Derandomizing Algorithms for Routing and Sorting on Meshes

Michael Kaufmann* Jop F. Sibeyn† Torsten Suel‡

Abstract

We describe a new technique that can be used to derandomize a number of randomized algorithms for routing and sorting on meshes. We demonstrate the power of this technique by deriving improved deterministic algorithms for a variety of routing and sorting problems. Our main results are an optimal algorithm for k-k routing on multi-dimensional meshes, a permutation routing algorithm with running time $2n + o(n)$ and queue size 5, and an optimal algorithm for 1-1 sorting.

1 Introduction

One of the main problems in the simulation of idealistic parallel computers by realistic ones is the problem of message routing through the sparse network of links connecting a set of processing units (PUs) among each other. In this paper, we consider the case of the $n \times n$ *mesh*, in which n^2 PUs are connected by a regular two-dimensional grid of bidirectional communication links. There may also be additional *wrap-around connections* between the two PUs at opposite ends of each row and each column of the network; this type of mesh is called *torus*. A mesh without wrap-around connections will be referred to as *square*. We assume the *MIMD model*, where in a single *step* each PU can perform an arbitrary amount of internal computation, and transmit one packet of information (of bounded length) to each of its neighbors.

The *routing problem* is the problem of rearranging a set of information packets in a network, such that every packet ends up at the PU specified in its destination address. This problem is of fundamental importance in the design of efficient algorithms for realistic models of parallel computation, as well as in the simulation of more powerful, idealistic models. Consequently, the routing problem has received considerable attention, and a variety of algorithms have been proposed for several variants of the problem. The performance of a routing algorithm is measured by its *running time* (the maximum time a packet may need to reach its destination) and its *queue size* (the maximum number of packets any node may have to store during the routing).

Most research has focused on the 1-1 routing problem, also called the *permutation routing problem*, in which each node is the origin and destination of at most one packet. However, in many practical applications a PU may have to communicate with a number of other PUs at the same time. This motivates the definition of the k-k routing problem, in which each PU can be the source and destination of up to k packets.

Another problem that involves the rearranging of packets within a processor network is the *sorting problem*. Again, several variants of the problem have been studied. In the 1-1 sorting problem, each PU initially holds a single packet, where each packet contains a key drawn from a totally ordered set. The packets have to be rearranged such that the packet with the key of rank i is moved to the PU with index i, for all i. In the k-k sorting problem, each PU is the source and destination of k packets.

In a routing problem, the destinations of the packets are given as part of the input; in a sorting problem, the destinations of the packets have to be computed as a function of the set of keys and the indexing of the PUs. However, there is also a close relationship between the two problems. Many routing algorithms involve the sorting of subsets of the packets, while many sorting algorithms use routing in intermediate steps of the computation.

1.1 Previous Work. A number of algorithms for the permutation routing problem have been proposed. A deterministic algorithm with running time $2 \cdot n - 2$ and constant queue size was given by Leighton, Makedon, and Tollis [14] and later refined by Rajasekaran and Overholt [15] and by Chlebus, Kaufmann, and Sibeyn [3], who achieve a queue size of 81. However, in addition to their large queue sizes, these algorithms suffer from a complicated control structure. Thus, a

*Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D-72076 Tübingen, Germany. E-mail: mk@informatik.uni-tuebingen.de

†Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany. E-mail: jopsi@mpi-sb.mpg.de. Partially supported by EC Cooperative Action IC-1000 (Project ALTEC: Algorithms for Future Technologies).

‡Department of Computer Sciences, University of Texas at Austin, Austin, TX 78712, USA. E-mail: torsten@cs.utexas.edu. Supported by Texas Advanced Research Program (TARP) Award #003658480.

simpler algorithm with a smaller queue size might be of more practical interest, even if its running time is slightly larger than $2 \cdot n - 2$. In this context, several fairly simple randomized algorithms with running time $2 \cdot n + \mathcal{O}(\log n)$ and small constant queue size have been proposed [7, 16].

Considerable attention has also been given to the problem of 1-1 sorting on two-dimensional meshes. In particular, Schnorr and Shamir [17] gave a $3 \cdot n + o(n)$ step algorithm for sorting into snake-like row-major order, and proved a nearly matching lower bound of $3 \cdot n - o(n)$, also independently discovered by Kunde [10]. In their model of the mesh, a PU may only hold a single packet at any time. However, this lower bound does not hold when PUs may hold more than one packet. For this model, Kaklamanis, Krizanc, Narayanan, and Tsantilas [5] gave a randomized $2\frac{1}{2} \cdot n + o(n)$ step algorithm for sorting into a block-wise indexing scheme. Kaklamanis and Krizanc [6] subsequently gave an improved version of this algorithm that runs in time $2 \cdot n + o(n)$, thus nearly matching the diameter lower bound. The best deterministic algorithm, due to Kunde [11], achieves a running time of $2\frac{1}{2} \cdot n + o(n)$ and a queue size of 2.

In [11], Kunde also showed that *k-k* sorting can be performed in $k \cdot n + o(k \cdot n)$ steps with a queue size of k. A randomized algorithm for *k-k* routing with running time $\max\{4 \cdot n, k \cdot n/2\} + o(k \cdot n)$ was given in [8]. This algorithm was improved and extended to sorting on squares and tori in [18], where it is shown that *k-k* sorting can be performed by a randomized algorithm in time $\max\{4 \cdot n, k \cdot n/2\} + o(k \cdot n)$ on a square, and in about half this time on a torus. This algorithm is based on the idea of Reif and Valiant and of Reischuk to randomly select a set of *splitters*. After sorting these splitters, the packets can estimate their rank and determine a corresponding preliminary destination.

1.2 Overview of the Paper.

We introduce a set of techniques that can be used to convert a number of randomized algorithms for routing and sorting into deterministic algorithms that achieve similar running times and queue sizes. Our techniques are very general, and seem to apply to many of the randomized algorithms for routing and sorting on meshes and related networks that have been proposed in the literature. We demonstrate the power of the technique by deriving improved deterministic algorithms for a variety of routing and sorting problems on meshes.

Our first result is an optimal algorithm for *k-k* sorting, with a running time of $k \cdot n/2 + (4 + o(1)) \cdot k^{5/6} \cdot n^{2/3}$ on a square and $k \cdot n/4 + (4 + o(1)) \cdot k^{5/6} \cdot n^{2/3}$ on a torus. The algorithm achieves a queue size of k, and does not make any copies of packets. In addition, it can easily be generalized to meshes of arbitrary dimensions. A similar algorithm was independently discovered by Kunde [12]. However, the lower order terms of his algorithm are larger, particularly for higher dimensions.

Next, we apply our techniques to several randomized routing algorithms recently proposed by Kaklamanis, Krizanc, and Rao [7]. As a result, we obtain an optimal deterministic routing algorithm for the square with a running time of $2 \cdot n + o(n)$ and a queue size of 5, as well as the first optimal deterministic algorithms for routing on the torus and the three-dimensional cube.

Finally, we demonstrate the power of our techniques by derandomizing an optimal randomized algorithm for 1-1 sorting on the square recently proposed by Kaklamanis and Krizanc [6]. The resulting deterministic algorithm has a running time of $2 \cdot n + o(n)$ and a queue size of about 25. In addition, we also obtain improved sorting algorithms for the torus and for three-dimensional networks.

Due to space constraints, we can only give a brief description of the main results. For complete proofs, and additional applications of our techniques, the reader is referred to [9, 19].

The remainder of the paper is organized as follows. The next section contains some additional definitions and useful results. In Section 3, we explain the basic idea behind our results, and show how it can be used in the construction of an optimal deterministic algorithm for *k-k* sorting. Section 4 contains our results for 1-1 routing. In Section 5, we present the optimal algorithm for 1-1 sorting on the two-dimensional mesh. Finally, Section 6 offers some concluding remarks.

2 Preliminaries

2.1 Machine Model.

Our model of computation is an $n \times n$ MIMD mesh with or without wrap-around connections. In the following, we refer to this machine simply as *mesh*. It consists of n^2 PUs arranged in a regular square grid, such that every PU is connected to (at most) four other PUs. The PU located at position (i, j) is referred to as $P_{i,j}$, where $P_{0,0}$ is the PU in the lower left corner. A one-dimensional square is called *chain*, and a square of dimension $d > 2$ is called *cube*.

The PUs operate in a synchronous fashion. In a single step of the computation, each PU can perform an arbitrary amount of internal computation, and communicate with all its neighbors. The only restriction is that at most one packet of bounded length can be transmitted across any edge in either direction. Thus, a PU in a square may send and receive up to four packets in a single step. We will assume that the running time of an algorithm is determined by the number of communication steps, and that the internal operations inside the PUs take a negligible amount of time (or can be performed simultaneously with the routing). Each PU has a *queue*, which temporarily stores packets that are not routed on immediately.

2.2 Indexing Schemes. In routing algorithms, the choice of the indexing of the PUs is irrelevant. In contrast, the running time of a sorting algorithm often depends on a particular indexing scheme. In this paper, we assume a so-called *blocked indexing scheme*, in which the mesh is partitioned into blocks of equal size, and the index of a PU is determined in the first place by the index of its block and in the second place by its index within this block. Under a *blocked snake-like row-major scheme*, the blocks are indexed by the snake-like row-major scheme, while the indexing inside the blocks can be arbitrary. As an example, we give a blocked snake-like row-major scheme of a 6×6 mesh with blocks of size 2×2:

26	27	30	31	34	35
24	25	28	29	32	33
22	23	18	19	14	15
20	21	16	17	12	13
2	3	6	7	10	11
0	1	4	5	8	9

2.3 Basics of Routing. We assume that a packet consists of a message plus some additional information that is needed to route the packet to its destination. For example, in the case of sorting each packet contains a key from a linearly ordered set. In the k-k sorting problem, the packets must be routed such that the PU with index i, $0 \leq i \leq n^2 - 1$, receives the packets with ranks $k \cdot i, k \cdot i + 1, \ldots, k \cdot (i + 1) - 1$.

We speak of *edge contention* if several packets residing in a PU have to be routed across the same connection. Contentions can be resolved by a priority scheme. Throughout this paper, we apply the *farthest-first strategy*, which gives priority to the packet that has the farthest distance to go.

We review a fundamental result from [8] concerning routing on a chain of length n under the farthest-first strategy. Let P_i be the PU with index i. For a given distribution of the packets over the PUs, define $h_r(i, j)$ as the number of packets passing from left to right through both P_i and P_j, and let T_r denote the number of steps required for the routing to the right.

LEMMA 2.1. *On a chain of length n, we have $T_r = \max_{i<j}\{j - i + h_r(i, j) - 1\}$.*

On two-dimensional meshes there are two basic modes of packet routing. A packet is routed *row-first* if it is first routed along the row to its destination column, and then along the column to its destination. A packet is routed *column-first* if it is first routed along the column to its destination row, and the along the row to its destination.

Considering the maximal distance a packet may have to travel, it follows that $d \cdot n - d$ is a lower bound for routing on d-dimensional cubes, the so-called *distance bound.* For k-k routing on a square, at least $k \cdot n/2$ steps are required if all packets located in the left half of the mesh have a destination in the right half; this is the so-called *bisection bound.* Together, these two bounds imply the following lemma.

LEMMA 2.2. *k-k routing and sorting on a d-dimensional cube requires at least* $\max\{d \cdot n - d, k \cdot n/2\}$ *steps.*

3 Basic Idea and k-k Sorting

Our derandomization technique is based on a combination of local sorting and 'unshuffling'. More precisely, we divide the mesh into blocks of size $n^\alpha \times n^\alpha$, with $0 < \alpha < 1$. The blocks are indexed. In sorting problems it is required that blocks with consecutive indices are adjacent. The packets in each block are sorted: in a routing problem the packets are sorted on the index of their destination block; in a sorting problem on their key. Then the packets of each block are distributed over the blocks. For a k-k problem, $k \geq 1$, this is done most frequently by routing the packet of rank i, $0 \leq i < k \cdot n^{2 \cdot \alpha}$, in block j, $0 < j \leq n^{2 - 2 \cdot \alpha} = N$, to position $j + \lfloor i/N \rfloor \cdot N$ in block $i \bmod N$. Notice that every PU is the destination of exactly k packets.

The utility of the above *sort-and-unshuffle* operation for sorting on meshes was previously observed by Schnorr and Shamir, who used it in the design of their $3 \cdot n + o(n)$ sorting algorithm in the single-packet model [17]. In the following, we will demonstrate that this operation can in many cases be employed as a 'substitute' for randomization. Following a scheme originally proposed by Valiant and Brebner [20], many randomized algorithms for routing on meshes start by sending the packets to random intermediate destinations. This has the effect of distributing packets with destinations close to each other evenly over the mesh. The sort-and-unshuffle operation simulates this effect in a deterministic manner.

3.1 Example. A simple example illustrates how sort-and-unshuffle can be applied to derandomize a sorting algorithm.

We consider the k-k sorting problem for $k = n^4$. We choose $\alpha = 0$, that is, each block consists of a single PU. The mesh is indexed with a snake-like row-major scheme. Consider the following simple randomized algorithm:

1. All packets are routed to a random destination; with probability 1/2 row-first, with probability 1/2 column-first.

2. The packets in each PU are sorted. The packets with rank i are routed to PU $\lfloor i/n^2 \rfloor$.

3. Sort all pairs of PUs $(2i, 2i+1)$, $0 \leq i < n^2/2$. Sort all pairs of PUs $(2i-1, 2i)$, $0 < i < n^2/2$. Repeat these steps $\mathcal{O}(1)$ times.

We refer to this algorithm as RANDSORT.

LEMMA 3.1. *After Step 2 of* RANDSORT*, each packet is at most $\mathcal{O}(1)$ PUs away from its destination, with high probability.*

Proof. Consider a packet p with destination in PU j, $0 \leq j < n^2$. There are less than $k \cdot (j+1)$ packets with destination before p. Each of these packets is routed in Step 1 with probability n^2 to PU j. Thus, using Chernoff bounds, the number of these packets in j can be bounded by $(j+1) \cdot k \cdot n^{-2} + \mathcal{O}((j \cdot k)^{1/2} \cdot 1/n)$. Hence, the maximum of the index of the PU to which p is routed in Step 2 is given by $j' = \lfloor ((j+1) \cdot k \cdot n^{-2} + \mathcal{O}((j \cdot k)^{1/2} \cdot 1/n))/(k \cdot n^{-2}) \rfloor = j + \mathcal{O}(1)$. □

With Lemma 2.1 and using Chernoff bounds it can be easily shown that the algorithm runs in $k \cdot n/2 + \mathcal{O}(k + (k \cdot n \cdot \log n)^{1/2})$ steps. The algorithm can be made deterministic by replacing Step 1 with

1′. The packets residing in each PU are sorted. The packet p with rank i, $0 \leq i \leq n^4 - 1$ is routed to $P_{i \bmod n^2}$. p is routed row-first if $i \bmod (2 \cdot n^2) < n^2$; else p is routed column-first.

We refer to this algorithm as HIGHKSORT. The correctness follows from the following lemma.

LEMMA 3.2. *After Step 2 of* HIGHKSORT*, each packet is at most one PU away from its destination.*

Proof. Consider a packet p that resides in some PU P after Step 1′. Since there are n^2 regularly interspaced packets from each PU in P, the rank of p among the packets that originally resided in any PU P_j, $0 \leq j < n^2$, can be estimated to within n^2. Hence, p can determine its global rank to within n^4, the number of packets residing in a single PU. □

Note that the way in which the packets are routed in Step 1′ and Step 2 ensures that the packets are distributed evenly.

THEOREM 3.1. HIGHKSORT *performs k-k sorting for $k = n^4$ in $k \cdot n/2 + \mathcal{O}(k)$ steps on a square.*

Proof. Under the routing operations in Step 1′ and Step 2 each PU sends exactly n^2 packets to each other PU. Each of these steps can be achieved in a perfectly regular way in $k \cdot n/4$ steps. Step 3 takes $\mathcal{O}(k)$ steps. □

The routing operations in Step 1′ and Step 2 are so regular that they can be easily implemented in such a way that no PU ever holds more than k packets. We can show the following lemma.

LEMMA 3.3. HIGHKSORT *can be implemented to run with maximal queue size k.*

Proof. We prove that each PU holds at most $k/2$ white packets at all times. Consider a phase in which the white packets move horizontally. The distribution is such that initially the number of packets that leave P over its right connection equals the number of packets that enter P over this connection. Because k is large, because of the distribution of the packets, and because of the farthest-first strategy, this guarantees that P receives a packet iff it sends a packet. □

Note that the movement of the packets in the deterministic algorithm is more regular than in the randomized one. The running times and queue sizes of the two algorithms are comparable.

3.2 Sorting for General k. HIGHKSORT can be easily modified into an algorithm that performs k-k sorting optimally for all $k \geq 8$. The mesh is divided into blocks with side length $n^{2/3}/k^{1/6}$. We assume that this number is an integer. Let $m = k^{1/3} \cdot n^{2/3}$ denote the number of blocks. The size of the blocks is chosen such that the number of packets in each block equals m^2. In principle we can now apply HIGHKSORT with the blocks playing the role of PUs, but we still have to specify precisely where every packet is going to be routed during Step 1′ and Step 2. We assume that the mesh is indexed with a blocked snake-like row-major scheme. We get the following algorithm:

1.a. The packets residing in each block are sorted. The intermediate destination of a packet p of rank i, $0 \leq i < m^2$, lies in block $i \bmod m$. If $i \bmod (2 \cdot m) < m$ then p is colored white, else p is colored black.

1.b. The blocks are divided into subblocks with side lengths $n^{1/3}/k^{1/3}$. The packets of each block are rearranged such that the packets with intermediate destination in block (i, j), $0 \leq i, j < k^{1/6} \cdot n^{1/3}$, appear in subblock (i, j).

1.c. The white packets are routed along the row to the column of blocks of their intermediate destination. The packets in the j_2th column of subblocks in the j_1th column of blocks, $0 \leq j_1, j_2 < k^{1/6} \cdot n^{1/3}$, are routed as a block to the j_1th column of subblocks in the j_2th column of blocks. The black packets are routed analogously.

1.d. The white packets are routed along the column to the block of their intermediate destination. The packets in the i_2th row of subblocks in the i_1th row of blocks, $0 \leq i_1, i_2 < k^{1/6} \cdot n^{1/3}$, are routed as a block to the i_1th row of subblocks in the i_2th row of blocks. The black packets are routed analogously.

2.a. The packets residing in each block are sorted. The preliminary destination of a packet p of rank i, $0 \leq i \leq m^2 - 1$, lies in block $\lfloor i/m \rfloor$. If i is even then p is colored white, else p is colored black.

2.b. *Identical to Step 1.b.*

2.c. *Identical to Step 1.c.*

2.d. *Identical to Step 1.d.*

3. Sort all pairs of blocks $(2i, 2i+1)$, $0 \leq 1 < m/2$. Sort all pairs of blocks $(2i-1, 2i)$, $0 < i < m/2$.

The correctness of this algorithm, called KKSORT, follows from the same argument as in the proof of Lemma 3.2, replacing 'PU' by 'block':

LEMMA 3.4. *After Step 2.d of* KKSORT*, each packet is either in its destination block, or in a block that is adjacent in the snake-like ordering of the blocks.*

The colorings in Step 1.a and Step 2.a assure that from each block precisely $m/2$ packets are routed row-first to each other block, and $m/2$ packets column-first. This is essential for the proof of

LEMMA 3.5. *Step 1.c, 1.d, 2.c and 2.d of* KKSORT *each take at most* $\max\{n, k \cdot n/8\}$ *steps on a square.*

Proof. We analyze the lemma for the routing along the row in Step 1.c of the white packets. The analyses for the black packets and for the other steps are analogous.

The PUs in block (i, j) send $m/2$ packets row-first to any other block. So, during Step 1.c $k^{1/6} \cdot n^{1/3} \cdot m/2 = k^{1/2} \cdot n/2$ packets are routed from block (i, j) to any block (i, j') in the same row of blocks. By the rearranging in Step 1.b these packets are evenly distributed over the rows of block (i, j). Hence, we can concentrate on the routing problem on a chain in which $k^{2/3} \cdot n^{1/3}/2$ packets are routed from each section of length $n^{2/3}/k^{1/6}$ to each other section. Within each section the packets are sorted according to the distance they have to go. In order to apply Lemma 2.1, we consider how many packets have to go from the first i sections to the last j sections, $i + j \leq k^{1/6} \cdot n^{1/3}$. These are $i \cdot j \cdot k^{2/3} \cdot n^{1/3}/2$ packets. The gap the packets have to bridge is $(k^{1/6} \cdot n^{1/3} - i - j) \cdot n^{2/3}/k^{1/6}$. Hence, the routing can be performed in $\max_{i,j}\{i \cdot j \cdot k^{2/3} \cdot n^{1/3}/2 + (k^{1/6} \cdot n^{1/3} - i - j) \cdot n^{2/3}/k^{1/6}\}$ steps. For $k \geq 8$ the maximum is assumed for $i = j = k^{1/6} \cdot n^{2/3}/2$, and equals $k \cdot n/8$. Otherwise the maximum is assumed for $i = j = 0$, and equals n. We do not have to consider 'cuts' within the sections because there the packets stand in the correct order. $\square$

Combining Lemma 3.4 and Lemma 3.5 gives

THEOREM 3.2. KKSORT *performs k-k sorting deterministically in* $\max\{4 \cdot n, k \cdot n/2\} + \mathcal{O}(k^{5/6} \cdot n^{2/3})$ *steps on a square.*

The constant of the lower-order term, $\mathcal{O}(k^{5/6} \cdot n^{2/3})$, depends on the applied algorithm for the local sorting operations. It is minimized when KKSORT itself is applied recursively. For the sorting in Step 3, the algorithm has to be modified slightly.

COROLLARY 3.1. *For* $k \geq 8$, *k-k sorting on a square can be performed in* $k \cdot n/2 + (4 + o(1)) \cdot k^{5/6} \cdot n^{2/3}$ *steps.*

Proof. Let $s = n^{2/3}/k^{1/6}$. During KKSORT, k-k sorting on squares of size $s \times s$ must be applied in Steps 1.a and 2.a. In Step 1.b and 2.b, the packets are rearranged within squares of size $s \times s$. Each of these four steps can be performed in $k \cdot s/2 + o(k \cdot s)$ steps. The sorting operations in Step 3 can be performed in $k \cdot s + o(k \cdot s)$ steps each. $\square$

The small constant of the additional term indicates that our results may be of practical value. The results of this section were stated only for squares. Generalizations for tori are immediate. As the routing steps on which the algorithms are based, and which determine the leading term of the sorting times, can be performed twice as fast on a torus, the problems on a torus can be solved almost twice as fast as on a square. Without further modification we can show with a refinement of Lemma 3.3 that the queue size is at most $k + 4$. When the packets are accurately timed we get

LEMMA 3.6. *For* $k \geq 2$, KKSORT *can be implemented to run with maximal queue size k.*

3.3 Generalizations. The ideas underlying our k-k sorting algorithm are very general, and can in fact be applied to large classes of networks. Interestingly, the resulting algorithm is a variation of Leighton's *Columnsort* algorithm [2, 13]. In this subsection, we briefly describe this generalized algorithm, and show how it can be efficiently implemented on multi-dimensional meshes. This leads to an algorithm for k-k sorting on meshes of arbitrary dimension whose running time matches the bisection lower bound to within a lower order additive term, as long as $d \leq \log N/(\alpha \cdot \log\log N)$ and $k \geq 4 \cdot d$, where $\alpha = 2/(\log 3 - 1)$ and d is the dimension of the mesh. The high-level structure of the generalized algorithm is as follows:

1.a. Sort within groups of $N^{2/3}/k^{1/3}$ PUs.

1.b. Route the packets to appropriate intermediate destinations.

2.a. Sort within groups of $N^{2/3}/k^{1/3}$ PUs.

2.b. Route the packets to their preliminary destinations.

3. Sort twice within groups of $2 \cdot N^{2/3}/k^{1/3}$ PUs.

Steps 1.a, 2.a, and 3 can be implemented by a recursive call to the algorithm. Note that Steps 1.b and 2.b of the algorithm are off-line routing problems corresponding to a sort-and-unshuffle operation. In the previous subsection, it was shown that these routing problems can be solved in optimal time, due to their highly regular structure. It can be shown that this is also the case for meshes of higher dimension.

Now divide the mesh into blocks of side length $n^{2/3}/k^{1/(3 \cdot d)}$. There are $m = k^{1/3} \cdot n^{d/3}$ blocks each holding $k^{2/3} \cdot n^{2/3 \cdot d} = m^2$ packets. The algorithm remains the same as KKSORT except that the coloring with two colors is replaced by a coloring with d colors: a packet which has rank i after the sorting in Step 1.a is given color $\lfloor (i \bmod d \cdot m)/m \rfloor$. In Step 3.a the packets get color $i \bmod d$. The packets are now routed along d independent paths: a packet of color c, $0 \leq c \leq d-1$ is routed first along axis c, then along axis $c+1$, and so on. The algorithm is applied recursively until the side length of the meshes is reduced to $\mathcal{O}(1)$. At that point we apply the hypercubic sorting algorithm of Cypher and Plaxton [4], which runs in $\mathcal{O}(k \cdot d \cdot \log d)$ steps. We refer to the resulting algorithm as HIGHDIMSORT.

THEOREM 3.3. *We apply* HIGHDIMSORT *to k-k sorting on a d-dimensional cube of side length n. If $d \cdot \log d = o(n^{2/3})$ and $k \geq 4 \cdot d$, then the sorting is performed in $k \cdot n/2 + (4 + o(1)) \cdot k^{1-1/(3 \cdot d)} \cdot n^{2/3}$ steps.*

Proof. The recurrence for the running time $T(k, N)$ of this algorithm is given by $T(k, N) = 2 \cdot (R(k, N) + T(k, N^{2/3}/k^{1/3}) + T(k, 2 \cdot N^{2/3}/k^{1/3}))$, where $R(k, n)$ denotes the number of steps needed in the off-line routing in Step 2 and Step 4. Solving the above recurrence with $R(k, n) = k \cdot n/4$, we see that the recursive sorting of the blocks takes time $T(k, N) = 4 \cdot k^{1-1/(3 \cdot d)} \cdot n^{2/3} + \mathcal{O}(k^{1-1/(3 \cdot d)} \cdot n^{4/9} + k \cdot d \cdot \log d)$. □

Hence, the running time of the above algorithm nearly matches the bisection lower bound. Note that the algorithm was designed under the assumption that each PU can communicate with all of its neighbors in a single step. For meshes of nonconstant dimension, this assumption might be considered somewhat unrealistic. However, through a straightforward simulation of this *multi-port model* on a weaker model in which a PU can only communicate with a single neighbor (the *single-port model*), we can also obtain efficient algorithms for the single-port model.

Finally, we point out that the performance of the algorithm does not really rely on the power of the Sharesort algorithm in [4], and that a similar result can also be shown using, for example, bitonic sorting [1]. In that case, the constant α is slightly larger.

4 Permutation Routing with Small Queues

In this section, we apply our techniques to an optimal randomized algorithm for permutation routing recently proposed by Kaklamanis, Krizanc, and Rao [7]. First, we give a description of their algorithm, which has a very simple structure.

Partition the mesh vertically into four quarters Q_0 to Q_3, where Q_i contains the columns $i \cdot n/4$ to $(i+1) \cdot n/4 - 1$. In the algorithm every packet is first routed along the row to an intermediate destination, where it turns into a column. In this column, the packet moves to its destination row, and then in the destination row to its final destination. The intermediate destination is chosen randomly according to the following rules.

- Packets in Q_0 and Q_1 with destinations in Q_0 or Q_1 choose intermediate destinations in Q_0.
- Packets in Q_0 and Q_1 with destinations in Q_2 or Q_3 choose intermediate destinations in Q_2.
- Packets in Q_2 and Q_3 with destinations in Q_0 or Q_1 choose intermediate destinations in Q_1.
- Packets in Q_2 and Q_3 with destinations in Q_2 or Q_3 choose intermediate destinations in Q_3.

It is shown in [7] that this routing scheme results in a running time of $2 \cdot n + \mathcal{O}(\log n)$ and a queue size of $\mathcal{O}(\log n)$, with high probability. The queue size can be improved to $\mathcal{O}(1)$ by applying a spreading technique described in [16]. An off-line version of the algorithm runs in time $2 \cdot n - 1$ with queue size 4.

The high-level structure of our deterministic algorithm is very similar. Instead of the randomization, we perform an appropriate sort-and-unshuffle operation. We also employ a more sophisticated spreading technique, hereafter referred to as *counter scheme*, in order to achieve a small, constant queue size.

1. Partition the mesh into blocks of size $n^{3/4} \times n^{3/4}$, and sort the packets in each block by their destination blocks, into row-major order. Here, it is assumed that the set of destination blocks is ordered in some arbitrary fixed way.

2. In each quarter Q_i, perform a sort-and unshuffle operation among the blocks of each row of blocks. This can be implemented by moving in each row the packet in position j to position $(j \bmod (\frac{n^{1/4}}{4})) \cdot n^{3/4} + \lfloor j/(\frac{n^{1/4}}{4}) \rfloor$, for $0 \leq j < n/4$.

3. Route the packets along the rows to their intermediate destinations according to the four rules given above, such that every packet travels a distance that is a multiple of $n/4$.

4. Again sort the packets in each block by their destination blocks, into row-major order.

5. Route the packets along the columns. In order to get to its destination block, a packet traveling along its column could turn in any of the $n^{3/4}$ consecutive rows passing through that block. The exact row across which a packet will enter its destination block is determined by the counter scheme described in the next step. The purpose of this scheme is to distribute the packets of each destination block evenly over the $n^{3/4}$ incoming rows, while at the same time maintaining a small, constant queue size.

6. The counter scheme: In each column we maintain $n^{1/2}$ counters, two for each of the $n^{1/2}/2$ destination blocks in the half of the mesh containing the column (all packets are already in the correct half of the mesh). The $n^{1/4}$ counters for any particular row of $n^{1/4}/2$ destination blocks are located in the $n^{1/4}/2$ PUs immediately above and below the $n^{3/4}$ rows passing through these destination blocks. Whenever a row element destined for a particular block arrives at one of the two corresponding counters, this counter is either increased by one, modulo $2 \cdot n^{1/2}$ (in the case of the counters above the destination rows), or decreased by one, modulo $2 \cdot n^{1/2}$ (in the case of the counters below the destination rows). The row across which the packet will enter its destination block is determined by the sum, modulo $n^{3/4}$, of the new counter value and a fixed offset value associated with each counter. A counter in column i of the half, $0 \leq i < n/2$, that corresponds to a destination block in the jth column of destination blocks, $0 \leq j < n^{1/4}/2$, is assigned the offset value $(i + j \cdot 2 \cdot n^{1/2}) \bmod n^{3/4}$.

7. Route the packets along the rows into their destination blocks using the farthest-first strategy. Each packet stops at the first PU in its destination block that has a free memory slot for an additional packet. It will be shown below that, due to the counter scheme, the incoming packets are evenly distributed over the rows of any destination block.

8. Perform local routing over a distance of $\mathcal{O}(n^{3/4})$ to bring every element to its final destination.

We first analyze the running time. Clearly, Steps 1, 4, and 8 only take $\mathcal{O}(n^{3/4})$ steps. Steps 2 and 3 can be overlapped by sending the packets directly to the locations they will assume at the end of Step 3. This takes between $n/2+\mathcal{O}(n^{3/4})$ and $3/4 \cdot n+\mathcal{O}(n^{3/4})$ steps, depending on the location of the block. As soon as a block has received all of its packets, it can perform the local sort in Step 4, and start with the column routing in Step 5. This routing problem is a $o(n)$-approximate 2-2 relation on a linear array, and can be routed in $n + o(n)$ steps (see [7]). Thus, Steps 5 and 6 of the algorithm will terminate between step $1\frac{1}{2} \cdot n+o(n)$ and step $1\frac{3}{4} \cdot n+o(n)$, depending on the location of the column. Assuming that Step 6 has distributed the packets evenly over the incoming rows of each destination block, Step 7 can be interpreted as the problem of routing an approximate 2-1 relation on a linear array of length $n/2$, where packets that have a distance of d to travel are not allowed to move before time $n/2 - d$. Thus, the above algorithm runs in time $2 \cdot n + o(n)$.

It remains to show that the packets are indeed evenly distributed after Step 6, and that the total queue size is bounded by 5. Consider a destination block D and two blocks B_1 and B_2 located in the same quarter and the same row of blocks. It can be shown that the number of packets with destination block D will differ by at most $n^{1/4} = o(n^{3/4})$ between B_1 and B_2, after Step 3. This implies that after Step 4, the number of packets with destination block D will differ by at most $2 \cdot n^{1/4}$ between any two columns in the quarter. There are $n^{3/2}$ packets with destination block D. Hence, any of the $n/4$ columns in the quarter can contain at most $n^{3/2}/(n/4) + 2 \cdot n^{1/4} \approx 4 \cdot n^{1/2}$ packets with destination block D, which are evenly distributed among $2 \cdot n^{1/2}$ rows by the counter technique (up to a difference of 1). Due to the assignment of offset values to the counters, packets with different destination blocks always turn in different PUs. This implies that at most 3 packets turn in any single PU. The elements in $n^{3/4}$ consecutive columns will be evenly distributed among all incoming rows of D, due to the $n^{3/4}$ different offset values of the $2 \cdot n^{3/4}$ counters corresponding to D. This implies that every PU of D will receive at most 2 packets. The maximum possible queue size of the algorithm is given by a scenario in which 3 packets have to turn in a given PU, while 2 other packets are temporarily passing through the PU during the routing in Step 5. This establishes the following result.

THEOREM 4.1. *There exists a deterministic routing algorithm for the two-dimensional mesh with a running time of* $2 \cdot n + o(n)$ *and a queue size of* 5.

Using similar ideas, we have also derandomized a number of other algorithms proposed in [7]. In particular, we can show the following results.

THEOREM 4.2. *Any* 2-2 *relation can be routed deterministically in time* $2 \cdot n + o(n)$ *with queue size* 10.

THEOREM 4.3. *There exists a deterministic algorithm for routing on the three-dimensional cube with running time* $3 \cdot n + o(n)$ *and queue size* 13.

THEOREM 4.4. *There exists a deterministic algorithm for routing on the two-dimensional torus with running time* $n + o(n)$ *and constant queue size.*

5 Optimal Deterministic Sorting on a Square

In this section, we apply our techniques to a randomized algorithm for 1-1 sorting with running time $2 \cdot n + o(n)$ recently described by Kaklamanis and Krizanc [6]. As a result, we obtain the first optimal deterministic algorithm for 1-1 sorting on the square. The fastest deterministic algorithm previously known runs in $2\frac{1}{2} \cdot n + o(n)$ steps, and is due to Kunde [11].

We start by giving a brief description of the randomized algorithm of Kaklamanis and Krizanc. In Subsection 5.2, we derive a simple deterministic algorithm with a running time of $2\frac{1}{4} \cdot n + o(n)$. Finally, Subsection 5.3 describes the optimal algorithm for the square, and lists some extensions to other types of meshes.

5.1 The Optimal Randomized Algorithm. We briefly review the optimal randomized algorithm of Kaklamanis and Krizanc [6]. The structure of their algorithm is quite complicated, and hence we necessarily omit a number of important details.

In the algorithm, the $n \times n$ mesh is partitioned into blocks B_i, $0 \leq i < n^{2-2\cdot\gamma}$, of size $n^\gamma \times n^\gamma$. In addition, we partition the mesh into quadrants Q_i, $0 \leq i < 4$, and subquadrants (quadrants of quadrants) T_i, $0 \leq i < 16$. We assume that the four central subquadrants are denoted as T_0 to T_3.

1. Each packet is selected as a presplitter with probability $n^{\epsilon/2}$, for some suitable $\epsilon > 0$. Each presplitter routes itself to a random position in a block B of size $n^{\epsilon/2} \times n^{\epsilon/2}$ at the center of the mesh. The presplitters are then sorted, and $n^{\epsilon/5}$ elements of equidistant ranks are selected as splitters.

2. The splitters are broadcast in T_1 to T_4.

3. Each packet declares itself either a *row element* or a *column element*, with equal probability. Each row element is then routed to a random location in its current row, within its subquadrant. Similarly, each column element is routed to a random position in its column. Each of the subquadrants T_0 to T_3 then receives a copy of the contents of all other subquadrants T_i, $0 \leq i < 16$.

4. The packets in each block B_i located in T_1 to T_4 are sorted. Using the splitters, each packet can then compute a preliminary destination that is close to its final destination, with high probability. A packet with a preliminary destination outside its current quadrant kills itself.

5. Each surviving packet is routed to a random location in the block containing its preliminary destination.

6. The global ranks of the splitters are computed, and broadcast throughout each quadrant.

7. Using the global ranks of the splitters, each packet can be routed to its final position.

Kaklamanis and Krizanc [6] show that by a clever interleaving of the steps, the above algorithm can be scheduled to run in $2 \cdot n + o(n)$ steps.

5.2 A Simple Non-Optimal Algorithm. We now describe a fairly simple deterministic algorithm that runs in $2\frac{1}{4} \cdot n + \mathcal{O}(n^{2/3})$ steps, and that does not use splitters. In the following, let the blocks B_i be of size $n^{2/3} \times n^{2/3}$.

To obtain the algorithm, we remove all steps that involve the computation and broadcasting of the splitter elements. The partial randomization in Step 3 is substituted by a complete sort-and-unshuffle operation in each subquadrant. This increases the running time of the algorithm by $n/4$, since only one phase of the sort-and-unshuffle operation can be scheduled in parallel with the overlapping of the packets into the subquadrants T_0 to T_3, as described in [6]. Hereafter, the packets in each block B_i in T_0 to T_3 are sorted again, and a preliminary destination is computed in the same way as in the algorithm KKSORT of Section 3. Using arguments similar to those in Lemma 3.2, it can be shown that the preliminary destination of each packet is at most one block away from its final destination. As before, any element with a preliminary destination outside its current quadrant kills itself.

The routing of the packets to their preliminary destinations is now actually much simpler than in the randomized algorithm, since we are guaranteed that approximately a quarter of the packets in each block B_i survive. This implies that the surviving elements are uniformly distributed within each center subquadrant, and that we can perform the routing in a simple, greedy fashion. This establishes the following result.

THEOREM 5.1. *Deterministic* 1-1 *sorting without splitters can be performed in* $2\frac{1}{4} \cdot n + \mathcal{O}(n^{2/3})$ *steps.*

5.3 The Optimal Algorithm. We now establish the main result of this section, a deterministic sorting algorithm with a running time of $2 \cdot n + o(n)$ and a queue size of approximately 25. Due to space constraints, we will not be able to give a formal proof of the claimed bounds on time and queue size.

Our algorithm follows closely the lines of the optimal randomized algorithm of [6], and simply replaces each randomized step with an appropriate deterministic one.

5.3.1 Splitter Selection and Routing. In Step 1 of the algorithm, randomization is used to select a set of presplitters and route it towards the center of the mesh. The following deterministic step can be substituted for Step 1.

1. In each block B_i, $0 \leq i < n^{2/3}$, sort the packets into row-major ordering and select the elements in column i as presplitters. Route all presplitters greedily towards the block B_j located at the center of the mesh, such that the presplitter of each block move in lock step. Then sort the set of presplitters, and select $n^{2/3}$ elements of equidistant ranks as splitters.

As the presplitters are routed with priority, they delay the routing of the other packets. However, it can be shown that this delay is small, since every edge of the mesh is traversed by at most $n^{2/3}$ presplitters. Using the splitters selected from the presplitters in the center of the mesh, each packet can estimate its rank to within $\mathcal{O}(n^{4/3})$. More precisely, the following lemma can be shown.

LEMMA 5.1. *Let s_i be the ith smallest splitter, $1 \leq i \leq n^{2/3}$. Then the global rank of s_i is within the range $i \cdot n^{4/3} \pm \mathcal{O}(n^{4/3})$.*

This implies that the preliminary destination computed by each surviving packet in Step 4 of the algorithm has a distance of $\mathcal{O}(n^{2/3})$ from its final destination (recall that we assume a blocked snake-like row-major indexing scheme). Hence, the local routing in Step 7 of the algorithm will route each packet to its correct destination in time $\mathcal{O}(n^{2/3})$.

5.3.2 Spreading Along Rows and Columns. In Step 3 of the randomized algorithm, the set of packets is partitioned into row elements and column elements, which are then randomized along the rows and columns, respectively, of each subquadrant. This step is overlapped with the copying of the packets into the center subquadrants T_0 to T_3, and is performed according to a rather ingenious schedule described in [6]. The purpose of this step is to make sure that after Step 4 the surviving packets are distributed over each center subquadrant in such a way that the routing into destination blocks in Step 6 can be performed efficiently.

In our deterministic algorithm, we first sort the packets in each block into row-major ordering. We then define the row and column elements as the packets with odd and even ranks, respectively. The randomization of the row elements along the rows of each subquadrant can be simulated along the lines of Step 2 of the deterministic routing algorithm in Section 4. The randomization of the column elements along the columns is done analogously.

After execution of this step, all packets with a common destination block are (approximately) evenly divided between the sets of row element and column element, and the row elements (resp. column elements) are evenly distributed over the columns (resp. rows) of each subquadrant.

5.3.3 Routing to Destination Blocks. We first give a brief description of the routing scheme used in Step 5 of the randomized algorithm. The routing consists of two overlapped phases. In the first phase, the row elements are routed in the columns, while the column elements are routed in the rows. In the second phase, the row elements are routed in the rows to their destination blocks, and the column elements are routed in the columns to their destination blocks.

If two packets in the same phase content for an edge, then priority will be given to the packet with the farthest total distance to travel. If the packets are from different phases, then a somewhat more complicated priority scheme is employed.

We use the same routing scheme in our deterministic algorithm, with one minor modification. In Step 5 of the randomized algorithm, each packet is routed to a random location inside its destination block. This ensures that at most $\mathcal{O}(\log n)$ packets turn in a single PU, and that at most $\mathcal{O}(\log n)$ packets are routed to the same destination PU. This queue size can be reduced to $\mathcal{O}(1)$ by applying a spreading technique described in [16]. In the deterministic algorithm, we use the counter scheme described in the algorithm of Section 4 to distribute the packets evenly over the incoming edges of each destination block. In addition, this scheme also ensures that the destinations of the elements in the second phase of the routing are evenly distributed inside each row and each column; this is crucial in the analysis of the running time.

It can be shown that the sort-and-unshuffle operation in Step 3 of the algorithm has distributed the surviving packets in such a way that the above routing scheme will route each packet to its destination block in $n + o(n)$ step. The formal proof of this claim is quite lengthy, and will hence be omitted. Altogether, we get the following result.

THEOREM 5.2. *There exists a deterministic algorithm for 1-1 sorting on the square with running time $2 \cdot n + o(n)$ and constant queue size.*

The following results can be derived in a similar fashion. The corresponding randomized results can be found in [6].

THEOREM 5.3. *There exists a deterministic algorithm for sorting on the three-dimensional cube with running time $3\frac{1}{2} \cdot n + o(n)$ and constant queue size.*

THEOREM 5.4. *There exists a deterministic algorithm for sorting on the two-dimensional torus with running time $1\frac{1}{4} \cdot n + o(n)$ and constant queue size.*

THEOREM 5.5. *There exists a deterministic algorithm for sorting on the three-dimensional torus with running time $2 \cdot n + o(n)$ and constant queue size.*

6 Concluding Remarks

In this paper, we have introduced a new technique that allows us to derandomize many of the randomized algorithms for routing and sorting on meshes that have been proposed in recent years. By applying this technique, we have obtained optimal or improved deterministic algorithms for a number of routing and sorting problems on meshes and related networks. The new technique is very general, and seems to apply to most of the randomized algorithms that have been proposed in the literature. In fact, as a result of this work, we are currently not aware of any randomized algorithm for routing and sorting on meshes and related networks whose running time cannot be matched, within a lower order additive term, by a deterministic algorithm.

This naturally raises the question whether randomization is of any help at all in the design of routing and sorting algorithms for these types of networks. In this context, we point out that some of the randomized algorithms still have a simpler control structure or smaller lower order terms than their deterministic counterparts, which repeatedly perform local sorting within blocks. Also, the results in this paper would not have been possible without the extensive study of randomized schemes for routing and sorting by a number of other authors, which has resulted in a variety of fast randomized algorithms [5, 6, 7, 8, 16, 18, 20].

Acknowledgements: The third author would like to thank Christos Kaklamanis, Greg Plaxton, and Rajmohan Rajaraman for helpful discussions.

References

[1] Batcher, K.E., 'Sorting Networks and their Applications,' *Proc. AFIPS Spring Joint Computer Conference*, pp. 307-314, 1968.

[2] Aggarwal, A., M.D. Huang, 'Network Complexity of Sorting and Graph Problems and Simulating CRCW PRAMs by Interconnection Networks,' *VLSI Algorithms and Architectures (AWOC 88)*, LNCS 319, pp. 50-59, ACM, 1992.

[3] Chlebus, B.S., M. Kaufmann, J.F. Sibeyn, 'Deterministic Permutation Routing on Meshes,' *Proc. 5th Symp. on Parallel and Distributed Proc.*, IEEE, 1993.

[4] Cypher, R., C.G. Plaxton, 'Deterministic Sorting in Nearly Logarithmic Time on the Hypercube and Related Computers,' *Proc. 22nd Symp. on Theory of Computing*, pp. 193-203, ACM, 1990.

[5] Kaklamanis, C., D. Krizanc, L. Narayanan, Th. Tsantilas, 'Randomized Sorting and Selection on Mesh Connected Processor Arrays,' *Proc. 3rd Symposium on Parallel Algorithms and Architectures*, pp. 17-28, ACM, 1991.

[6] Kaklamanis, C., D. Krizanc, 'Optimal Sorting on Mesh-Connected Processor Arrays,' *Proc. 4th Symposium on Parallel Algorithms and Architectures*, pp. 50-59, ACM, 1992.

[7] Kaklamanis, C., D. Krizanc, S. Rao, 'Simple Path Selection for Optimal Routing on Processor Arrays,' *Proc. 4th Symposium on Parallel Algorithms and Architectures*, pp. 23-40, ACM, 1992.

[8] Kaufmann, M., S. Rajasekaran, J.F. Sibeyn, 'Matching the Bisection Bound for Routing and Sorting on the Mesh,' *Proc. 4th Symposium on Parallel Algorithms and Architectures*, pp. 31-40, ACM, 1992.

[9] Kaufmann, M., J.F. Sibeyn, 'Derandomizing Sorting Algorithms on Meshes,' *Unpublished Manuscript*, 1993.

[10] Kunde, M., 'Lower Bounds For Sorting on Mesh-Connected Architectures', *Acta Informatica*, 24, pp. 121-130, 1987.

[11] Kunde, M., 'Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound', *Proc 31th Symposium on Foundations of Computer Science*, pp. 141-150, IEEE, 1991.

[12] Kunde, M., 'Block Gossiping on Grids and Tori: Deterministic Sorting and Routing Match the Bisection Bound,' *Proc. European Symp. on Algorithms*, LNCS 726, pp. 272-283, 1993.

[13] Leighton, F. T., 'Tight Bounds on the Complexity of Parallel Sorting,' *IEEE Trans. Comp.*, 34, pp. 344-354, 1985.

[14] Leighton, F. T., F. Makedon, I.G. Tollis, 'A $2n-2$ Step Algorithm for Routing in an $n \times n$ Array with Constant Queue Sizes,' *Proc. 1st ACM Symposium on Parallel Algorithms and Architectures*, pp. 328-335, 1989.

[15] Rajasekaran, S., Overholt, R., 'Constant Queue Routing on a Mesh,' *Journal of Parallel and Distributed Computing*, 15, pp. 160-166, 1992.

[16] Rajasekaran, S., Tsantilas, T., 'Optimal Routing Algorithms for Mesh-Connected Processor Arrays,' *Algorithmica*, 8, pp. 21-38, 1992.

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

[18] Sibeyn, J.F., M. Kaufmann, 'k-k Sorting on Meshes,' *Unpublished Manuscript*, 1992.

[19] Suel, T., 'Optimal Deterministic Routing and Sorting on Mesh-Connected Arrays of Processors,' Technical Report TR-93-18, University of Texas at Austin, 1993.

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