Sample Sort on Meshes

Jop F. Sibeyn[*]

Abstract

In this paper various algorithms for sorting on processor networks are considered. We focus on meshes, but the results can be generalized easily to other decomposable architectures. We consider the k-k sorting problem in which every PU initially holds k packets. We present well-known randomized and deterministic splitter-based sorting algorithms. We come with a new deterministic sorting algorithm which performs much better than previous ones. The number of routing steps is reduced by a refined deterministic splitter selection. Hereby deterministic sorting might become competitive with randomized sorting in practice.

1 Introduction

1.1 Problem and Machine

Meshes. One of the most thoroughly investigated interconnection schemes for parallel computation 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. Numerous parallel machines with mesh topologies have been built, and various algorithmic problems have been analyzed on theoretical models of the mesh.

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 field. On a mesh, the PUs can be identified by simply giving their coordinates. 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 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. In the k-k sorting problem each PU holds at most k packets. Thus, in a routing problem the destinations of the packets are given as part of the input, while in a sorting problem, the destinations have to be computed from the key values. For an introduction on the problems of routing and sorting, and a survey of basic results, we refer to Leighton's book [22].

1.2 Earlier Work and Results

Routing and Sorting. 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 $\max\{2 \cdot d \cdot n, k \cdot n\} + o(k \cdot n)$ steps have been presented [15, 16, 17, 19, 26]. 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 [26] 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 $2^4 \leq n^d \leq 2^{16}$ and $10^4 \leq k \cdot n^d \leq 10^7$.

Often randomized algorithms are simpler and faster than deterministic algorithms. For example, on a hypercube with N PUs, randomized 1-1 sorting can be performed in $\mathcal{O}(\log N)$ [23], whereas the best deterministic algorithm takes $\mathcal{O}(\log N \cdot$

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

$\log \log N$) steps [9]. This has lead to a widespread believe that the development of deterministic algorithms is just a theoretical game. 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 [24]. It was applied to sorting on networks in [23, 14]. Most sorting algorithms on meshes implicitly or explicitly apply sample sort. That is, the mesh is divided into non-overlapping **submeshes**, and then somehow they proceed according to the following scheme:

```
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.
```

If the routing in Step 5 consists of two phases, a (pseudo-) randomization and a greedy routing, then the first phase can be overlapped with Step 1, 2 and 3.

Splitters in Deterministic Algorithms. In recent deterministic algorithms [19, 17, 26], 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. In this paper we reintroduce splitters in deterministic sorting algorithms. By reducing their number to a minimum, applying a variant of 'successive sampling' (term used in [3], see Section 5 for a description of the method), handling and comparing the packets with them is considerably cheaper than before.

In the context of selection and ranking, comparable splitter selection methods have been used before. The idea originates with Cole and Yap. In [8] they give a parallel comparison model algorithm for finding the median based on successive sampling. For selection on a hypercube it has been applied by Berthomé ea. [3]; for selection on a PRAM by Chaudhuri, Hagerup and Raman [4]. Our variant turns out to resemble most the application in [3]. 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.

1.3 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.

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.

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 $I : [0, n - 1]^2 \rightarrow [0, n^2 - 1]$. 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

0	1	2	9	10	11	18	19	20
3	4	5	12	13	14	21	22	23
6	7	8	15	16	17	24	25	26
45	46	47	36	37	38	27	28	29
48	49	50	39	40	41	30	31	32
51	52	53	42	43	44	33	34	35
54	55	56	63	64	65	72	73	74
57	58	59	66	67	68	75	76	77
60	61	62	69	70	71	78	79	80

Figure 1: A blocked 'snake-like' indexing for a 9×9 mesh. The blocks have size 3×3. Within the blocks the PUs are indexed in row-major order.

assume that the submeshes are indexed such that block j is adjacent to block $j+1$ for all $0 \leq j < n^2/m^2 - 1$. An example is given in Figure 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 [26]).

In some subroutines we will sort the packets 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 $P_{\lfloor r/(k \cdot n) \rfloor, r \bmod n}$. This is

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15
16	17	18	19
20	21	22	23
24	25	26	27
28	29	30	31
32	33	34	35
36	37	38	39
40	41	42	43
44	45	46	47

Figure 2: The semi-layered order for 3-3 sorting on a 4×4 mesh.

the indexing that is obtained when the $n \times n$ mesh with k memory positions per PU is perceived as a $k \cdot n \times n$ mesh with row-major indexing. See Figure 2.

Chernoff Bounds We use a variant of Chernoff bounds [5] to bound the tail probabilities of binomial distributions. Let X be the random variable denoting the number of heads in n independent flips of a biased coin, where the probability of a head in each coin flip is p. Thus, X has binomial distribution $B(n,p)$. Then for all $0 < h < n \cdot p$, we have [11]

$$\begin{aligned} Pr\{X \geq n \cdot p + h\} &\leq e^{-h^2/(3 \cdot n \cdot p)}, \\ Pr\{X \leq n \cdot p - h\} &\leq e^{-h^2/(2 \cdot n \cdot p)}. \end{aligned}$$

Finally, we say that an event A happens **with high probability** if $Pr(A) \geq 1 - n^{-\epsilon}$, for some $\epsilon > 0$. All results claimed for randomized algorithms hold with high probability. The following lemma follows directly from the above bounds:

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 $|X_i - p \cdot n| = \mathcal{O}((p \cdot n \cdot \log n)^{1/2})$ for all $0 \leq i < t$, with high probability.*

Randomizations. We consider the operation of sending all packets to random destinations:

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

Proof: Consider a two-dimensional mesh. An essential idea is to 'color' half of the packets white and the other half black. The white packets are first routed along the rows to their destination columns, and then along the columns to their destinations. The black packets go 'column-first'. If several packets compete for the use of the same connection, then the packet which has to go farthest gets priority (farthest-first strategy). For more details we refer to [15]. □

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 [17] 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 $\lfloor r/N \rfloor$.

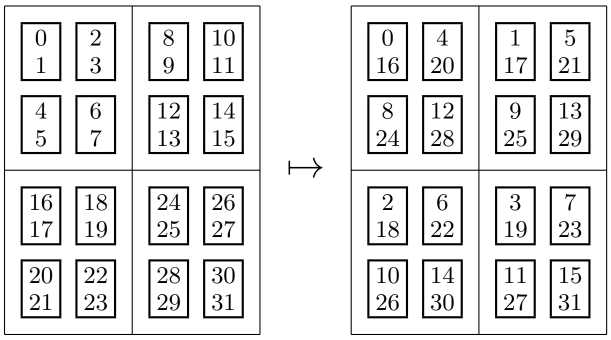

Figure 3: A $(16, 4, 2)$-unshuffle on a two-dimensional mesh. The 32 packets are indicated by the value r from Definition 2. Here we assumed that the submeshes and the PUs within the submeshes are indexed in row-major order.

An unshuffle is a completely regular routing operation, under which from every block the same number of packets has to go to every block. An example is given in Figure 3. 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. This means that no conflict-resolution strategy is required. If such a routing is supported by hardware, it might be much faster than a randomization. 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 in Section 1. 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 $\mathcal{O}(\log N \cdot N/M)$ 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,

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

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 [27, 15]. 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 [16] *randomized k-k sorting can be performed in*

$$T_{R1}(k, d, n) = k \cdot n/2 + \mathcal{O}(k^{1-\frac{1}{2 \cdot d}} \cdot n^{1/2} \cdot \log^{\frac{1}{2 \cdot d}}(k \cdot n^d)),$$

steps, for all $k \geq 8 \cdot d$.

Proof: We must perform sorting and rearrangement operations in submeshes with side lengths $n' = ((N \cdot \log N)^{1/2}/k)^{1/d}$. These take $k \cdot n'$ steps. Substituting $N = k \cdot n^d$ gives the theorem. □

The algorithm in [16] 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

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

Again we should have that the inaccuracy equals the number of splitters, and thus we should take

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

This idea is the basis of [17, 19], and was already present in [20, 25].

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

Theorem 2 [26] *Deterministic k-k sorting can be performed in*

$$T_{D1}(k, d, n) = k \cdot n/2 + \mathcal{O}(k^{1-\frac{1}{3\cdot d}} \cdot n^{2/3}),$$

steps, for all $k \geq 4 \cdot d$.

$T_{\mathrm{D1}} > T_{\mathrm{R1}}$, but the routing operations are simpler: they are perfectly regular off-line operations and the packets can be scheduled in such a way that no contentions for connections arise.

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 [23].

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. The following lemma states that the subsplitters are almost optimal:

Lemma 4 *With the computed set of subsplitters,*

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

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 larger than $2 \cdot \alpha \cdot \log N \cdot N/M = \mathcal{O}(N/M')$ without selected subsplitters. □

4.2 Application for Meshes

In a practical algorithm subsplitter selection has been used for sorting on meshes by Hightower, Prins and Reif [12]. The idea was also present in [15], 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. We give a high-level description:

```
Algorithm RANDSORT
1. SUBSPLITTERS is performed on the whole mesh. The number of selected subsplitters is $M' = M/\log N$.
2. The splitters 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 $M = N^{1/2} \cdot \log N$,* RANDSORT *performs k-k sorting on two-dimensional meshes in*

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

for all $k \geq 26$.

Proof: Next to the routing in Step 5 of BASIC-SORT, there are three operations that may determine the costs: sorting the M splitters, sorting in $n' \times n'$ submeshes, and sorting in $n'' \times n''$ submeshes.

Sorting M numbers can be performed in $\mathcal{O}(M/n)$ steps. Sorting in $n' \times n'$ meshes takes $\mathcal{O}((k \cdot M/\log N))$ steps, and sorting in $n'' \times n''$ meshes takes $\mathcal{O}((k \cdot N \cdot \log N/M))$ steps. Substitute $M = N^{1/2} \cdot \log N$, and $N = k \cdot n^2$.

The constant 26 is determined as follows: in order not to get an additional term $\mathcal{O}(n)$ for small k, the first phase of the routing in Step 5 of BASIC-SORT (randomizing the packets), must be overlapped with Step 1, 2 and 3. The randomization takes $k \cdot n/4 + \mathcal{O}((k \cdot n \cdot \log(k \cdot n))^{1/2})$. For small k, Step 1, 2 and 3 take $4^1\!/_2 \cdot n + 2 \cdot n + 2 \cdot n$ steps together. $\square$

Many details remain to be spelled out before RANDSORT can actually be used, but there are no fundamental problems.

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.[1] On the other hand, the deterministic algorihm of Section 3.2 gives a much larger additional term, reducing its practicality to very large k and n. In this section we present a refined deterministic splitter selection method. In Section 7 we show how it can be applied to obtain a deterministic algorithm with performance comparable to RANDSORT.

We reconsider the selection of the splitters. Suppose that we are given a large set $\mathcal{S}$ of packets with keys. $\#\mathcal{S} = N$. Without loss of generality we assume that all keys are different.[2] We want to select a very good splitter set of some size M. The optimum would be that the splitters regularly subdivide $\mathcal{S}$. That is, by comparison with the splitters we could estimate the rank of a packet accurate up to N/M. Computing such a splitter set appears to require almost as much time as sorting. Nevertheless, we can come close to this in much less time. Similar methods were used before in [8, 3, 4] (see Section 1.2 for a short discussion).

5.1 Selection Method

First we give a high-level description of the algorithm without considering details of the network. $\mathcal{S}$ is 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 (see Section 7 for an example on meshes with $y = 4$).

5.2 Analysis

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*

[1]Sorting with an algorithm of the type of BASIC-SORT takes at least $k \cdot n/2 + \Omega(k^{3/4} \cdot n^{1/2})$. We give a sketch of a proof. $k \cdot n/2$ are needed for the routing. If M (sub-) splitters are selected, then they somehow must be compared with the packets. This is at least as expensive as sorting all subsets of M elements. The resulting inaccuracy is at least N/M. This implies that finally we still have to sort subsets of size N/M. The costs are minimal for $M = \sqrt{N}$, in that case the additional operations take $\Theta(k \cdot (\sqrt{N}/k)^{1/2}) = \Theta(k^{3/4} \cdot n^{1/2})$ steps.

[2]Each PU P_i can attach to the packet in its memory position j the unique identifier $k \cdot i + j$. The identifiers can be used as secondary sorting criterion.

$\mathcal{T}_{p,t}$ 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 $\mathcal{T}_{p,t}$. Then

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

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,

$$\begin{aligned} R_{p,t} &\geq \sum_{i=1}^{y} \alpha_i \cdot y^{t-1} \\ &= r_p \cdot y^t. \end{aligned}$$

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

$$\begin{aligned} R_{p,t} &\leq \alpha_1 \cdot y^{t-1} + \delta_{t-1} \\ &+ \sum_{i=2}^{y} ((\alpha_i + 1) \cdot y^{t-1} - 1 + \delta_{t-1}) \\ &= r_p \cdot y^t + y \cdot \delta_{t-1} + (y-1) \cdot (y^{t-1} - 1). \end{aligned}$$

Thus, δ_t is given by the following recurrence:

$$\begin{aligned} \delta_1 &= 0, \\ \delta_t &= y \cdot \delta_{t-1} + (y-1) \cdot (y^{t-1} - 1). \end{aligned}$$

By induction we proof that

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

for all $t > 1$. Suppose (5) holds for some $t - 1$, then

$$\begin{aligned} \delta_t &\leq (t-2) \cdot (y-1) \cdot y^{t-1} \\ &+ (y-1) \cdot (y^{t-1} - 1) \\ &= (t-1) \cdot (y-1) \cdot y^{t-1} - y + 1. \quad \square \end{aligned}$$

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

Theorem 4

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

Proof: Denote by $\mathcal{M}$ the set of selected splitters. And for any packet p by R_p its rank among the N packets in $\mathcal{S}$. 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,

$$r_p \cdot N/M < R_q < (r_p + x) \cdot N/M. \quad \square$$

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.

5.3 Refinement

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. This may have a positive effect on the cost of handling them, once they are selected.

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

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

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 Several Applications

We briefly consider the possibilities of application of the deterministic splitter selection on RAMs, PRAMs and hypercubes. The results are not fabulous and are mainly given for the sake of completeness. They are potential starting points for further research. More interesting results for meshes are presented in Section 7.

6.1 RAMs

For sorting N numbers on a sequential RAM computer we can apply a recursive algorithm based on the described splitter selection. Counting the number of required comparisons, such an algorithm can impossibly beat merge sort: merge sort runs in $N \cdot \log N - N + 1$ comparisons, which is less than $N/2$ away from the theoretical optimum ($\log N!$). Several other algorithms are in practice even better. Therefore, the main purpose of this section is theoretically in nature: to show that the algorithm actually has runtime $\mathcal{O}(N \cdot \log N)$.

Analysis. $M = (N \cdot \log N)^{1/2}$ numbers are selected as splitters. The selection involves sorting all subsets of M elements and then repeatedly merging and reducing pairs of them (we take $y = 2$). The merging takes $\mathcal{O}(N)$ steps all together. Hereafter the problem is reduced to sorting sets of less than $(N \cdot \log N)^{1/2}$ elements. Denote by $T(N)$ the number of comparisons required to sort a set of N numbers, then we have

Theorem 6 *For sorting sequentially,*

$$T(N) = \mathcal{O}(N \cdot \log N).$$

Proof: $T(N)$ is given by the following recurrence:

$$T(N) = 2 \cdot (N/\log N)^{1/2} \cdot T((N \cdot \log N)^{1/2}) + \mathcal{O}(N).$$

The recursion can be finished with a constant number of comparisons as soon as N has come below a threshold value. As an induction hypothesis we propose

$$T(N) = \alpha(N) \cdot N \cdot \log N.$$

This gives the condition

$$\alpha(\sqrt{N}) \cdot N \cdot (\log N + 2 \cdot \log\log N - 2) + \mathcal{O}(N) \leq \alpha(N) \cdot N \cdot \log N. \qquad (6)$$

For $\alpha(N)$ a constant function, this would not hold, but we can choose $\alpha(N)$ a slowly growing function bounded by a constant. We consider sufficiently large N, such that (6) can be simplified to

$$\alpha(N) - \alpha(\sqrt{N}) \geq \log^{-1/2} N.$$

This condition is satisfied by

$$\alpha(N) = \sum_{i=0}^{\log\log N} 2^{-i/2}.$$

So $\alpha(N) < 1/(1 - 1/\sqrt{2}) \simeq 3.41$. □

Implementation We have written a Pascal program implementing the algorithm. No further recursion is applied for segments of length six and less. We used it to sort random inputs. For sorting $4.194.304 = 2^{22}$ numbers, it takes about one hour of computation on a Sparc-10 workstation. For random inputs, the best choice for M is $M = sqrtN$. The number of comparisons is growing slowly from $2.41 \cdot N \cdot \log N$ for $N = 2048$ to $2.62 \cdot N \cdot \log N$ for $N = 16.777.216 = 2^{24}$. We never found that the estimated rank of a packet was wrong by more than $9 \cdot N/M$.

An hour for sorting 2^{22} numbers is extremely slow. Optimizing the program and using C instead of Pascal would speed the program up considerably. But, even a fully optimized variant will not be super fast. From the beginning this was not our goal, rather we wanted to obtain a feeling for the size of the leading constant. Its relatively small value suggests that under certain circumstances implementations on processor networks may be competitive.

In comparison with other *deterministic* sequential algorithms, our algorithm has one advantage: at all times it operates on sets consisting of only approximately $\sqrt{N}$ numbers. If paging is applied, this means that large problems do not imply continuously swapping pages. We actually observed this phenomenon in our implementation.

6.2 PRAMs

Basic Result. On a PRAM, the time $T(N)$ for sorting N numbers by an algorithm based on the deterministic selection of $M = (N \cdot \log N)^{1/2}$ splitters, satisfies the following recurrence

$$T(N) = 2 \cdot T((N \cdot \log N)^{1/2}) + T_{\text{sel}}(N). \quad (7)$$

Here $T_{\text{sel}}(N)$ is the time for selecting the splitters. In our analysis we need

Lemma 6 [18] *On a CREW PRAM with $N/\log\log N$ PUs, two sorted arrays of length N each can be merged to one sorted array of length $2 \cdot N$ in $\log\log N$ time.*

With (7) this immediately yields

Lemma 7 *On a CREW PRAM with $P \leq N/\log^2\log N$ PUs, N numbers can be sorted in $N \cdot \log N/P$ time.*

Proof: $T_{\text{sel}}(N) = \mathcal{O}(\log N \cdot \log\log N)$, for all $P \geq N/(\log N \cdot \log\log N)$. Now use induction to proof that $T(N) = \mathcal{O}(\log N \cdot \log^2\log N)$ for $P = N/\log^2\log N$. □

The lemma states that for P not too close to N the algorithm is work optimal. We will modify SELECT1, and refine our analysis to obtain a work-optimal algorithm running in $\mathcal{O}(\log\log\log N \cdot \log N)$ time.

Modified Splitter Selection. The splitters are selected as in SELECT1, by repeatedly applying REDUCE. However, the parameter y is no longer a constant: in every iteration of REDUCE, the number of candidate-splitters is halved, but the number of available PUs remains constant. The resulting surplus of PUs can be used to increase the number of sorted arrays that are merged at the same time:

```
      Algorithm SELECT2
  $i := 0$;
  Sort subsets of $M$ packets;
  **while** there are arrays to merge **do**
      apply REDUCE with $\lambda = 2^{2^i}$;
      apply REDUCE with $\lambda = 2^{2^i}$;
      $i := i + 1$.
```

The number of reduce operations is much smaller than in SELECT1: less than $\log\log N$.[3] This also means that the inaccuracy is much smaller than before. Analogously to the proof of Lemma 5 it can be shown that

$$\mathit{inac}(N, M) < \log\log N \cdot N/M.$$

Thus, the optimal choice for M is

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

In our estimate of $T_{\text{sel}}(N)$, the time for performing SELECT2, we need the following result:

Lemma 8 *On a CREW PRAM with $y^2 \cdot N$ PUs, y sorted arrays consisting of N elements each can be merged to one sorted array of $y \cdot N$ elements in $\mathcal{O}(\log\log N + \log y/\log\log y)$ time.*

Proof: The algorithm is analogous to the algorithm for merging two arrays [18]. For two arrays A, B, denote by (A, B) the array of ranks, giving for each element $a \in A$ its rank among the elements in B. In our case we have arrays $A_1, \ldots, A_y$, and we determine the $(y-1) \cdot y$ arrays (A_i, A_j), for all $1 \leq i \neq j \leq y$. In [18] it is shown how N PUs are sufficient to compute one particular array (A_i, A_j) in $\log\log N$ time, so $(y-1) \cdot y \cdot N$ PUs are enough to compute all of them in parallel within the same time bounds.

Now we have to compute the ranks of all elements among the $y \cdot N$ elements. This can be done by adding its ranks in all y lists together. Trivially applying a binary-tree algorithm for this would be to slow for our purposes. We apply

Fact 1 [7] *On a CREW PRAM with N PUs, N numbers in the range $0, 1, \ldots, N$ can be added together in $\mathcal{O}(\log N/\log\log N)$ time.*

For computing each of the $y \cdot N$ sums of y numbers we have y PUs available, so these sums can be computed in $\mathcal{O}(\log y/\log\log y)$ time. □

[3] There are $2 \cdot x$ calls to REDUCE, where x is the smallest number satisfying $(\prod_{i=0}^{x} 2^{2^i})^2 \geq N/M$.

Lemma 9 *On a CREW PRAM with N PUs,*

$$T_{sel}(N) = \mathcal{O}(\log N / \log\log N).$$

Proof: For an easier analysis assume that we have $4 \cdot N$ PUs. Generally, assume that in the i-th iteration, $i = 0, 1, \ldots$, of the above loop there are $2^{2^{i+1}} \cdot M$ PUs for each array of M elements. For $i = 0$ this condition is satisfied. In the i-th iteration, the number of lists is reduced by a factor $2^{2^{i+1}}$. So, in the $(i+1)$-st iteration there are $2^{2^{i+2}}$ PUs for every array. Thus, according to Lemma 8, an application of REDUCE in the i-th iteration can be performed in $\mathcal{O}(\log\log M + 2^i/i)$ time. Summing over all $\mathcal{O}(\log\log N/M)$ iterations gives a time consumption bounded by

$$\begin{aligned} & \log^2 \log N + \sum_{i=0}^{\log\log N} 2^i/i \\ \lesssim \; & \log N / \log\log N \end{aligned}$$

□

With a trivial modification to SELECT2 (starting with $\log\log N$ applications of REDUCE with $y = 2$), the number of PUs can be reduced by a factor $\log N / \log\log N$. The selection cannot be performed faster having more PUs:

Lemma 10 *If the number of available processors is polynomial, then*

$$T_{sel}(N) = \Omega(logN / \log\log N).$$

Proof: We must somehow merge N/M sorted subsets S_i, $1 \leq i \leq N/M$, of M elements each, and select the right elements out of them. Suppose that all elements in S_i lie within a range that does not overlap with the range in which the numbers of S_j lie, for all $i \neq j$. Then the final set of M elements holds one element from every S_i. These elements are sorted. Clearly this merging and selection is not easier than taking the smallest elements and sorting them. But sorting N/M numbers takes at least $\Omega(\log(N/M)/\log\log(N/M))$ time for any number of PUs that is polynomial in M [1]. For $M = N^\epsilon$, for some $\epsilon > 0$, this gives the desired result. □

Lemma 9 is the main ingredient in the proof of

Theorem 7 *On a CREW PRAM with $P \leq N/\log\log\log N$ PUs, N numbers can be sorted in $N \cdot \log N / P$ time.*

Proof: Instead of (7), the time for sorting N numbers $T_{\text{sort}}(N)$, is given by

$$T_{\text{sort}}(N) = 2 \cdot T_{\text{sort}}((N \cdot \log\log N)^{1/2}) + T_{\text{sel}}(N).$$

Suppose that $P = N/\log\log\log N$. Then $T_{\text{sel}}(N) = \mathcal{O}(\log N/\log\log N)$. Now the proof that $T_{\text{sort}} = \mathcal{O}(\log\log\log N \cdot \log N)$ is finished similar to the proof of Theorem 6. The factor $\log\log\log N$ originates from a sum $\sum_{i=1}^{\log\log N} 1/i$. □

For all practical purposes $\log\log\log N \leq 3$, but theoretically it is unsatisfactory that we did not obtain a $\mathcal{O}(\log N)$-sorting algorithm. Though this has been established before [6], our algorithm would constitute a conceptually simple alternative.

6.3 Hypercubes

On a hypercube the efficiency depends on the efficiency of the implementation of REDUCE. Denoting the time for routing the packets to their preliminary destinations by $T_{\text{route}}(N)$, the recurrence is of the form

$$T(N) \geq 2 \cdot T(M) + T_{\text{route}}(N) + T_{\text{sel}}(N).$$

For the case that every PU holds exactly one packet we cannot hope to achieve better than $\mathcal{O}(\log N \cdot \log\log N)$ because $T_{\text{route}}(N) \geq \log N$. But even an $\log N \cdot \log\log N$ algorithm might be a great achievement: it might constitute a more practical alternative for the algorithm of Cypher and Plaxton [9]. A necessary condition is that somehow we manage to perform the splitter selection in $\mathcal{O}(\log N)$ steps.

More promising, and practically more relevant, are the possibilities for the case that every PU of the hypercube holds a fairly large number k of packets. In that case the iterations of REDUCE become geometrically cheaper. If k exceeds the number of PUs, then many operations can be performed internally. See the next section for an analysis of these phenomena in the case of meshes. It is not hard to prove that

Theorem 8 *On a one-port hypercube with $P = 2^n$ PUs, and $N = k \cdot P$ packets, $k \geq \log N$,*

$$T(N) = \mathcal{O}(k \cdot \log N \cdot \log\log N).$$

7 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 [26].

7.1 Rank Estimation

We describe how the packets can efficiently obtain an estimate of their ranks. 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. If $m = 1$, then only retain the elements 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 elements with ranks $4 \cdot j$, for $1 \leq j \leq M$. After iteration i every PU should hold exactly $M/(4^{i+1} \cdot m^2)$ packets.

3. If $M/(4^i \cdot m^2) = 1$ then sort the remaining packets in the whole mesh and retain the packets 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 packets 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 $\sum_{l<j} A_l + a_{i,j} + r$.

Notice that, contrary to BASIC-SORT, the splitters are discarded at an early stage.

How many (routing) steps does ESTIMATE take? We need to make an assumption about the sorting in submeshes. We use a slightly simplified estimate:

Lemma 11 [26] *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.*

Step 1 is a k-k sorting in $m \times m$ meshes:

Lemma 12 *Step 1 takes $2 \cdot \sqrt{k \cdot M}$ steps.*

Step 2 is the hearth of the selection. It corresponds to KKMERGE from [26]. The packets are kept in semi-layered order at all times. This facilitates the merging. During 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 $\text{odd}(k' \cdot i + r + j)$.

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 [26].

Lemma 13 *Step 2 of* ESTIMATE *can be performed in $4^1\!/\!_2 \cdot \sqrt{k \cdot M}$ steps.*

Proof: In every iteration, n' becomes twice as large, and k' is divided by four. So, all iterations together take twice as many steps as the first iteration. For the first iteration, $n' = 2 \cdot m = 2 \cdot \sqrt{M/k}$, and $k' = k$. □

Step 3 is at worst a 1-1 sorting in the whole mesh:

Lemma 14 *Step 3 takes at most* $4^1\!/\!_2 \cdot n$ *steps.*

Broadcasting can be performed efficiently:

Lemma 15 *Step 4 requires at most* $3/4 \cdot \sqrt{k \cdot M'} + 2 \cdot n$ *steps.*

Proof: The splitters are first concentrated in each quadrant, then in each subquadrant, and so on. In the very last compression, we have four $m \times m$ submeshes. In each there are $M/4$ splitters, $k/4$ splitters per PU. The splitters from each submesh are first copied to the two adjacent submeshes. This takes $k \cdot m/4$ steps. Then each submesh copies half of the packets it just received from an adjacent submesh to the other adjacent submesh (in such a way that every submeshes receives everything). This takes $k \cdot m/8$ steps. The complete operation takes twice as long. If $M < n^2$, then first some routing has to be performed. □

In Step 5 one should exploit that the packets in the submeshes and the splitters were already sorted before:

Lemma 16 *Step 5 requires at most* $2^1\!/\!_2 \cdot \sqrt{k \cdot M'}$ *steps.*

Proof: Step 5 can be performed by first merging the packets and the splitters in every submesh. This can be achieved in $2 \cdot \sqrt{k \cdot M}$ steps. Then the splitters can easily detect their predecessors. Finally the computed numbers $\alpha_{i,j}$ have to be routed within the submeshes. By their special arrangement this only means spreading them out. □

Step 6 is a multiple parallel prefix operation:

Lemma 17 *Step 6 requires at most* $\sqrt{k \cdot M'} + 2 \cdot n$ *steps.*

Proof: It can be organized by first adding along rows and then along columns. A factor two is gained if half of the packets work orthogonally at all times. □

Step 7 is similar to Step 5:

Lemma 18 *Step 7 requires at most* $2^1\!/\!_2 \cdot \sqrt{k \cdot M'}$ *steps.*

We summarize the effects of ESTIMATE:

Lemma 19 ESTIMATE *runs in* $6^1\!/\!_2 \cdot \sqrt{k \cdot M} + 6^3\!/\!_4 \cdot \sqrt{k \cdot M'} + \mathcal{O}(n)$ *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: Summing over all steps gives the time consumption. The properties are established in Step 7. □

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

$$inac(N, M) = \log(N/M) \cdot N/M. \qquad (8)$$

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. So, on the contrary: determining the precise positions of the packets as early as possible helps reducing the overall routing time.

7.2 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 (8) 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 20 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 *routing* algorithm which is substantially faster than a *sorting* algorithm. Randomizedly we can apply the algorithm from [15]. 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 [21] 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 21 *For k-k sorting on* $n \times n$ *meshes with* $M \simeq N^{1/2}$,

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

For the values of N we consider, with $\log N \simeq 20$, a good choice for M is

$$M = (\log N \cdot N)^{1/2}/4 \tag{9}$$

Theorem 9 *The sorting algorithm based on* ESTIMATE *and* COMPLETE, *with M as in (9), yields*

$$\begin{aligned} T_{D2}(k,n) &< T_{route}(k,n) + \mathcal{O}(n) \\ &+ 22 \cdot k^{3/4} \cdot n^{1/2} \end{aligned}$$

Proof: Substitute the value of M and $N = k \cdot n^2$, and estimate $\log N < 24$. □

7.3 Evaluation

The result of Theorem 9 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. There are several remarks here:

- Possibly the constant can be more than halved by applying the algorithm recursively for sorting in the submeshes.
- If $k > n^2$, then the submeshes are reduced to single PUs, and the local sorting can be performed internally.
- Most other algorithms have larger exponents to the k and n.

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. T_{R1}, T_{D1}, T_{R2} and T_{D2} denote the sorting times for the algorithms of Section 3.1, Section 3.2, Section 4.2 and Section 7, respectively. Then we have

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

The randomized algorithm with subsplitter choice appears to outperform all others. For the deterministic algorithms we consider a small and a big instance. For $n = 8$ and $k = 1000$, we have $T_{\text{D1}} = 1.58 \cdot k \cdot n$ and $T_{\text{D2}} = 1.31 \cdot k \cdot n$. For $n = 64$ and $k = 4000$, we have $T_{\text{D1}} = 0.63 \cdot k \cdot n$ and $T_{\text{D2}} = 0.35 \cdot k \cdot n$. This should be compared with the routing time, about $k \cdot n/4$ for average-case inputs. The relative difference in the total sorting time is considerable, about 50% for large inputs.

How the algorithms perform exactly in practice cannot be accurately estimated. Many factors play a role. One of the most important is how many local operations are actually operations that are performed on the packets that stand within a single PU. The new deterministic algorithm shares with the randomized algorithms the property that this happens for $k > n^2$. For the simpler deterministic algorithm k has to be at least n^4.

8 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 [26].

A similar two-fold situation is reported to occur in practice: Diekmann e.a. [10] 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 [2].

By its simple structure and its regular routing operations our algorithm is suited for actual implementation. It would be interesting to compare a well-tuned version with the sample-sort algorithm applied by Diekmann e.a. We do not expect to be faster, but the difference should be negligible for sufficiently large k. In that case one might prefer the added certainty of a deterministic algorithm.

Acknowledgement

Torsten Suel gave valuable pointers to previous work in this field, and made some critical remarks.

References

[1] Beame, P., J. Hastad, 'Optimal Bounds for Decision Problems on the CRCW PRAM,' *Journal of the ACM*, 36, pp. 643–670, 1989.

[2] 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.

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

[4] Chaudhuri, S., T. Hagerup, R. Raman, 'Approximate and Exact Deterministic Parallel Selection,' *Proc. 18th Symp. on Mathematical Foundations of Computer Science*, LNCS 711, pp. 352–361, 1993.

[5] Chernoff, H., 'A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations,' *Annals of Mathematical Statistics*, 23, pp. 493–507, 1952.

[6] Cole, C., 'Parallel Merge Sort,' *SIAM Journal of Computing*, 17(4), pp. 770–785, 1988.

[7] Cole, R., U. Vishkin, 'Faster Optimal Parallel Prefix Sums and List Ranking,' *Information and Control*, 81, pp. 334–352, 1989.

[8] Cole, C., C.K. Yap, 'A Parallel Median Algorithm,' *Information Processing Letters*, 20, pp. 137–139, 1985.

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

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

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

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

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

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

[15] 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.

[16] Kaufmann, M., J.F. Sibeyn, 'Randomized Multi-Packet Routing and Sorting on Meshes,' unpublished manuscript, 1992. Submitted to Algorithmica.

[17] 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.

[18] Kruskal, C., 'Searching, Merging and Sorting in Parallel Computation,' IEEE Trans. on Computers, C-32, pp. 942–946, 1983.

[19] 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.

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

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

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

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

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

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

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

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