Sample Sort on Meshes

Jop F. Sibeyn[*]

Abstract

Sorting on interconnection networks has been solved 'optimally'. However, the 'lower-order' terms are so large that they dominate the overall time-consumption for many practical problem sizes. Particularly for deterministic algorithms, this is a serious problem.

In this paper a refined deterministic sampling strategy is presented, by which the additional term of the presented deterministic sorting algorithm is hardly larger than the one of the best randomized algorithm.

1 Introduction

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 field. In a *sorting problem*, instead of a destination address each packet contains a key from a totally ordered set, and they have to be rearranged such that the packet of rank i ends up in the memory position with index i with respect to some fixed *indexing* of the memory positions. A routing problem in which each PU is the source and destination of at most k packets is called a k-k routing problem. A k-k sorting problem is defined similarly. For an introduction on the problems of routing and sorting, and a survey of basic results, we refer to Leighton's book [15].

One of the dominant interconnection schemes for parallel computers is the $n \times n$ *mesh*, in which n^2 processing units, *PUs*, are connected by a two-dimensional grid of communication links. Its immediate generalizations are d-dimensional $n \times \cdots \times n$ meshes. While meshes have a large diameter in comparison to the various hypercubic networks, they are nonetheless of great importance due to their simple structure and efficient layout.

Earlier Work. On d-dimensional $n \times \cdots \times n$ meshes a lower bound of $k \cdot n/2$ for k-k routing and k-k sorting is implied by the bisection width of the network. Several algorithms running in $\max\{2 \cdot d \cdot n, k \cdot n/2\} + o(k \cdot n)$ steps have been presented [9, 10, 12, 11, 19]. For $k \geq 4 \cdot d$, they match the lower-bound up to a lower order term, which is usually called *optimality*. This 'optimality' is a doubtful notion if we take into account that the lower order term typically is $10 \cdot k^{5/6} \cdot n^{2/3}$ or larger. It means that only for fairly large input sizes these algorithms outperform more down-earth algorithms with sub-optimal performance (see [19] for more details). In this paper we consider randomized and deterministic sorting algorithms. Our special attention goes to the size of the 'lower-order' term. Implicitly we assume rather small n and fairly large k as these reflect the practically important cases. Reasonable orders of magnitude are $10 \leq n^d \leq 10^4$ and $10^3 \leq k \leq 10^7$.

Often randomized algorithms are simpler and faster than deterministic algorithms. One might even believe that deterministic algorithms are just of theoretical importance. In this paper we show that for sorting on meshes this is not necessarily so.

Sample Sort. As a parallel sorting strategy, sample sort was developed by Reischuk [17]. It was applied to sorting on networks in [16, 8]. Most sorting algorithms on meshes implicitly or

[*]Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany, Email: jopsi@mpi-sb.mpg.de. URL: http://www.mpi-sb.mpg.de/~jopsi/

explicitly apply sample sort. That is, the mesh is divided into non-overlapping *submeshes*, and then somehow proceed as follows:

```
Algorithm BASIC-SORT
1. Select a small subset of the packets as splitters. Broadcast the splitters to all submeshes.
2. In every submesh, for every splitter determine how many packets are smaller.
3. In every submesh, determine the exact global ranks of the splitters by adding together the
   locally computed numbers.
4. Estimate the ranks of the packets by comparison with the splitters.
5. Route the packets to their preliminary destinations.
6. Complete the sorting locally, exploiting the fact that the ranks of the splitters are known.
```

Results. In recent deterministic algorithms [12, 11, 19], the splitters have become obsolete and were omitted: if the packets are suitably redistributed (unshuffled), then the rank of a packet can be estimated by comparing its value with the other packets in its submesh. Here we reintroduce splitters in deterministic sorting algorithms. The use of splitters allows to decouple the routing from the rank estimation. As routing can be performed with a smaller lower-order term than the conventional sorting algorithms, this may be profitable. The number of splitters is reduced to a minimum, by applying a variant of 'successive sampling' (term used in [2]), handling them and comparing the packets with them is cheaper than before.

In the context of selection and ranking, comparable splitter selection methods have been used before. In [4] Cole and Yap give an algorithm for finding the median based on successive sampling. For selection on a hypercube it has been applied by Berthomé ea. [2]; for selection on a PRAM by Chaudhuri, Hagerup and Raman [3]. Our variant resembles most the application in [2]. Application of successive sampling for meshes requires specific adaptation to the features of the network. It appears that we are the first to apply it for sorting.

Contents. We start with preliminaries. Then we consider the presented basic sorting algorithm in more detail, giving basic deterministic and randomized variants and refinements thereof. Our main deterministic sorting algorithm is presented in Section 5 et seq. Many details had to be omitted due to a lack of space. These can be found in [20].

2 Preliminaries

Model of Computation. A d-dimensional *mesh* consists of $N = n^d$ processing units, *PUs* laid out in a d-dimensional grid of side length n. 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 of the computation, a PU can perform an unbounded amount of internal computation, and exchange a bounded amount of data with each of its neighbors. This basic amount is called a *packet* and consists of some data plus the information for routing it to its destination. In some papers this model is called 'MIMD' model.

Indexing Schemes. We only consider two-dimensional meshes, the definitions for higher dimensional meshes are analogous. Let $P_{i,j}$ be the PU located in row i and column j. Here position $(0, 0)$ lies in the upper left corner of the mesh. $[x, y]$ denotes the set $\{x, x+1, \ldots, y\}$.

The index of a PU is determined by an *indexing scheme*, a bijection $I : [0, n-1]^2 \to [0, n^2-1]$. For a given indexing, P_i denotes the PU with index i. The most natural indexing is the *row-major* order under which $P_{i,j}$ has index $i \cdot n + j$. In a k-k sorting problem, the packet of rank i has to be moved to the PU with index $\lfloor i/k \rfloor$. Throughout this paper we assume some *blocked* indexing scheme. That is, the mesh is regularly divided into $m \times m$ submeshes for some m, and the PUs in the submesh with index j, $0 \leq j < n^2/m^2$, have indices in $[j \cdot m^2, (j+1) \cdot m^2 - 1]$. We assume that the submeshes are indexed such that block j is adjacent to block $j+1$ for all $0 \leq j < n^2/m^2 - 1$. Such a blocked indexing is particularly suited for sorting algorithms like BASIC-SORT (but more general indexings can be used as well, as was shown in [19]). Sometimes the packets are sorted in *semi-layered* order. This means that the packet with rank r, $0 \leq r \leq k \cdot n^2$, stands in memory position $\lfloor r/n \rfloor \bmod k$ of $P_{\lfloor r/(k \cdot n) \rfloor, r \bmod n}$: the indexing obtained when perceiving the $n \times n$ mesh with k memory positions per PU as a $k \cdot n \times n$ mesh with row-major indexing.

Randomness. An event A happens *with high probability* if $Pr(A) \geq 1 - n^{-\epsilon}$, for some $\epsilon > 0$. All our results for randomized algorithms hold with high probability.

We use Chernoff bounds to bound the tail probabilities of binomial distributions, $B(n, p)$. Using the estimates in [6], it is easy to derive that

Lemma 1 *Let $X_0, \ldots, X_{t-1}$ be random variables with $t = poly(n)$ and $X_i = B(n, p)$ for $0 \leq i < t$. Then $|X_i - p \cdot n| = O((p \cdot n \cdot \log n)^{1/2})$ for all $0 \leq i < t$, with high probability.*

Definition 1 *A k-randomization is a distribution of packets in which initially every PU holds at most k packets, that have to be routed to randomly chosen destinations.*

Using Lemma 1, it is easy to show that

Lemma 2 **[9]** *On d-dimensional meshes, if $k \geq 4 \cdot d$, then k-randomizations can be routed in $k \cdot n/4 + O((k \cdot n \cdot \log n)^{1/2})$ steps, with high probability.*

Unshuffles. We formally define the 'handing-out operation' under which the packets are regularly redistributed over the whole mesh. This operation is the deterministic counterpart of a randomization. See [11] for details.

Definition 2 *Consider a processor network of N PUs and a division in blocks with M PUs each. Suppose that every PU holds k packets. Consider the packet p in position i, $0 \leq i < k$, in PU j, $0 \leq j < N/M$, in block l, $0 \leq l < M$. Let $r = l \cdot N/M + k \cdot j + i$. Then, under the (N, M, k)-unshuffle, p has to be routed to block $r \bmod M$, and there to PU $\lfloor r/M \rfloor \bmod (N/M)$, and in this PU to position $\lfloor r/N \rfloor$.*

On meshes, it can be easily figured out how to schedule the packets such that never two packets are competing for the use of the same connection. Using such a schedule the routing can be performed without loosing a single step:

Lemma 3 *On a d-dimensional mesh, if $k \geq 4 \cdot d$, and the number of packets in a submesh is a multiple of the number of submeshes, then an unshuffle can be performed in $k \cdot n/4$ steps.*

3 Basic Sample Sort

We consider a randomized and a deterministic sample-sort algorithm of the type of BASIC-SORT. Both algorithms are well-known and only the essential points are recalled. The number of packets to sort is N.

Definition 3 *The* inaccuracy *of an algorithm of the type of* BASIC-SORT, *is the maximum difference between the rank of a packet as estimated in Step 4, and its actual rank.*

3.1 Randomized Sample Sort

Each key is selected as splitter independently and uniformly with probability M/N, for some $0 \leq M \leq N$. Choosing too few splitters means that the inaccuracy becomes too large; choosing too many of them means that handling them and ranking the keys among them becomes too expensive. Denote the resulting inaccuracy by *inac*(N, M). Using elementary probability theory it can easily be estimated that any two consecutive splitters lie at most $O(\log N \cdot N/M)$ apart, with high probability. Thus, using that the exact ranks of the splitters are determined before the packets are compared with them to estimate their ranks,

$$inac(N, M) = O(\log N \cdot N/M), \tag{1}$$

with high probability. We can take the number of splitters as large as the inaccuracy: then the sorting operations in Step 2 and Step 6 can be performed in submeshes of approximately the same size. The result is a well-balanced algorithm with minimal total cost. Solving gives

$$M_{R1} = (N \cdot \log N)^{1/2}.$$

For Step 5 of BASIC-SORT we can first route all packets to random positions [21, 9]. From there they are routed to their preliminary destinations. This second routing is approximately the inverse of a randomization. Working out the details gives

Theorem 1 **[10]** *For randomized k-k sorting, for all $k \geq 8 \cdot d$.*

$$T_{RI}(k, d, n) = k \cdot n/2 + \mathcal{O}(k^{1-\frac{1}{2 \cdot d}} \cdot n^{1/2} \cdot \log^{\frac{1}{2 \cdot d}}(k \cdot n^d)).$$

The algorithm in [10] is slightly different. There the ranks of the packets are estimated before the global ranks of the splitters are determined. In this way the routing and the other operations can be maximally overlapped, and there is no additional term $\mathcal{O}(n)$ already for $k \geq 4 \cdot d$. The price is that much more splitters must be selected to assure a sufficient accuracy.

3.2 Deterministic Sample Sort

Instead of selecting approximately M splitters randomly, we can also sort the packets that stand in submeshes holding M' packets each, and selecting from these packets those with ranks $i \cdot N/M$, for $0 \leq i < M' \cdot M/N$. In order to balance the costs of the various routing and sorting operations it is best to choose $M' = M$. By comparison with the splitters the rank of a packet among the packets from a submesh can be determined up to N/M positions. There are N/M submeshes, so

$$inac(N, M) = (N/M)^2. \tag{2}$$

The inaccuracy should equal the number of splitters, and thus we should take

$$M_{D1} = N^{2/3}.$$

This idea is the basis of [11, 12], and was already present in [13, 18].

In a deterministic algorithm, the routing in Step 5 of BASIC-SORT can be achieved by performing two suitable unshuffles. This yields

Theorem 2 **[19]** *For deterministic k-k sorting, for all $k \geq 4 \cdot d$,*

$$T_{DI}(k, d, n) = k \cdot n/2 + \mathcal{O}(k^{1-\frac{1}{3 \cdot d}} \cdot n^{2/3}).$$

4 Subsplitter Selection

There is a method to reduce the number of splitters considerably. It can be used to reduce the amount of work in PRAM algorithms, and also in network algorithms it is advantageous. Probably this method has been applied for the first time by Reif and Valiant in [16].

4.1 Selection Method

Suppose that we have selected M splitters randomly, as we did in Section 3.1. The resulting inaccuracy is expressed in (1). Now consider the following procedure to select the splitters:

```
Algorithm SUBSPLITTERS
1. Select each packet as splitter with probability $M/N$.
2. Sort all splitters.
3. Select the elements with ranks $i \cdot \log N$, for all $1 \leq i \leq M/\log N$ as subsplitters.
```

Let $M' = M/\log N$ be the number of subsplitters. They are almost optimal:

Lemma 4 *With the computed set of subsplitters,*

$$inac(N, M') = \mathcal{O}(N/M').$$

Proof: With help of the Chernoff bounds we bound the maximum number of packets that lie between any pair of consecutive subsplitters. Consider an arbitrary subset of $\alpha \cdot \log N \cdot N/M$, which are consecutive in the sorted order. The expected number of selected splitters from among them is $\alpha \cdot \log N$. For sufficiently large α at least $\log N$ of them are actually selected. Hence there are no intervals of length $2 \cdot \alpha \cdot \log N \cdot N/M = \mathcal{O}(N/M')$ without subsplitters. □

4.2 Application for Meshes

Subsplitter selection has been used for sorting on meshes by Hightower, Prins and Reif [7]. The idea was also present in [9], but there it was not applied in a very profitable way. It is important to perform the sorting in Step 2 of SUBSPLITTERS on the whole mesh such that all PUs hold on the average only M/n^d splitters during the sorting.

We consider some details of an implementation on two-dimensional $n \times n$ meshes:

```
Algorithm RANDSORT
1. SUBSPLITTERS is performed on the whole mesh. The number of selected subsplitters is
   $M' = M/\log N$.
2. The subsplitters are broadcast to all $n' \times n'$ submeshes, with $n' = (M'/k)^{1/2}$.
3. Perform Step 2 through 6 of BASIC-SORT. The operations in Step 6 are performed in
   $n'' \times n''$ submeshes, with $n'' = \mathcal{O}((N/(k \cdot M'))^{1/2})$.
```

Theorem 3 *For $M = N^{1/2} \cdot \log N$ and $k \geq 26$,* RANDSORT *performs k-k sorting on two-dimensional meshes in*

$$T_{R2}(k, n) = k \cdot n/2 + \mathcal{O}((k \cdot n)^{1/2} \cdot (k^{1/4} - \log^{1/2}(k \cdot n))).$$

5 Better Splitter Selection

The improved randomized algorithm RANDSORT of Section 4.2 leaves little to desire: it is fairly simple, and the additional term is almost as small as we could hope for. On the other hand, the deterministic algorithm of Section 3.2 gives a much larger additional term. In this section we present a refined deterministic splitter selection method. In Section 6 we show how it can be applied in a deterministic algorithm with performance comparable to RANDSORT. Similar methods have been used before in [4, 2, 3].

First we give a high-level description of the algorithm without considering details of the network. The packets are divided in N/M subsets of size M each. Suppose that $N/M = y^x$, for some y and x. Thus,

$$x = \log(N/M)/\log y. \tag{3}$$

The subsets are sorted. Then, the following procedure is repeated x times:

```
Algorithm REDUCE
1. Merge $y$ subsets together.
2. Only retain elements $p$ with ranks $r_p = j \cdot y$, for all $1 \leq j \leq M$.
```

The M packets that finally come out of this process are called the *splitters*. REDUCE is an almost trivial operation which can be performed efficiently on networks.

We prove that the ranks of the packets gradually become less accurate, but not too much:

Lemma 5 *Let $1 \leq t \leq x$, denote the number of so far performed iterations of* REDUCE. *Consider a packet p, which has rank r_p in its actual subset $T_{p,t}$ of M elements. Let $R_{p,t}$ denote the rank of p in the subset of $y^t \cdot M$ elements that has been reduced to $T_{p,t}$. Then*

$$r_p \cdot y^t \leq R_{p,t} \leq (r_p + (t-1) \cdot (1 - 1/y)) \cdot y^t. \tag{4}$$

Proof: In the first iteration, y subsets of M packets each are sorted together and the packets with ranks $i \cdot y$, for $1 \leq i \leq M$ are selected. Clearly the rank $R_{p,1}$ of the packet p with rank r_p among the M selected packets equals $y \cdot r_p$, and hence (4) is satisfied for $t = 1$.

We proof the lower bound on $R_{p,t}$ by applying induction on t. So, assume that $R_{p,t-1} \geq r_p \cdot y^{t-1}$, for all p for some $t > 1$. Consider some packet p with rank r_p in $\mathcal{T}_{p,t}$. Denote the y subsets that were merged to obtain $\mathcal{T}_{p,t}$ by $\mathcal{A}_i$, $1 \leq i \leq y$. Without loss of generality we may assume that $p \in \mathcal{A}_1$. Define $\alpha_i = \#\{\text{packets } q \in \mathcal{A}_i | q \leq p\}$ (here we identified a packet and its key). Remind that all keys are different. By the selection in Step 2 of REDUCE, we know that $\sum_{i=1}^{y} \alpha_i = r_p \cdot y$, and thus, applying the induction assumption, $R_{p,t} \geq \sum_{i=1}^{y} \alpha_i \cdot y^{t-1} = r_p \cdot y^t$.

Let δ_t be the number such that $R_{p,t} \leq r_p \cdot y^t + \delta_t$. That $\delta_1 = 0$ was shown above. For $t > 1$ we have $R_{p,t} \leq \alpha_1 \cdot y^{t-1} + \delta_{t-1} + \sum_{i=2}^{y}((\alpha_i + 1) \cdot y^{t-1} - 1 + \delta_{t-1}) = r_p \cdot y^t + y \cdot \delta_{t-1} + (y-1) \cdot (y^{t-1} - 1)$. Thus, δ_t is given by the following recurrence: $\delta_1 = 0$ and $\delta_t = y \cdot \delta_{t-1} + (y-1) \cdot (y^{t-1} - 1)$. By induction we proof that

$$\delta_t \leq (t-1) \cdot (y-1) \cdot y^{t-1}. \tag{5}$$

for all $t > 1$. Suppose (5) holds for some $t - 1$, then $\delta_t \leq (t-2) \cdot (y-1) \cdot y^{t-1} + (y-1) \cdot (y^{t-1} - 1) = (t-1) \cdot (y-1) \cdot y^{t-1} - y + 1$. □

From this we obtain an estimate on the quality of the selected splitters:

Theorem 4

$$inac(N, M) < \frac{y-1}{y \cdot \log y} \cdot \log(N/M) \cdot N/M.$$

Proof: Denote by $\mathcal{M}$ the set of selected splitters. And for any packet p by R_p its rank among the N packets. REDUCE is iterated x times, for x as in (3). Hence, omitting the factor $(1 - 1/y)$, for any $p \in \mathcal{M}$, with rank r_p in $\mathcal{M}$, we know that $R_p = R_{p,x}$ satisfies $r_p \cdot N/M \leq R_p \leq (r_p + x - 1) \cdot N/M$. For any packet $q \notin \mathcal{M}$, we can find $p, p' \in \mathcal{M}$, such that $r_{p'} = r_p + 1$ and $p < q < p'$. So, $R_p < R_q < R_{p'}$, and hence, $r_p \cdot N/M < R_q < (r_p + x) \cdot N/M$. □

The optimum is reached for $M = \Theta((N \cdot \log N)^{1/2})$. For this M, the inaccuracy is of the same order as the number of elements on which the merge operations are performed. In the spirit of Section 4, it may be profitable to reduce the number of selected splitters further without substantially impairing the accuracy of the estimated ranks.

In view of the inaccuracy given in Theorem 4, the number of splitters can be reduced by a factor $\mathcal{O}(\log(N/M))$. We will see that a reduction by a factor

$$z = (1 - \frac{y-1}{y \cdot \log y}) \cdot \log(N/M)$$

is a reasonable choice. We summarize the splitter-selection procedure:

```
    Algorithm SELECT1
1.  Sort subsets of $M$ packets;
2.  for $i := 1$ to $x$ do
        apply REDUCE;
3.  only retain elements with ranks
    $j \cdot z$ for all $1 \leq j \leq M/z$.
```

With the chosen z, the inaccuracy is hardly larger than before:

Theorem 5

$$inac(N, M/z) < \log(N/M) \cdot N/M.$$

Proof: The proof is analogous to the proof of Lemma 4. Important is that $(y-1)/(y \cdot \log y) \cdot \log(N/M) + z = \log(N/M)$. □

6 Sorting on 2D Meshes

We consider in detail a k-k sorting algorithm of the type of BASIC-SORT for two-dimensional $n \times n$ meshes. $N = k \cdot n^2$. We apply REDUCE for the splitter selection. We take $y = 4$, and use a merge operations from [19].

6.1 Rank Estimation

We present an algorithm to efficiently obtain an estimate of the ranks of the packets. For convenience we assume that all occurring numbers divide each other nicely, particularly we assume that M, the number of selected splitters, is a power of four.

Algorithm ESTIMATE

```
1. $m := \max\{1, \sqrt{M/k}\}$. Sort the packets in all $m \times m$ submeshes. Copy all packets to splitters. If $m = 1$, then only retain the splitters with ranks $i \cdot k/M$, for $1 \leq i \leq M$.
2. Repeat the following operation for $i := 0$ to $\log(n/m) - 1$ as long as $M/(4^i \cdot m^2) \geq 4$: merge four $2^i \cdot m \times 2^i \cdot m$ submeshes, and retain only the splitters with ranks $4 \cdot j$, for $1 \leq j \leq M$. After iteration $i$ every PU should hold exactly $M/(4^{i+1} \cdot m^2)$ splitters.
3. Sort the remaining splitters in the whole mesh and retain the splitters with ranks $j \cdot N/M$, for $1 \leq j \leq M$.
   $z := \max\{1, 5/8 \cdot \log(N/M)\}$, $M' := M/z$. $m' = \max\{1, \sqrt{M'/k}\}$. Only retain the splitters with ranks $j \cdot z$, for $1 \leq j \leq M'$.
4. Broadcast the splitters to all $m' \times m'$ submeshes $B_i$, $1 \leq i \leq M'/N$.
5. In every $B_i$, $1 \leq i \leq M'/N$, for every splitter $p_j$, $1 \leq j \leq M'$, determine the number $\alpha_{i,j} = \#\{\text{packets } q \text{ in } B_i | p_{j-1} < q \leq p_j\}$. Place this number in PU $\lfloor j/k \rfloor$ of $B_i$. Discard the splitters.
6. Add the numbers $\alpha_{i,j}$ together such that afterwards the numbers $a_{i,j} = \sum_{l<i} \alpha_{l,j}$ and $A_j = \sum_{i=1}^{M'/N} \alpha_{i,j}$ stand in PU $\lfloor j/k \rfloor$ of $B_i$ for all $i$ and $j$.
7. In every $B_i$, $1 \leq i \leq M'/N$, for every packet $q$, with $p_{j-1} < q \leq p_j$ and with rank $r$ among the packets counting for $\alpha_{i,j}$, determine its preliminary rank as $\sum_{l<j} A_l + a_{i,j} + r$.
```

In Step 1, the size of the initial submeshes is determined. Afterwards, the number of splitters in every submesh is exactly M. Step 2 corresponds to REDUCE. In every iteration the number of splitters in every PU is halved and eventually may become one. In that case the selection of the splitters is completed in Step 3. The number of splitters is reduced by a factor z in order to reduce the further costs to handle them. In Step 4, 5 and 6, the global ranks of the splitters are computed and made available in all $m' \times m'$ submeshes. Hereafter, in Step 7, the ranks of the packets can be estimated by local comparisons. All together, ESTIMATE corresponds to Step 1 through 4 of BASIC-SORT. Here, contrary to BASIC-SORT, the splitters are discarded early.

Lemma 6 [19] *For all n, $k \geq 4$, k-k sorting on an $n \times n$ mesh can be performed in $2 \cdot k \cdot n$ steps.* 1-1 *sorting can be performed in* $4^1\!/\!_2 \cdot n$ *steps.*

During Step 2 the packets are kept in semi-layered order. This facilitates the merging. At a certain stage of the merging and pruning, let $n'/2$ be the size of the submeshes, and k' the number of packets hold by every PU. Then we perform

Algorithm KKMERGE

```
1. $P_{i,j}$, $0 \leq i, j < n'$, sends its packet with rank $r$, $0 \leq r < k'$, to $P_{i,(j+n'/2) \bmod n'}$ if $\mathrm{odd}(k' \cdot i + r + j)$.
2. In all columns, sort the packets.
3. In every $P_{i,j}$, $0 < i \leq n' - 1$, $0 \leq j \leq n' - 1$, copy the smallest packet to $P_{i-1,j}$. In every $P_{i,j}$, $0 \leq i < n' - 1$, $0 \leq j \leq n' - 1$, copy the largest packet to $P_{i+1,j}$.
```

4. Sort the rows from left to right or vice versa depending on the position of this $n' \times n'$ submesh in the next merge.

5. In every row, throw away the n' packets with the smallest and the n' packets with the largest indices. From the remaining packets only retain those whose ranks is a multiple of four. Route the packets such that they come to stand in semi-layered order.

For the correctness of KKMERGE it is important that, by the semi-layered indexing, our merging corresponds to a 1-1 merge on a $k' \cdot n' \times n'$ mesh. It is not hard to estimate that KKMERGE essentially takes $9/8 \cdot k' \cdot n'$ steps. A reduction can be achieved by combining the routing of Step 5 of iteration i and Step 1 of iteration $i+1$. Further details are given in [19].

Lemma 7 ESTIMATE *runs in* $6^1\!/\!_2 \cdot \sqrt{k \cdot M} + 6^3\!/\!_4 \cdot \sqrt{k \cdot M'} + O(n)$ *steps. Afterwards the preliminary ranks satisfy the following properties:*

1. *Every packet q has a unique preliminary rank r_q.*

2. *For any splitter p and packets q_1, q_2, $q_1 < p \leq q_2$ implies $r_{q_1} < r_{q_2}$.*

Proof: Step 1 is a k-k sorting in $m \times m$ meshes and takes $2 \cdot \sqrt{k \cdot M}$ steps. Step 2 of ESTIMATE can be performed in $4^1\!/\!_2 \cdot \sqrt{k \cdot M}$ steps. Step 3 is at worst a 1-1 sorting in the whole mesh and takes at most $4^1\!/\!_2 \cdot n$ steps. The broadcasting in Step 4 can be performed in at most $3/4 \cdot \sqrt{k \cdot M'} + 2 \cdot n$ steps. In Step 5 one should exploit that the packets and the splitters were already sorted before. Then it can be performed in $2^1\!/\!_2 \cdot \sqrt{k \cdot M'}$ steps. Step 6 is a multiple parallel prefix operation, which requires at most $\sqrt{k \cdot M'} + 2 \cdot n$ steps. Step 7 is similar to Step 5, and requires at most $2^1\!/\!_2 \cdot \sqrt{k \cdot M'}$ steps. □

The uniqueness of the preliminary ranks assures that the subsequent routing of the packets to their preliminary destinations is a perfect k-k routing. The second property means that after this routing all packets which lie between any two splitters stand in PUs with consecutive indices. From Theorem 5, we know that the number of packets between two splitters is at most

$$inac(N, M) = \log(N/M) \cdot N/M. \tag{6}$$

One might think that the properties are more than needed and mean a waste of routing steps. However, the global ranks of the splitters have to be determined anyway, and at some time the packets must find out their precise positions by one more comparison with them.

6.2 Completing the Sorting

ESTIMATE correspond to Step 1 and 2 of BASIC-SORT. It remains to route the packets to their preliminary ranks and to sort the subsets of packets that fall between two splitters.

For the sorting we use a blocked-indexing scheme. The blocks have size $b \times b$, with $b = (\log(N/M) \cdot N/(2 \cdot k \cdot M))^{1/2}$. By (6) this means that the packets stand either in their destination block, or in the preceding or succeeding block. Thus, the sorting can be completed as follows:

Algorithm COMPLETE

1. Sort the packets in all $b \times b$ blocks.
2. Merge the packets in all pairs of blocks (B_i, B_{i-1}), with i even.
3. Merge the packets in all pairs of blocks (B_i, B_{i-1}), with i odd.

Lemma 8 COMPLETE *completes the sorting in* $5/\sqrt{2} \cdot (k \cdot \log(N/M) \cdot N/M)^{1/2}$ *steps.*

Proof: Step 1 takes $2 \cdot k \cdot b$ steps. Step 2 and 3 can be performed in $3/2 \cdot k \cdot b$ steps each. □

There remains one point to settle: how do we route the packets to their preliminary destinations? A problem is that deterministically there is no known *routing* algorithm which is substantially faster than a *sorting* algorithm. Randomizedly we can apply the algorithm from [9]. As we intend to develop in this paper a new approach for sorting in practice, we might even assume that the input is more or less random. In that case we could apply an easy greedy-routing strategy (see [14] for an analysis of a special case). We leave this issue open. The optimal choice should be made depending on the application and the values of k and n. Denote the time consumption for k-k routing by $T_{\text{route}}(k, n)$. Adding all together yields

Lemma 9 *For k-k sorting on $n \times n$ meshes with $M \simeq N^{1/2}$,*

$$T_{D2}(k,n) < T_{route}(k,n) + O(n) + (6.5 + 17.1/\log^{1/2} N) \cdot \sqrt{k \cdot M} + 2.5 \cdot (k \cdot \log N \cdot N/M)^{1/2}.$$

Theorem 6 *For $N \leq 2^{24}$, the sorting algorithm based on* ESTIMATE *and* COMPLETE, *with $M = (\log N \cdot N)^{1/2}/4$ yields*

$$T_{D2}(k,n) < T_{route}(k,n) + O(n) + 22 \cdot k^{3/4} \cdot n^{1/2}.$$

6.3 Evaluation

The result of Theorem 6 shows an additional term of the same order of magnitude as we find in the result of Theorem 3. However, the constant 22 looks rather disappointing. Let us compare the obtained algorithm with the other three algorithms in this paper. We have not analyzed their constants, but making the same assumptions (k-k sorting takes $2 \cdot k \cdot n$ steps and $\log N = 24$), they can be estimated fairly reliably as follows:

$$\begin{aligned} T_{R1}(k,n) &\simeq T_{\text{route}} + 25 \cdot k^{3/4} \cdot n^{1/2}, & T_{R2}(k,n) &\simeq T_{\text{route}} + 12 \cdot k^{3/4} \cdot n^{1/2}, \\ T_{D1}(k,n) &\simeq T_{\text{route}} + 10 \cdot k^{5/6} \cdot n^{2/3}, & T_{D2}(k,n) &\simeq T_{\text{route}} + 22 \cdot k^{3/4} \cdot n^{1/2}. \end{aligned}$$

For an average sized problem T_{D2} is considerably smaller than T_{D1}. For example, consider the case $n = 16$, $k = 10{,}000$, then $T_{D1} = 1.11 \cdot k \cdot n$, and $T_{D2} = 0.80 \cdot k \cdot n$. Even more important is that in the refined algorithm, all local operations may be performed internally in a single PU for $k = O(n^2)$, whereas for the original algorithm, this desirable state is reached only for $k = O(n^4)$. So, particularly in the range $n^2 \leq k \leq n^4$, the new algorithm will be much faster. Practically, this range is higly relevant. The improved randomized algorithm outperforms all others, but, for sufficiently large k and n, the difference with the improved deterministic algorithm is small, and in that case the stability of a deterministic algorithm may be profitable.

7 Conclusion

We proposed a sorting algorithm based on an improved deterministic splitter-selection method. It clearly outperforms earlier deterministic algorithms. Only for sorting small numbers of packets bitonic- or merge-sort algorithms perform better, for example the algorithm presented in [19]. A similar two-fold situation is reported to occur in practice: Diekmann e.a. [5] considered implementations of sorting algorithms on a Parsytec GCel with up to 1024 PUs. They found that bitonic sort is the best if there are less than 1000 packets per PU, while sample sort is better for larger numbers of packets. Similar observations were made for the CM-2 (a hypercubic network) in [1].

In [20] we consider some possible applications of the refined deterministic splitter selection for sorting on RAMs, PRAMs and hypercubes. Especially for sorting on RAMs it is a good strategy. Most promising though, appear to be applications for real parallel computers: by its simple structure and its regular routing operations our algorithm can easily be implemented. As for real problems we may assume that k exceeds the number of PUs, all local operations can be performed in single PUs. Currently the algorithm is being implemented on an Intel Paragon.

Acknowledgement: Torsten Suel gave valuable pointers to previous work in this field.

References

[1] 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 SPAA*, pp. 3–16, ACM, 1991.

[2] Berthomé, P., A. Ferreira, B.M. Maggs, S. Perennes, C.G. Plaxton, 'Sorting-Based Selection Algorithms for Hypercubic Networks,' *Proc. 7th IPPS*, pp. 89–95, IEEE, 1993.

[3] Chaudhuri, S., T. Hagerup, R. Raman, 'Approximate and Exact Deterministic Parallel Selection,' *Proc. 18th MFCS*, LNCS 711, pp. 352–361, Springer-Verlag, 1993.

[4] Cole, C., C.K. Yap, 'A Parallel Median Algorithm,' *IPL*, 20, pp. 137–139, 1985.

[5] Diekmann, R., J. Gehring, R. Lüling, B. Monien M. Nübel, R. Wanka, 'Sorting Large Data Sets on a Massively Parallel System,' *Proc. 6th SPDP*, pp. 2–9, IEEE, 1994.

[6] Hagerup, T., C. Rüb, 'A Guided Tour of Chernoff Bounds,' Inf. Proc. Lett. 33, 305–308, 1990.

[7] Hightower, W.L., J.F. Prins, J.H. Reif, 'Implementations of Randomized Sorting on Large Parallel Machines,' *Proc. 4th Symp. on Parallel Algorithms and Architectures*, pp. 158–167, ACM, 1992.

[8] Kaklamanis, C., D. Krizanc, L. Narayanan, Th. Tsantilas, 'Randomized Sorting and Selection on Mesh Connected Processor Arrays,' *Proc. 3rd SPAA*, pp. 17–28, ACM, 1991.

[9] 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.

[10] Kaufmann, M., J.F. Sibeyn, 'Randomized Multipacket Routing and Sorting on Meshes,' *Algorithmica*, 17, pp. 224-244, 1997.

[11] 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.

[12] Kunde, M., 'Block Gossiping on Grids and Tori: Deterministic Sorting and Routing Match the Bisection Bound,' *Proc. 1st ESA*, LNCS 726, pp. 272–283, Springer-Verlag, 1993.

[13] Leighton, F.T., 'Tight Bounds on the Complexity of Parallel Sorting,' *IEEE Transactions on Computers, C-34(4)*, pp. 344–354, 1985.

[14] Leighton, T., 'Average Case Analysis of Greedy Routing Algorithms on Arrays,' *Proc. 2nd Symp. on Parallel Algorithms and Architectures*, pp. 2–10, ACM, 1990.

[15] Leighton, F.T., *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*, Morgan Kaufmann, 1991.

[16] Reif, J.H., L.G. Valiant, 'A logarithmic time sort for linear size networks,' *Journal of the ACM*, 34(1), pp. 68–76, 1987.

[17] Reischuk, R., 'Probabilistic Parallel Algorithms for Sorting and Selection,' *SIAM Journal of Computing*, 14, pp. 396–411, 1985.

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

[19] Sibeyn, J.F., 'Desnakification of Mesh Sorting Algorithms,' accepted 9-1996 for *SIAM Journal on Computing*. Preliminary version in *Proc. 2nd European Symp. on Algorithms*, LNCS 855, pp. 377–390, Springer-Verlag, 1994. Full version in *Techn. Rep. MPI-I-94-102*, Max-Planck-Institut für Informatik, Saarbrücken, Germany, 1994.

[20] Sibeyn, J.F., 'Sample Sort on Meshes,' *Techn. Rep. MPI-I-95-1012*, Max-Planck Institut für Informatik, Saarbrücken, Germany, 1995.

[21] Valiant, L. G., 'A Scheme for Fast Parallel Communication,' *SIAM Journal on Computing*, 11, pp. 350–361, 1982.