October 1993

k-k Routing, *k-k* Sorting, and Cut Through Routing on the Mesh

Sanguthevar Rajasekaran
University of Pennsylvania

Abstract

In this paper we present randomized algorithms for *k-k* routing, *k-k* sorting, and cut through routing. The stated resource bounds hold with high probability. The algorithm for *k-k* routing runs in [*k/2*]*n+o*(*kn*) steps. We also show that *k-k* sorting can be accomplished within [*k/2*] *n+n+o*(*kn*) steps, and cut through routing can be done in [3/4]*kn*+[3/2]*n+o*(*kn*) steps. The best known time bounds (prior to this paper) for all these three problems were *kn+o*(kn).

[kn/2] is a known lower bound for all the three problems (which is the bisection bound), and hence our algorithms are very nearly optimal. All the above mentioned algorithms have optimal queue length, namely k+o(*k*). These algorithms also extend to higher dimensional meshes.

$k - k$ routing, $k - k$ Sorting, and Cut Through Routing On The Mesh

MS-CIS-91-93
GRASP LAB 290

Sanguthevar Rajasekaran

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

October 1991

$k - k$ Routing, $k - k$ Sorting, and Cut Through Routing on the Mesh

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

Abstract In this paper we present randomized algorithms for $k-k$ routing, $k-k$ sorting, and cut through routing. The stated resource bounds hold with high probability. The algorithm for $k - k$ routing runs in $\lceil \frac{k}{2} \rceil n + o(kn)$ steps, the time bound of the best known previous algorithm being $kn + o(kn)$ due to Rajasekaran & Raghavachari [22] (randomized) and Kunde [10](deterministic). We also show that $k - k$ sorting can be accomplished within $\lceil \frac{k}{2} \rceil n + n + o(kn)$ steps, and cut thorugh routing can be done in $\frac{3}{4}kn + \frac{3}{2}n + o(kn)$ steps. [10]'s algorithm for $k - k$ sorting has a time bound of $kn + o(kn)$, and [22]'s randomized algorithm for cut through routing runs in $kn + o(kn)$ time. These were the best known algorithms prior to this paper.

$\frac{kn}{2}$ is a known lower bound for all the three problems (which is the bisection width), and hence our algorithms are very nearly optimal. All the above mentioned algorithms have optimal queue length, namely $k + o(k)$. These algorithms also extend to higher dimensional meshes.

1 Introduction

1.1 Packet Routing

Fixed connection machines are some of the most practical models of parallel computing, as infered from the parallel computers available today. A fixed connection machine is usually represented as a directed graph whose nodes correspond to processing elements, and whose edges correspond to communication links. The speed of a parallel computer is determined by 1) the computing power of component processors, and 2) the speed of inter-processor communication. Nowadays the computing power of individual processing elements can be made arbitrarily high owing to the decline in hardware costs. Thus the speed of any parallel machine crucially depends on how fast the inter-processor communication is.

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* [9]. 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* [15].

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.

1.3 Mesh Connected Computers

The fixed connection machine assumed in this paper is the *Mesh Connected Computer*. The basic topology of a two dimensional Mesh is an $n \times n$ square grid with one processor per grid point (see figure 1). Except for processors at the boundary, every other processor is connected to its neighbors to the *left, right, above*, and *below* through bidirectional links. Variations in this topology are possible depending on whether one or more of the following connections are allowed : 1) vertical wrap arounds, 2) horizontal wrap arounds, and 3) connections to diagonal neighbors. In this paper we only consider the Mesh with the basic topology. 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.

Mesh connected computers (MCCs) have drawn the attention of computer scientists in recent times

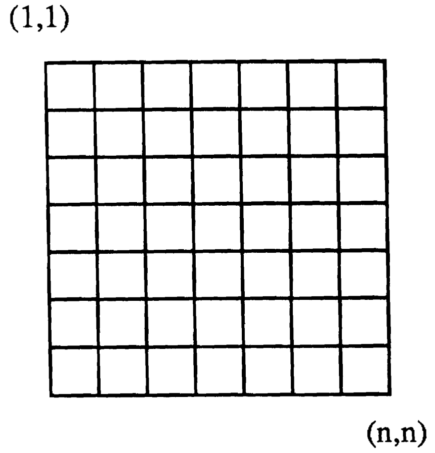

FIG.1. An $n \times n$ Mesh Connected Computer

because of their many special properties. Some of the special features of MCCs 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 Known and New Results

Many optimal algorithms (both deterministic and randomized) have been derived for store and forward routing. See e.g., [29, 24, 6, 8, 13, 21, 20]. The problem of multipacket routing on the Mesh was first studied by Kunde and Tensi [27] who presented an efficient algorithm for $k-k$ routing (with a time bound of $\frac{5}{4}kn + O(\frac{kn}{q})$ and a queue length of q (for any $1 \leq q \leq n$)).

Makedon and Simvonis [15] initiated the study of cut through routing on the Mesh and presented both a deterministic algorithm (with a time bound of $\frac{3}{2}kn + O(\frac{kn}{q})$) and a randomized algorithm (with a time bound of $kn + O(\frac{kn}{q})$), for a queue length of q (for any $1 \leq q \leq n$). Later Rajasekaran and Raghavachari [22] showed that both multipacket and cut through routing can be performed in $kn + O(k \log n)$ steps using a randomized algorithm, the queue length being $O(k)$. Recently Kunde [10] has presented an algorithm for $k-k$ routing whose time bound is $kn + o(kn)$ and whose queue length is k, for any $k \geq 4$. He also proved that a sequence of k permutations can be routed in $\lceil \frac{k}{2} \rceil n + o(kn)$ steps (for any $k \geq 8$) and obtained similar results for routing on r-dimensional meshes as well. However this algorithm can not be used for the general $k-k$ routing unless one is willing to perform certain 'off-line' computing. In this paper we present an algorithm for $k-k$ routing with a run time of $\lceil \frac{k}{2} \rceil n + o(kn)$ and an algorithm for cut through routing with a run time of $\frac{3}{4}kn + \frac{3}{2}n + o(kn)$, for any $k \geq 8$. Both these algorithms are randomized and need a queue length of only $k + o(k)$, with high probability.

As far as $k-k$ sorting on the Mesh is concerned, several optimal algorithms can be found in the literature for $1-1$ sorting [27, 7, 26, 14, 12] (all of which have a run time of $3n + o(n)$). Recently Kaklamanis, Krizanc, Narayanan, and Tsantilas [4] showed that $1-1$ sorting can be accomplished within $2.5n + o(n)$ steps and constant queue length using a randomized algorithm. Later, Kunde [10] matched this time bound with a deterministic algorithm and a queue length of 2. Park and Balasubramanian [18] proved that $2-2$ sorting can be performed on an $n \times n$ Mesh in an optimal $3n+o(n)$ steps. Subsequently Kunde [10] gave an algorithm for $k-k$ sorting that runs in $\lceil k/4 \rceil 2n + kn/2 + o(kn)$ time with a queue length of k, for any $k \geq 4$. We give in this paper a randomized algorithm for $k-k$ sorting which runs in $\lceil \frac{k}{2} \rceil n + n + o(kn)$ time and which has a queue length of $k + o(k)$, for any $k \geq 8$. The queue length of our routing and sorting algorithms is only $k + O(1)$ if $k = O(n^{\nu})$ for some constant $\nu < 1$. In practice k is usually a constant and hence it may be safe to assume this queue length to be $k + O(1)$.

We also show that our routing and sorting algorithms apply to higher dimensional Meshes. Several algorithms exist for off-line routing (see e.g., [2, 5, 17, 19]). In [11], Leighton analyzes the expected behavior of certain greedy algorithms for packet routing. For an excellent treatise on sorting and routing algorithms for the Mesh, the reader is refered to Leighton [12]. Since $\frac{kn}{2}$ is a lower bound for all the three problems we consider in this paper [9, 10], our algorithms are very nearly optimal.

1.5 Some Definitions

We say a randomized algorithm uses $\widetilde{O}(g(n))$ 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. Similar definitions apply to $\widetilde{o}(g(n))$ and other such 'asymptotic' functions.

By *high probability* we mean a probability of $\geq 1 - n^{-\alpha}$ for any $\alpha \geq 1$ (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 [3] 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 The Queue Line Lemma

In the process of packet routing in a network, the time taken by any packet to reach its destination is dictated by two factors: 1) the *distance* between the packet's origin and destination, and 2) the number of steps (also called the *delay*) the packet waits in queues. The *Queue Line Lemma* enables one to compute an upper bound on the delay of any packet.

Consider the set of paths $\mathcal{P}$ taken by the packets. Two packets are said to *overlap* if they share at least one edge in their paths. The set of paths is said to be *nonrepeating* if for any two paths in $\mathcal{P}$, the following statement holds: If these two paths meet, share some successive edges, and diverge, then they will never meet again. The following lemma is due to Valiant and Brebner [29]:

Lemma 2.1 *The amount of delay any packet q suffers waiting in queues is no more than the number of distinct packets that overlap with q, provided the set of paths taken by packets is nonrepeating.*

3 Routing on a Linear Array

In this section we study different routing problems on a linear array. These results will help us analyze routing algorithms on the Mesh. As will be shown, routing on a Mesh can be broken into a constant number of phases, where in each phase routing is performed either along the rows or along the columns.

Problem 1. Each node of a linear $n-$array has $\leq k$ packets initially and each node is the destination of $\leq k$ packets. Send all the packets to their destinations sending at the most one packet along any edge in a single step.

The following lemma can be proved using the proof technique of [24] and [9]:

Lemma 3.1 *If we use the furthest destination first priority scheme, Problem 1 can be solved in time $\leq \frac{kn}{2}$.*

Problem 2 The number of packets destined for any successive i nodes of a linear array is $\leq ki + f(n)$. Route the packets.

The following lemma also can be proven along the same lines as that of lemma 3.1.

Lemma 3.2 *If we use the furthest destination first priority scheme, Problem 2 can be solved in $\frac{kn}{2} + f(n)$ steps.*

4 A Simple Algorithm for $k - k$ Routing and Cut Through Routing

In this section we show that both $k - k$ routing and cut through routing can be accomplished within roughly $\frac{3}{4}kn + \tilde{o}(kn)$ steps on an $n \times n$ Mesh, the queue length being $k + \tilde{o}(k)$.

The algorithm to be presented is very similar to the original algorithm of Valiant and Brebner [29]. There are three phases in the algorithm. Let q be any packet whose origin is (i, j) and whose destination is (r, s).

In 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 upto this node. In phase II, q travels along row i' upto column s. Finally, in phase III the packet reaches its destination traversing along column s. Use the furthest destination first priority scheme for all the three phases.

Lemma 4.1 *The above algorithm has a run time of $\frac{3}{2}kn + \tilde{o}(kn)$ (when applied to $k - k$ routing), the queue length being $k + \tilde{o}(k)$.*

Proof. In phases I and II, the number of packets that are destined for any successive i nodes is $B(kn, i/n)$ and $B(kin, 1/n)$ respectively. Using Chernoff bounds, this number is $ki + \tilde{O}(\sqrt{kn \log n})$. In phase III the number of packets destined for any successive i nodes is exactly ki. Thus phase I and phase II can be completed in $\frac{kn}{2} + \tilde{o}(kn)$ steps each. Also phase III can be done within $\frac{kn}{2}$ steps (cf. lemma 3.2).□

Random Coloring. The time bound of the above algorithm can be improved to $\frac{3}{4}kn + \tilde{o}(kn)$ using the following trick (which has been used in previous works [9, 15, 22]). Realize that the above algorithm is *uniaxial*, i.e., at any given time either only the row edges are used or the column edges are used. The idea is to make use of the unused edges also.

At the beginning color each packet as white or black by flipping a 2-sided unbiassed coin. The white packets use the above mentioned algorithm without any modification, where as the black packets exploit the unused edges. To be more precise, in phase I, a black packet 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 black and white packets. Also, both the number of black packets and the number of white packets is $B(kn^2, 1/2)$. Thus w.h.p. these two numbers will be nearly the same.

Further, the number of black (white) packets that will participate in the row routing of phases I and III (phase II) is $B(kn, 1/2)$ each. Also, the number of white (black) packets that participate in column routing of phases I and III (phase II) is $B(kn, 1/2)$ each. Thus using lemma 3.2 we can show that each of the three phases can be completed in $\frac{kn}{4} + \tilde{o}(kn)$ steps.

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 [24] 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 ideas employed in the next two sections. Therefore we have the following

Theorem 4.1 *$k - k$ routing can be completed in $\frac{3}{4}kn + \tilde{o}(kn)$ steps on an $n \times n$ Mesh, the queue length being $k + \tilde{o}(k)$.*

Corollary 4.1 *Cut through routing can be performed using the above algorithm within $\frac{3}{4}kn + \frac{3}{2}n + \tilde{o}(kn)$ steps, the queue length being $k + \tilde{o}(k)$.*

Proof. The only change in the analysis is that in Lemma 3.2, the time bound for cut through routing is $\frac{kn}{2} + \frac{n}{2} + f(n)$ (cf. Lemma 3.1 in [22]).

5 An Optimal Algorithm for $k - k$ Routing

In this section we show that $k - k$ routing can be performed within $\lceil \frac{k}{2} \rceil n + \tilde{o}(kn)$ steps, for any $k \geq 8$. The main idea behind the algorithm is the concentrated regular data streams introduced by Kunde [10]. In particular, the following theorem due to Kunde is an essential part of our algorithm. We also show that the same algorithms are applicable on higher dimensional Meshes.

Theorem 5.1 *A sequence of k permutations can be routed on an $n \times n$ Mesh such that the time bound is $\lceil \frac{k}{2} \rceil n + o(kn)$ and the queue length is k (for any $k \geq 8$).*

Consider the general $k - k$ routing problem. There are exactly k packets starting from any node and exactly k packets destined for any node. Using Hall's theorem [12], it follows that one can decompose the general $k - k$ routing problem into a sequence of k permutations. But then it involves a certain amount of off-line work. Here lies the difficulty in applying theorem 5.1 for the general $k - k$ routing.

To begin with, say we color each packet in a node with a random color (from a collection of k colors; call these colors $1, 2, \ldots, k$). One would expect that packets of the same color form 'more or less' a permutation. This is the main idea behind our algorithm. Another difficulty in using theorem 5.1 is that this algorithm is applicable only if we have a sequence of k **full** permutations. This is not a serious problem either as we will show.

For the problems of $k - k$ sorting and $k - k$ routing, the packets can be assumed to form k disjoint layers [9, 10]. Any indexing should be defined on the triple (ℓ, r, c) where r and c are the row and column numbers of any node and ℓ is the layer within this node.

The Algorithm

Step 1

Color the k packets in each node randomly from out of the colors $1, 2, \ldots, k$.

Step 2

Partition each row into slices of n^ϵ (for some constant $\epsilon < 1$) successive nodes each. Sort the packets in each slice w.r.t. to their colors, the indexing used being lexicographic of the tuple (ℓ, c).
(* The effect of this sorting is to distribute all the packets of the same color uniformly among the n^ϵ nodes. This can be done in $O(kn^\epsilon)$ steps. Each color corresponds to a layer. *)

Step 3

If there are more than one packets of the same color in any node, retain only one packet of this color. Call all the retained packets as *special packets*. Call the rest of the packets as *stray packets*.
(* This takes k time units. W.h.p., the number of stray packets in any slice will be $\leq kn^{\epsilon'}$, for some constant $\epsilon' < \epsilon$ (as shown in lemma 5.1). *)

Step 4

In any node if there are less than k special packets, create *dummy* packets of appropriate colors such that there is exactly one packet (either special or dummy) of each color in every node. A dummy packet has a random destination in the Mesh.
(* This can also be done in k units of time. *)

Step 5

Send each stray packet to a random node in its slice.
(* This can be done in $kn^{\epsilon'}$ steps w.h.p. *)

Step 6

Route all the non-stray packets (there are exactly k of them in each node) using Kunde's algorithm [10]. Notice that non-stray packets of the same color still need not form a full permutation. Kunde's algorithm for routing a sequence of k permutations indeed sorts the packets w.r.t. their destination addresses (in some appropriate indexing scheme). Non-stray packets use the same algorithm, i.e., they are sorted. At the end of this sorting step, each special packet will be most n^{δ} steps away from its final destination, for some constant $\delta < 1$ (as will be shown in Lemma 5.3).
(* Sorting can be performed in $\lceil \frac{k}{2} \rceil n + o(kn)$ steps (see Theorem 5.1), the queue length being k. *)

Step 7

After the above sorting step, delete all the dummy packets and send each special packet to its correct destination.
(* This task can be accomplished using any of the existing sorting or routing algorithms in $O(kn^{\delta})$ steps, since any such sorting or routing is only local to a sub-mesh of size $n^{\delta} \times n^{\delta}$. The queue length can also be kept as k. *)

Step 8

Route the stray packets using the greedy algorithm. If a packet originates from (i, j) whose destination is (r, s), it traverses along row i upto (i, r)

and then travels along column s upto (r, s).
(* This routing can be done in $2n + \tilde{o}(kn)$ steps with a queue length of $\tilde{o}(k)$ if there were no other packets in the Mesh (see Lemma 5.4 below). *)

The correctness of the above algorithm is quite clear. The fact that its time bound is $\lceil \frac{k}{2} \rceil n + o(kn)$ is established in the following sequence of lemmas.

Lemma 5.1 *The number of stray packets in any slice of n^ϵ nodes is $\leq kn^{\epsilon'}$ for some constant $\epsilon' < \epsilon$.*

Proof. Consider any color i. The number of packets of color i in a given slice is $B(kn^\epsilon, 1/k)$. The expectation of this random variable is n^ϵ. Using Chernoff bounds, this number is $n^\epsilon + \tilde{O}(\sqrt{n^\epsilon \log n})$. □

As an immediate consequence of the above lemma we get the following

Lemma 5.2 *The number of stray packets along any row or any column (at the end of Step 5) is $\leq kn^\gamma$, for some constant $\gamma < 1$ w.h.p.*

Lemma 5.3 *After the sorting done in Step 6, each special packet will be at the most n^δ (for some constant $\delta < 1$) distance away from its final destination.*

Proof. The indexing scheme used in Kunde's algorithm is 'block-wise snake-like row major', where the block is of size $n^{2/3} \times n^{2/3}$.

Consider a special packet q whose destination node is (j, s). After the sorting in Step 6, q can only be displaced from its actual destination by dummy packets whose destination is in row j or below. But the total number of stray packets in the whole Mesh is $\leq n^{4/3}$ w.h.p. for a proper choice of ϵ. Thus q can only be displaced by one block in the worst case w.h.p. □.

The following lemma shows that the stray packets can be routed easily using the greedy algorithm so as to ensure $\tilde{o}(k)$ queue length, provided there are no other packets in the Mesh.

Lemma 5.4 *In an $n \times n$ Mesh with k packets in each node, let each packet be selected for routing with probability $\frac{1}{n^\psi}$ (for some constant $0 < \psi < 1$). Let a similar statement hold for the packets destined for any node. Then, these packets can be routed in $2n + \tilde{o}(kn)$ steps, the queue length being $\tilde{o}(k)$, using the greedy algorithm.*

Proof. is similar to that of Theorem 5.1 in [24]. □

The above algorithm as mentioned has a time bound of $\lceil \frac{k}{2} \rceil n + \tilde{o}(kn) + 2n$. But as such, it is assumed that Step 6 and Step 8 are disjoint. We can overlap these two steps and hence obtain the following

Lemma 5.5 *If Step 6 and Step 8 in the above algorithm are overlapped, the time bound can be reduced to $\lceil \frac{k}{2} \rceil n + \tilde{o}(kn)$.*

Proof. Give the highest priority to stray packets. The number of stray packets that will traverse along any column or any row is $\leq kn^{\gamma}$ w.h.p. according to lemma 5.2. Therefore the maximum delay that any non-stray packet suffers because of stray packets is $\widetilde{O}(kn^{\gamma})$ (for some constant $\gamma < 1$). Also, the queue length increase due to overlap is only $\widetilde{o}(k)$. □

The above results yield the following

Theorem 5.2 *$k - k$ routing on an $n \times n$ Mesh can be performed within $\lceil \frac{k}{2} \rceil n + \widetilde{o}(kn)$ steps, the queue length being $k + \widetilde{o}(k)$.*

Corollary 5.1 *If $k = O(n^{\nu})$ for some constant $\nu < 1$, the queue length of the above algorithm is only $k + \widetilde{O}(1)$.*

Proof. If $kn^{\epsilon'}$ (see Lemma 5.1) is $o(1)$, the number of stray packets in any node at the end of Step 5 is only $\widetilde{O}(1)$. Also notice that Step 5 could be performed before Step 4. □

The following theorem pertains to $k - k$ routing on r-dimensional Meshes.

Theorem 5.3 *$k - k$ routing on an $r-$ dimensional Mesh can be performed within $\lceil \frac{k}{r} \rceil \frac{rn}{2} + \widetilde{O}(krn^{(r-1)/r})$ steps, the queue length being $k + \widetilde{o}(k)$. If $k = O(n^{\nu})$, the queue length is only $k + \widetilde{O}(1)$.*

Proof. will appear in the final version.

6 $k - k$ Sorting on the Mesh

The problem is to sort a Mesh in which there are k packets to start with in each node. Many optimal algorithms have been proposed in the literature for $1-1$ sorting under various Mesh models [27, 7, 26, 14]. Park and Balasubramanian [18] proved that $2 - 2$ sorting can be completed in $3n + o(n)$ steps. Later Kunde [10] showed that $k - k$ sorting can be done in $kn + o(kn)$ steps, keeping the queue length as k. The indexing used is layer last blockwise snake-like row-major ordering. We also assume the same indexing scheme.

In this section we show that $k-k$ sorting can be accomplished within $\lceil \frac{k}{2} \rceil n + \widetilde{o}(kn) + n$ steps, for any $k \geq 8$. The queue length is $k + \widetilde{o}(k)$. If $k = O(n^{\nu})$ for some $\nu < 1$, the queue length is only $k + \widetilde{O}(1)$. We will make use of the optimal $k - k$ routing algorithm presented in section 5 as a subroutine. The time bound of our algorithm very nearly matches the bisection width of $\frac{kn}{2}$.

Summary. Random sampling has played a vital role in the design of parallel algorithms for comparison problems (including sorting and selection). Reischuk's [25, 23] sorting algorithm is a good example. Given n keys, the idea is to: 1) randomly sample n^{ϵ} (for some constant $\epsilon < 1$) keys, 2) sort this sample (using any nonoptimal algorithm), 3)partition the input using the sorted sample as splitter keys, and 4) to sort each part separately in parallel. Similar ideas have been used in many other works as well (see e.g., [23]).

Let $X = k_1, k_2, \ldots, k_n$ be a given sequence of n keys and let $S = \{k'_1, k'_2, \ldots, k'_s\}$ be a random sample of s keys 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 [25, 23] 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 $\widetilde{O}(\frac{n}{s+1} \log n)$.*

Kaklamanis, Krizanc, Narayanan, and Tsantilas [4] recently implemented a similar algorithm on the Mesh to show that $1-1$ sorting can be done on the Mesh in $2.5n$ steps (for a constant queue length). Next we describe our algorithm which is similar to that of [4]. A random sample is chosen and sorted. The sorted sample is broadcast to the whole Mesh, using which each key can compute an approximate rank. Now each key is routed to its approximate destination using the algorithm of section 5. By then, the global rank of each splitter key is computed and broadcast. The global ranks of the splitter keys enable each key to compute its global rank. Finally, every key is sent to its actual destination (which will be very close to its approximate destination w.h.p.)

Algorithm $k-k$ Sorting

Step 1

Each key includes itself as a sample key in S with probability $\frac{1}{n^{\epsilon}}$, for some constant $\epsilon < 1$. Concentrate all the sample keys in the middle of the Mesh in a block of size $n^{\delta} \times n^{\delta}$ (where δ is a constant $> 1 - \frac{\epsilon}{2}$), and sort this block.
(* The number of keys in the sample is $k\widetilde{\theta}(n^{2-\epsilon})$. The cocentration can be done in $0.5n + \widetilde{o}(kn)$ steps, the queue length being $\widetilde{o}(k)$. Sorting takes $O(kn^{\delta})$ steps. *)

Step 2

Partition the Mesh into blocks of size $n^{\delta'} \times n^{\delta'}$ for some $\delta' > \delta$. Broadcast a copy of the sorted *sample block* to all the blocks of the Mesh. Sort all the keys in each block.
(* Broadcasting takes $0.5n + \widetilde{o}(kn)$ steps. Sorting can be performed in $O(kn^{\delta'})$ steps. The queue increases in each block by $\widetilde{o}(k)$ because of the sample keys. *)

Step 3

Perform a segmented prefix sum operation within each block to determine an approximate rank for each key and also compute the partial rank of each splitter key in this block.
(* Can be done in $O(kn^{\delta'})$ steps. After this step, each key knows the X_j it belongs to. *)

Step 4

Compute the global rank of each splitter key and broadcast this information to each block in the Mesh.
(* This is done by summing up the partial ranks computed in Step 3 for each sample key. Summing is done as the sample blocks traverse toward the center of the Mesh. Within $0.5n + \widetilde{o}(kn)$ steps the global ranks of all the splitter keys will be available at the center. In another $0.5n + \widetilde{o}(kn)$ steps these ranks can be broadcast (as a block). *)

Step 5

Using the $k - k$ routing algorithm of section 5 route each packet to a random node in a block that corresponds to the approximate rank of the key.
(* This can be shown to take $\lceil \frac{k}{2} \rceil n + \widetilde{o}(kn)$ steps, the queue length being $k + \widetilde{o}(k)$. At the end of this step each packet is at most one block away from its actual destination w.h.p. *)

Step 6

Sort each block and make use of the global ranks of the splitter keys to compute the global rank of each key in the block.
(* Can be completed in $O(kn^{\delta'})$ time units. *)

Step 7

Route each packet to its actual destination.
(* This step takes $O(kn^{\delta'})$ time. *)

The following lemmas establish the correctness of the algorithm.

Lemma 6.2 *The number of keys in the sample is $k\widetilde{\theta}(n^{2-\epsilon})$.*

Lemma 6.3 *The approximate destination computed in Step 3 for each key is at the most one block away from the actual destination of the key.*

Proof. This is a consequence of Lemma 6.1. □

Theorem 6.1 *The above algorithm for $k - k$ sorting runs in $\lceil \frac{k}{2} \rceil n + \widetilde{o}(kn) + n$ steps, the queue length being $k + \widetilde{o}(k)$.*

Proof Sketch. As stated the algorithm seems to take $\lceil \frac{k}{2} \rceil n + \widetilde{o}(kn) + 2n$ time. But we can overlap Step 4 and Step 5 giving the highest priority to the sample block. Also notice that the sample block has only $\widetilde{o}(k)$ keys per node.

If Step 4 and Step 5 are overlapped and the highest priority is given to sample block, the additional queue length increase is only $\widetilde{o}(k)$. More over, the additional delay any packet suffers in Step 5 due to the sample block is only $O(n^{\delta})$. Thus n time units can be saved because of overlapping. □

Corollary 6.1 *If $k = O(n^{\nu})$ for some constant $\nu < 1$, the queue length of the above sorting algorithm is only $k + \widetilde{O}(1)$.*

Proof. If $n^{2\delta}$ is asymptotically larger than $kn^{2-\epsilon}$, the number of keys in the sample block in Step 1 (and after) is $\widetilde{O}(1)$. $\square$

Similar bounds can be obtained for $k-k$ sorting on an $r-$dimensional Mesh as well.

Theorem 6.2 *$k-k$ sorting on an $r-$ dimensional Mesh can be performed within $\lceil \frac{k}{r} \rceil \frac{rn}{2} + \widetilde{O}(krn^{(r-1)/r})$ steps, the queue length being $k + \widetilde{o}(k)$. If $k = O(n^{\nu})$, the queue length is only $k + \widetilde{O}(1)$.*

Proof. will appear in the final version.

7 Conclusions

In this paper we have presented randomized algorithms for $k-k$ routing, $k-k$ sorting, and cut through routing on the Mesh. These algorithms have time bounds which match or very nearly match the bisection width of $\frac{kn}{2}$. An interesting open problem is if there exist deterministic algorithms with similar behavior.

References

[1] Angluin, D., and Valiant, L.G., 'Fast Probabilistic Algorithms for Hamiltonian Paths and Matchings,' Journal of Computer and Systems Science, 18, 1979, pp. 155-193.

[2] Annexstein, F., and Baumslag, M., 'A Unified Approach to Off-Line Permutation Routing on Parallel Networks,' Proc. Symposium on Parallel Algorithms and Architectures, pp. 398-406, July 1990.

[3] Chernoff, H., '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.

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

[5] Krizanc, D., 'A Note on Off-line Permutation Routing on a Mesh Connected Processor Array,' Proc. International Conference on Computing and Information, 1991, pp. 418-421.

[6] Krizanc, D., Rajasekaran, S., and Tsantilas, T. 'Optimal Routing Algorithms for Mesh Connected Processor Arrays,' Proc. VLSI Algorithms and Architectures: AWOC, 1988. Springer-Verlag Lecture Notes in Computer Science #319, pp. 411-22. To appear in Algorithmica.

[7] Kumar, M., and Hirschberg, D., 'An Efficient Implementation of Batcher's Odd-Even Merge Algorithm and its Application to Parallel Sorting Schemes,' IEEE Trans. Comp., 32 (1983).

[8] Kunde, M. 'Routing and Sorting on Mesh Connected Processor Arrays,' Proc. VLSI Algorithms and Architectures: AWOC 1988. Springer-Verlag Lecture Notes in Computer Science #319, Springer-Verlag, pp. 423-33.

[9] Kunde, M., and Tensi, T. 'Multi-Packet Routing on Mesh Connected Arrays,' Proc. ACM Symposium on Parallel Algorithms and Architectures, 1989, pp. 336-343.

[10] Kunde, M., 'Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound,' to be presented in the IEEE Symposium on Foundations of Computer Science, 1991.

[11] Leighton, T., 'Average Case Analysis of Greedy Routing Algorithms on Arrays,' Proc. ACM Symposium on Parallel Algorithms and Architectures, pp. 2-10, July 1990.

[12] Leighton, T., *Introduction to Parallel Algorithms and Architectures: Arrays–Trees–Hypercubes*, Morgan-Kaufmann Publishers, San Mateo, California, 1992.

[13] Leighton, T., Makedon, F., and Tollis, I.G. 'A $2n-2$ Step Algorithm for Routing in an $n \times n$ Array With Constant Size Queues,' Proc. ACM Symposium on Parallel Algorithms and Architectures, 1989, pp. 328-35.

[14] Ma,Y., Sen, S., Scherson, D., 'The Distance Bound for Sorting on Mesh Connected Processor Arrays is Tight,' In Proc. 27th IEEE Symposium on Foundations of Computer Science, 1986, pp. 255–263.

[15] Makedon, F., Simvonis, A., 'On bit Serial packet routing for the mesh and the torus.' Proc. third Symposium on Frontiers of Massively Parallel Computation, 1990, pp. 294-302.

[16] Makedon, F., and Simvonis, A., 'Fast Parallel Communication on Mesh Connected Machines with Low Buffer Requirements,' Proc. International Conference on Computer Design, 1990.

[17] Nassimi, D., and Sahni, S., 'An Optimal Routing Algorithm for Mesh Connected Parallel Computers,' J. ACM, 27 (1980) pp. 6–29.

[18] Park, A., and Balasubramanian, K., 'Reducing Communication Costs for Sorting on Mesh-Connected and Linearly Connected Parallel Computers,' J. of Parallel and Distributed Computing, 9, 1990, pp. 318-322.

[19] Raghavendra, C.S., and Kumar, V.K.P., 'Permutations on Illiac IV-Type Networks,' IEEE Trans. Comp., 35 (1986) pp. 662–669.

[20] Rajasekaran, S., 'Randomized Algorithms for Packet Routing on the Mesh,' to appear in *Advances in Parallel Algorithms*, Blackwell Scientific Publications, 1991.

[21] Rajasekaran, S., and Overholt, R. 'Constant Queue Routing on a Mesh,' Proc. Symposium on Theoretical Aspects of Computer Science, 1990. Springer-Verlag Lecture Notes in Computer Science #480, pp. 444-455. To appear in JPDC.

[22] Rajasekaran, S., and Raghavachari, M., 'Optimal Randomized Algorithms for Multipacket and Cut Through Routing on the Mesh,' to be presented in the IEEE Symposium on Parallel and Distributed Processing, Dallas, Texas, Dec. 1991.

[23] Rajasekaran, S., and Sen, S., 'Random Sampling Techniques and Parallel Algorithms Design,' in *Synthesis of Parallel Algorithms*, editor: Reif, J.H., Morgan-Kaufmann Publishers, San Mateo, California, 1992.

[24] Rajasekaran, S., and Tsantilas, T. '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.

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

[26] Schnorr, C.P., and Shamir, A., 'An Optimal Sorting Algorithm for Mesh Connected Computers,' In Proc. 18th ACM Symposium on Theory of Computing, 1986, pp. 255-263.

[27] Thompson, C.D., and Kung, H.T., 'Sorting on a Mesh Connected Parallel Computer,' Comm. ACM, 20 (1977) pp. 263-270.

[28] Valiant, L.G., 'A Scheme for Fast Parallel Communication,' SIAM J. Comp. 11 (1982), pp. 350-361.

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