Beyond the Worst-Case Bisection Bound: Fast Sorting and Ranking on Meshes*

Michael Kaufmann[1], Jop F. Sibeyn[2], Torsten Suel[3]

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, 72076 Tübingen, Germany. Email: mk@informatik.uni-tuebingen.de.

[2] Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany. Email: jopsi@mpi-sb.mpg.de.

[3] NEC Research Institute, 4 Independence Way, Princeton, NJ 08540, USA. Email: torsten@research.nj.nec.com.

Abstract. Sorting is an important subroutine in many parallel algorithms and has been studied extensively on meshes and related networks. If every processor of an $n \times n$ mesh is the source and destination of at most k elements, then sorting requires at least $k \cdot n/2$ steps in the worst-case, and simple algorithms have recently been proposed that nearly match this bound. However, this lower bound does not extend to non-worst-case inputs, or weaker definitions of sorting that are sufficient in many applications. In this paper, we give algorithms and lower bounds for several such problems.

We first present a very simple scheme for k-k routing that performs optimally under both average-case and worst-case inputs. As an application of this scheme, we describe a simple k-k sorting algorithm based on sample sort that nearly matches this bound.

The main part of the paper considers several 'sorting-like' problems. In the ranking problem, the ranks of all elements have to be determined, but there is no requirement about their final positions. We describe an algorithms running in time $(1 + o(1)) \cdot k \cdot n/4$ steps, which is nearly optimal under the considered model of the mesh. We show that the integer versions of the sorting and ranking problems, where keys are drawn from $\{0, \ldots, m-1\}$, can be solved asymptotically faster than the general problems for small values of m. A related problem, the excess counting problem, can be solved in $O(n)$ steps in many interesting cases.

1 Introduction

One of the most thoroughly investigated interconnection schemes for parallel computation is the $n \times n$ *mesh*, in which a set of n^2 processing units (PUs) is connected by a two-dimensional grid of communication links. While the mesh has a large diameter in comparison to the various hypercubic networks, it is nonetheless of great importance due to its simple structure and efficient layout. A number of parallel machines with mesh topology have been built, and a variety of algorithmic problems have been analyzed on theoretical models of the mesh.

1.1 Routing and Sorting

Routing and sorting are probably the two most extensively studied algorithmic problems on fixed-connection networks. In a routing problem, a set of *packets* has to be redistributed in the network such that every packet ends up at the PU specified in its destination address. Here, the address of a PU is determined by some fixed numbering of the PUs called an *indexing scheme*. A routing problem in which each PU is the source and destination of at most k packets is called a k-k routing problem.

In the k-k sorting problem, instead of a destination address each packet contains a key from a totally ordered set, and the packets have to be rearranged such that the packet of rank i ends up at the PU with index $\lfloor i/k \rfloor$, for all i. Thus, in a routing problem the destinations of the packets are given as part of the input, while in a sorting problem, the

* Part of this work was done while the third author was visiting the Max-Planck-Institut.

destinations have to be computed from the given key values. For an introduction into the problems of routing and sorting, and a survey of basic results, we refer the reader to [13].

There is a trivial lower bound of $2 \cdot n - 2$ for routing and sorting on the two-dimensional mesh due to the diameter of the network. For large k, this bound is dominated by the lower bound of $k \cdot n/2$ for k-k routing and sorting due to the bisection width of the network, and several algorithms running in $(1 + o(1)) \cdot k \cdot n/2$ steps have recently been presented [8, 10, 9, 24].

Note that, while the diameter lower bound of $2 \cdot n - 2$ (or $n - 1$ for algorithms that output the result at a single PU) actually applies to most non-trivial algorithmic problems on the mesh, the bisection bound of $k \cdot n/2$ does not seem to extend to any general class of problems with k items per PU. As a trivial example, we point out that computing the prefix sums takes $2 \cdot n - 2$ steps with one item per PU, and $2 \cdot n + O(k)$ steps with k items per PU. On the other hand, the bisection lower bound of $k \cdot n/2$ for k-k sorting relies on the assumption that the final positions of the packets are given by a fixed indexing scheme, and that we can thus force all $k \cdot n^2/2$ packets initially located in one half of the network to move to the other half. If we drop this assumption, and only require the algorithm to compute the ranks of the packets, then this simple lower bound no longer applies.

1.2 Sorting-Like Problems.

In this paper, we study several problems that are closely related to sorting, but that are not covered by the above worst-case bisection bound for k-k sorting. These problems are motivated by the observation that in many applications of parallel sorting it is actually not necessary to perform a sorting operation in its most general form, but it suffices to solve a slightly weaker problem.

One example of such a problem is the *ranking* problem, already alluded to above, in which the ranks of all packets have to be computed, but the final positions of the packets are not fixed. Many applications, e.g., in scientific computing, use ranking as a subroutine. In fact, the 'Integer Sort' of the NAS Parallel Benchmarks [1], which is probably the most widely used set of benchmark problems for parallel machines, requires only a ranking, rather than sorting, of the keys.

Other problems considered in this paper are the *integer sorting*, *integer ranking*, and *excess counting* problems. Among other things, they have applications in PRAM emulations, list ranking, and unbalanced communication problems on fixed-connection networks. In the following, we describe and motivate these problems in more detail.

Ranking. Let $\mathcal{P} = \{p_0, \ldots, p_{m-1}\}$ be a set of packets, where each packet p_i contains a value x_i drawn from some totally ordered set X. The ranking problem is the problem of computing, for each packet p_i, the function $Rank\,(p_i, \mathcal{P}) = |\{j \mid x_j < x_i \vee (x_j = x_i \wedge j < i)\}|$.

Note that a sorting problem can be solved by first ranking the packets, and then routing each packet to the destination corresponding to its rank. In the case of the sequential RAM model, as well as in the shared-memory PRAM, the time for permuting the data in the second step is dominated by that for computing the ranks. However, this is not the case for fixed-connection networks of bounded degree such as the mesh, where routing can take a significant portion of the time for sorting. On such networks, ranking can sometimes be performed more efficiently than sorting. Given the close relation between routing, sorting, and ranking described above, we can also think about the ranking problem as capturing some of those features that distinguish sorting from routing.

As we will show, ranking takes only about half as much time as sorting on the mesh. While the definition of ranking allows the final positions of the packets to be arbitrary, our algorithm in fact arranges the packets in a fairly regular pattern. More precisely, the packets are arranged in blocks of side length $o(n)$, such that each block contains only packets with consecutive ranks; we refer to this as a *blocked ranking*. As it turns out, a blocked ranking is advantageous in some applications. For example, a blocked ranking is required as part of the list ranking algorithm for fixed-connection networks in [26].

Integer Sorting and Ranking. The *integer sorting* and *integer ranking* problems are special cases of the sorting and ranking problems where the values of the keys are restricted to the set $\{0, \ldots, m-1\}$. This problem is motivated by the observation that in many applications the keys are drawn from a fairly small set of values, often not much larger than the input size. (In fact, this assumption is part of the definition of the 'Integer Sort' in the NAS Parallel Benchmarks [1].) For example, each key could be the index of a PU, or a pointer into an array or other data structure.

We have to be a little bit careful in the definition of the two integer problems. We assume that in the *integer ranking* problem, each PU has to compute the ranks of the packets initially located in it. In the integer sorting problem, at the end of the computation the PU with index i has to contain the key value belonging to the packet of rank $\lfloor i/k \rfloor$. Note that the definition of integer sorting is fairly weak in that it does not require any way of determining the origins of the key values in the final output. In fact, it turns out that integer sorting is asymptotically faster than integer ranking. We will show that both problems can be solved asymptotically faster than general sorting as long as m is not significantly larger than $k \cdot n^2$.

Excess Counting. A problem related to integer sorting is the *excess counting* problem, which asks to mark all packets whose key values appear more than t times. Excess counting can be used to detect imbalances, say, in irregular communication problems, or in the distribution of key values. The application that originally motivated our interest in this problem arises in the context of a deterministic PRAM emulation on the mesh [18]. The algorithm maintains $2 \cdot c - 1$ copies of each memory location, for some constant c, that are distributed over the network. In order to access a memory location, it suffices to access c of the $2 \cdot c - 1$ copies [17]. In a simulation of the algorithm of [18], requests for all $2 \cdot c - 1$ copies are generated, and then excess counting is used to identify PUs that receive a large number of requests and eliminate some of these requests, thus speeding up the routing of the remaining packets [16].

The running time of our algorithm for excess counting only depends on the threshold t, and is independent of m, the size of the set of possible key values. If $t = \Omega(n^\epsilon)$, for some $\epsilon > 0$, then excess counting can be solved in time proportional to the diameter of the network. This is significantly faster than the corresponding integer sorting operation (which could also be used to solve the problem).

1.3 Standard vs. Arbitrary Model

Before we give an overview of the contributions of this paper, a few remarks about our model of computation are in order. A common assumption in the literature on routing and sorting is that packets are 'atomic entities' whose contents can only be accessed and manipulated in a very restricted fashion. More precisely, each packet consists of a destination address or key value plus some additional data, and the algorithm can read the destination or key but cannot read or alter the data. In each step, a single packet plus $O(\log n)$ bits of auxiliary information can be transmitted across any edge.

In the case of sorting, it is also commonly assumed that the only way to access a key value is by means of a comparison with another key located in the same (or a neighboring) PU. This model, which we will refer to as the *standard model*, is the one assumed in our results for general sorting and ranking.

However, the standard model does not seem appropriate for problems such as integer sorting and excess counting, which are more efficiently solved by non-comparison-based methods. For these problems, we assume a less restrictive model, called the *arbitrary model*, which allows unrestricted access and manipulation of the input. If the keys are chosen from $\{0, \ldots, m-1\}$, then we assume that $\log m + O(\log n)$ bits can be transmitted across a communication link in a single step; this allows us to compare the running times with those for general ranking under the standard model. (All results can also be transformed into a 'bit model', by multiplying the stated bounds by a factor of $\log m + O(\log n)$.)

Many lower bounds for the standard model, say the $k \cdot n/2$ worst-case lower bound for k-k routing and sorting, follow directly from the observation that only a bounded number of packets can cross a given bisection of the network in a single step; we call such a lower bound a *standard bisection bound*. However, when proving bisection-based lower bounds in a less restrictive model, we have to be more careful, as it may not be immediately clear how much information has to be exchanged between different areas of the network in order to compute the given function (e.g., see [7]). In the case of the arbitrary model, lower bounds can be established with combinatorial arguments similar to those used in VLSI theory [14]; we call such a bound an *information-theoretic bisection bound*.

1.4 Overview of this Paper

In this paper, we study several problems related to routing, sorting, ranking, and counting on the mesh. In addition to providing algorithms and lower bounds, we also attempt to motivate and discuss these problems in a more general context.

In Section 2 we review some recent algorithms for k-k routing and sorting on meshes, and describe their relation to the total-exchange operation, an important communication primitive in parallel computation. We present a modified scheme for k-k routing that nearly matches the bisection bound for both average-case and worst-case inputs. Finally, we discuss an application of this scheme to the efficient implementation of sorting algorithms on parallel machines.

Section 3 considers the ranking problem with unrestricted key length. We prove a lower bound of $k \cdot n/4$ in the standard model (where k is the number of items per PU), and provide nearly matching randomized and deterministic upper bounds.

In Sections 4 and 5 we consider the integer sorting, integer ranking, and excess counting problems under the arbitrary model. We give nearly tight bounds for all of these problems, and show that in most cases they can be solved asymptotically faster than the general sorting problem.

1.5 Model of Computation

Our model of computation is the d-dimensional *mesh*. It consists of $N = n^d$ *processing units* (PUs) laid out in a d-dimensional grid of side length n, where every PU is connected to each of its (at most) $2 \cdot d$ immediate neighbors by a bidirectional communication link. We assume that in a single step a PU can exchange a bounded amount of data with each of its neighbors, as defined in the following paragraph. For the sake of simplicity, we assume that a PU can perform an unbounded amount of internal computation in each step. Unless stated otherwise, all presented algorithms can be adapted to run in the same time bounds on a model with bounded internal computation, provided the input size is at most polynomial in n.

In the *standard model*, assumed in Sections 2 and 3, a single packet, plus $O(\log n)$ bits of auxiliary information, can be transmitted across a communication link in each step. The only way of accessing a key value in the standard model is by performing a comparison with another key located in the same PU. In the *arbitrary model*, assumed in Sections 4 and 5, the key values can be manipulated in an arbitrary fashion, and up to $\log m + O(\log n)$

bits of data can be transmitted across a directed communication link in each step, where the key values are chosen from $\{0, \ldots, m-1\}$.

Throughout the paper, we focus on the case $d = 2$. For most results, generalizations to meshes of higher dimension, as well as toroidal networks, follow immediately. We also assume that $k = \omega(1)$, which is the most interesting case, and which allows for a more succinct statement of many of our upper bounds.

2 Routing, Sorting, and the Total-Exchange Operation

In this section, we briefly review the basic ideas underlying the recent optimal algorithms for k-k routing and sorting on the mesh [8, 10, 9]. In particular, we focus on the relation of these algorithms to the total-exchange operation [2], a communication primitive used in many parallel algorithms. We then present a very simple routing scheme based on the total-exchange operation which achieves nearly optimal performance on both average-case and worst-case inputs, and which we believe to be of practical interest. Finally, as an application of our routing scheme, we describe an efficient implementation of a sorting algorithm based on sample sort. In the description of the algorithms in this section, we omit most details about their exact implementation on the mesh, in order to keep the presentation as simple and general as possible.

2.1 Total-Exchange Operation

The *total-exchange* operation (also sometimes called *personalized all-to-all* or *gossiping*) is a communication primitive in which each PU of a parallel machine sends a different message of fixed length to each other PU. If each message consists of w packets of data, then we say that the total-exchange has size w. The total-exchange operation arises in many parallel applications [2], and its simple and regular structure allows for a very efficient implementation (relative to the available bandwidth) [5]. As a consequence, the total-exchange operation has also been studied with respect to its ability to efficiently perform other, more general communication patterns (e.g., see [5, 20]).

Given a partition of a mesh into blocks, we can also define the *blocked total-exchange* operation, in which the blocks play the role of the PUs in the total-exchange, that is, each block sends a fixed amount of data to each other block.

Several optimal randomized [8] and deterministic [10, 9] schemes for k-k routing and sorting on the mesh have recently been proposed. The randomized schemes are based on work by Valiant [27] and Reif and Valiant [21], while the deterministic schemes can be viewed as efficient implementations of Leighton's Columnsort [12]. The basic structure of these algorithms consists of two phases, with each phase containing a blocked total-exchange. This is most explicitly described in the deterministic algorithms in [9, 10], where the network is partitioned into m sufficiently large blocks, and each block B_i is subdivided into m *buckets*, $b_{i,0}, \ldots, b_{i,m-1}$. In every blocked total-exchange, the content of bucket $b_{i,j}$ in B_i is sent to bucket $b_{j,i}$ in B_j. Following is a high-level description of the algorithms.

1. In the first phase, each block distributes its packets 'approximately evenly' over its buckets. Then it calls a blocked total-exchange of size $w = k \cdot N/m^2$. The distribution of the packets over the buckets can either be done deterministically by sorting the packets inside each block B_i and placing the packet of rank j into bucket $b_{i,j \bmod m}$, or we can use randomization.

2. In the second phase, each block B_i sorts its packets, and places the packet of rank j into bucket $b_{i,\lfloor j/w \rfloor}$. Then another blocked total-exchange of size w is performed. Afterwards, local operations are used to bring the packets to their destinations.

It can be shown that after the second blocked total-exchange, each packet is within one block of its final destination (assuming that the blocks are sufficiently large, and that blocks with consecutive indices are adjacent in the network). Note that in the case of routing, we can simply place each packet into the bucket corresponding to its destination block during the second phase. This results in an approximately equal number of packets in each bucket, and the size of the second total-exchange is then determined by (an upper bound on) the maximum number of packets in any bucket.

The running time of this scheme is dominated by the time for the two blocked total-exchange operations. In [9, 10] it is shown that a blocked total-exchange of size $k \cdot n^2/m^2$ can be performed in $(1+o(1)) \cdot k \cdot n/4$ steps on the $n \times n$ mesh. This is also clearly optimal, and leads to a running time of $(1 + o(1)) \cdot k \cdot n/2$ for k-k routing and sorting.

2.2 Routing with Good Average-Case and Worst-Case Performance

The approach described in the previous subsection is nearly optimal with respect to worst-case inputs, in which all $k \cdot N$ packets have to cross the bisection of the network. However, the running time is a factor of 2 away from the lower bound with respect to average-case inputs, in which only half of the packets have to cross the bisection.

It is not difficult to modify the scheme such that it runs in optimal time on average-case inputs, by essentially omitting the first phase [11]. (This is because the sole purpose of the first phase is to distribute the packets evenly over the network, thus reducing a worst-case to an average-case problem.) Unfortunately, this one-phase algorithm has a miserable worst-case performance. Of course, one could add an additional step to the algorithm that scans the input to decide whether it should be treated as worst case or average case. While this approach may solve the problem from a purely theoretical point of view, it seems unlikely that it would lead to any practical and elegant routing schemes.

In the following, we describe a simple refinement of the above two-phase routing scheme that has both good average-case *and* good worst-case performance. A somewhat similar idea, although in a different context, is described in [20]. For the remainder of this subsection, we restrict our attention to routing problems. Our scheme consists of the following two phases.

1. In the first phase, each block B_i computes w_i, the minimum, over all blocks B_j, of the number of packets that have to be sent from B_i to B_j. It then places w_i packets with destination B_j into each bucket $b_{i,j}$, and distributes all remaining packets evenly over the buckets, either by sorting with respect to destination blocks, or through randomization. Then a blocked total-exchange of size $w = k \cdot N/m^2$ is performed. This delivers (at least) $m \cdot w_i$ packets from each B_i to their destination blocks, and distributes the remaining $k \cdot N - m \cdot \sum_i w_i$ packets evenly over the network.

2. In the second phase, each B_i places all remaining packets with destination in B_j into bucket $b_{i,j}$. Then it calls a blocked total-exchange operation of size approximately $w' = k \cdot N/m^2 - \sum_i w_i/m$. Afterwards, local operations can be used to bring the packets to their final destinations.

It can be shown that for a random input we have $w' = o(w)$ with high probability. Hence, the total running time for this case is dominated by the cost of the first total-exchange, and we obtain the following result.

Theorem 1 *The described scheme performs k-k routing on meshes in $(1 + o(1)) \cdot k \cdot n/2$ steps for all distributions, and in $(1 + o(1)) \cdot k \cdot n/4$ steps on the average.*

Thus, we achieve optimality for both average-case and worst-case inputs. In addition, the algorithm also seems to perform well on inputs that are ‘between average and worst case’, although it does not achieve optimality on every possible input,

2.3 Application to Sorting

As a simple application of our routing scheme, we can obtain a sorting algorithm based on sample sort [22, 21, 3] that performs optimally on both worst-case and average-case inputs. Informally speaking, sample sort uses randomly [22, 21, 3] or deterministically [9] selected splitters in order to reduce a sorting problem to a routing problem. In many cases, the time for solving this routing problem dominates the time for selecting and handling the splitters. (Although for realistic problem sizes the latter may also be significant, see [25] for a discussion.) The routing problem resulting from this reduction can be either worst case or average case (or somewhere in between), depending on the input to the sorting problem. By applying the routing scheme from the previous subsection, we obtain the following result.

Theorem 2 *There are simple randomized and deterministic algorithms that perform k-k sorting on meshes in* $(1 + o(1)) \cdot k \cdot n/2$ *steps for all distributions, and in* $(1 + o(1)) \cdot k \cdot n/4$ *steps on the average.*

While in our algorithm the routing times for the average case and worst case differ by only a factor of two, this gap can be significantly larger in actual implementations of sample sort on parallel machines (e.g., see [3, 4]), due to the algorithm used in the routing phase (and partly also due to the router). Because most parallel machines can efficiently implement highly regular communication patterns such as the total-exchange, we believe that the simple routing scheme in Section 2.2 may provide a practical solution to the problem of worst-case key distributions in sample sort and related algorithms.

3 Ranking

In this section, we give nearly tight bounds for ranking in the standard model. We first prove a lower bound, and then present an algorithmic scheme that leads to nearly optimal randomized and deterministic solutions. We assume that every PU initially holds k packets.

3.1 Lower Bound

We prove a simple lower bound for ranking in the standard model, where we can only compare two key values located in the same PU:

Theorem 3 *Any randomized or deterministic algorithm for ranking requires at least* $k \cdot n/4$ *steps on the standard model of the mesh.*

Proof: Partition the packets into pairs (l_i, r_i), $0 \leq i < k \cdot n^d/2$, where each l_i is initially located in the left half, and each r_i is initially located in the right half of the network. Consider all assignments of key values to the packets such that $val(l_i) = 3 \cdot i + 1$ and $val(r_i) \in \{3 \cdot i, 3 \cdot i + 2\}$, for all $0 \leq i < k \cdot n^d/2$. Thus, in order to compute the rank of l_i, it is necessary to perform a comparison between l_i and r_i. Since a comparison can only be performed between packets located in the same PU, at least one packet in each of the $k \cdot n^d/2$ pairs has to cross the bisection. As at most $2 \cdot n^{d-1}$ packets can cross the bisection in a single step, the theorem follows. □

In the arbitrary model, we have to be more careful, since an algorithm might try to perform a comparison between packets in different areas of the network with only partial knowledge of their key values. In fact, there are well-known randomized protocols that compare two κ-bit values located in different processors by communicating $o(\kappa)$ bits between the processors [28]. If the range of key values is sufficiently large, then these protocols can be used to obtain asymptotically faster algorithms for ranking in the arbitrary model.

3.2 Basic Scheme

We now present our basic algorithmic scheme for ranking, which can be seen as an extension of the well-known sample sort algorithm [3, 21, 22]. We will later use this scheme to obtain nearly optimal randomized and deterministic solutions.

Partition the mesh into g square blocks of equal size, called *G-blocks*, where $g = \omega(1)$. Let $f = \omega(g)$ be a multiple of g. (Suitable choices for g and f are discussed further below.) Our basic scheme consists of the following steps.

1. Select a global set of $s = \omega(f)$ approximately evenly spaced splitter elements, and compute the exact global ranks of the splitters. Then broadcast the splitters and their ranks to all G-blocks.

2. Use the splitters to estimate for every packet p_i its ranks r_i, and define $R_i = \lfloor f \cdot r_i/(k \cdot n^2) \rfloor$. Each R_i, $0 \leq R_i < f$, is the index of the *destination interval* of p_i.

3. Compute a suitable assignment of destination intervals to G-blocks, such that f/g intervals are assigned to each G-block.

4. Route each packet to the G-block to which its destination interval was assigned.

5. Complete the ranking by locallysorting the packets in the G-blocks.

Lemma 1 *Step 1, 2 and 5 can be performed in $o(k \cdot n)$ steps. At most $(1 + o(1)) \cdot k \cdot n^2/f$ packets are allocated to any destination interval.*

Proof: The selection of the splitters and computation of their global ranks in Step 1 can be performed in $o(k \cdot n)$ steps using either randomized [8, 21, 22] or deterministic [9] sampling techniques. Step 2 can be performed in time $o(k \cdot n)$ by sorting the packets together with the splitters in the G-blocks.

Every destination interval is assigned approximately $k \cdot n^2/f$ packets, up to a small inaccuracy due to packets that lie between two splitters that are in different destination interval. This inaccuracy is bounded by the maximum number of packets between any two splitters. If s is chosen sufficiently larger then f, then this is number is lower-order compared to $k \cdot n^2/f$. □

It remains to show how we can assign the destination intervals to the G-blocks such that the routing in Step 4 can be performed efficiently. The assignment must be such that the routing is as 'balanced' as possible. More precisely, we want to minimize the maximum number of packets that have to be sent from any G-block to any other G-block. This goal is justified by the following lemma.

Lemma 2 *If at most $(1 + o(1)) \cdot k \cdot n^2/g^2$ packets have to be routed between any two G-blocks, then the routing in Step 4 can be performed in $(1 + o(1)) \cdot k \cdot n/4$ steps.*

Proof: Apply a blocked total-exchange of size $(1 + o(1)) \cdot k \cdot n^2/g^2$ with respect to the G-blocks, as defined in Section 2.1. □

Thus, if we can efficiently find a good assignment, then we immediately obtain a ranking algorithm running in $(1 + o(1)) \cdot k \cdot n/4$ steps. We formalize the problem as follows. Let $M = k \cdot N = k \cdot n^2$ be the total number of packets, and let $A = (a_{i,j})$ be a $g \times f$ matrix, where entry $a_{i,j}$ gives the number of packets in the ith G-block belonging to the jth destination interval. It follows that

$$0 \leq a_{i,j} \leq M/f, \text{ for all } 0 \leq i < g, 0 \leq j < f,$$
$$\textstyle\sum_j a_{i,j} = M/g, \text{ for all } 0 \leq i < g, \text{ and}$$
$$\textstyle\sum_i a_{i,j} = M/f, \text{ for all } 0 \leq j < f.$$

We have to partition the set of columns into g disjoint subsets of f/g columns each, such that every row sum in every subset is at most $(1+o(1))$ times the average value. More formally, we have to construct an $f \times g$ zero-one matrix $S = (s_{i,j})$, with

$$\sum_j s_{i,j} = 1, \text{ for all } 0 \leq i < g, \qquad \sum_i s_{i,j} = f/g, \text{ for all } 0 \leq j < f, \tag{1}$$

satisfying

$$(A \cdot S)_{i,j} = (1 + o(1)) \cdot M/g^2, \text{ for all } 0 \leq i, j < g. \tag{2}$$

3.3 Randomized Solution

We now show that a randomly chosen matrix S satisfies (2) with high probability (i.e., with failure probability at most $M^{-\epsilon}$, for some $\epsilon > 0$). A similar idea was mentioned in [3] in the context of an implementation of sample sort on the CM-5, where the authors propose to 'randomize the locations of the buckets' in order to make the performance of the algorithm independent of the key distribution.

Thus, we randomly select a matrix S satisfying (1) and prove that (2) holds with high probability given an appropriate choice of f and g. Note that (2) imposes g^2 conditions that must be satisfied simultaneously. If any single condition is violated with probability at most $M^{-\epsilon}/g^2$, then the probability that any of them is violated is at most $M^{-\epsilon}$.

Lemma 3 *If $f = \omega(\ln^{1/3} M)$, then a randomly selected zero-one matrix S that satisfies (1) also satisfies (2) with high probability.*

Proof: We can bound the probability for a single condition by analyzing the following situation. Given a multi-set $\mathcal{T}$ of f numbers between 0 and M/f whose sum is equal to M/g, we have to bound the probability that the sum of the elements of a random subset $\mathcal{S}$ of size f/g exceeds $M/g^2 + t$, for $t = o(M/g^2)$. The expected value of this sum is M/g^2.

A majorization argument shows that the probability that the sum of the values of the elements in $\mathcal{S}$ exceeds $M/g^2 + t$ is smaller than the probability that the sum of f/g independently selected elements X_i, with $0 \leq X_i \leq M/f$ and with expected value M/g^2, exceeds $M/g^2 + t$ (see [23] and the references therein). Applying Hoeffding's Inequality [6] this probability can be estimated as follws.

$$Pr(\sum_i X_i \geq M/g^2 + t) \leq \exp(-2 \cdot f^3 \cdot t^2/(g \cdot M^2)).$$

So, we should take

$$t(f, g, M) = c \cdot M \cdot \ln^{1/2} M \cdot g^{1/2}/f^{3/2},$$

for some constant c, to get the desired bound on the probability. In order that $t(f, g, M) = o(M/g^2)$, we must have $\ln^{1/2} M \cdot g^{5/2} = o(f^{3/2})$. Since the only condition on g is $g = \omega(1)$, the lemma follows. □

3.4 Deterministic Solution

We now derive a deterministic solution by performing a (very simple) derandomization [15] of the above algorithm. To do so, we show that the parameters f, g in the randomized solution can be chosen such that the sample space is of size $O(N)$ and the success probability is non-zero. We can then search the entire sample space by assigning a constant number of samples to each of the N PUs.

Lemma 4 *Let k be polynomial in N. If $f = \log N/\log\log N$ then it suffices to test at most N matrices satisfying (1) in order to find one that also satisfies (2).*

Proof: Lemma 3 implies that for $f = \omega(\ln^{1/3} M)$ a randomly selected S satisfies (2) with high probability. Hence, there *exists* an S that satisfies (2). The size $s(f,g)$ of the sample space can be estimated as

$$s(f,g) = \frac{f!}{((f/g)!)^g} < f^f.$$

Thus, if we choose $f \leq \log N / \log\log N$, then $f^f < N$. Assuming that k is polynomial in N, the condition $f = \omega(\ln^{1/3} M)$ is also satisfied. □

Since the matrices that have to be tested can be generated in a systematic way, the tests can be performed in a distributed fashion. Thus, Lemma 4 implies that every PU has to generate and test only one $f \times g$ zero-one matrix. This takes $\mathcal{O}(f \cdot g^2)$ time. Since $f < \log N$, this time is negligible in comparison to the routing time. Combining all results from Lemma 1 to 4, we obtain the main result of this section.

Theorem 4 *If every PU initially holds k packets, then the ranking problem can be solved deterministically in time $(1 + o(1)) \cdot k \cdot k/4$ on the $n \times n$ mesh.*

It follows from Lemma 1 that at most $(1 + o(1)) \cdot k \cdot n^2/g$ packets are routed to any G-block. If desired, we can redistribute the packets in $o(k \cdot n)$ steps such that every PU holds exactly k packets at the end.

The large lower-order terms make the deterministic algorithm impractical for reasonable values of n and k.[4] The probabilistic algorithm, on the other hand, has a fairly simple structure and may be of practical interest.

4 Integer Ranking and Sorting

In this section, we consider the integer ranking and integer sorting problems, where the keys are restricted to the set $\{0, \ldots, m-1\}$. We will show that both problems can be solved asymptotically faster than general sorting, provided that m is not much larger than $k \cdot n/2$. Throughout this section, we assume the arbitrary model of the mesh, which is more useful than the comparison-based standard model for problems with restricted key size. We begin with the following lower bound (which, of course, also extend to the more restricted standard model.)

Theorem 5 *If $m = O(k \cdot n^2)$, then any deterministic algorithm for integer sorting requires $\Omega(n + \sqrt{m \cdot k}/\log n)$ steps on the arbitrary model of the mesh.*

Proof: (Sketch) Consider any two square blocks B_0 and B_1 of side length $\sqrt{\frac{m}{2k}}$ in the mesh. Let X be the set of those inputs where every possible key value appears exactly once in $B_0 \cup B_1$, while all keys in the rest of the network have value 0. Note that every input in X results in the same output. If we do not distinguish between inputs that can be obtained from each other by only permuting keys within B_0 or within B_1, then X contains $\binom{m}{m/2}$ distinct inputs.

A simple crossing-sequence argument (e.g., see [14]) shows that at least $\log(|X|) = \Omega(m)$ bits of data have to be communicated between B_0 and B_1, since otherwise there is an input π that coincides with some $\pi_0 \in X$ in B_0, and with some $\pi_1 \in X$, $\pi_0 \neq pi_1$, in the rest of the network, and that results in the same output as π_0 and π_1. The lower bound then follows from the fact that B_0 and B_1 have only $O(\sqrt{m/k})$ outgoing edges, each of which can transmit at most $m + O(\log n)$ bits in each step. □

[4] Going through the proofs of the above lemmas, we find that for $f = \log N/\log\log N$, we can take $g = \log^{1/3} N$. On the mesh this gives lower-order terms bounded by $\mathcal{O}(k \cdot n/\log^{1/3} n)$.

For the case of integer ranking, the above lower bound can be strenghtened slightly. We obtain the following result, the proof of which is omitted.

Theorem 6 *If* $m = O(k \cdot n^2)$, *then any deterministic algorithm for integer ranking requires* $\Omega(n + \sqrt{m \cdot k} \cdot \frac{\log(n \cdot k/m)}{\log n}$ *steps on the arbitrary model of the mesh.*

We now describe algorithms for integer sorting and ranking whose running times nearly match the lower bounds. Our algorithms are based on counting sort, and rely on an efficient implementation of a restricted form of the 'multiprefix' operation described in [19]. We first present a simple, but non-optimal, algorithm that solves both integer sorting and ranking, and then explain how the algorithm can be modified into optimal solutions for each problem.

In the following, we assume that $k \cdot n^2 > m > k$. (If $m \leq k$, then the problem can be solved in $O(n + k)$ steps by a simple pipelined prefix operation.) The algorithm consists of the following steps.

1. Partition the mesh into blocks of side length $\sqrt{m/k}$, and determine in each block the number a_i of occurrences of key value i, for $0 \leq i < m$. Arrange the numbers inside each block such that a_i is contained in PU $P_{i \bmod k}$ of the block.

2. Combine the values a_i from all blocks in the mesh by repeatedly 'merging' groups of four adjacent blocks. That is, add the corresponding a_i values from the four blocks, and distribute the resulting values over the larger block.

3. Use a simple prefix operation to compute the ranks of the key values.

4. In the case of integer sorting, simple routing and segmented broadcast operations can be used to bring the key values to the correct PUs. For integer ranking, we have to deliver the rank information to the PUs initially holding the keys. To do so, we reverse the 'merging' in Step 2 by performing a downwards pass in the merging tree (as in standard prefix algorithms).

Step 1 of the algorithm can be performed in $O(\sqrt{m \cdot k})$ steps using local sorting. It can be shown that the running time for Steps 2 and 4 is dominated by the time needed for the lowest level of the merging tree, which is also $O(\sqrt{m \cdot k})$. Thus, the algorithm runs in time $O(\sqrt{m \cdot k})$.

The non-optimality of this simple algorithm is due to the inefficient representations of the key values as a collection of values a_i in the upward pass in Step 2, and the rank values in the downward pass in Step 4. To improve the running time of the algorithm, we assume that the keys and ranks in Steps 2 and 4 are given as sorted lists. Note that a sorted list of ν key values from $\{0, \ldots, \mu - 1\}$ can be represented with $O((1 + \log(\mu/\nu)) \cdot \nu)$ bits. Using such a representation, and starting with blocks of side length $\sqrt{m/k}/\log n$ in Step 1, we can reduce the time for Steps 2 and 4 to $O(\sqrt{m \cdot k}/\log n)$ and $O(\sqrt{m \cdot k} \cdot \frac{\log(n \cdot k^2/m)}{\log n}$, respectively. This establishes the following results.

Theorem 7 *If* $m = O(k \cdot n^2)$, *then integer sorting and integer ranking can be performed in time* $O(n + \sqrt{m \cdot k}/\log m)$ *and* $O(\sqrt{m \cdot k} \cdot \frac{\log(n \cdot k^2/m)}{\log n}$, *respectively, on the arbitrary model of the mesh.*

Thus, the results asymptotically match the lower bounds. The algorithms can be adapted to the case where $m = o((k \cdot n^2)^{1+\epsilon})$ for any constant $\epsilon > 0$, for which they still achieve an asymptotic improvement over general sorting.

5 Excess Counting

In this section, we consider the excess counting problem, where each PU holds k colored packets, and we want to mark those packets whose colors occur more than some threshold t times. We start with a lower bound. The proof uses an information-theoretic bisection-bound argument applied to some corner section, similar to those in Section 4.

Lemma 5 *The excess-counting problem with threshold t requires that some connections transfer $\Omega(k \cdot n/\sqrt{t})$ bits.*

Proof: Divide the mesh in a corner $\mathcal{C}$ and the remainder of the network $\mathcal{R}$. $\mathcal{C}$ has size $(n/(2 \cdot t)^{1/2}) \times (n/(2 \cdot t)^{1/2})$, and contains $x = k \cdot N/(2 \cdot t)$ packets. All colors occurring in $\mathcal{C}$ are unique. In $\mathcal{R}$ there are $k \cdot N \cdot (1 - 1/(2 \cdot t)) > 2 \cdot x \cdot (t-1)$ packets.

Suppose that $2 \cdot x$ colors occur exactly $t-1$ times in $\mathcal{R}$, and that the x colors occurring in $\mathcal{C}$ are chosen from these $2 \cdot x$ colors. Then clearly the complete information on the colors occurring in $\mathcal{C}$ has to be sent into $\mathcal{R}$. There are $\binom{2 \cdot x}{x}$ possible choices of these x colors Thus, at least $\log\left(\binom{2 \cdot x}{x}\right) = \Omega(x)$ bits have to be transferred over the $2 \cdot n/(2 \cdot t)^{1/2}$ connections between $\mathcal{C}$ to $\mathcal{R}$, and the result follows. $\square$

In the following, we derive an algorithm whose running time comes comes quite close to the above lower bound. Note that a trivial possibility is to sort the packets on their colors, and then count the packets of each color; this can be done in $(1+o(1)) \cdot k \cdot n/2 + 2 \cdot n$ steps. A factor of two can be gained by applying the ranking algorithm of Section 3 (using the fact that the algorithm produces a *blocked* ranking). If the number of possible colors is very large, and t small, then this is the best algorithm we know. If the number of possible colors is not too large, then we can apply the techniques of Section 4.

In the following, we present an alternative scheme that runs in time independent of the number of possible colors, and that is considerably faster than sorting for sufficiently large values of t. Its structure differs significantly from the algorithm for integer sorting. The algorithm is inspired by the following observation.

Observation 1 *Consider a network that is partitioned into x subnetworks. If a color occurs more than t times in total, then in some subnetwork it occurs at least $\lceil t/x \rceil$ times.*

We assume the arbitrary model. Initially the network is partitioned into t subnetworks with N/t PUs each, called *blocks* (assume that t divides N). The packets in each subnetwork are sorted on their colors and the frequency of each color is determined. All occurring colors are *candidates*. Iteratively, the candidates in pairs of subnetworks are merged together. More precisely, inductively we assume that the following invariant holds at all times:

Invariant 1 *After i merges, and for any subnetwork S with $2^i \cdot N/t$ PUs, a color c is a candidate in S iff it occurs at least 2^i times. There are at most $k \cdot N/t$ candidates in S, and the candidates and their frequencies are known in all blocks in S.*

Intially Invariant 1 holds. Finally, for $i = \log t$, the invariant states that a packet is a candidate iff it occurs more than t times in the whole network, and that the candidates are known in every block. In that case, one more local operation suffices to mark all packets with colors that occur more than t times.

Assume that Invariant 1 holds after merge i. Then the following steps are performed to merge two subnetworks S_1 and S_2 of 2^i blocks each (we only describe the operations in S_2, as the situation in S_1 is symmetric):

Algorithm EXCESS-MERGE

1. The candidates from S_1 and their frequencies are broadcast to all blocks of S_2.

2. In each block of S_2, determine the frequency of the candidates from S_1.

3. The frequencies of the candidates of S_1 in the blocks of S_2 are added up, and made available in every block. These numbers are then added to their frequencies in S_1.

4. All old candidates from S_1 and S_2 that occur more than 2^{i+1} times are selected as candidates.

Theorem 8 *By iterating* EXCESS-MERGE *for* $i = 0, 1, \ldots, \log t - 1$, *excess counting with threshold* t *can be performed in time* $\mathcal{O}(n + \log t \cdot k \cdot n/\sqrt{t})$ *on an* $n \times n$ *mesh.*

Proof: The correctness follows from the invariant, which is obviously restored after every merge. To analyze the running time, we consider the described merging of two subnetworks consisting of 2^i blocks each. The blocks have size $n/\sqrt{t} \times n/\sqrt{t}$ and the subnetworks have size $(2^{\lfloor i/2 \rfloor} \cdot n/\sqrt{t}) \times (2^{\lceil i/2 \rceil} \cdot n/\sqrt{t})$. Step 1 takes $\mathcal{O}((2^{i/2} + k) \cdot n/\sqrt{t})$ steps (distance plus bisection bound). Step 2 can be implemented by sorting in every block S, and Step 3 is similar to a broadcast. Thus, this application of EXCESS-MERGE takes $\mathcal{O}((2^{i/2} + k) \cdot n/\sqrt{t})$ steps. By summing over all $\log t$ iterations we obtai the result. □

The result can be generalized to a bound of $O(d \cdot n + \log t \cdot k \cdot n/t^{1/d})$ for excess determination on a d-dimensional mesh. Note that our scheme outperforms sorting for all $t = \omega(1)$. To the best of our knowledge, the deterministic sequential complexity of this problem is the same as that of sorting. (In a randomized setting, better performance can be achieved by applying hashing and bucket-sort.) So our algorithm gives an interesting example of a problem for which an adaptation of a sequential algorithm does not lead to a good parallel algorithm for the mesh. Also, we see here that problems which are apparently of about the same complexity sequentially, might have a substantially different complexity on the mesh.

Acknowledgement

We thank Greg Plaxton for helpful discussions about the material in Section 2.

References

1. Bailey et al., 'The NAS Parallel Benchmarks,' *Tech. Rep. RNR-94-007*, NASA Ames Research Center, 1994.
2. Bertsekas, D. P., J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods*, Prentice-Hall, 1989.
3. Blelloch, G. E., C. E. Leiserson, B. M. Maggs, C. G. Plaxton, S. J. Smith, M. Zagha, 'A Comparison of Sorting Algorithms for the Connection Machine CM-2,' *Proc. 3rd Symp. on Parallel Algorithms and Architectures*, pp. 3–16, ACM 1991.
4. Dusseau, A. C., 'Modeling Parallel Sorts with LogP on the CM-5,' *Tech. Rep. CSD-94-829*, University of California at Berkeley, 1994.
5. Hinrichs, S., C. Kosak, D. R. O'Hallaron, T. M. Stricker, R. Take, 'An Architecture for Optimal All-to-All Personalized Communication,' *Proc. 6th Symp. on Parallel Algorithms and Architectures*, pp. 310–319, ACM, 1994.
6. Hofri, M., *Probabilistic Analysis of Algorithms*, Springer, 1987.
7. Iwama, K., E. Miyano, Y. Kambayashi, 'Routing Problems on the Mesh of Buses,' *Proc. 3rd Int. Symp. on Algorithms and Computation*, LNCS 650, pp. 155–164, Springer, 1992.
8. Kaufmann, M., S. Rajasekaran, J. F. Sibeyn, 'Matching the Bisection Bound for Routing and Sorting on the Mesh,' *Proc. 4th Symp. on Parallel Algorithms and Architectures*, pp. 31–40, ACM, 1992.
9. Kaufmann, M., J. F. Sibeyn, T. Suel, 'Derandomizing Algorithms for Routing and Sorting on Meshes,' *Proc. 5th Symp. on Discrete Algorithms*, pp. 669–679 ACM-SIAM, 1994.
10. Kunde, M., 'Block Gossiping on Grids and Tori: Deterministic Sorting and Routing Match the Bisection Bound,' *Proc. 1st European Symp. on Algorithms*, LNCS 726, pp. 272–283, Springer, 1993.
11. Kunde, M., R. Niedermeier, K. Reinhardt, P. Rossmanith, 'Optimal Average Case Sorting on Arrays ,' *Proc. 12th Symp. on Theoretical Aspects of Computer Science*, pp. 503–513, Springer, 1995.

12. Leighton, F. T., 'Tight Bounds on the Complexity of Parallel Sorting,' *IEEE Transactions on Computers*, C-34(4), pp. 344–354, 1985.
13. Leighton, F. T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*, Morgan Kaufmann, 1991.
14. T. Lengauer, 'VLSI Theory,' in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, J. van Leeuwen (ed.), pp. 805–833, Elsevier/MIT Press, 1990.
15. Luby, M., 'A Simple Parallel Algorithm for the Maximal Independent Set Problem,' *SIAM Journal on Computing*, 15, pp. 1036–1053, 1986.
16. Meyer, U., J.F. Sibeyn, 'Simulating the Simulator: Deterministic PRAM Simulation on a Mesh Simulator,' *Proc. Eurosim '95*, F. Breitenecker and I. Husinsky (Eds), Elsevier, 1995, to appear.
17. Mehlhorn, K., U. Vishkin, 'Randomized and Deterministic Simulations of PRAMs by Parallel Machines with Restricted Granularity of Parallel Memories,' *Acta Informatica*, 9(1), pp. 29–59, 1984.
18. Pietracaprina, A., G. Pucci, J. F. Sibeyn, 'Constructive Deterministic PRAM Simulation on a Mesh-Connected Computer,' *Proc. 6th Symp. on Parallel Algorithms and Architectures*, pp. 248–256, ACM, 1994.
19. Ranade, A., S. N. Bhatt, S. L. Johnsson, 'The Fluent Abstract Machine', *Advanced Research in VLSI: Proc. 5th MIT Conference*, pp. 71–94, MIT Press, 1988.
20. Rao, S. B., T. Suel, Th. Tsantilas, M. Goudreau, 'Efficient Communication Using Total-Exchange', *Proc. 9th International Parallel Processing Symposium*, pp. 544–550, IEEE, 1995.
21. Reif, J. H., L. G. Valiant, 'A logarithmic time sort for linear size networks,' *Journal of the ACM*, 34, pp. 68–76, 1987.
22. Reischuk, R., 'Probabilistic Parallel Algorithms for Sorting and Selection,' *SIAM Journal of Computing*, 14, pp. 396–411, 1985.
23. Schmidt, J.P., A. Siegel, A. Srinivasan, 'Chernoff-Hoeffding Bounds for Applications with Limited Independence,' *Proc. 4th Symp. on Discrete Algorithms*, pp. 331–340, ACM-SIAM, 1993.
24. Sibeyn, J. F., 'Desnakification of Mesh Sorting Algorithms,' *Proc. 2nd European Symp. on Algorithms*, LNCS 855, pp. 377–390, Springer, 1994.
25. Sibeyn, J.F., 'Sample Sort on Meshes,' *Tech. Rep. MPI-I-95-1012*, Max-Planck Institut für Informatik, Saarbrücken, Germany, 1995.
26. Sibeyn, J. F., 'List Ranking on Interconnection Networks,' *Tech. Rep. MPI-I-95*, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1995, to appear.
27. Valiant, L. G., 'A Scheme for Fast Parallel Communication,' *SIAM Journal on Computing*, 11, pp. 350–361, 1982.
28. Yao, A. C., 'Some Complexity Questions Related to Distributive Computing,' *Proc. 11th Symp. on the Theory of Computing*, pp. 209–213, ACM 1979.