# Playing Tetris on Meshes and Multi-Dimensional Shearsort[*]

Mirosław Kutyłowski[†] Rolf Wanka[‡]

TR-97-029

August 1997

## Abstract

Shearsort is a classical sorting algorithm working in rounds on 2-dimensional meshes of processors. Its elementary and elegant runtime analysis can be found in various textbooks. There is a straightforward generalization of Shearsort to multi-dimensional meshes. As experiments turn out, it works fast. However, no method has yet been shown strong enough to provide a tight analysis of this algorithm. In this paper, we present an analysis of the 3-dimensional case and show that on the $l \times l \times l$-mesh, it suffices to perform $2 \log l + 10$ rounds while $2 \log l + 1$ rounds are necessary. Moreover, tools for analyzing multi-dimensional Shearsort are provided.

[*]Accepted for presentation on the 8th International Symposium on Algorithms and Computation (ISAAC), 1997.

[†]Department of Mathematics and Computer Science, Paderborn University, 33095 Paderborn, Germany. e-mail: `mirekk@uni-paderborn.de`. Supported by KBN grants 2 1197 91 01 and 8 S503 002 07, DFG grant Di 412/2-1, and by Volkswagen-Stiftung. A part of this work has been done when this author was affiliated with Computer Science Institute, University of Wrocław, Poland.

[‡]ICSI, 1947 Center Street, Berkeley, CA 94704, USA. e-mail: `wanka@icsi.berkeley.edu`. Supported by DFG-SFB 376 "Massive Parallelität," by EU ESPRIT Project 20244 (ALCOM-IT), and by DFG Leibniz Grant Me 872/6-1.

## 1 Introduction

Networks of processing elements (PUs) with multi-dimensional mesh topology have been the subject of intensive theoretical research motivated by the fact that many realizations of multiprocessor systems have the communication structure of a mesh. A variety of efficient algorithms has been designed to run on such an architecture, among them many algorithms for such a basic problem as sorting (e. g., see [7]).

In this paper, we examine multi-dimensional Shearsort, a very simple, but hard to analyze sorting algorithm for higher-dimensional meshes that turns out to have a rich combinatorial structure.

**Mesh Architecture.** The $d$-dimensional $m_d \times \cdots \times m_2 \times m_1$-*mesh* is the graph $(V, E)$, where $V = \{P[i_d, \ldots, i_2, i_1] : i_k \in \{1, \ldots, m_k\}\}$ is the set of nodes. Two nodes $P[i_d, \ldots, i_2, i_1]$ and $P[i'_d, \ldots, i'_2, i'_1]$ are connected by an edge called *link* if there is an $s$, $s \leq d$, such that $i_j = i'_j$ for $j \neq s$ and $|i_s - i'_s| = 1$. We refer to the set $\{P[i_d, \ldots, i_{k+1}, s, i_{k-1}, \ldots, i_1] : s \leq m_k\}$ of nodes as a *row in dimension k*. The $l \times l \times l$-mesh is called a *cube* of size $l$. The *jth level* of the cube is the set of nodes with third index equal to $j$, i. e., $\{P[j, i_2, i_1] : i_2 \leq l, i_1 \leq l\}$.

We assume that every node of the mesh can store exactly one key. Sorting these keys is the process of relocating them so that finally their ordering agrees with some predefined order "$\prec$" of the nodes. The 2-dimensional Shearsort uses the *snake-like* order depicted in Fig. 1. The key property of this order is that every two neighboring rows in dimension 1 are ordered in opposite directions. For the 3-dimensional mesh, we apply a similar approach (see Fig. 1):

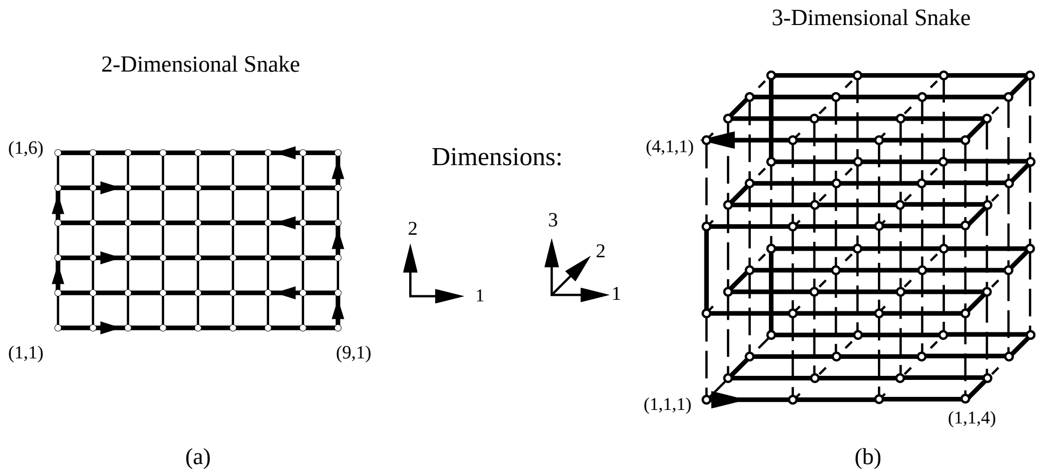


Figure 1: Snake-like order of (a) a 2-dimensional mesh and (b) a 3-dimensional mesh.

(1) Every level is ordered by a 2-dimensional snake-like order,

(2) every two neighboring levels are ordered in opposite directions,

(3) the nodes of level $j$ precede all nodes of level $j + 1$, for every $j$; the first node of level $j + 1$ lies above the last node on level $j$.

**Compare-Exchange and Oblivious Algorithms.** In the design of sorting algorithms for networks of processing units, it is reasonable to perform operations that are as simple as possible.The reason is that complex operations may require powerful (expensive) PUs and might be slow in practice. A *compare-exchange step* is such a simple operation. It is determined by a matching $M$ on the set of nodes, where each $(P, P') \in M$ corresponds to a link in the network. For each $(P, P') \in M$, if $x$ and $x'$ are stored in $P$ and $P'$, resp., then after performing the operation, $P$ stores $\min(x, x')$ and $P'$ stores $\max(x, x')$. A *compare-exchange algorithm* is an algorithm consisting of compare-exchange steps.

Many sorting algorithms use *routing* in addition to compare-exchange steps: the contents of the network is permuted according to a given permutation that is fixed during the network design. That means that these permutations do not depend on the sequence of keys to be sorted. Such algorithms are called *oblivious* sorting algorithms.

**Previous Oblivious Sorting Algorithms on Meshes.** Routing (besides compare-exchange steps) has been used by Schnorr and Shamir [11] to design an asymptotically time optimal algorithm that sorts $l^2$ keys on the $l \times l$-mesh in $3l + o(l)$ steps. For $d$-dimensional meshes ($d$ being a constant), Kunde [6] presented an asymptotically time optimal algorithm: On the $m_d \times \cdots \times m_1$-mesh, it runs in time $2\sum_{i=1}^{d-1} m_i + m_d + o(\sum_{i=1}^{d} m_i)$. One of the mesh architectures that has received special attention is the hypercube, i. e., the $d$-dimensional $2 \times \cdots \times 2$-mesh. The asymptotically fastest oblivious algorithm for the $d$-dimensional hypercube by Plaxton [8] sorts $2^d$ keys in time $O(2^{O(\sqrt{\log d})})$. (Note that there are sorting algorithms that work asymptotically faster, but they are not oblivious. They duplicate keys, communicate more than one key per step, perform permutations depending on the inputs, and make sometimes certain assumptions on the representation of the keys.)

Unfortunately, the low order terms and constant factors, resp., hidden in the "O"-notation in the runtime bounds of the algorithms mentioned above are quite large. Moreover, the logical structures of these algorithms are complicated; the algorithms spend much time on routing. For these reasons, these algorithms are not well suited for practical implementations.

On the other hand, there are simple algorithms like Batcher's Bitonic Sort [1] for the $d$-dimensional hypercube with runtime $\frac{1}{2}d(d+1)$ and a generalization of Bitonic Sort to arbitrary meshes by Corbett and Scherson [3]. Though the runtimes of these algorithms are not asymptotically optimal, the involved constants are very small. So the algorithms are fast for realistic input sizes. The behavior of these algorithms is easy to analyze due to their recursive structure allowing elegant inductive proofs.

In order to ease the implementation, *periodic* sorting algorithms have been considered, i. e., algorithms repeating the same sequence of compare-exchange steps called a *round*. A classical algorithm of this kind is Shearsort by Scherson, Sen, and Shamir [10, 9]. It runs on $m_2 \times m_1$-meshes in time $(m_2 + m_1) \cdot (\lceil \log m_2 \rceil + 1)$. In another classical paper, Dowd et al. [4] present the periodic balanced sorting algorithm running on the $d$-dimensional hypercube in $d^2$ steps. These algorithms repeat sorting of rows in various dimensions until the contents of the network becomes sorted with respect to the snake-like order. In the worst case, both algorithms are slower only by a factor of about 2 than the simple non-periodic methods mentioned before.

In [2], an approach closely related to Shearsort has been implemented on the MasPar MP-1 parallel computer. Its practical performance is able even to beat Bitonic Sort under certain circumstances. This is an evidence that a theoretical approach must be sometimes revised to adhere to the real world.

**Multi-Dimensional Shearsort.** Both Shearsort and the periodic balanced sorting algorithm are special cases of a simple and elegant algorithm for arbitrary multi-dimensional meshes with snake-like order where sorting of rows in all dimensions is repeated until the input is sorted according to the snake-like order:

ALGORITHM. (Multi-Dimensional Shearsort) Let $M$ be the $m_d \times \cdots \times m_1$-mesh. Shearsort consists of rounds: each round consists of stages $1, 2, \ldots, d-1, d$. At stage $i$, $i \leq d$, each row in dimension $i$ is sorted using Odd-Even Transposition Sort [5, p. 241]. The direction in which a single row is sorted agrees with the order on the row induced by the snake of the mesh (for the 3-dimensional case see Fig. 2). The rounds are repeated until $M$ is sorted according to the snake-like order.

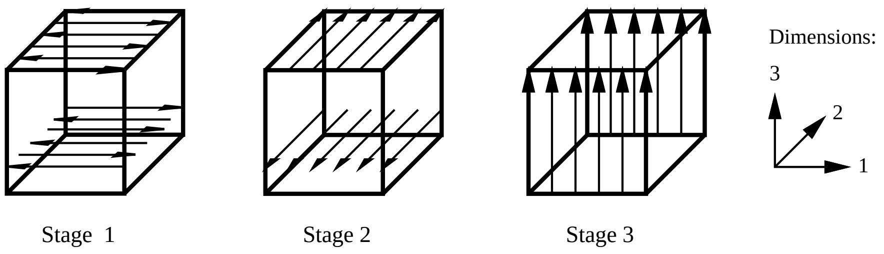


Figure 2: A round of the 3-dimensional Shearsort

**New Results.** We present almost matching upper and lower bounds on the number of rounds that the 3-dimensional Shearsort needs to sort on the cube.

**Theorem 1** *Let $l \in \mathbb{N}$ be arbitrary. Then the 3-dimensional Shearsort requires at most $2 \log l + 10$ rounds on the $l \times l \times l$-mesh in the worst case.*

The above bound is almost tight since there are simple inputs requiring $2\lfloor \log l \rfloor + 1$ rounds as stated in the following example (its analysis is tedious, but elementary, and therefore omitted):

**Example 2** *Let $l \in \mathbb{N}$ be even, but not a power of 2. Let the set of nodes $\{P[i_3, i_2, 1] : 1 \leq i_3, i_2 \leq 2^{\lfloor \log l \rfloor}\}$ store 0's, and let all other nodes store 1's. Then the 3-dimensional Shearsort needs $2\lfloor \log l \rfloor + 1$ rounds to sort this input.*

Theorem 1 can be generalized in many ways to arbitrary 3-dimensional meshes, but the analysis requires more technical details (to be presented in a full version of the paper). A generalization for higher dimensions is possible, too, but in this case we do not obtain such tight results.

The paper is organized as follows. In Section 2, we recall some facts and make observations on the 2-dimensional Shearsort to be applied for the 3-dimensional case. The most interesting fact in this part is a combinatorial property called *Tetris Lemma.* In Section 3, we develop techniques of projecting the contents of the cube into two dimensions and bookkeeping of *dirty rows* that lead to a proof of Theorem 1.

## 2 Auxiliary Results on 2-Dimensional Shearsort

### 2.1 General Observations.

**Lemma 3 (0-1 Principle [5, p. 224])** *If a compare-exchange algorithm sorts all inputs consisting solely of* 0*'s and* 1*'s, then it sorts arbitrary inputs.*

Due to the 0-1 Principle, we shall mainly consider inputs consisting of 0's and 1's only, also called 0-1 *inputs*. The 0-1 Principle can be shown by applying the following fundamental property we shall use later, too.

**Lemma 4 (Homomorphic invariance)** *Let $h$ be a monotonic function. Let $\mathcal{A}$ be a compare-exchange algorithm working on a network $M$ and let $\mathcal{A}(\vec{x})$ denote the contents of $M$ after executing $\mathcal{A}$ on input $\vec{x}$. Then $h(\mathcal{A}(\vec{x})) = \mathcal{A}(h(\vec{x}))$, for every $x$.*

*Proof.* The proof is by induction on the number of compare-exchanges performed. As $h$ is monotonic, $\max(h(x), h(x')) = h(\max(x, x'))$ and $\min(h(x), h(x')) = h(\min(x, x'))$. So the lemma holds for a single compare-exchange operation. Let $B$ be all compare-exchange operations of $A$ except the last one $C$. Then by the induction hypothesis $h(\mathcal{A}(\vec{x})) = h(C(B(\vec{x}))) = C(h(B(\vec{x}))) = C(B(h(\vec{x}))) = \mathcal{A}(h(\vec{x}))$. $\square$

The next proposition is obvious, nevertheless it is extremely useful.

**Lemma 5 (Permutation Trick)** *Assume that at a given moment of the execution of a compare-exchange algorithm, the nodes of the network can be partitioned into subsets $T_1, \ldots, T_k$, and that each $T_i$ is subject to a separate sorting process. Then we may arbitrarily permute the keys inside each set $T_i$ prior to sorting them without influencing the outcome of sorting the sets $T_i$.*

### 2.2 Dirty Rows and the Classical Results.

The analysis of 2-dimensional Shearsort in [10] is based on the following concept of *clean* and *dirty rows*:

**Definition 6** *Let $M$ be a (multi-dimensional) mesh. If every node of a row in dimension $j$ of $M$ contains the same key $s$, then we call this row a* clean row *or in more detail an $s$-*row. *A row containing at least two different keys is called a* dirty row.

**Definition 7** *Let $M$ be a 2-dimensional mesh with a* 0-1 *input. We say that $M$ is $h$-*clean, *if $M$ contains a number of* 0*-rows at the bottom of $M$ and a number of* 1*-rows at the top of $M$, and at most $h$ dirty rows between these two blocks of clean rows.*

In order to illustrate the notions introduced, we prove the following lemma:

**Lemma 8** *Assume that a 2-dimensional mesh $M$ stores a* 0-1 *input. If $M$ contains $k$ zero-rows* (1*-rows), then after sorting the columns, $M$ contains $k$ zero-rows* (1*-rows) at the bottom (top) of $M$. These rows remain there after any number of steps of Shearsort. So if $M$ is $h$-clean, then it remains $h$-clean after executing any number of rounds of Shearsort.*

*Proof.* By the Permutation Trick, we may permute the rows of $M$ so that the 0-rows are placed at the bottom of $M$. Then during sorting a column no 0 of the $k$ lowest positions in a column can be moved. The rest follows immediately. $\square$

The upper bound on the runtime of the 2-dimensional Shearsort follows from the following lemma [10]:

**Lemma 9 ([10])** *Let a* 0-1 *input be given to the* $l \times m$*-mesh* $M$*. Then after round* $t$ *of Shearsort,* $M$ *is* $\lceil l/2^t \rceil$*-clean.*

By Lemma 9, after round $\lceil \log l \rceil$ the mesh is 1-clean and by Lemma 8 it will remain 1-clean afterwards. Then it suffices to perform one additional round to sort the dirty row. Hence, the following theorem holds:

**Theorem 10 ([10])** *Shearsort requires at most* $\lceil \log l \rceil + 1$ *rounds to sort on the* $l \times m$*-mesh.*

*Proof of Lemma 9.* The proof is by induction on $t$. For $t = 0$, it is obviously true. Let us assume it is true for $t$. Let $s$ be the number of rows in the block of 0-rows at the bottom of $M$. After sorting the rows at round $t+1$, we use the Permutation Trick: we sort each pair of rows $s+i$ and $s+i+1$ columnwise, for every odd $i$. Since the rows of a pair have been sorted in opposite directions, for every pair we get at least one clean row (see Fig. 3). By Lemma 8, while sorting the columns at round $t+1$, the 0-rows and 1-rows are moved downwards and upwards, resp. There are at most $\lceil \frac{1}{2} \cdot \lceil l/2^t \rceil \rceil = \lceil l/2^{t+1} \rceil$ dirty rows left, hence $M$ becomes $\lceil l/2^{t+1} \rceil$-clean. $\square$

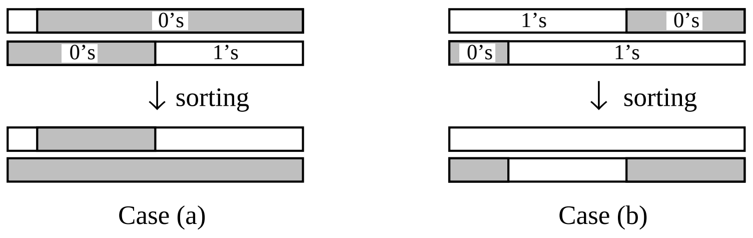


Figure 3: At least one clean row is obtained from two dirty rows

Later we shall use the following variation of Lemma 9:

**Lemma 11** *Assume that there are* $p$ *occurrences of a key* $u$ *in the* $l \times m$*-mesh. Then after* $t$ *rounds of Shearsort, there is a set of at most* $2 \cdot \lceil l/2^t \rceil + p/m$ *contiguous rows such that below them all keys stored are smaller than* $u$ *and above them all keys stored are greater than* $u$*.*

*Proof.* If we replace the input keys smaller than $u$ by 0's and the remaining keys by 1's, then after round $t$ there are at most $\lceil l/2^t \rceil$ dirty rows. By Lemma 4, this means that in the original mesh after round $t$ there are at most $\lceil l/2^t \rceil$ rows with at least one key smaller than $u$ and at least one key not smaller than $u$. Similarly, there are at most $\lceil l/2^t \rceil$ rows with at least one key not greater than $u$ and at least one key greater than $u$. There are at most $p/m$ rows containing only $u$'s. It follows that $u$ may occur in at most $2 \cdot \lceil l/2^t \rceil + p/m$ rows. $\square$

## 2.3 The Tetris Lemma

Sorting the columns resembles the Tetris game: we hope to fill as many rows at the bottom as possible with 0's. Below we define "pieces" used for our "Shearsort game".

**Definition 12** *(i) A* segment *with endpoints $i_1, i_2$ in a 2-dimensional mesh is a set of nodes of the form $\{P[j,i] : i_1 \le i < i_2\}$ for some $j$. A segment is a $u$-*segment *if all its nodes store the key $u$.*

*(ii) Let $k \in \mathbb{N}$. We say that key $u$ is $k$-dispersed in a 2-dimensional mesh $N$, if the set of nodes of $N$ storing $u$ can be partitioned into at most $k$ disjoint segments.*

For a given key $u$, we define its *canonical segments* storing $u$ on the $l \times m$-mesh: If $P[j,i]$ stores $u$ and $P[j,i-1]$ does not, or if $i = 1$, then a canonical segment starts at $P[j,i]$. This segment is as long as possible: its right endpoint is at $P[j,i']$, where $i' = \min\{s : s > i, P[j,s]$ does not store $u\}$. If $i'$ does not exist, the endpoint is $m+1$. Obviously, the canonical segments are disjoint and contain all $u$'s.

By sorting the rows of the $l \times m$-mesh containing 0's, 1's and $\frac{1}{2}$'s the key $\frac{1}{2}$ becomes $l$-dispersed – simply each row contains at most one segment. We show that it remains $l$-dispersed after sorting the columns.

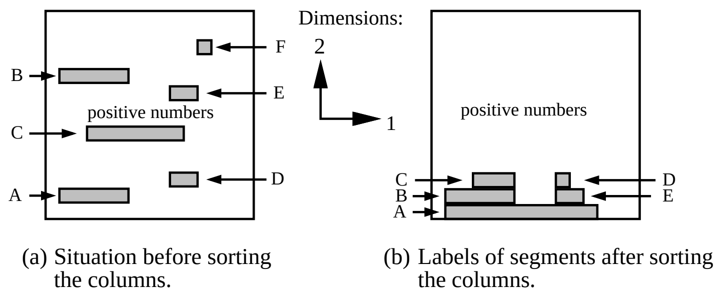
**Lemma 13 (Tetris Lemma)** *Let $N$ be the $l \times m$ mesh.*

*(a) Let 0 be the minimum key stored in $N$. If 0 is $k$-dispersed in $N$, then sorting the columns of $N$ preserves this property. The endpoints of the new segments can be taken from the set of the endpoints of the canonical segments existing before sorting the columns.*

*(b) Let $N$ contain keys 0, 1 and $\frac{1}{2}$ only and assume each row of $N$ sorted. Then after sorting the columns, key $\frac{1}{2}$ is $l$-dispersed in $N$.*

*Proof.* (a) Intuitively, (a) can be shown by induction on the number of 0-segments. Consider Fig. 4 and inject in subfigure (b) a further 0-segment from above. No matter where this additional segment "falls down," the number of 0-segments cannot be increased by more than 1. We show this in a more rigorous way: Let $e_1, \ldots, e_s$ be all endpoints of the original $k$ segments of 0's in $N$ with $e_1 < e_2 < \ldots < e_s$. Additionally, we consider a dummy column 0 storing 1's and put $e_0 = 0$. Let $f(i)$ denote the number of 0's in the column $i$ of $N$. Obviously, $f$ is constant on each interval $[e_j, e_{j+1})$. Let us consider the canonical 0-segments existing after sorting the columns of $N$. These segments called *new segments* have endpoints from the set $\{e_1, \ldots e_s\}$. Indeed, since 0 is the minimum key, the 0's in each column form a block starting at the bottom of the column. Therefore, if a new segment has an endpoint at a column $i$, then $f(i) \ne f(i-1)$. So $i$ must be one of the points $e_1, \ldots, e_s$.

It remains to show that the number of new segments does not exceed $k$. Let us label the original $k$ segments containing 0's with $k$ different labels. We label the new segments with the same labels in the following way (see Fig. 4): Consider a $j$, $j \le s$. A new segment starts at column $e_j$, if $f(e_j) > f(e_j - 1)$. If $f(e_j) = f(e_j - 1) + q$, then there are at least $q$ original segments starting at $e_j$ (there were even more such segments, if there were original segments with the left endpoint $e_j$). The $q$ new segments starting at column $e_j$ get labels of the original segments starting at column $e_j$, each one receiving a different label (see Fig. 4). Since every new segment gets one label out of $k$ and no label is used twice, the number of new segments does not exceed $k$.



Dimensions:
2
1
B
C
A
F
E
D
positive numbers
positive numbers
C
B
A
D
E
(a) Situation before sorting the columns.
(b) Labels of segments after sorting the columns.

Figure 4: Preserving the number of 0-segments during sorting of columns

(b) We label all 0- and 1-segments in $N$ existing before sorting the columns so that no label is used twice. We consider a dummy column 0 (column $m+1$) with 0's (1's) in the odd rows and 1's (0's) in the even rows. Since the rows of $N$ are sorted, at each row there is at most one 0- or 1-segment disjoint with the first column. Such segments will be called *proper segments*. Hence we need at most $l$ labels to label the proper segments.

We consider the canonical sets of 0-segments, 1-segments and $\frac{1}{2}$-segments after sorting the columns; these segments will be called the *new segments*. We label the new 0- and 1-segments in a similar way as in the part (a).

We assign the labels of the new $\frac{1}{2}$-segments. If such a segment ends at $P[i,j]$, then we take the label of the new segment that has the left endpoint at $P[i,j]$. Because of dummy column $m+1$, there is always such a segment on the right hand side of $P[i,j]$. In this way, every new $\frac{1}{2}$-segment receives a label and no two $\frac{1}{2}$-segments receive the same label. It suffices to check that at most $l$ labels have been assigned to the new $\frac{1}{2}$-segments. Note that no $\frac{1}{2}$-segment ends at column 0 (there are no $\frac{1}{2}$'s in column 0), hence the new $\frac{1}{2}$-segments receive the labels from the 0- and 1-segments with the left endpoints in the columns 2 through $m$. These labels are the labels of proper segments, so there are at most $l$ of them. □

## 3 Main Technical Analysis

One may try to generalize the approach of Lemma 9 of halving the number of dirty rows to the 3-dimensional case. We may try to reduce the number of dirty rows in dimension 1. In the 2-dimensional mesh, this number is nearly halved in each round. However, one gets into trouble on the 3-dimensional mesh, since after sorting along dimension 1 the dirty rows inside a level need not form a contiguous block. In an extreme case when every second row is dirty, their number will be not reduced at all. Therefore, we also need to inspect the location of dirty rows.

### 3.1 Projections of 3-Dimensional Meshes

**Definition 14** *Let $M$ be a cube of size $l$ with a* 0-1 *input. Let $R_{i,j} = \{P[i,j,s] : 1 \le s \le l\}$ denote a row in dimension 1. We consider the 2-dimensional $l \times l$-mesh consisting of nodes*

$N[i,j]$, $i \leq l, j \leq l$. *Its contents is defined as follows:*

*(i) If $R_{i,j}$ is a 0-row (1-row), then $N[i,j]$ contains a zero (a one).*

*(ii) If $R_{i,j}$ is a dirty row, then $N[i,j]$ contains $\frac{1}{2}$.*

*Proj$(M)$ denotes this 2-dimensional mesh with these keys and is called the* projection *of $M$.*

Note that the projection of the 3-dimensional snake is the 2-dimensional snake, i.e., if $P[i,j,s] \prec P[i',j',s']$, then $N[i,j] \prec N[i',j']$. The idea is that once $R_{i,j}$ becomes a clean row, then Shearsort will move it around $M$ as a whole row. Similar movements can be observed in *Proj*$(M)$. Hence, in order to trace the progress of the algorithm, we shall observe the changes in *Proj*$(M)$. The stages Shearsort executes on the 3-dimensional mesh $M$ have their counterparts on *Proj*$(M)$: Sorting along dimension 2 (3) in $M$ corresponds to sorting the rows (columns) of *Proj*$(M)$. Sorting the rows in dimension 1 in $M$ corresponds to an empty step on *Proj*$(M)$. If $A$ is a sequence of rounds of Shearsort on $M$, then let *Proj*$(A)$ denote the sequence of corresponding rounds on *Proj*$(M)$. Let $A(M)$ denote the result of executing $A$ on $M$.

**Lemma 15** *Let $A$ be a sequence of rounds of the 3-dimensional Shearsort on $M$. Then Proj$(A(M))$ may be obtained from Proj$(A)($Proj$(M))$ by replacing some number of $\frac{1}{2}$'s by 0's and 1's.*

*Proof.* Obviously, it suffices to prove the lemma for the three single stages of the 3-dimensional Shearsort. For Stage 1, it is obvious, since sorting a row in dimension 1 does not change the fact whether a given row is dirty, hence *Proj*$(M)$ does not change. For Stage 2, assume that $M$ contains $i$ zero-rows and $j$ one-rows in dimension 1 at level $k$. Equivalently, row $k$ of *Proj*$(M)$ contains $i$ zeroes and $j$ ones. While sorting the rows of *Proj*$(M)$ these 0's and 1's are moved into the opposite sides of row $k$. Stage 2 of the 3-dimensional Shearsort acts on each level of $M$ separately. So if we look at level $k$ of $M$ as a 2-dimensional mesh, then Stage 2 sorts the columns of this 2-dimensional mesh. By Lemma 8, $i$ zero-rows and $j$ one-rows will be retained and moved into opposite sides of level $k$ (possibly some new clean rows emerge as a result of sorting the columns). Afterwards *Proj*$(M)$ contains at least $i$ zeroes and $j$ ones on the sides of row $k$. At these positions, all 0's and 1's obtained by sorting the rows of *Proj*$(M)$ can be found.

The proof for Stage 3 is similar. □

## 3.2 Eliminating $\frac{1}{2}$'s

We shall trace the behavior of Shearsort on $M$ by examining the contents of *Proj*$(M)$. The next lemma is a straightforward generalization of Lemma 9:

**Lemma 16** *Let $M$ be a 3-dimensional mesh, whose rows in dimension 1 are sorted. If Proj$(M)$ contains $r$ occurences of $\frac{1}{2}$'s located in $s$ segments, then after sorting the rows of $M$ in dimension 2, there are at most $\frac{r+s}{2}$ occurrences of $\frac{1}{2}$'s in Proj$(M)$.*

*Proof.* Let $S_i$ denote the $i$th $\frac{1}{2}$-segment existing immediately before sorting the rows of $M$ in dimension 2. Let $u_i$ be the number of nodes in $S_i$. As in the proof of Lemma 9, we partition the rows in dimension 1 corresponding to the $\frac{1}{2}$'s in $S_i$ into pairs, in each pair two rows sorted in two different directions. (If $u_i$ is odd, then there is one row left.) Then inside each pair we sort the elements along dimension 2. Thereby, as on Fig. 3, we get at least one 0- or 1-row. Together we reduce the number of dirty rows corresponding to $S_i$ from $u_i$ to $\lceil u_i/2 \rceil \leq u_i/2 + \frac{1}{2}$. Then we sort the rows of $M$ in dimension 2. By the Permutation Trick, the result is the same as without performing the additional steps described above. So after sorting rows in dimension 2 the number of dirty rows in dimension 1 does not exceed $\sum_{i=1}^{s}(\frac{u_i}{2} + \frac{1}{2}) = \frac{r+s}{2}$. □

Let $d_t$ denote the number of $\frac{1}{2}$'s in *Proj*$(M)$ after the second stage of round $t$ of the Shearsort algorithm is executed on the cube $M$ of size $l$. Assume the $\frac{1}{2}$'s be $s_t$-dispersed at this moment. We may upper bound $d_t$ using the following lemma which depends crucially on the Tetris Lemma:

**Lemma 17** *For each* $t \geq 1$, $d_{t+1} \leq \frac{1}{2}(d_t + s_t)$.

*Proof.* By Lemma 15 the contents of *Proj*$(M)$ after round $t$ may be obtained by taking *Proj*$(M)$ after sorting the rows in dimension 2 at round $t$, sorting the columns of *Proj*$(M)$ and replacing certain $\frac{1}{2}$'s, say $r$ of them, by 0's and 1's. Before sorting the columns of *Proj*$(M)$, $\frac{1}{2}$is $s_t$-dispersed. By the Tetris Lemma (Lemma 13), sorting the columns of this 2-dimensional mesh leaves the $\frac{1}{2}$'s $s_t$-dispersed. The real contents of *Proj*$(M)$ at this moment becomes at most $(s_t + r)$-dispersed. Indeed, each new 0 or 1 may cut an existing segment of $\frac{1}{2}$'s into at most two pieces. The number of $\frac{1}{2}$'s after round $t$ equals $d_t - r$. So by Lemma 16, we have $d_{t+1} \leq \frac{1}{2}((d_t - r) + s_t + r) = \frac{1}{2}(d_t + s_t)$. □

Since there are $l$ rows in *Proj*$(M)$ and each row is sorted after performing the second stage, $\frac{1}{2}$ is always $l$-dispersed in *Proj*$(M)$ at this moment. Hence we get the following corollary.

**Corollary 18** *For each* $t \geq 1$, $d_{t+1} \leq \frac{1}{2}(d_t + l)$. *So* $d_{t+j} \leq d_t/2^j + l$ *for* $j \geq 1$.

## 3.3 Runtime Analysis

Now we have all tools to prove Theorem 1. Let *Proj*$(M)$ be $z_t$-clean after round $t$. I. e., there are at most $z_t$ (contiguous) dirty rows in *Proj*$(M)$. The idea of the proof is to upper bound $z_t$ and $d_t$ by applying Lemma 11 and Corollary 18 repeatedly. Our aim is to find a moment $t'$ such that $z_{t'} = O(1)$ and $d_{t'} = O(1)$. Then the computation may be easily terminated in a few rounds. For this purpose, we divide Shearsort into phases, each consisting of some number of rounds. The trick is that during each phase the techniques used to estimate the decrease of $z_t$ and $d_t$ differ a little bit. It is surprising that these estimations give such a tight result. Let $k = \lceil \frac{1}{2} \log l \rceil$.

**Phase 1:** $2k + 1$ *rounds.*
Obviously, $d_1 \leq l^2$. So by Corollary 18, $d_{i+1} \leq l^2/2^i + l$. Therefore, $d_{2k+1} \leq 2l$. Now we estimate $z_{2k+1}$. Consider *Proj*$(M)$ after sorting the rows of $M$ in dimension 2 at round $k+1$. Then there are $d_{k+1} \leq l\sqrt{l} + l$ occurrences of $\frac{1}{2}$in *Proj*$(M)$. Let us perform $k$ rounds of the 2-dimensional Shearsort on this mesh. By Lemma 11, after these rounds there are at most $2\lceil l/2^k \rceil + \sqrt{l} + 1 \leq 3\sqrt{l} + 3$ rows that are above 0-rows at the bottom of the mesh and below 1-rows at the top of the mesh. By Lemma 15, it also holds for the contents of *Proj*$(M)$ after executing round $2k + 1$ of 3-dimensional Shearsort on $M$. Hence $z_{2k+1} \leq 3\sqrt{l} + 3$.

**Phase 2:** $k+3$ *rounds.*
Our goal is to achieve $z_t \leq 3$. Consider *Proj*$(M)$ at the beginning of Phase 2. Since $\log z_{2k+1} + 1 \leq k+3$, it suffices to perform $k+3$ rounds of the 2-dimensional Shearsort on *Proj*$(M)$ to sort it, hence in particular to get all $\frac{1}{2}$'s stored in at most 3 rows. So by Lemma 15, all $\frac{1}{2}$'s of *Proj*$(M)$ after round $3k+4$ are stored in at most 3 rows. By Lemma 11,

$$z_{2k+1+j} \leq \frac{d_{2k+1}}{l} + 2\left\lceil \frac{z_{2k+1}}{2^j} \right\rceil \leq 6 + \frac{3\sqrt{l}}{2^{j-1}}.$$

Hence by Corollary 18, $d_{2k+1+j+1} \leq \frac{1}{2}(d_{2k+1+j} + z_{2k+1+j}) \leq \frac{1}{2}d_{2k+1+j} + \frac{3\sqrt{l}}{2^j} + 3$. This yields $d_{2k+1+j} \leq \frac{2l}{2^j} + \frac{3j\sqrt{l}}{2^{j-1}} + 6$. In particular $d_{3k+4} \leq \frac{1}{4}\sqrt{l} + \frac{3}{4}(k+3) + 6$.

As we see, at the end of Phase 2 the contents of $M$ is sorted except for at most 3 levels. Inside these 3 levels there are at most $\frac{1}{4}\sqrt{l} + \frac{3}{4}(k+3) + 6$ dirty rows in dimension 1. Since the number of dirty rows is small it can be reduced fast during the next phase.

**Phase 3:** $k$ *rounds.*
By Corollary 18, we get $d_{3k+4+j} \leq \frac{1}{2^j}d_{3k+4} + 3$. So $d_{4k+4} \leq 5$.

**Phase 4:** 5 *rounds.*
A simple case inspection shows that 5 rounds suffice to finish sorting of $M$. $\square$

## References

[1] K. E. Batcher. Sorting networks and their applications. In *AFIPS Conf. Proc. 32*, pp. 307–314, 1968.

[2] K. Brockmann and R. Wanka. Efficient oblivious parallel sorting on the MasPar MP-1. In *Proc. 30th Hawaii International Conference on System Sciences (HICSS)*, Vol. I, pp. 200–208, 1997.

[3] P. F. Corbett and I. D. Scherson. Sorting in mesh connected multiprocessors. *IEEE Transactions on Parallel and Distributed Systems* 3 (1992) 626–632.

[4] M. Dowd, Y. Perl, M. Saks, and L. Rudolph. The periodic balanced sorting network. *Journal of the ACM* 36 (1989) 738–757.

[5] D. E. Knuth. *The Art of Computer Programming, Volume 3: Sorting and Searching* (Addison-Wesley, Reading, 1973).

[6] M. Kunde. Optimal sorting on multi-dimensionally mesh-connected computers. In *Proc. 4th Symposium on Theoretical Aspects of Computer Science (STACS)*, pp. 408–419, 1987.

[7] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes* (Morgan Kaufmann, San Mateo, 1992).

[8] C. G. Plaxton. A hypercubic network with nearly logarithmic depth. In *Proc. 24th ACM Symposium on Theory of Computing (STOC)*, pp. 405–416, 1992.

[9] I. D. Scherson and S. Sen. Parallel sorting in two-dimensional VLSI models of computation, *IEEE Transactions on Computers* 38 (1989) 238–249.

[10] I. D. Scherson, S. Sen, and A. Shamir. Shear-sort: A true two-dimensional sorting technique for VLSI networks, in *Proc. 15th IEEE International Conference on Parallel Processing (ICPP)*, 1986, pp. 903–908.

[11] C. P. Schnorr and A. Shamir. An optimal sorting algorithm for mesh-connected computers. In *Proc. 18th ACM Symposium on Theory of Computing (STOC)*, pp. 255–263, 1986.