S. Raghavendra, Array Processor with Multiple Broadcasting, Journal of Parallel and Distributed Computing 4, 1987, pp. 173-190.
- [19] M. Kunde, Routing and Sorting on Mesh Connected Processor Arrays, in Proc. VLSI Algorithms and Architectures: AWOC 1988. Springer-Verlag Lecture Notes in Computer Science #319, Springer-Verlag, pp. 423-33.
- [20] M. Kunde and T. Tensi, Multi-Packet Routing on Mesh Connected Arrays, in Proc. ACM Symposium on Parallel Algorithms and Architectures, 1989, pp. 336-343.
- [21] M. Kunde, Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound, in Proc. IEEE Symposium on Foundations of Computer Science, 1991.

- [22] T. Leighton, Average Case Analysis of Greedy Routing Algorithms on Arrays, in Proc. ACM Symposium on Parallel Algorithms and Architectures, pp. 2-10, July 1990.
- [23] T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*, Morgan-Kaufmann Publishers, San Mateo, California, 1992.
- [24] T. Leighton, F. Makedon, and I.G. Tollis, A $2n - 2$ Step Algorithm for Routing in an $n \times n$ Array With Constant Size Queues, in Proc. ACM Symposium on Parallel Algorithms and Architectures, 1989, pp. 328-35.
- [25] J. Y-T. Leung and S. M. Shende, Packet Routing on Square Meshes with Row and Column Buses, in Proc. IEEE Symposium on Parallel and Distributed Processing, Dallas, Texas, Dec. 1991, pp. 834-837.
- [26] J. Y-T. Leung and S. Shende, to appear in Journal of Parallel and Distributed Computing, 1992.
- [27] H. Li and Q. Stout, *Reconfigurable Massively Parallel Computers*, Prentice-Hall Publishers, 1991.
- [28] F. Makedon and A. Simvonis, On bit Serial packet routing for the mesh and the torus, in Proc. third Symposium on Frontiers of Massively Parallel Computation, 1990, pp. 294-302.
- [29] O. Menzilcioglu, H.T. Kung, and S.W. Song, Comprehensive Evaluation of a Two-Dimensional Configurable Array, Proc. 19th Symposium on Fault Tolerant Computing, 1989, pp. 93-100.
- [30] 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.
- [31] 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.
- [32] K. Nakano, D. Peleg, and A. Schuster, Constant-time Sorting on a Reconfigurable Mesh, Manuscript, 1992.
- [33] R. Paturi, S. Rajasekaran, and J.H. Reif, The Light Bulb Problem, submitted for publication, 1992.
- [34] C.S. Raghavendra and V.K.P. Kumar, Permutations on Illiac IV-Type Networks, IEEE Trans. Comp., 35 (1986) pp. 662-669.
- [35] S. Rajasekaran, Randomized Parallel Selection, Proc. Tenth International Conference on Foundations of Software Technology and Theoretical Computer Science, 1990. Springer-Verlag Lecture Notes in Computer Science 472, pp. 215-224.

- [36] S. Rajasekaran, $k-k$ Routing, $k-k$ Sorting, and Cut Through Routing on the Mesh, Technical Report, Department of CIS, University of Pennsylvania, Philadelphia, PA 19104, October 1991.
- [37] S. Rajasekaran, Randomized Algorithms for Packet Routing on the Mesh, in *Advances in Parallel Algorithms*, Blackwell Scientific Publications, 1992, pp. 277-301.
- [38] S. Rajasekaran and T. McKendall, Permutation Routing and Sorting on the Reconfigurable Mesh, Technical Report MS-CIS-92-36, Department of Computer and Information Science, University of Pennsylvania, May 1992.
- [39] S. Rajasekaran and R. Overholt, Constant Queue Routing on a Mesh, in Proc. Symposium on Theoretical Aspects of Computer Science, 1990. Springer-Verlag Lecture Notes in Computer Science #480, pp. 444-455. Also in Journal of Parallel and Distributed Computing 15, 1992, pp. 160-166.
- [40] S. Rajasekaran, and M. Raghavachari, Optimal Randomized Algorithms for Multipacket and Cut Through Routing on the Mesh, Proc. IEEE Symposium on Parallel and Distributed Processing, Dallas, Texas, Dec. 1991. To appear in Journal of Parallel and Distributed Computing.
- [41] S. Rajasekaran and J.H. Reif, Derivation of Randomized Sorting and Selection Algorithms, Technical Report, Aiken Computing Lab., Harvard University, 1985.
- [42] S. Rajasekaran, and S. Sen, Random Sampling Techniques and Parallel Algorithms Design, in *Synthesis of Parallel Algorithms*, editor: Reif, J.H., Morgan-Kaufmann Publishers, San Mateo, California, 1992.
- [43] S. Rajasekaran and Th. Tsantilas, An Optimal Randomized Routing Algorithm for the Mesh and A Class of Efficient Mesh like Routing Networks, Proc. 7th Conference on Foundations of Software Technology and Theoretical Computer Science, 1987. Springer-Verlag Lecture Notes in Computer Science #287, pp. 226-241.
- [44] S. Rajasekaran and Th. Tsantilas, Optimal Routing Algorithms for Mesh-Connected Processor Arrays, *Algorithmica*, vol. 8, 1992, pp. 21-38.
- [45] J.H. Reif and L.G. Valiant, A Logarithmic Time Sort for Linear Size Networks, *Journal of the ACM*, 34(1), 1987, pp. 60-76.
- [46] R. Reischuk, Probabilistic Parallel Algorithms for Sorting and Selection, *SIAM Journal of Computing*, 14(2), 1985, pp. 396-411.
- [47] C.P. Schnorr and A. Shamir, An Optimal Sorting Algorithm for Mesh Connected Computers, in Proc. 18th ACM Symposium on Theory of Computing, 1986, pp. 255-263.

- [48] A. Schuster, Dynamic Reconfiguring Networks for Parallel Computers: Algorithms and Complexity Bounds, Ph.D. Thesis, Computer Science Department, Technion-Israel Institute of Technology, August 1991.
- [49] Q.F. Stout, Broadcasting on Mesh-Connected Computers, Proc. 1982 Conference on Information Science and Systems, pp. 85-90.
- [50] Q.F. Stout, Mesh-Connected Computers with Broadcasting, IEEE Trans. Computers 32, 1983, pp. 826-830.
- [51] Q.F. Stout, Meshes with Multiple Buses, Proc. IEEE Symp. on Foundations of Computer Science, 1986, pp. 264-273.
- [52] X. Thibault, D. Comte, and P. Siron, A Reconfigurable Optical Interconnection Network for Highly Parallel Architecture, Proc. Symposium on the Frontiers of Massively Parallel Computation, 1989.
- [53] C.D. Thompson and H.T. Kung, Sorting on a Mesh Connected Parallel Computer, Comm. ACM, 20 (1977) pp. 263-270.
- [54] L.G. Valiant, Parallelism in Comparison Problems, SIAM Journal on Computing, vol.14, 1985, pp. 348-355.
- [55] L.G. Valiant and G.J. Brebner, Universal Schemes for Parallel Communication, in Proc. 13th ACM Symposium on Theory of Computing, 1981, pp. 263-277.
- [56] U. Vishkin, An Optimal Parallel Algorithm for Selection Algorithm, Unpublished manuscript, 1983.
- [57] B.F. Wang, G.H. Chen, and F.C. Lin, Constant Time Sorting on a Processor Array with a Reconfigurable Bus System, Information Processing Letters, 34(4), April 1990.