Improved Bounds for Routing and Sorting on Multi-Dimensional Meshes

*Torsten Suel**

Department of Computer Science
University of Texas at Austin

Abstract

We show improved bounds for 1–1 routing and sorting on multi-dimensional meshes and tori. In particular, we give a fairly simple deterministic algorithm for sorting on the d-dimensional mesh of side length n that achieves a running time of $3dn/2+o(n)$ without making any copies of the elements. We give deterministic algorithms with running times of $5dn/4+o(n)$ and $3dn/4+o(n)$ for the d-dimensional mesh and torus, respectively, that make one copy of each element. We also show lower bounds for sorting with respect to a large class of indexing schemes, under a model of the mesh where each processor can hold an arbitrary number of packets. Finally, we describe algorithms for permutation routing whose running times come very close to the diameter lower bound.

1 Introduction

The mesh-connected arrays of processors are one of the most thoroughly investigated classes of interconnection schemes for parallel processing. While the networks in this class have a large diameter in comparison to the various hypercubic networks, they are nonetheless of great importance due to their simple structure and their good performance in many applications. A variety of algorithmic problems have been analyzed as to their complexity on theoretical models of the mesh; probably the two most extensively studied problems are those of routing and sorting. For a detailed introduction into these problems, and an overview of the most important classes of interconnection schemes, the reader is referred to [14].

*Email: `torsten@cs.utexas.edu`.

Supported by the Texas Advanced Research Program under Grant Nos. 003658–480 and 003658–461, and by a Schlumberger Graduate Fellowship.

0

Much of the previous work on meshes has concentrated on the one-dimensional and two-dimensional cases. Particularly the two-dimensional networks are of great practical importance due to their efficient layout, and have been used as the basis for a number of existing parallel machines. In contrast, the meshes of dimension $d > 2$ (hereinafter referred to as *multi-dimensional meshes*) have received somewhat less attention. While the problems of routing and sorting on these networks have previously been studied by a number of authors, there are still considerable gaps between the best upper and lower bounds.

In this paper, we focus on the problems of 1–1 routing and sorting on multi-dimensional meshes with constant dimension d. Formally, a d-dimensional mesh of side length n consists of $N = n^d$ processors, where each processor is identified by a d-tupel $(p_1, \ldots, p_d)$ in $[n]^d$. (We use $[n]$ to denote the set $\{0, \ldots, n-1\}$.) Two processors $P = (p_0, \ldots, p_{d-1})$ and $Q = (q_0, \ldots, q_{d-1})$ are connected by a bidirectional communication link iff there exists an i in $[d]$ with $|p_i - q_i| = 1$ and $p_j = q_j$ for all j in $[d]$ with $j \neq i$. The d-dimensional torus is obtained from the d-dimensional mesh by adding *wrap-around* edges between all pairs of processors $(p_0, \ldots, p_{d-1})$ and $(q_0, \ldots, q_{d-1})$ such that there exists an i in $[d]$ with $|p_i - q_i| = n - 1$ and $p_j = q_j$ for all j in $[d]$ with $j \neq i$.

We assume that the processors in the network operate in a synchronous mode. In a single step of the computation, each processor can perform some fixed amount of internal computation, and transmit one packet of information (of bounded size) to each of its neighbors. A processor can hold at most a constant number of packets at any time.

The *routing problem* is the problem of rearranging a set of packets in a network, such that every packet ends up at the processor specified in its destination address. A routing problem in which each processor is the source and destination of at most k packets is called a k–k routing problem. Most research has focused on the 1–1 routing problem, also called the *permutation routing problem.* In the k–k sorting problem, we assume that each processor initially holds k packets, where each packet contains a key drawn from some totally ordered set X. Our goal is to rearrange the packets in such a way that the packet with the key of rank i is moved to the processor with *index* $\lfloor i/k \rfloor$, for all i. As in the case of routing, most of the research has focused on the case $k = 1$.

The *index* of a processor in the network is determined by an *indexing scheme*. Formally, an *indexing scheme* for a d-dimensional mesh of side length n is a bijection $\mathcal{I}$ from $[n]^d$ to $[n^d]$. If $\mathcal{I}(P) = k$ for some processor $P = (p_0, \ldots, p_{d-1})$ in $[n]^d$ and some k in $[n^d]$, then we say that processor P has index k. The problem of sorting an input with respect to an indexing scheme $\mathcal{I}$ is to move every packet y of the input to the processor with index $\mathcal{I}(\mathrm{Rank}\,(y, X))$, where $\mathrm{Rank}\,(y, X) \stackrel{\text{def}}{=} |\{x \in X \mid x < y\}|$. An example of a commonly used indexing scheme for the two-dimensional mesh is the *row-major indexing scheme*, which is given by indexing the processors from the left to the right, and from the top row to the bottom row. Another important and closely related indexing scheme is the *snake-like row-major ordering* defined by $\mathcal{I}(i_1, j_1) < \mathcal{I}(i_2, j_2) \Leftrightarrow (i_1 < i_2) \vee ((i_1 = i_2) \wedge (((i_1 \text{ odd}) \wedge (j_1 < j_2)) \vee ((i_1 \text{ even}) \wedge ((j_1 > j_2))))$.

Both of these indexing schemes can be naturally extended to multi-dimensional networks; see [11] for a formal definition. Sorting algorithms on the mesh are usually designed with a particular indexing scheme in mind, and techniques suitable for one indexing scheme may not work well at all for others. In the algorithms presented in this paper, we assume a *blocked snake-like* indexing scheme. This indexing scheme is defined by partitioning the mesh into blocks of side length n^α, and using a snake-like indexing scheme inside each block, while the blocks themselves are ordered according to another snake-like indexing. Similar blocked indexing schemes have recently been used in a number of fast sorting algorithms (e.g., see [3, 4, 6, 12]).

An obvious lower bound for the running time of any algorithm for 1–1 routing or sorting is given by the diameter D of the network. (That is, $D = d(n-1)$ for the d-dimensional mesh and $D = dn/2$ for the d-dimensional torus.) The performance of an algorithm for routing or sorting on theoretical models of the mesh is commonly measured by its running time, its *queue size* (that is, the maximum number of packets any node has to store during the algorithm), and of course by factors such as the overall simplicity of the algorithm and the demands it places on the local control hardware or software. In this paper, we will focus on the time complexity of routing and sorting on meshes and tori of arbitrary constant dimension, and we will express our results in terms of the diameter D of the network.

1.1 Previous Results

A variety of sorting algorithms for the two-dimensional mesh have been proposed, starting with the work of Orcutt [16] and Thompson and Kung [18]. In particular, Schnorr and Shamir [17] gave a $3n + o(n)$ time algorithm for sorting into row-major order, and showed a nearly matching lower bound of $3n - o(n)$, also independently discovered by Kunde [7]. This lower bound assumes a model of the mesh, hereinafter referred to as the *single-packet* model, in which each processor can only hold a single packet at any time. The lower bound technique can also be extended to indexing schemes other than row-major [8]; the best general lower bound for arbitrary indexing schemes is currently at $2.27n$ [2].

However, these lower bounds do not hold for the *multi-packet* model assumed in this paper, in which any processor can hold a constant number of packets at a time. For this model, the only known lower bound is given by the network diameter of $2n - 2$, and recently algorithms with running time $2n + o(n)$ have been obtained [3, 6]. In the case of the torus, a lower bound of $3n/2 - o(n)$ has been shown for the single-packet model [8, 17]. In contrast, the fastest algorithm for the multi-packet model runs in $5n/4 + o(n)$ steps [3, 6].

Early examples of sorting algorithms for multi-dimensional meshes were given by Thompson and Kung [18] and Nassimi and Sahni [15]. A lower bound of $2D - n - o(n)$ for sorting on multi-dimensional meshes was established for the single-packet model [7, 17]; the bound applies to most of the indexing schemes used in the literature. An algorithm with a running time of $2D - n + o(n)$ was subsequently described by Kunde [9].

No non-trivial lower bounds are known for sorting on d-dimensional meshes in the multi-packet model. The best upper bound in this model is currently at $2D - 5n/2 + o(n)$. (This result can be obtained from the $7n/2 + o(n)$ time sorting algorithms for the three-dimensional mesh in [3, 6].) Hence, for large values of d, this upper bound is still nearly a factor of 2 away from the diameter lower bound. This is also true in the case of the d-dimensional torus, where the best upper bound is currently at $2D - n + o(n)$. (This bound is implied by the $2n + o(n)$ time sorting algorithms for the three-dimensional torus in [3, 6].)

Slightly better results have been obtained for permutation routing on d-dimensional networks. For this problem, Kunde [10] has described algorithms that run in time $(d + (d-2)(1/d)^{1/(d-2)} + \epsilon)n$ on the mesh, and in about half that time on the torus. For networks of low dimension, this is a significant improvement over previous results. However, for larger dimensions this result is again nearly a factor of 2 away from the diameter lower bound. In fact, even for off-line routing no better results are currently known.

For the problems of k–k routing and sorting on d-dimensional networks, there are obvious lower bounds of $kn/2$ for the mesh and $kn/4$ for the torus, due to the bisection width of the networks. For $k \geq 4r$, several randomized and deterministic algorithms have recently been proposed that match this lower bound, within a lower order additive term [5, 6, 12].

1.2 Overview of the Paper

In this paper, we show improved bounds for 1–1 routing and sorting on multi-dimensional meshes and tori. Our first result is a deterministic algorithm for sorting on multi-dimensional meshes of side length n that achieves a running time of $3D/2 + o(n)$. The algorithm has a fairly simple structure, and does not make any copies of the packets. Next, we show that the running time of the algorithm can be reduced to $5D/4 + o(n)$ by making one copy of each packet. A similar technique is then applied to the multi-dimensional torus, leading to a deterministic algorithm with a running time of $3D/2 + o(n)$. In contrast, the fastest previously known sorting algorithms required $2D - 5n/2 + o(n)$ steps on the mesh and $2D - n + o(n)$ steps on the torus.

Thus, our algorithms improve significantly over all previous results for sorting, and in fact even for off-line routing, on multi-dimensional meshes and tori. The ideas underlying our algorithms are quite simple, but the techniques used in the design and analysis are somewhat different from those in previous papers on multi-dimensional networks. While we restrict our attention in this paper to constant values of d, the claimed time bounds also hold for a limited range of networks of non-constant dimension.

In addition, we show lower bounds for sorting with respect to a large class of indexing schemes, under a model of the mesh where each processor can hold an arbitrary number of packets. Our lower bounds are the first non-trivial lower bounds for sorting in the multi-packet model of the mesh, and they imply that our upper bounds are nearly optimal on networks of sufficiently high constant dimension under a large class of indexing schemes. (In fact, we are not aware of any fast sorting algorithm for multi-dimensional networks that uses an indexing scheme not covered by our lower bound.) Using similar ideas, we can also establish a lower bound for selection on multi-dimensional meshes.

Finally, we describe algorithms for permutation routing on multi-dimensional meshes and tori whose running times nearly match the diameter lower bound. In particular, our algorithms achieve a running time of $D + \epsilon n$, for any $\epsilon > 0$ and d sufficiently large (depending on ϵ).

Due to space constraints, some of the proofs have been omitted from this abstract. A more detailed description of our results will be given in a later version of the paper.

The remainder of this paper is organized as follows. Section 2 contains some useful definitions and lemmas. Section 3 gives improved algorithms for sorting on multi-dimensional meshes and tori. Section 4 contains our lower bounds. Section 5 describes our algorithms for permutation routing. Finally, Section 6 lists some open questions for future research.

2 Preliminaries

In this section, we give some useful definitions and lemmas. We begin with a brief review of a "derandomization" technique for routing and sorting algorithms recently proposed in [6]. In Subsection 2.2, we state some results on greedy routing of random permutations.

2.1 Randomization and Unshuffling

In the following, we describe a "derandomization" technique proposed in [6], which can be used to convert many randomized algorithms for routing and sorting on meshes into deterministic algorithms. The technique is based on an operation called *sort-and-unshuffle*. The purpose of this operation is to evenly distribute packets with similar destinations (in the case of routing) or similar ranks (in the case of sorting) over some large region of the network, using a combination of local sorting and "unshuffling".

In the following, we assume a blocked snake-like indexing scheme on a d-dimensional mesh, where the blocks have side length n^α with $2/3 \leq \alpha < 1$. In the first step of the sort-and-unshuffle operation, the packets are sorted inside each block. In the second step, the packets of each block are distributed evenly over all the blocks. This is done by routing the packet of rank i, $0 \leq i < n^{d\alpha}$, in block j, $0 < j \leq n^{d(1-\alpha)}$, to position $j + \lfloor i/n^{d(1-\alpha)} \rfloor \cdot n^{d(1-\alpha)}$ in block $i \bmod n^{d(1-\alpha)}$. Note that this second step is an off-line routing problem; the particular permutation that has to be routed will be referred to as *unshuffle* permutation. (Note that if we lay out the processors of the network in a chain according to the indexing function, then this permutation is identical to an $(n^{d(1-\alpha)})$-way unshuffle operation on the chain.)

The structure of this unshuffle permutation exhibits many of the "nice" properties commonly associated with "average" or "random" permutations. In particular, the unshuffle permutation has the property that the destinations of the packets in any region of the network are approximately evenly distributed over the entire network. As a consequence, the unshuffle permutation can in most cases be routed as efficiently as a random permutation.

(In fact, this claim can be formalized under certain conditions.)

It was shown in [6] that the sort-and-unshuffle operation can in many cases be employed as a "substitute" for randomization. Following a scheme originally proposed by Valiant and Brebner [19], many randomized algorithms for routing and sorting 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 network. The sort-and-unshuffle operation simulates this effect in a deterministic manner. Using this relationship between randomization and the sort-and-unshuffle operation, the algorithms of this paper can be described in both randomized and deterministic terms.

2.2 Some Results on Greedy Routing

Routing is used as an important subroutine in many sorting algorithms for fixed-connection networks. In the algorithms presented in this paper, we need efficient routing schemes for random permutations (in the randomized case) and for unshuffle permutations (in the deterministic case). Our routing schemes are based on an extension of the standard *greedy* routing scheme. This routing scheme routes a packet on a d-dimensional mesh by moving it greedily towards its destination along edges of increasing dimension. In the case of edge contentions, priority is given to the packet with the farthest distance to travel.

In our extension of this routing scheme, k permutations are simultaneously routed by running d "copies" of the above greedy routing scheme. More precisely, we partition the set of packets into d sets $S_0, \ldots, S_{d-1}$ of approximately equal size, such that the origins and destinations of the packets in each set are approximately evenly distributed over the entire network. This can be done either in a randomized way, by having each packet choose a random set S_i, or in a deterministic way, by locally sorting blocks of side length $o(n)$, and defining S_i as the set of packets with a local rank y such that $y \bmod d = i$. The packets in set S_i are then routed along edges of increasing dimension modulo d, starting with dimension i and ending with dimension $(i-1) \bmod d$.

It is a natural question to ask how many random permutations can be routed simultaneously under the above routing scheme. To make this question more precise, we define the notions of *diameter-optimal* and *distance-optimal* routing. We say that a routing algorithm on a d-dimensional mesh is *diameter-optimal* if all packets are delivered to their destination in time $D + o(n)$, where D is the diameter of the network. We say that a routing algorithm is *distance-optimal* if each packet is delivered in time $S + o(n)$, where S is the distance between the source and the destination of the packet.

In the context of an optimal randomized algorithm for k–k sorting, it was shown by Kaufmann, Rajasekaran, and Sibeyn [5] that up to $4d$ random permutations can be routed diameter-optimally on d-dimensional meshes and tori, with high probability. The same bound can also be shown for the unshuffle permutation, leading to the optimal deterministic algorithm for k–k sorting in [6]. However, these results cannot be extended to the case of distance-optimal routing.

For the standard greedy routing scheme, Leighton [13] has shown that a random permutation can be routed distance-optimally on d-dimensional meshes and tori, with high probability. For the extended greedy routing scheme, we can show the following result.

Lemma 2.1 Up to $2d$ random permutations can be routed distance-optimally on the d-dimensional torus, with high probability.

Proof: (Sketch) Due to the structure of the torus, it suffices to consider the movement of the packets within a single dimension of the network. Initially, we assign 2 of the $2d$ random permutations to each dimension of the torus. Note that the destinations of the packets in each dimension are evenly distributed over all processors in the ring, and the distances between sources and destinations are evenly distributed over $[n/2]$. Whenever a packet reaches its destination in the ring, it moves on to the next higher dimension. On the other hand, we can assume that any new packet that arrives in the current dimension chooses a random destination within the ring. Thus, we can analyze the movement of the packets within the current dimension by considering an appropriate dynamic routing problem, as defined by Leighton [13]. □

Unfortunately, this analysis does not extend to the case of the d-dimensional mesh without wrap-around edges. In fact, it is not difficult to show that even d random permutations cannot be routed distance-optimally on the d-dimensional mesh under the extended greedy routing scheme. Using a more complicated analysis, we can show the following results for meshes without wrap-around edges.

Lemma 2.2 For $d \geq 3$, two random permutations can be routed distance-optimally on the d-dimensional mesh, with high probability.

Lemma 2.3 Up to $\lfloor d/2 \rfloor$ random permutations can be routed distance-optimally on the d-dimensional mesh, with high probability.

Lemmas 2.1, 2.2, and 2.3 also extend to the case of the unshuffle permutation. We believe that the result of Lemma 2.3 can be slightly improved, but we do not yet have an exact bound on the number of random permutations that can be routed distance-optimally on the d-dimensional mesh.

3 Upper Bounds for Sorting

In this section, we give improved algorithms for 1–1 sorting on multi-dimensional meshes and tori. In the first subsection, we describe the basic ideas underlying our algorithms. Subsections 3.2 and 3.3 contain our algorithms for multi-dimensional meshes and tori, respectively.

3.1 Basic Ideas

The basic ideas underlying our algorithms are quite simple. Consider the case of the d-dimensional mesh, and let C denote the set of processors that have a distance of at most $D/4$ from the center. It is not difficult to see that exactly half of the processors of the network are contained in this *center region* C. Also, no processor in C has a distance of more than $3D/4$ from any other processor of the network.

These observations lead to the following idea for a fast sorting algorithm. In the first phase, we concentrate all packets into the center region C, in such a way that packets of similar ranks are evenly distributed over C. This can be done either in a randomized or in a deterministic fashion. Next, we locally sort the packets inside each block of side length n^α (as defined by the blocked indexing scheme) that is contained in C. Since all packets were evenly distributed over the center region in the first phase, we can use the local ranks of the packets to obtain good approximations of the global ranks, and hence the final destinations, of all packets. In the third phase, we route each packet to some location in the block containing its approximate final destination. In the fourth phase, we use local sorting to bring each packet to its final destination.

Note that no packet has to travel a distance of more than $3D/4$ in the first or the third phase. Thus, if we can show that the routing in these two phases can be done distance-optimally, then the above scheme runs in time $3D/2 + o(n)$.

In the next subsection, we give a more detailed description of a deterministic algorithm based on the ideas presented in this section. We show that the routing problems in the first and third phase of the algorithm can be reduced to the simultaneous routing of several unshuffle permutations. We also present an even faster algorithm that makes one copy of each packet. In Subsection 3.3, we use similar ideas to obtain algorithms for the d-dimensional torus.

3.2 Sorting on Multi-Dimensional Meshes

In the following, we give fast deterministic sorting algorithms based on the ideas described in the previous subsection. We assume a blocked snake-like indexing scheme with blocks of side length n^α. In addition, we also assume an arbitrary fixed numbering of the $n^{d(1-\alpha)}/2$ blocks located in the center region C, independent of the indexing scheme. We begin with the following simple algorithm:

Algorithm SimpleSort:

(1) Sort the packets in each block of side length n^α.

(2) Distribute the packets in each block evenly over all blocks in C. This is done by routing the packet of rank i, $0 \le i < n^{d\alpha}$, in block j, $0 < j \le n^{d(1-\alpha)}$, to position $j + \lfloor i/n^{d(1-\alpha)} \rfloor \cdot n^{d(1-\alpha)}$ in block $i \bmod (n^{d(1-\alpha)}/2)$ in C. (Here, the numbering of the destination blocks is with respect to the arbitrary fixed numbering of the blocks in C.) Note that each processor in C receives exactly two packets.

(3) Sort the packets in each block of side length n^α in C.

(4) Send the packets in each block in C towards their destinations. This is done by routing the packet of rank i, $0 \le i < 2n^{d\alpha}$, in block j, $0 < j \le n^{d(1-\alpha)}/2$ of C to position $j + (i \bmod 2n^{d(2\alpha-1)}) \cdot n^{d(1-\alpha)}/2$ in block $\lfloor i/(2n^{d(2\alpha-1)}) \rfloor$. (Here, the numbering of the source blocks is with respect to the arbitrary fixed numbering of the blocks in C.) Note that each processor in the network receives exactly one packet.

(5) Perform two steps of odd-even transposition sort between neighboring blocks.

The correctness of the above algorithm is implied by the following lemma, which can be proved along the lines of Lemma 3.2 in [6].

Lemma 3.1 After Step (4) of Algorithm SimpleSort, each packet is at most one block away from its destination.

Next, we analyze the running time of the above algorithm. Clearly, Steps (1), (3), and (5) each run in time $O(n^\alpha) = o(n)$. For the routing in Step (2), the following can be shown.

Lemma 3.2 Step (2) of Algorithm SimpleSort can be reduced to the routing of two partial unshuffle permutations.

Proof: Consider the packets of a single unshuffle permutation π. Let S be the set of processors that contain a packet with destination in C. In each block, exactly half of the processors are in S, and the destinations of the packets in these processors are evenly distributed over all blocks in C. Let π' be the partial permutation consisting only of the packets that are initially located in S. After π' has been routed, we move the remaining half of the packets to the processors in S. By routing another instance of the partial unshuffle permutation π', we can now distribute these remaining packets evenly over the blocks in C. Of course, the two instances of π' can also be started simultaneously. $\Box$

By Lemma 2.2, we know that two partial unshuffle permutations can be routed distance-optimally on meshes of dimension $d \geq 3$. Since no packet has to travel a distance of more than $3D/4$, this implies that the routing is completed in time $3D/4 + o(n)$. Also note that the routing problem in Step (4) is exactly the inverse of the problem in Step (2), and therefore runs within the same time bound. This establishes the following result.

Theorem 3.1 For any constant d, there exists a deterministic sorting algorithm for the d-dimensional mesh with a running time of $3D/2 + o(n)$ that does not make any copies of the packets.

By Lemma 2.3, up to $\lfloor d/2 \rfloor$ unshuffle permutations can be routed distance-optimally on d-dimensional meshes. By modifying Algorithm SimpleSort appropriately, we can use this extra bandwidth to establish the following result for k–k sorting.

Corollary 3.1.1 If $k \leq \lfloor d/4 \rfloor$, then there exists a deterministic algorithm for k–k sorting on the d-dimensional mesh with a running time of $3D/2 + o(n)$ that does not make any copies of the packets.

We can also get a slight improvement in the running time for 1–1 sorting, by concentrating the packets into a smaller center region C. In general, however, the running time of this improved algorithm is still $(3/2 - \epsilon)D$, for all $\epsilon > 0$ and d sufficiently large (depending on ϵ).

Corollary 3.1.2 Let $C(r)$ be the set of processors of distance at most r from the center point. If $|C(r)| \geq 2n^d/d$, then there exists a deterministic sorting algorithm for the d-dimensional mesh with a running time of $D + 2r + o(n)$.

Next, we show that the time for 1–1 sorting can be reduced to $5D/4 + o(n)$ by making one copy of each packet. To do so, we modify Algorithm SimpleSort appropriately; the resulting algorithm is called CopySort. Steps (1), (3), and (5) remain the same as in Algorithm SimpleSort. The routing in Step (2) of SimpleSort is augmented as follows. As before, we distribute the packets evenly over the blocks in C. In addition, we make one copy of each packet, and route this copy to a processor in the unique block of the center region C that is located exactly on the opposite side of the center point than the destination processor of the original in this step, and that has the same distance from the center point. The routing of the copies can be done simultaneously with the routing of the originals, and the entire Step (2) can be implemented by routing four partial unshuffle permutations. By Lemma 2.3, the routing is completed in $3D/4 + o(n)$ steps for $d \geq 8$. The following lemma can be shown using simple geometric arguments.

Lemma 3.3 After Step (3) of Algorithm CopySort, no processor in the network is more than a distance of $D/2 + o(n)$ away from both the original and the copy of any packet.

In Step (4) of CopySort, we first delete either the original or the copy of each packet, depending on which one is farther away from the destination. Then the remaining packets are routed towards their destination. It can shown that this routing can again be implemented by four partial unshuffle permutations. By Lemma 3.3, no packet has to travel more than a distance of $D/2$. This establishes the following result.

Theorem 3.2 For any constant $d \geq 8$, there exists a deterministic sorting algorithm for the d-dimensional mesh with a running time of $5D/4 + o(n)$.

For larger values of d, this result can again be slightly improved by concentrating into a smaller center region. Alternatively, we can also adapt the algorithm to k–k sorting with $k \leq \lfloor d/8 \rfloor$.

3.3 Sorting on Multi-Dimensional Tori

In this subsection, we adapt the ideas of the previous subsections to the case of the d-dimensional torus. We describe a modification of the Algorithm CopySort from the previous subsection, which we refer to as TorusSort. As before, Steps (1), (3), and (5) perform local sorting operations. In Step (2), we distribute the packets evenly over the entire network. In addition, we also make a copy of each packet, and route this copy to a processor in the unique block in the network that is $D/2$ steps away from the destination processor of the original packet in this step. Step (2) can be implemented by routing two full unshuffle permutations; the routing takes time $D + o(n)$. Then the following lemma can be shown using simple geometric arguments.

Lemma 3.4 After Step (3) of Algorithm TorusSort, no processor in the network is more than a distance of $D/2 + o(n)$ away from both the original and the copy of any packet.

As before, half of the packets are deleted in Step (4), and the remaining packets are routed towards their destination. This routing can be implemented by two partial unshuffle permutations. By Lemma 3.4, no packet has to travel more than a distance of $D/2 + o(n)$. Using Lemma 2.1 we obtain the following result.

Theorem 3.3 For any constant d, there exists a deterministic sorting algorithm for the d-dimensional torus with a running time of $3D/2 + o(n)$.

By modifying Algorithm TorusSort appropriately, and using the extra bandwidth supplied by Lemma 2.1, we can establish the following result.

Corollary 3.3.1 For any constant d, there exists a deterministic algorithm for d–d sorting on the d-dimensional torus with a running time of $3D/2 + o(n)$.

Alternatively, we can also get a slight improvement in the running time for 1–1 sorting. As an example, we can obtain a fairly simple algorithm for the two-dimensional torus that uses four copies of each packet and runs in time $1.375n$. In general, however, the running time of the improved algorithm is still $(3/2 - \epsilon)D$, for all $\epsilon > 0$ and d sufficiently large (depending on ϵ).

4 Lower Bounds for Sorting

In this section, we establish lower bounds for sorting on multi-dimensional meshes and tori under the multi-packet model. The lower bounds hold for a large class of indexing schemes, including most of the indexing schemes used in the literature. Our lower bound technique is an extension of the *Joker Zone* argument of Kunde [7] and Schnorr and Shamir [17] to the multi-packet model. An important difference is that our lower bounds are based on edge capacity arguments, and do not place any limits on the number of packets that can be held inside a single processor. We begin the section with a few definitions.

We say that an indexing scheme $\mathcal{I}$ of the d-dimensional mesh is *compatible* if there exists a $\beta < 1$ such that for every index $i \in [n^d - n^{\beta d}]$, the set of processors with indices in $\{i, \ldots, i + n^{\beta d} - 1\}$ contains a complete $(d-1)$-dimensional subnetwork of side length n. (Informally speaking, this means that a compatible indexing scheme has the property that a joker zone of $n^{\beta d}$ packets suffices to move the final destination of a packet to any processor within a $(d-1)$-dimensional sub-network.) Note that the natural extensions of the row-major, snake-like, blocked row-major, and blocked snake-like indexing schemes to multi-dimensional networks are compatible indexing schemes. In the remainder of this section, we assume an arbitrary compatible indexing scheme with associated constant β.

We use $C_{d,\gamma}$ to denote the processors of a d-dimensional diamond of radius $(1-\gamma) \cdot D/4$ around the center of a d-dimensional mesh. (That is, the set of processors that have a distance of at most $(1-\gamma) \cdot D/4$ from the center.) The number of processors in $C_{d,\gamma}$ is denoted by $V_{d,\gamma}$, and the number of processors on the surface of $C_{d,\gamma}$ is denoted by $S_{d,\gamma}$. Then the following bounds can be shown.

Lemma 4.1 For any d and any $\gamma > 0$, we have

$$V_{d,\gamma} \leq e^{-\gamma^2 d/4} \cdot n^d$$

and

$$S_{d,\gamma} \leq \frac{8}{\gamma} \cdot e^{-\gamma^2 d/16} \cdot n^{d-1}.$$

4.1 Sorting without Copying

We first establish a lower bound for sorting under the restriction that no copies of the packets can be made. Our main lemma for this case is as follows.

Lemma 4.2 Let d and γ be chosen such that

$$d \cdot S_{d,\gamma} \cdot \left(\left(\frac{1}{2} + \frac{1-\gamma}{4} \right) \cdot D - dn^{\beta} \right) < n^d - V_{d,\gamma}$$

holds for large enough n. If no copying of packets is allowed, then sorting on the d-dimensional mesh with respect to an arbitrary compatible indexing scheme takes at least $D + (1-\gamma) \cdot D/2 - n - dn^{\beta}$ steps.

Proof: Consider the computation of an arbitrary sorting algorithm up to time $(\frac{1}{2} + \frac{1-\gamma}{4}) \cdot D - dn^{\beta}$. At most $d \cdot S_{d,\gamma}$ packets can enter the diamond $C_{d,\gamma}$ in each step. Thus, the above inequality implies that not all of the $n^d - V_{d,\gamma}$ packets that are initially outside the diamond can have entered up to this point.

Now consider an arbitrary packet located outside the diamond at time $(\frac{1}{2} + \frac{1-\gamma}{4}) \cdot D - dn^{\beta}$. This packet has a distance of at least $(\frac{1}{2} + \frac{1-\gamma}{4}) \cdot D$ from at least one of the corners of the network. (Otherwise, the packet would be in the diamond.) Thus, the present position of the packet is independent of the content of a block of side length n^{β} located in that corner.

As we assume a compatible indexing scheme, the content of this block can force the destination of the packet to be in any processor of a $(d-1)$-dimensional sub-network of side length n. There exists a processor in this sub-network that has a distance of at least $(\frac{1}{2}+\frac{1-\gamma}{4})\cdot D-n$ from the current position of the packet. Hence, at least $(\frac{1}{2}+\frac{1-\gamma}{4})\cdot D-n$ additional steps are needed under some assignment of values to the corner block. □

Theorem 4.1 If no copying of packets is allowed, then for every $\epsilon \geq 0$ there exists a d_0 such that for all $d \geq d_0$, sorting on the d-dimensional mesh with respect to a compatible indexing scheme takes at least $(3/2-\epsilon)D$ steps.

To establish this theorem, we use Lemma 4.1 to show that the condition in Lemma 4.2 holds for $\gamma = 3\epsilon$ and d sufficiently large (depending on ϵ), The claim then follows by a direct application of Lemma 4.2. Together with Theorem 3.2, this result establishes a separation between the complexities of sorting with and without copying, for large values of d. Unfortunately, Lemma 4.1 does not give any good bounds for small values of d. In this case, we can show lower bounds by adapting our argument to the particular network in question. In particular, we can establish the following theorem.

Theorem 4.2 If no copying of packets is allowed, then for $d \geq 5$ the diameter bound cannot be asymptotically matched under any compatible indexing scheme.

For the torus, it can be shown that the lower bounds for the single-packet model also extend to the multi-packet model, assuming that no copying is allowed. Informally speaking, the reason is that the torus does not have a center point towards which the packets could be routed.

4.2 Sorting with Copying

Our lower bound techniques can also be extended to a model in which unlimited copying of packets is allowed. For this case, we obtain the following result.

Theorem 4.3 If unlimited copying of packets is allowed, then for every $\epsilon \geq 0$ there exists a d_0 such that for all $d \geq d_0$, sorting on the d-dimensional mesh with respect to a compatible indexing scheme takes at least $(5/4-\epsilon)D$ steps.

We only describe the main ideas in the proof of the above theorem. The basic idea for this lower bound is that we choose the center diamond small enough such that only a small fraction of the packets can be routed into this diamond. Next, we argue that the edge bandwidth of the network does not allow every packet to distribute a large number of copies of itself over the network. (Formally, the number of communication steps required to route copies of a packet to a number of locations in the network is lowerbounded by the length of a minimal "broadcast tree" connecting these locations.) This implies that an appropriate loading of the joker zones can force the rank of a packet to be such that no copy is close to its destination.

However, this technique does not give any nontrivial lower bounds for reasonable values of d. We expect that some results for smaller d can be obtained by adapting our argument to the particular low-dimensional network in question. We plan to report the results of such an analysis in a later version of this paper. In the case of the torus, we obtain the following result.

Theorem 4.4 If unlimited copying of packets is allowed, then for every $\epsilon \geq 0$ there exists a d_0 such that for all $d \geq d_0$, sorting on the d-dimensional torus with respect to a compatible indexing scheme takes at least $(3/2-\epsilon)D$ steps.

The lower bounds can also extended to many noncompatible indexing schemes. In fact, it is not difficult to show that the above bounds hold for the vast majority of all possible indexing schemes. (A similar result for the single-packet model was described by Kunde [8].) Of course, such a result is not a very good measure for the generality of our lower bounds, since most indexing schemes are highly irregular and thus unsuitable for any efficient sorting scheme. More important in this respect is that we are not aware of any fast sorting algorithm that assumes an indexing scheme not covered by our lower bound. This indicates that any such algorithm would probably be quite different from those currently known.

4.3 Selection

Using similar ideas, we can also show a lower bound of $(9/16-\epsilon)\cdot D$ for the problem of selecting the median at the center processor of a high-dimensional mesh. A trivial lower bound for this problem is given by the radius of the network. (That is, $D/2$ for the multi-dimensional mesh and D for the multi-dimensional torus.)

By Lemma 4.1, we know that for any $\epsilon > 0$ and any sufficiently large d, only a small fraction of the packets can enter $C_{d,\epsilon}$ in the first $D/2$ steps of any algorithm. Let x be any processor outside $C_{d,\epsilon}$. Then the set of processors that have a distance of at most $(5/16-2\epsilon)\cdot D$ from x contains only a small fraction of the n^d processors in the network. This means that up to time $(5/16-2\epsilon)\cdot D$, no packet located outside $C_{d,\epsilon}$ can be "ruled out" as the median element. Hence, up to $(1-\epsilon)\cdot D/4$ additional steps are necessary to move the median to the center processor, and we get the following result.

Theorem 4.5 For every $\epsilon \geq 0$ there exists a d_0 such that for all $d \geq d_0$, selection on the d-dimensional mesh takes at least $(9/16 - \epsilon)D$ steps.

An upper bound of $D + o(n)$ can be obtained by a modification of the sorting algorithms in Section 3. For large values of d, this result can be improved to $(3/4+\epsilon)\cdot D$. On the multi-dimensional torus, a running time of $(1+\epsilon)\cdot D$ can be achieved for large d, thus coming very close to the trivial lower bound of D.

5 Permutation Routing

The lower bounds established in the previous section are restricted to the case of sorting. In this section, we show the existence of algorithms for permutation routing on multi-dimensional networks that nearly match the diameter lower bound. The algorithms are based on similar ideas as the sorting algorithms in Subsection 3. In particular, they use a similar reduction to the distance-optimal routing of a number of unshuffle permutations.

Consider the following idea for a randomized routing algorithm. For a packet with source processor x and destination processor y, we define $S(x,y)$ as the set of processors that have a distance of at most $D/2$ from both x and y. Note that $S(x,y)$ is non-empty for all x and y. Thus, a simple two-phase algorithm could route a packet with source x and destination y by first sending the packet to a random processor in $S(x,y)$, and then to its destination y. If we could solve the resulting two routing problems distance-optimally, then we would obtain a total running time of $D + o(n)$ for the algorithm.

Unfortunately, we do not know how to reduce these two routing problems to a small number of random or unshuffle permutations. To do so, we have to modify the above algorithm slightly. We define $S_\nu(x,y)$ as the set of processors that have a distance of at most $D/2+\nu$ from both x and y. In the first phase of the algorithm, we now route each packet with source x and destination y to a random processor in $S_\nu(x,y)$. In the corresponding deterministic algorithm, we partition the network into blocks of side length n^α, and distribute all packets with source in block X and destination in block Y evenly over $S_\nu(X,Y)$, the set of blocks that have a distance of at most $D/2+\nu$ from both block X and block Y.

If we choose ν such that $k \cdot |S_\nu(X,Y)| \geq n^d$ holds for all blocks X and Y, then we can reduce each phase of the algorithm to the simultaneous routing of k unshuffle permutations. For a block X, define $c(X)$ as the corner processor that is closest to X. Then we can lowerbound $S_\nu(X,Y)$ by $S_\nu(c(X),c(Y))$. An analysis shows that for $d \geq 4$ and $\nu = n/2$, we have $\lfloor d/2 \rfloor \cdot |S_\nu(c(X),c(Y))| \geq n^d$, and hence we can reduce each phase of the algorithm to the routing of $\lfloor d/2 \rfloor$ unshuffle permutations. Using Lemma 2.3, we obtain the following result.

Theorem 5.1 For all d, there exists a deterministic algorithm for permutation routing on the d-dimensional mesh with a running time of $D + n + o(n)$.

The routing scheme can be easily adapted to the multi-dimensional torus. For $d \geq 4$ and $\nu = n/16$, we have $2d \cdot |S_\nu(X,Y)| \geq n^d$, and by Lemma 2.1 we obtain the following result.

Theorem 5.2 For all d, there exists a deterministic algorithm for permutation routing on the d-dimensional torus with a running time of $D + n/8 + o(n)$.

Finally, a sharper analysis using bounds similar to those in Lemma 4.1 shows that in high-dimensional meshes (tori), most processors have a distance of around $D/2$ from any particular corner (any particular processor). This means that as d increases, we can choose smaller and smaller values for ν.

Theorem 5.3 For all $\epsilon > 0$, there exists a d_0 such that for all $d \geq d_0$, permutation routing can be done in time $D + \epsilon n$ on d-dimensional meshes and tori.

6 Concluding Remarks

In this paper, we have shown improved bounds for routing and sorting on multi-dimensional meshes and tori. While our bounds are nearly tight for high-dimensional networks, we do not obtain very good bounds for networks of small, fixed dimension. In particular, it is an interesting open question whether there exists an optimal algorithm for sorting on the two-dimensional mesh that does not make any copies, or whether any optimal sorting algorithm exists for some $d \geq 3$.

We are currently investigating whether the lower bounds for sorting can be extended to arbitrary indexing schemes. One possible approach to this problem is to adapt some of the techniques that have been used to show lower bounds for arbitrary indexing schemes in the single-packet model [1].

It would also be nice to obtain algorithms for permutation routing that match the diameter bound more closely. For example, one might try to overlap the two routing phases of the algorithm in Section 5, and bound the running time of the resulting algorithm. Finally, it is an open question whether the diameter and bisection bounds can be matched simultaneously for routing on networks of dimension $d \geq 2$.

Acknowledgements

I would like to thank Greg Plaxton and Rajmohan Rajaraman for helpful discussions.

References

[1] Y. Han and Y. Igarashi. Time lower bounds for parallel sorting on multidimensional mesh-connected processor arrays. *Information Processing Letters*, 33:233–238, 1990.

[2] Y. Han, Y. Igarashi, and M. Truszczynski. Indexing functions and time lower bounds for sorting on a mesh-connected computer. *Discrete Applied Mathematics*, 36:141–152, 1992.

[3] C. Kaklamanis and D. Krizanc. Optimal sorting on mesh-connected processor arrays. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 50–59, July 1992.

[4] C. Kaklamanis, D. Krizanc, L. Narayanan, and T. Tsantilas. Randomized sorting and selection on mesh-connected processor arrays. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 17–28, July 1991.

[5] M. Kaufmann, S. Rajasekaran, and J. F. Sibeyn. Matching the bisection bound for routing and sorting on the mesh. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 31–40, July 1992.

[6] M. Kaufmann, J. Sibeyn, and T. Suel. Derandomizing algorithms for routing and sorting on meshes. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 669–679, January 1994.

[7] M. Kunde. Lower bounds for sorting on mesh-connected architectures. *Acta Informatica*, 24:121–130, 1987.

[8] M. Kunde. Bounds for 1-selection and related problems on grids of processors. In *Proceedings of the 4th International Workshop on Parallel Processing by Cellular Automata and Arrays (PARCELLA)*, pages 298–307. Springer, 1988.

[9] M. Kunde. Routing and sorting on mesh-connected arrays. In J. H. Reif, editor, *VLSI Algorithms and Architectures: Proceedings of the 3rd Aegean Workshop on Computing,* Lecture Notes in Computer Science, volume 319, pages 423–433. Springer, 1988.

[10] M. Kunde. Balanced routing: Towards the distance bound on grids. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 260–271, July 1991.

[11] M. Kunde. Concentrated regular data streams on grids: Sorting and routing near to the bisection bound. In *Proceedings of the 32nd Annual IEEE Symposium on Foundations of Computer Science*, pages 141–150, October 1991.

[12] M. Kunde. Block gossiping on grids and tori: Deterministic sorting and routing match the bisection bound. In *Proceedings of the 1st Annual European Symposium on Algorithms*, September 1993.

[13] F. T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 2–10, July 1990.

[14] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes.* Morgan-Kaufmann, San Mateo, CA, 1991.

[15] D. Nassimi and S. Sahni. Bitonic sort on a mesh-connected parallel computer. *IEEE Transactions on Computers*, C–28:2–7, 1979.

[16] S. E. Orcutt. *Computer Organization and Algorithms for Very-High Speed Computations.* PhD thesis, Department of Computer Science, Stanford University, September 1974.

[17] C. P. Schnorr and A. Shamir. An optimal sorting algorithm for mesh-connected computers. In *Proceedings of the 18th ACM Symposium on Theory of Computing*, pages 255–263, May 1986.

[18] C. D. Thompson and H. T. Kung. Sorting on a mesh-connected parallel computer. *Communications of the ACM*, 20:263–271, 1977.

[19] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, May 1981.