Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound

Manfred Kunde *
Institut für Informatik
TU München
Arcisstr. 21, D-8000 Munich 2, Germany

Abstract

Sorting and routing on r-dimensional $n \times ... \times n$ grids of processors is studied. Deterministic algorithms are presented for $h - h$ problems, $h \geq 1$, where each processor initially and finally contains h elements. We show that the classical $1-1$ sorting can be solved with $(2r-1.5)n+o(n)$ transport steps, i.e. in about $2.5n$ steps for $r = 2$. The general $h - h$ sorting problem, $h \geq 4r - 4$, can be solved within a number of transport steps that asymptotically differs by a factor of at most 3 from the trivial bisection bound. Furthermore we show that the bisection bound is asymptotically tight for sequences of h permutation routing problems, $h = 4cr$, $c \geq 1$, and for so-called off-line routing.

1 Introduction

In this paper we present deterministic sorting and routing algorithms on mesh-connected arrays where the number of parallel data transfers is asymptotically near to the minimum.

An $n_1 \times ... \times n_r$ mesh-connected array or grid is a set $mesh(n_1, \ldots, n_r)$ of $N = n_1 n_2 \ldots n_r$ identical processors where each processor $P = (p_1, \ldots, p_r)$, $0 \leq p_i \leq n_i - 1$, is directly connected to its nearest neighbours only. A processor $Q = (q_1, \ldots, q_r)$ is called nearest neighbour of P if and only if the distance between them is exactly 1. For a grid without wrap-around connections the distance is given by $d(P,Q) = |p_1 - q_1| + \cdots + |p_r - q_r|$. For grids with wrap-around connections (tori) we define $d_{wrap}(P,Q) = \min(|p_1 - q_1|, n_1 - |p_1 - q_1|) + \cdots + \min(|p_r - q_r|, n_r - |p_r - q_r|)$. The control structure of the grid of processors is thought to be of the MIMD type.

*This research was supported in part by the Deutsche Forschungsgemeinschaft, Grant No. Ku658/1

1.1 Sorting and Routing

For the basic $1 - 1$ sorting problem we assume that at most N elements, also called packets, from a linearly ordered set are initially loaded into the N processors, each receiving at most one element. For denoting the place into which an element has to be sorted the processors are thought to be indexed by a certain one-to-one mapping g from $mesh(n_1, \ldots, n_r)$ onto $\{0, \ldots, N-1\}$. With respect to this function the sorting problem is to move the i-th smallest element to the processor indexed by $i - 1$ for all $i = 1, \ldots, N$.

For the *multi-packet sorting problem* more than one elements are loaded into the N processors. A fundamental subproblem in this context is the $h - h$ sorting problem where each processor contains at most h packets and finally receives at most h packets. The packets are now assumed to lie in at most h disjoint layers within the processors. For the $h - h$ sorting problem h layers are assumed, numbered from 0 to $h - 1$. For a processor P and a layer j the pair (P, j) is called a place. In this case it is not sufficient to have only an indexing for the processor. The r-dimensional mesh *mesh* can now be viewed as a virtual $(r+1)$-dimensional grid *mesh* $\times$ *layer* of places supplied with a respective indexing g^*, a bijective mapping from *mesh* $\times$ *layer* onto $\{0, \ldots, hN - 1\}$. The sorting problem is now to transport i-th smallest element to the place indexed with $i - 1$. In this paper we will mainly discuss *full* $h - h$ problems where the grid contains exactly hN elements.

For the full $h - h$ *routing problem* hN elements or packets are loaded in the N processors. Each packet has a destination address specifying the processor to which it has to be sent. Each processor is destination of exactly h packets. The routing problem is to transport each packet to its address. Note that for a given indexing g one can supply each packet with one of the indices of the places concerning to destination processor of that packet. In this manner the full $h-h$ routing

problem can be viewed as $h-h$ sorting problem and therefore be solved by a sorting algorithm.

For deriving exact lower and upper bounds for sorting or routing the possible steps for computations must be described carefully. The following conventional model is used: Interchanges of data between two directly neighbouring processors and data shifts on cycles of processors are allowed. The main restriction is that during one step interval at most one packet (as an atomic unit) can be transported on each directed channel between neighbouring processors. Packets may be stored in a processor until a limited buffer is filled. If for an $h-h$ problem at most h packets may be buffered at each processor before and after each step we speak of model 1. For many of the sorting algorithms this first model is used. In all other cases we speak of model 2 (with specified buffer size $f(n) \geq 2$). In this case before and after each step each processor contains at most $f(n)$ packets.

For complexity considerations only external transport steps are counted, i.e. operations within a processor, especially between different layers, are ignored.

1.2 Previous and new results

For 2-dimensional $n \times n$ meshes without wrap-arounds several $1-1$ sorting algorithms have been proposed for the first model or can easily be transformed into algorithms which fit for that model [12,14,15]. The fastest ones need about $3n+o(n)$ steps for snake-like indexing [14,15] or blockwise snake-like indexing [12]. For such indexings the algorithms are asymptotically optimal in the first model as shown in [3,14]. In this paper we show that the above $1-1$ sorting problem can be solved deterministically in $2.5n+o(n)$ transport steps with respect to both snake-like and blockwise snake-like indexing, provided the grid has a buffer of two elements at each processor (i.e. in the second model). Recently Kaklamanis et al. [2] proposed a randomized algorithm sorting with $2.5n+o(n)$ steps with high probability. Note that this number of steps cannot be reached by any sorting (with respect to snake-like indexings) on an $n \times n$ mesh with buffer of only one packet [3,14]. That means that the lower and upper bounds heavily depend on the model in use.

Park and Balasubramanian [11] recently mentioned that on $n \times n$ meshes the optimal number of $3n+o(n)$ steps can also be obtained for $2-2$ sorting (with buffer 2), i.e. for a doubled loading. In this paper we show that for $h-h$ sorting (with buffer $h \geq 2$) sorting with respect to blockwise snake-like indexing can be done by $\lceil h/4 \rceil 2n + hn/2 + O(hn^{2/3})$ transport steps. That is, for example, $3n+O(n^{2/3})$ transport steps for $h=2$, $3.5n+O(n^{2/3})$ steps for $h=3$, and $4n+O(n^{2/3})$ transport steps for $h=4$. Moreover, these algorithms are *uniaxial*, i.e. during one time step all processors communicate along one coordinate axis only. In the next section we show that the *multi-section bound* of hn steps is valid for uniaxial sorting algorithms. That means, for all $h \equiv 0 \bmod 4$ our uniaxial sorting algorithms cannot be improved (as far as uniaxiality is demanded).

On wrap-around meshes a lower bound of $3n/2$ steps was shown for $1-1$ sorting for the first computation model (in [5]). This bound is also valid for $h-h$ sorting. The fastest $1-1$ sorting algorithms for blockwise snake-like indexing need $2n+o(n)$ steps [12]. In the third section we show that these numbers of steps can also be obtained for $2-2$ sorting. Moreover, for $h-h$ sorting (with buffer $h \geq 2$) sorting with respect to blockwise snake-like indexing can be done by $\lceil h/4 \rceil n + hn/2 + O(hn^{2/3})$ transport steps.

The best previous algorithm for $1-1$ sorting on an $n \times \cdots \times n$ cube, $r \geq 3$, needs only $(2r-1)n+o(n)$ steps for sorting [4]. This is asymptotically optimal for this type of sorting if we have a buffer of only one packet [3]. If a buffer of 2 packets is provided we can show that only $(2r-1.5)n+o(n)$ steps are sufficient. For $r=3$, for example, a $1-1$ problem can deterministically be sorted with $4.5n+o(n)$ steps. Moreover, we can show that $h-h$ sorting with a buffer of $h \geq 2$ elements can uniaxially be solved with $\lceil h/4 \rceil 2(r-1)n + hn/2 + O(hrn^{r-1/r})$ transport steps. That means, $(2r-1)n + O(n^{r-1/r})$ transport steps are sufficient for $h=2$, $(2r-0.5)n + O(n^{r-1/r})$ steps for $h=3$, and $2rn + O(n^{r-1/r})$ transport steps for $h=4$. For all $h \equiv 0 \bmod 4$ this is optimal for uniaxial algorithms. For non-uniaxial algorithms better results can be obtained: at most $\lceil h/(4(r-1)) \rceil 2(r-1)n + hn/2 + O(hrn^{r-1/r})$ transport steps are sufficient. Hence for all $h \equiv 0 \bmod (4(r-1))$ the number of transport steps asymptotically differs by a factor of only 2 from the simple *bisection bound* of $hn/2$ steps. For all other h the difference factor is at most 3. Neglecting the low order term this means that the complexity of sorting of large loadings tends to be independent from the dimension r.

All the results on $h-h$ sorting are immediately valid for full $h-h$ packet routing. Surprisingly enough they present in part so far known best solutions for different packet disciplines. For $1-1$ routing on an two-dimensional grid $2.5n+o(n)$ is the smallest known number of steps for a buffer of 2. If each buffer may contain more than 58 packets then routing can be done in the optimal number of $2n-2$ steps [10,13]. For full

$h-h$ packet routing on r-dimensional grids with large h the results of this paper are even better than those presented quite recently [7,9] where a buffer of $f(n)$ is necessary, f a strictly increasing, sublinear function.

For sequences of h permutation routing problems on an r-dimensional $n \times ... \times n$ grids, $r \geq 2$, the improvement is remarkable. The best results before were presented in [9] with $\lceil h/r \rceil 2(r-2)n + o(n)$ steps and a buffer of $f(n) > hr$ packets. In this paper we show that with the smallest possible buffer of h packets $\lceil h/(4r) \rceil 2(r-1)n + \lceil h/(2r) \rceil n + O(hrn^{r-1/r})$ transport steps are only needed. For large h this is a reduction to nearly a quarter. Moreover, for all h with $h \equiv 0 \bmod 4r$ the number of transport steps asymptotically matches the trivial bisection bound. That means that sequences of permutations present a first class of non-trivial packet routing problems on r-dimensional grids where this trivial bound can be matched. For example, on two-dimensional $n \times n$ grids 8 arbitray permutations can be solved in only $4n+o(n)$ steps. For arbitrary dimension $r \geq 3$ we get the first asymptotic match of the bisection bound with $4r$ permutations.

The above results have also immediate consequences for packet routing where the packets are no longer considered as atomic units. Let us assume that packets with a size of s bits need $\rho s + t_{set}$ time per each transport step, where ρ is a transfer rate and t_{set} is a set-up time. If the splitting of packets into smaller subpackets is allowed then the $1-1$ permutation routing problem on an r-dimensional $n \times ... \times n$ grid can be done within $\rho sn/2 + 2rnt_{set} + O(hrn^{r-1/r})$ transport time. For very small set-up time t_{set} this time is asymptocally optimal, since it is near to the theoretical lower transfer time bound of $\rho sn/2$. The best time for unsplit case is the result in [10,13] with $(2n-2)\rho s + (2n-2)t_{set}$. Compared with this approach we can obtain a reduction to nearly a quarter. With respect to the best previous split result of time $\rho sn + 2t_{set}n + o(n)$ (in [9]) we present an improvement of about 50 percent.

Furthermore, the results can be used for the so-called off-line routing (see for example [1]) where it is allowed to determine a route, as good as possible, for all packets in advance and outside the given grid. We show that $h-h$ routing on an r-dimensional grid with a buffer of h elements can be solved by an off-line algorithm with $\lceil h/(4r) \rceil 2(r-1)n + \lceil h/(2r) \rceil n + O(hrn^{r-1/r})$ transport steps for h elements. For all h with $h \equiv 0 \bmod 4r$ the number of transport steps matches asymptotically the bisection bound and is therefore optimal. This result and those mentioned before illustrate that it may be hard or even impossible to find a better lower bound than the bisection bound for general $h-h$ routing.

We also show that on meshes with wrap-around-connections for nearly all of the problems discussed above only half the numbers of steps are sufficient for the corresponding solutions.

2 Preliminaries

In this section some basic definitions, lower bounds, and results on 1-dimensional arrays are presented. We will concentrate on those grids where all sidelengths are equal, that is $n_i = n$ for all $i = 1, ..., r$.

2.1 Subgrids and indexings

As various types of intervals of processors are often dealt with a formal notation for them and some useful abbreviations are introduced:

Definition 1 (intervals of processors) *Let* $[a_i, b_i]$ *denote an interval of integers with* $0 \leq a_i \leq b_i < n$. *Then* $([a_1, b_1], \ldots, [a_i, b_i], \ldots, [a_r, b_r]) = \{(p_1, \ldots, p_i, \ldots, p_r) \mid a_i \leq p_i \leq b_i, i = 1, \ldots, r\}$. *If* $a_i = b_i = p_i$ *then* $p_i = [a_i, b_i]$ *is used. The interval* $[0, n-1]$ *is abbreviated by* $*$. *An* interval of processors along the i-th axis, *called an* i-tower, *is described by* $(p_1, \ldots, p_{i-1}, *, p_{i+1}, \ldots, p_r)$.

As special submeshes so-called *blocks* of various dimensions r are frequently used.

Definition 2 (blocks) *Let a* block parameter k *be an even integer such that* $2 \leq k^r \leq n$ *holds and* k^r *is a divisor of* n. *For this* k *an* r*-dimensional cube of processors is divided into* k^r *blocks. For arbitrary* $k_i, 0 \leq k_i \leq k-1$, $i = 1, \ldots, r$ *a* block *is defined by* $B(k_1, \ldots, k_r) = ([k_1 n/k, (k_1+1)n/k - 1], \ldots, [k_r n/k, (k_r+1)n/k - 1])$. *For a fixed* k_r, $0 \leq k_r \leq k-1$, *the union of blocks* $\bigcup_{i=1}^{r-1} \bigcup_{k_i=0}^{k-1} B(k_1, \ldots, k_i, \ldots, k_r)$ *is called the* k_r*-th* plane of blocks. *For fixed* $k_i, i = 1, \ldots, r-1$, *the set* $\bigcup_{k_r=0}^{k-1} B(k_1, \ldots, k_{r-1}, k_r)$ *is called a* column of blocks *or* tower of blocks.

That is, each block has sidelength $b = n/k$ and contains $(n/k)^r$ processors. Note that a processor $P = (p_1, \ldots, p_r)$ lies in block $B(\lfloor kp_1/n \rfloor, \ldots, \lfloor kp_r/n \rfloor)$. Obviously, there are exactly k^{r-1} columns of blocks in the whole mesh. Assume that these columns of blocks are numbered from 0 to $k^{r-1}-1$ by the snake-like lexicographical indexing (as given by the next

definition). Then let $B_{ij} = B(k_1, \ldots, k_{r-1}, k_r)$ denote that block which is in the i-th plane of blocks and in the j-th column of blocks, i.e. $i = k_r$ and $j = lex\_snake_{r-1,k}(k_1, \ldots, k_{r-1})$.

For the sorting problem several index functions have been used in the literature. In order to get more flexibility we will introduce indexings for all kinds of r-dimensional tuple cubes with different side lengths.

Definition 3 (indexings)
1. The lexicographical indexing lex *is defined by* $lex_{r,n}(p_1, p_2, \ldots, p_r) := \sum_{i=1}^{r} p_i n^{r-i}$.
The reversed lexicographical indexing rev *is given by* $rev_{r,n}(p_1, p_2, \ldots, p_r) := lex_{r,n}(p_r, \ldots, p_2, p_1)$.
2. The snake-like version of an indexing g *is given by*
$g\_snake_{r,n}(p_1, \ldots, p_r) =$
$\begin{cases} p_1 n^r + g\_snake_{r-1,n}(p_2, \ldots, p_r) & \text{even } p_1 \\ (p_1+1)n^r - 1 - g\_snake_{r-1,n}(p_2, \ldots, p_r) & \text{odd } p_1. \end{cases}$
3. For $P = (p_1, \ldots, p_r)$ *let* $k_i = \lfloor kp_1/n \rfloor$, $i = 1, \ldots, r$. *Then the* blockwise indexing $g\_block_{r,n,k}$ *with respect to indexing* $g_{r,k}$ *and* $g'_{r,n/k}$ *is given by*
$g\_block_{r,n,k}(p_1, \ldots, p_r) = (n/k)^r g_{r,k}(k_1, \ldots, k_r) + g'_{r,n/k}(p_1 \bmod (n/k), \ldots, p_r \bmod (n/k))$.
If $g_{r,k} = lex\_snake_{r,k}$ *and* $g'_{r,n/k} = lex_{r,n/k}$ *we speak of the* blockwise snake-like indexing.
4. For h layers the layer last indexing of places g_L with respect to an indexing of processors g *is defined by* $g_L(P, j) = hg(P) + j$, *and*
the layer first indexing g_F *is given by*
$g_F(P, j) = n^r j + g(P)$, $P \in$ mesh, $0 \le j \le h-1$.

The subscripts r, n, k are omitted if it is obvious which actual values are meant. For submeshes the indexings are used in the corresponding manner.

2.2 Lower bounds

Depending on the loading some trivial lower bounds can easily be derived: $h-h$ sorting and $h-h$ routing on an r-dimensional cube with sidelength n needs at least
a) $rn - r$ transport steps (*distance bound*).
b) $hn/2$ transport steps (*bisection bound*).
(See for example [9]).

Further bounds can be derived when additional, restricting properties of algorithms are taken into account. The next definition describes a property which is helpful in designing complex algorithms.

Definition 4 (uniaxial algorithm) *An algorithm is said to be* uniaxial, *if there is a function* α *from step index* ($\in \mathbb{N}$) *to* $\{1, \ldots, r\}$ *such that in clock step* j *processors* P *and* Q *may communicate iff* $P - Q = \pm u_{\alpha(j)}$ *(where* u_i *is the* i*-th unit vector). The axis* $\alpha(j)$ *is called the* active *axis at step* j.

Lemma 5 (multi-section bound) Uniaxial $h-h$ *sorting and* uniaxial $h-h$ *routing on an* r*-dimensional cube with sidelength* n *needs at least* $hrn/2$ *steps.*

The technical proof can be found in [8]. The lemma shows that the bisection bound cannot be matched by any uniaxial sorting (or routing) algorithms, provided $r \ge 2$.

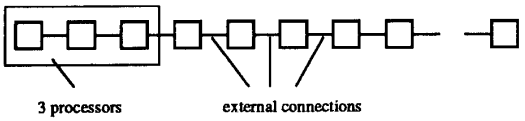
2.3 Sorting on one-dimensional arrays

For $1-1$ sorting a simple, well-known algorithm on a linear array of n processors is the *odd-even transposition sort* (*oets*). We will generalize this algorithm to $h-h$ sorting problems on linear arrays of n processors. The index of places (i, j) in processor i and layer j, $0 \le j \le h-1$, is $id_L(i, j) = ih + j$, the layer last indexing with respect to the identity. Let $cont_t(i, j)$ be the contents of layer j in processor i at time t and let $place_t(ih + j) = place_t(id_L^{-1}(i, j)) = cont_t(i, j)$. Further, let $compex_t(l, l+1)$ stand for the parallel operation
$place_t(l+1) = max(place_{t-1}(l), place_{t-1}(l+1))$
$place_t(l) = min(place_{t-1}(l), place_{t-1}(l+1))$.

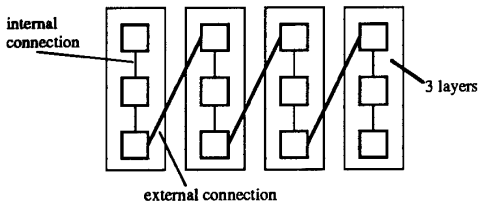
The following algorithm for h layers is called *zig-zag sort* (see figure 1). A similar algorithm for $h = 2$ was recently presented by Park and Balasubramanian [11].

Algorithm 6 Zig-Zag-Sort
For $t = 1$ **to** $hn/2$ **do**
1. **Internal (even) step** t*:*
for all $l = 1, ..., \lceil hn/2 \rceil - 1$, $0 \ne 2l \bmod h$*:*
$compex_t(2l-1, 2l)$.
2. **Internal (odd) step** t*:*
for all $l = 0, ..., \lceil hn/2 \rceil - 1$, $0 \ne (2l+1) \bmod h$*:*
$compex_t(2l, 2l+1)$.
3. **External (odd) step** t*:*
for all l *with* $2l + 1 \equiv 0 \bmod h$, $0 \le l \le \lceil hn/2 \rceil - 1$*:*
$compex_t(2l, 2l+1)$.
4. **External (even) step** t*:*
for all l *with* $2l \equiv 0 \bmod h$, $1 \le l \le \lceil hn/2 \rceil - 1$*:*
$compex_t(2l-1, 2l)$.

Lemma 7 *Sorting on a linear array of* n *processors, each one containing* h *elements,* $h \ge 2$, *can be done in the optimal number of* $hn/2$ *transport steps.*



array of 3n processors for the odd-even transposition sort



array of n processors with 3 layers for the zig-zag sort

Figure 1: Zig-Zag-Sort

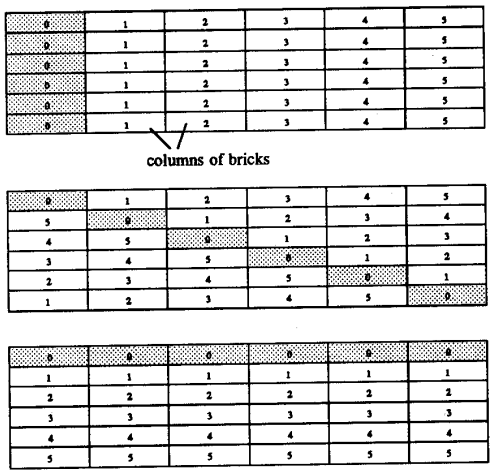


Figure 2: Column-to-row-transportation of bricks

Proof:
See figure 1. Note that the internal steps cause no external transportation and therefore cost 0 transport steps. A more detailed discussion can be found in [8]. □

3 Sorting on two-dimensional arrays

3.1 Concentrated regular data streams

As a first consequence of lemma 7 let us mention that an arbitrary full $h-h$ packet routing problem, $h \geq 2$, on a linear array of n processors can be solved in $hn/2$ steps. This is optimal for arbitrary problems. For certain regular routing problems we will show that better results can be obtained. These regular problems where the destinations of the packets can easily be calculated will be solved frequently by fast routing procedures. The procedures will also be integrated into sorting algorithms for higher-dimensional meshes. The following definitions are therefore given in the context of r-dimensional grids.

Definition 8 (bricks) *Let a, x, y be integers with x a divisor of n, a adivisor of n/x, and $0 \leq y \leq n/(ax)-1$. An $a \times n/x$ brick along the first and the u-th axis, $2 \leq u \leq r$, is an r-dimensional submesh* $\text{brick}_{1,u,y}(i,j) = ([ia+yax,(i+1)a-1+yax], I_2, \ldots, I_u, \ldots, I_r)$ *with $I_u = [jn/x,(j+1)n/x-1]$, $I_v = [0,n-1] = *$, for all $v \neq u$, $0 \leq i \leq x-1$, and $0 \leq j \leq x-1$.*

In the beginning we assume $y = 0$ and $u = 2$. (The subscripts will be omitted, since they are usually obvious by the context.) The size of the bricks will depend on the sidelength of the grid and the block parameter k. For different integers s, $1 \leq s \leq r$, we will handle bricks of shape $k^{r-1} \times n/k$, i.e. x equals the block parameter k, and $a = k^{r-1}$.

Now have a look at a $k \times k$ array of $k^{r-1} \times n/k$ bricks or, which is the same, a $k^s \times n$ array of processors. We will try to map all bricks from the same column j of bricks to row j of bricks. This can be done in several ways. Let π be any permutation on $0, \ldots, k-1$.

Algorithm 9 (Column-to-row transportation)

$\text{ctr}(k^{r-1}, n/k, \pi)$
For all $i,j = 0,\ldots,k-1$ and all $l = 0,\ldots,n/k-1$ do in parallel:
1. shift brick (i,j) onto brick $(i,(j+\pi(i)) \bmod k)$
i.e. for all processors in brick (i,j) shift contents of processor $(p_1,p_2,\ldots,p_s)$
to processor $(p_1,((p_2+\pi(i))n/k) \bmod n,\ldots,p_s)$.
2. map brick $(i,(j+\pi(i)) \bmod k)$
onto brick $(j,(j+\pi(i)) \bmod k)$
i.e. transport contents of processor $(p_1,(p_2 + \pi(i)n/k) \bmod n,\ldots,p_s)$ to processor
$((p_1 + (j-i)k^{r-1}) \bmod k^s, (p_2 + \pi(i)n/k) \bmod n,\ldots,p_s)$.

By the above algorithm *ctr* the j-th column of bricks is mapped onto the j-th row of bricks. The inverse mapping $rtc = ctr^{-1}$ is called a row-to-column mapping. Note that the number of transport steps needed for both the mappings is the same. The case where π is the identity is illustrated in figure 2. The

transport cost for at most one packet at each processor is depending on the maximal distance. For a torus connection the shift costs $n/2$ steps and the second phase k^s steps. If there are no wrap-arounds at most $n - n/k + k^s$ steps are sufficient. Note that we can easily map the j-th column of bricks onto the $\delta(j)$-th row of bricks for an arbitrary permutation δ. This can happen by an additional phase at the end of algorithm 9 where each row j is transported to row $\delta(j)$. This would cost another k^s steps. However, it is not hard to integrate that phase into the last phase of the above algorithm such that no extra cost occurs.

The above algorithm gives some freedom in selecting π. We will use this to handle two layers in sequel, layer 0 with π and layer 1 with permutation π^* with $\pi^*(i) = (k/2 - \pi(i)) \bmod k$. Let us first restrict to tori of processors.

Algorithm 10 (two-layer-ctr)
two-ctr$(k^{r-1}, n/k, \pi)$
for all $i, j = 0, \ldots, k-1$ *do in parallel:*
1. if $\pi(i) \leq k/2 - 1$ **then**
a) shift layer 0 *of brick* (i,j) *onto brick* $(i, (j + \pi(i)) \bmod k)$ *in clockwise direction.*
b) at time $\pi(i)n/k$*: shift layer* 1 *of brick* (i,j) *onto brick* $(i, (j + (k/2 - \pi(i))) \bmod k)$ *in clockwise direction.*
if $\pi(i) \geq k/2$ **then**
c) shift layer 0 *of brick* (i,j) *onto brick* $(i, (j + \pi(i)) \bmod k)$ *in counter-clockwise direction.*
d) at time $(k - \pi(i))n/k$*: shift layer* 1 *of brick*(i,j) *onto brick* $(i, (j + (k/2 - \pi(i))) \bmod k) = (i, (j - (k - (k/2 - \pi(i)))) \bmod k)$ *in counter-clockwise direction.*
2. *transport contents of layer* 0 *in brick* $(i, (j + \pi(i)) \bmod k)$ *into brick* $(j, (j + \pi(i)) \bmod k)$*, and contents of layer* 1 *in brick* $(i, (j + (k/2 - \pi(i))) \bmod k)$ *into brick* $(j, (j + (k/2 - \pi(i))) \bmod k)$.

The above algorithm costs only $n/2 + k^s$ steps on a torus. Although the above algorithm is quite busy for any given π, it does not exploit the capacity of the grid totally. Observe that in any row i with $\pi(i) \leq k/2 - 1$ all the time only connections in clockwise direction are used and the counter-clockwise direction is free whereas for any row i' with $\pi(i') \geq k/2$ the clockwise direction is free. Hence we can handle another two layers simultaneously with an adequate permutation π', exactly with $\pi'(i) = k - 1 - \pi(i)$, which behaves absolutely complementary to π with respect to channel requirements.

Algorithm 11 (four-layer-ctr)
four-ctr$(k^{r-1}, n/k, \pi)$
Do in parallel: two-ctr$(k^{r-1}, n/k, \pi)$ *and*
two-ctr$(k^{r-1}, n/k, k - 1 - \pi)$

Note that the above method allows us to distribute simultaneously four layers of elements contained in k columns of bricks to corresponding k rows of bricks. On a torus both mappings can be performed uniaxially within $n/2 + 2k^s$ transport steps. Furthermore, the inverse operation *four-rtc* = *four-ctr*$^{-1}$ is easily performed in the same number of steps.

Lemma 12 *Four-layer-ctr* four-ctr *and its inverse* four-rtc *on a* $k \times k$ *array of* $k^{s-1} \times n/k$ *bricks can uniaxially be performed in*
a) $n/2 + 2k^s$ *transport steps on a torus, and*
b) $n + 2k^s$ *transport steps on a grid without wrap-arounds.*

For tori of processors the lemma is obvious by the above discussion. The technical proof for grids without wrap-arounds can be found in [8].

3.2 Multi-packet sorting

We will now show that the highly concentrated data streams of the four-layer column-to-row transportation algorithm are very useful for $h - h$ sorting with $h \geq 2$. The sorting procedures will use sorting of $n^{2/3} \times n^{2/3}$ blocks, i.e. the block parameter is $k = n^{1/3}$. The structure of the algorithm is similar to those presented for $1 - 1$ sorting [12,14,15]. The indexing used for the whole sorting algorithm is layer last blockwise snake-like indexing.

Algorithm 13 (h-layer-sort)
sort(n, n, h)
1. Sort all $n^{2/3} \times n^{2/3}$ *blocks with respect to* rev_F *(layer first reversed lexicographical indexing).*
2. In all $n^{1/3} \times n$ *submeshes* $([jn^{1/3}, (j+1)n^{1/3} - 1], *)$*,* $O \leq j \leq n^{2/3}$*, perform a column-to-row mapping of the corresponding* $1 \times n^{2/3}$ *bricks.*
3. Sort all columns of blocks downwards with respect to layer last lexicographical indexing lex_L *(for* $n \times n^{2/3}$ *meshes).*
4. In all $n^{2/3} \times n$ *submeshes* $([jn^{2/3}, (j+1)n^{2/3} - 1], *)$*,* $0 \leq j \leq n^{1/3} - 1$*, perform a row-to-column mapping of the corresponding* $n^{1/3} \times n^{2/3}$ *bricks such that neighbouring rows of bricks are mapped onto neighbouring blocks in the snake-like indexing of the blocks.*
5. Sort with respect to layer last blockwise snake-like indexing all neighbouring pairs of blocks
a) with block indices $2i$ *and* $2i + 1$*,* $i = 0, \ldots, k^2/2$*,*
b) with block indices $2i - 1$ *and* $2i$*,* $i = 1, \ldots, k^2/2$.

Theorem 14 *On a two-dimensional $n \times n$ grid with a buffer of h elements, $h \geq 2$, the $h-h$ sorting problem (with respect to blockwise snake-like lexicographical indexing) can uniaxially be solved with*
a) $3n + O(n^{2/3})$ *transport steps for* $h = 2$.
b) $3.5n + O(n^{2/3})$ *transport steps for* $h = 3$.
c) $\lceil h/4 \rceil 2n + hn/2 + O(hn^{2/3})$ *steps for* $h \geq 4$.
d) For all $h \equiv 0 \bmod 4$ *the number of steps asymptotically matches the multi-section bound.*

The lengthy proof is omitted here and can be found in [8].

The algorithm can be altered for the pure snake-like indexing easily. Instead of the phases 4 and 5 the altered algorithm has only to perform $hn/2 + O(n^{2/3})$ steps of the zig-zag sort along the total snake of the mesh. This can be done uniaxially in $hn/2 + O(n^{2/3}) +$ 2 steps.

Corollary 15 *$h-h$ sorting with respect to snake-like lexicographical indexing can uniaxially be solved with*
1. $3n + O(n^{2/3})$ *transport steps for* $h = 2$.
2. $4n + O(n^{2/3})$ *transport steps for* $h = 3$.
3. $\lceil h/4 \rceil n + hn + O(hn^{2/3})$ *transport steps for* $h \geq 4$.

For $h = 2$ the above result and the corresponding result in corollary 16 have also been reported by Park and Balasubramanian [11].

It is not hard to see that algorithm 13 can be extended to arbitrary $n_1 \times n_2$ grids.

Corollary 16 *On a two-dimensional $n_1 \times n_2$ grid with a buffer of h elements, $h \geq 2$, the $h-h$ sorting problem can uniaxially be solved with*
1. $(\lceil h/4 \rceil + h/2)n_2 + hn_1/2 + O(hn^{1/2})$ *transport steps with respect to snake-like lexicographical indexing, and*
2. $\lceil h/4 \rceil 2n_2 + hn_1/2 + O(hn^{1/2})$ *transport steps with respect to blockwise snake-like lexicographical indexing.*

3.3 2.5n sorting on an $n \times n$ array

The highly concentrated stream of data in the row-to-column mapping only promises benefits if there are more than two elements at each processor. At a first glance for $1-1$ sorting we can not exploit this advantage. However, by a kind of data concentration we transform the $1-1$ problem into a $2-2$ subproblem. This will be done by the next algorithm.

Algorithm 17 (center-concentration)
$\mathrm{conc}([0, n-1], 1/2)$
For all $i = 0, \ldots, n-1$ *do in parallel:*
transport packet from place $(i, 0)$
to place $(n/4 + \lfloor i/2 \rfloor, i \bmod 2)$.

It is not hard to see that the operation costs $n/4$ transport steps. Furthermore, note that the inverse operation $expand = conc^{-1}$ needs the same number of steps.

The basic structure of the following $1-1$ sorting algorithm is similiar to those $3n + o(n)$ algorithms in [14,15]. The difference lies mainly in an additional prephase (concentration of data), a postphase (expansion of data), and the application of the zig-zag sort on columns and rows with two layers instead of odd-even transposion sort sorting linear arrays with only one layer.

Algorithm 18
1. In all rows $(p_1, [0, n-1])$, $p_1 = 0, \ldots, n-1$, *concentrate elements with* $\mathrm{conc}((p_1, [0, n-1]), 1/2)$
2. Sort the $n \times n/2$ *grid* $(*, [n/4, 3n/4 - 1])$ *with two layers by the corresponding algorithm* $\mathrm{sort}(n_1, n_2, 2)$ *for snake-like lexicographical indexing*
3. Expand all rows.

Theorem 19 *On an $n \times n$ grid with a buffer for two elements $1-1$ sorting with respect to snake-like lexicographical indexing can uniaxially be done with* $2.5n + O(n^{2/3})$ *transport steps .*

Proof:
Note that the concentration and expansion only cost $n/4 + n/4 = n/2$ steps. The sorting of the $n \times n/2$ grid with two layers in the middle of the whole mesh needs $2n + O(n^{2/3})$ steps as stated in corollary 16. □

Note that the above result could only be obtained, because more than one element can be stored at each processor and processors may be empty during the sorting process. For $1-1$ sorting with respect to snake-like lexicographical indexing where after each step exactly one element is contained in each processor $3n - o(n)$ is still a lower bound [3,5,14].

4 Sorting on r-dimensional grids, $r \geq 3$

In this section we will give a unified approach for sorting, valid for grids with dimension $r \geq 3$.

For the r-dimensional case we need a generalization of the column-to-row mapping. It is called block-to-plane transportation, since it maps r-dimensional $n/k \times \ldots \times n/k$ blocks onto $(r-1)$-dimensional $n \times \ldots \times n$ hyperplanes. For this purpose we will handle $k^{s-1} \times n/k$ bricks, $1 \leq s \leq r$. That means, operations take place on a $k \times k$ array of $k^{s-1} \times n/k$ bricks. Let $k = n^{1/r}$, i.e. $n/k = n^{1-1/r}$.

Algorithm 20 (block-to-plane-transportation)

For $s = r-1$ **downto** *1* **do begin**
For all $k^{s-1} \times n \times ... \times n$ *submeshes*
$([jk^s, (j+1)k^s - 1], *, ..., *)$, $0 \le j \le n/k^s - 1$,
do a column-to-row mapping along the axes 1 *and* $r+1-s$ *on* $k \times k$ *arrays of* $k^{s-1} \times n/k$ *bricks.*
end

Lemma 21 *The block-to-plane algorithm uniaxially transports the contents of each r-dimensional* $n^{1-1/r} \times ... \times n^{1-1/r}$ *block* $\text{block}(k_1, ..., k_r)$ *into an* $(r-1)$ *dimensional plane* $(lex(k_1, ..., k_r), *, ..., *)$
within $(r-1)n + O(k^{r-1})$ *steps.*

For $r = 2$ the lemma is already shown in the last section. For $r \ge 3$ the technical proof is omitted here. A similar technique has already been presented in [4]. Note that the mapping of blocks onto planes $(lex\_snake(k_1, ..., k_r), *, ..., *)$ and the corresponding inverse plane-to-block transportation can be done with the same number of steps.

Algorithm 22 (r-dimensional sorting)
1. Sort all $n^{r-1/r} \times ... \times n^{r-1/r}$ *blocks with respect to layer first reversed lexicographical indexing* rev_F.
2. In all $n^{r-1/r} \times n ... \times n$ *submeshes* $([jn^{r-1/r}, (j+1)n^{r-1/r} - 1], *, ..., *)$, $0 \le j \le n^{1/r} - 1$, *perform a block-to-plane mapping.*
3. Sort all $n \times n^{r-1/r} \times ... \times n^{r-1/r}$ *towers of blocks downwards (with respect to layer last lexicographical indexing* lex_L*).*
4. In all $n^{r-1/r} \times n \times ... \times n$ *submeshes* $([jn^{r-1/r}, (j+1)n^{r-1/r} - 1], *, ..., *)$ *perform a plane-to-block mapping such that neighbouring planes* $(p_1, *, ..., *)$ *are mapped onto neighbouring blocks in the snake-like indexing of the blocks.*
5. Sort (with respect to layer last blockwise snake-like indexing) all neighbouring pairs blocks
a) with block indices $2i$ *and* $2i+1$, $i = 0, ..., k^r/2 - 1$,
b) with block indices $2i-1$ *and* $2i$, $i = 1, ..., k^r/2 - 1$.

With the above algorithm we can obtain solutions for several $h-h$ sorting problems.

Theorem 23 $h-h$ *sorting on an r-dimensional* $n \times n \times ... \times n$ *grid with a buffer of* $max(2, h)$ *elements can uniaxially be done with*
$\lceil h/4 \rceil 2(r-1)n + hn/2 + O(hrn^{r-1/r})$ *transport steps. For uniaxial algorithms the number of transport steps is asymptotically optimal for all h with* $h \equiv 0 \bmod 4$.

The proof is omitted here and can be found in [8].

It is not hard to see that the presented methods can be extended to r-dimensional $n_1 \times n_2 \times \cdots \times n_r$ meshes. Neglecting low order terms the number of steps needed by the corresponding algorithm is $hn_1/2 + \lceil h/4 \rceil 2(n_2 + ... + n_r)$ on meshes without wrap-arounds, $h \ge 2$

For the $1-1$ problem one can us the same trick of concentrating the elements in the center. This leads to a $2-2$ problem on an $n/2 \times n \times ... \times n$ grid. By this approach the $1-1$ problem can be solved within $(2r - 1.5)n + O(rn^{r-1/r})$ steps.

In the case of wrap-around connections the number of transport steps can be reduced to nearly the half. The reason for this lies in the fact that block-to-plane transportation can be done on tori in half the step number. Halving the number of steps for sorting on one-dimensional rings is more difficult. However, if there are enough layers then we can use a sorting procedure which can sort h layers in $n \times n/k$ stripes in about $\lceil h/4 \rceil n$ steps. This procedure consists of a column-to-block algorithm, sorting of the blocks, and a final block-to-column transportation. Further details will appear in the full paper. Here we only state the result without proof.

Theorem 24 $h-h$ *sorting on an r-dimensional torus can uniaxially be done with*
$\lceil h/4 \rceil rn + O(hrn^{r-1/r})$ *transport steps.*

5 Packet routing and sorting

5.1 Matching the bisection bound

A helpful tool for exploiting the abilities of a network of processors is that of orthogonal overlapping of uniaxial algorithms as demonstrated in [7,9]. Note that all the algorithms presented so far in this paper are uniaxial. For the following overlapped, non-uniaxial algorithms we introduce a kind of tuple rotation: The *i-th rotation of an indexing g* is given by $g\_rot_i(p_1, ..., p_r) := g(p_{i+1}, ..., p_r, p_1, ..., p_i)$.

The overlapped sorting algorithms consist of at most r incarnations of the same basic uniaxial algorithm $SORT_1$. Let $\alpha_1(t)$ be the active axis of $SORT_1$ at step t. Then let $\alpha_j(t) = (\alpha_1(t) + j - 2) \bmod r + 1$ be the active axis of the j-th *incarnation* $SORT_j$, $1 \le j \le r$. Now colour all elements which lie in the beginning in layer i, $0 \le i \le h-1$, with *colour* $i \bmod r$. Let the j-th incarnation SORT_j handle all elements of colour $j-1$ (i.e. the colour-$(j-1)$ subproblem). For incarnation j the corresponding indexings are $(j-1)$-th rotations of those indexings as used for the first incarnation.

Algorithm 25 (Overlapped Sorting)
r-overlapped-sort(*mesh*)
for all $j = 1, \ldots, r$ *do in parallel: apply* Sort$_j$ *to the colour-*$(j-1)$ *subproblem.*

Since for a sorting problem all elements in the mesh are to be sorted with respect to a universal indexing valid for all layers in the mesh, this overlapped approach can not be used directly for sorting, because different incarnations of the sorting algorithm use different rotated indexings. However, the overlapping technique can at once be applied for natural subclasses of the full $h-h$ packet routing problem, namely for sequences of h permutations.

Theorem 26 *On an r-dimensional* $n \times \ldots \times n$ *grid with a buffer of h elements a sequence of h permutations can (non-uniaxially) be routed with* $\lceil h/(4r) \rceil 2(r-1)n + \lceil h/(2r) \rceil n + O(hrn^{r-1/r})$ *steps. For all* $h \equiv 0 \bmod 4r$ *the number of transport steps asymptotically matches the bisection bound.*

Proof:
Apply algorithm 25 to the problem. Note that each colour-j subproblem consists of at most $\lceil h/r \rceil$ layers, each layer containing a permutation problem. The result now follows immediately by theorem 23. □

5.2 Packet Splitting

Until now we considered packets as atomic units. The time needed for a transport step was just a unit, in our case just 1. However, if the packets have a size of s bits then it seems to be a realistic assumption that the transport of a packet needs $\rho s + t_{set}$ time, where ρ is a transfer rate (depending on the technical abilities of the channels) and t_{set} is a set-up time needed by the processor to prepare a packet for the transfer. If the splitting of packets into smaller subpackets is allowed then we obtain the following theorem.

Theorem 27 *Permutation routing on an r-dimensional* $n \times \ldots \times n$ *grid can be done within* $\rho sn/2 + 2rnt_{set} + O(hrn^{r-1/r})$ *transport time. For very small* t_{set} *this time is asymptotically optimal.*

Proof:
Split each packet into $4r$ subpackets, each of size $s/(4r)$. Let these subpackets lie in layers $0, \ldots, 4r-1$. Note that we now have a sequence of $4r$ full permutation routing problems. By theorem 26 this problem can be solved with $2rn + O(hrn^{r-1/r})$ transport steps, each one consuming $\rho s/(4r) + t_{set}$ time. Thus $\rho sn/2 + 2rnt_{set} + O(hrn^{r-1/r})$ time is needed in total. Furthermore, we can apply a bisection argument for the lower bound. If we have a problem where $n^r/2$ packets of size s have to be transported through n^{r-1} channels with transfer rate ρ, then we need at least $(sn^r/2) \;/\; (n^{r-1}/\rho) = \rho sn/2$ time (only to transfer $sn^r/2$ bits without any administration overhead). Thus for a very small set-up time we match this lower transfer bound. □

By the above result all previous results have been improved. The best time for unsplit case on an $n \times n$ grid is the result in [10,13] with $(2n-2)\rho s + (2n-2)t_{set}$. Compared with this approach we can obtain a reduction to nearly a quarter. With respect to the best previous split result of time $\rho sn + 2t_{set}n + o(n)$ (in [9]) we presented an improvement of about 50 percent. Furthermore, neglecting the set-up times the result of theorem 27 is nearly independent of the dimension r. Finally, note that theorem 27 demonstrates that splitting the packets into more than $4r$ subpackets will not lead to a further improvement.

5.3 Overlapping and $h-h$ sorting

The overlapping technique can also be used for the the general $h-h$ sorting on meshes with dimension $r \geq 3$. Note that the block-to-plane transportation and its inverse are uniaxial algorithms working along $r-1$ axes in this case. That means that we can take $r-1$ incarnations of these procedures to transport the contents of $(r-1)h$ layers in the same number of steps as h layers before. Following this approach we can state

Theorem 28 $h-h$ *sorting,* $h \geq 2$, *on an r-dimensional grid,* $r \geq 3$, *with a buffer of h elements can be solved by an overlapped algorithm with at most* $\lceil h/(4(r-1)) \rceil 2(r-1)n + hn/2 + O(hrn^{r-1/r})$ *steps.*

Hence for all $h \equiv 0 \bmod (4(r-1))$ the number of transport steps asymptotically differs by a factor of only 2 from the simple *bisection bound* of $hn/2$ steps. For all other h the difference factor is at most 3. Neglecting the low order term this means that the complexity of sorting of large loadings tends to be independent from the dimension r.

5.4 Off-line routing

The results discussed above can also be used for the so-called off-line routing (see for example [1]) where it is allowed to determine a route, as good as possible, for all packets in advance and outside the given grid.

Theorem 29 *On an r-dimensional grid with a buffer of h elements $h-h$ routing can be solved off-line in $\lceil h/(4r)\rceil 2(r-1)n + (\lceil h/r\rceil/2)n + O(hrn^{r-1/r})$ steps. For all h with $h \equiv 0 \bmod 4r$ the number of transport steps matches asymptotically the bisection bound.*

Proof:
As demonstrated in [1] it is possible to reorganize the elements within the processors in such a way that each layer contains a full permutation routing problem. Thus our theorem follows from theorem 26. □

It is still open whether the bisection bound can be matched for off-line routing with less than $4r$ layers. For example, for 2-dimensional $n \times n$ meshes the general result is $\lceil h/8\rceil 2n + \lceil h/4\rceil n + O(n^{2/3})$. I.e., the best result for 4 layers is $2n + n = 3n$ steps whereas in this case the bisection bound is with $2n$ steps (the same as the distance bound).

6 Conclusion

In this paper we discussed deterministic sorting and routing on r-dimensional meshes. For two-dimensional meshes a $2.5n + o(n)$ sorting algorithm for the classical $1-1$ problem was presented. For the general $h-h$ sorting problem it was shown that sorting asymptotically needs at most three times as many steps as the bisection bound. We showed that routing sequences of h permutations presents a first natural class of non-trivial routing problems where the trivial bisection bound can be matched. Moreover, it was shown that theoretical lower transfer time bounds may asymptotically be matched for permutation routing, provided splitting of packets is allowed and set-up times are neglectible. Furthermore, for many cases of the $h-h$ routing problem off-line solutions were presented which also match the bisection bound.

It should be mention that in the meantime some further improvement could be obtained. Optimal solutions can be obtained not only for the cases with $h \equiv 0 \bmod 4r$ (as shown in this paper), but also for all h with $h = cr, c \geq 4$. This has been shown in [8]. There the reader may also find all the proofs which had to be omitted in this paper because of the lack of space.

The results of this paper could be obtained by the combination of several methods: Obtaining local information (by sorting of blocks), distributing this information uniformly in concentrated regular streams of packets, and overlapping of uniaxial algorithms. It might be interesting to analysize, whether these techniques are helpfull for other fixed size networks.

References

[1] Annexstein, F.and Baumslag, M. A unified approach to off-line permutation routing on parallel networks. Proc. of SPAA 90. Iland of Crete, 1990, pp. 398–406.

[2] Kaklamanis, C., Krizanc, D., Narayanan, L. and Tsantilas, T. Randomized Sorting and Selection on Mesh-Connected Processor Arrays. Proceedings of SPAA 91. Hilton Head, South Carolina, 1991.

[3] Kunde, M. Lower bounds for sorting on mesh-connected architectures. Acta Informatica, **24** (1987), pp. 121–130.

[4] Kunde, M. Routing and Sorting on Mesh-Connected Arrays. In J.H.Reif (ed.). Proceedings of the 3rd AWOC 88, Lect. Notes Comp. Sci. 319, Springer, Berlin, 1988, pp. 423–433.

[5] Kunde, M. Bounds for for l-Selection and Related Problems on Grids of Processors. J. of New Generation Computer Systems, **2** (1989), pp. 129–143.

[6] Kunde, M. Packet Routing on Grids of Processors. In Djidjev (ed.), Optimal Algorithms, Proceedings, Lect. Notes Comp. Sci., vol. 401. Springer, Berlin, 1989, pp. 254-265.

[7] Kunde, M. Balanced Routing: Towards the Distance Bound on Grids. Proceedings of SPAA 91. Hilton Head, South Carolina, 1991.

[8] Kunde, M. Routing and Sorting on Grids. Habilitationsschrift. TU Munich, Munich, 1991, 177 pp..

[9] Kunde, M. and Tensi, T. k-k routing on multidimensional mesh-connected arrays. J. of Parallel and Distributed Computing, **11** (1991), pp. 146-155

[10] Leighton, T., Makedon, F. and Tollis, I. A 2n-2 Step Algorithm for Routing in an n x n Array with Constant Size Queues. Proc. of SPAA 89. Santa Fe, 1989, pp. 328–335.

[11] Park, A. and Balasubramanian, K. Reducing Communication Costs for Sorting on Mesh-Connected and Linearly Connected Parallel Computers. J. of Parallel and Distributed Computing, **9** (1990), pp. 318–322

[12] Ma, Y., Sen, S. and Scherson, I.D. The distance bound for sorting on mesh-connected processor arrays is tight. Proceedings FOCS 86, pp. 255–263.

[13] Rajasekaran, S. and Overholt, R. Constant Queue Routing on a Mesh. In Choffrut, C. and Jantzen, M. (eds.), Proceedings of STACS 91,Lect. Notes Comp. Sci., vol. 480, Springer, Berlin, 1991, pp. 444–455.

[14] Schnorr, C.P. and Shamir, A. An optimal sorting algorithm for mesh-connected computers. Proceedings STOC 1986. Berkley, 1986, pp. 255–263.

[15] Thompson, C.D. and Kung, H.T. Sorting on a mesh-connected parallel computer. CACM **20** (1977), pp. 263–270.