Overview of Mesh Results

Jop F. Sibeyn
Max-Planck-Institut für Informatik
Im Stadtwald
66123 Saarbrücken
Germany.
Email: jopsi@mpi-sb.mpg.de.

Abstract

This paper provides an overview of lower and upper bounds for algorithms for mesh-connected processor networks. Most of our attention goes to routing and sorting problems, but other problems are mentioned as well. Results from 1977 to 1995 are covered. We provide numerous results, references and open problems. The text is completed with an index.

This is a worked-out version of the author's contribution to a joint paper with Miltos D. Grammatikakis, D. Frank Hsu and Miro Kraetzl on multicomputer routing, submitted to the Journal of Parallel and Distributed Computing.

Contents

1 Mesh-Connected Multicomputers

A two-dimensional $n \times n$ **mesh** consists of $N = n^2$ processors (PUs) arranged in a two-dimensional $n \times n$ grid. Each PU is connected to its (at most) four neighbors.

Meshes and their direct generalizations are attractive because of their regularity, scalability and conceptual simplicity. Therefore, they are very easy to program and to construct in VLSI. Their major draw-backs are their large diameter and small bisection width: $2 \cdot n - 2$ and n, respectively, for two-dimensional meshes. The latter causes n to be the maximum speed-up that can be acheived with n^2 PUs in problems without much 'locality'.

The PUs can be indicated by giving their coordinates within the mesh; the PU at position (i, j), $0 \leq i, j < n$, is denoted $P_{i,j}$. Here position $(0, 0)$ lies in the upper-left corner. In sorting problems the time also depends on the indexing scheme of the PUs. An **indexing scheme** is a bijection $I : \{0, 1, \ldots, n-1\}^2 \to \{0, 1, \ldots, n^2 - 1\}$, that attributes to every PU a unique index. In the common **row-major indexing** $P_{i,j}$ has index $i \cdot n + j$. In the **column-major indexing** $P_{i,j}$ has index $i + j \cdot n$. In the **snake-like row-major indexing** the indexing of the odd rows is reversed. In a **shuffled row-major indexing** $P_{i,j}$, with $i = \sum_{i=0}^{\log n-1} i_k \cdot 2^k$ and $j = \sum_{k=0}^{\log n-1} j_k \cdot 2^k$, has index $I(i, j) = \sum_{k=0}^{\log n-1} (2 \cdot i_k + j_k) \cdot 2^{2 \cdot k}$. These indexing schemes are illustrated in Figure 1.

0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

0	4	8	12
1	5	9	13
2	6	10	14
3	7	11	15

0	1	2	3
7	6	5	4
8	9	10	11
15	14	13	12

0	1	4	5
2	3	6	7
8	9	12	13
10	11	14	15

Figure 1: Indexings of a 4×4 mesh. From left to right: row-major, column-major, snake-like row-major, shuffled row-major indexing.

Several routing models can be distinguished. In the **SIMD model**, in every routing step there is one direction (leftwards, upwards, rightwards or downwards) in which the PUs can send: every PU sends and receives at most one packet. In the **uni-axial model** in every routing step there is one axis along which the communication is performed (horizontal or vertical). Along this axis each PU can send to both its neighbors and receive from them. In the **MIMD model** (also called **bi-axial model** or **full-port model**) the PUs can communicate with all their neighbors in a single step. In any case the amount of information that can be sent over a connection in a single step is limited to one packet plus some information necessary for guiding it to its destination ($\mathcal{O}(\log n)$ bits).

All definitions carry over to non-square $m \times n$ meshes. A **torus** is a mesh with so-called **wrap-around connections**, which connect $P_{i,0}$ with $P_{i,n-1}$ and $P_{0,j}$ with $P_{n-1,j}$. Tori have the advantage that they are uniform: there are no corners that may have to be treated in a special way. The diameter of a torus is only half the diameter of the corresponding mesh, and their bisection width is twice as large. Therefore, on tori most problems can be solved twice as fast. A torus can be embedded in a mesh with dilation 2, see

0	9	1	8	2	7	3	6	4	5

Figure 2: The embedding of a circular array (one-dimensional torus) of length 10 into a linear array (one-dimensional mesh). The dilation is 2. The embedding of a torus into a mesh with dilation 2 is realized analogously, by 'folding' in both directions.

Figure 2. This embedding enables us to run any torus algorithm on a mesh with delay factor 2 (except for the SIMD model). So, any optimal torus algorithm, that is an algorithm matching the distance or bisection bound, automatically yields an optimal mesh algorithm. The opposite is not true: there are

problems that cannot be solved twice as fast on a torus as on a mesh. 1-1 sorting on one-dimensional arrays is the simplest example [64]. A **d-dimensional $n \times \cdots \times n$ mesh** consists of $N = n^d$ arranged in a d-dimensional cube. The **d-dimensional torus** is analogous, with wrap-around connections.

For problems on meshes and tori, we are not only interested in the **routing time** of an algorithm, the maximum required number of steps, but just as much in its **queue size**, the maximum number of packets that may have to be stored in a single PU during the execution of the algorithm. The queue size is of importance because of physical limitations, and because we assume that the computation time is negligible in comparison with the routing time. This assumption is justified only if the number of computation steps is small. If the queue is long, then selecting the right element from it may require a lot of computation.

2 Routing on Meshes

In this section we consider several variants for the **routing problem** on meshes. In the routing problem every PU holds a certain number of packets equipped with the index of a destination PU. These packets must be routed to their destinations assuring a small routing time and small queues. We will not attempt to list all results for all variants and all models. Most mesh results carry over to tori and MIMD results carry over to SIMD with the due time factors between them.

The theory on routing is rich and next to the various routing models, there are several other distinctions. In **off-line routing** , the source/destination distribution of the packets is globally known, and an optimal routing schedule can be precomputed. This in contrast with **on-line routing** , in which initially a PU only knows the destinations of its own packets. On-line routing is further distinguished in **adaptive routing**, in which the PUs may decide how to route the packets based on the information that is gathered during the routing, and **oblivious routing**, in which the track of a packet is completely determined once it is sent away. Oblivious routing has the advantage that the decisions during the routing are simpler: a PU only has to decide which packet has to be sent first over a connection, but not over which connection to send them. A disadvantage is its lack of flexibility, and therefore oblivious routing cannot guarantee good performance in many cases. In [44] Krizanc proofs $\Omega(n^2)$ lower-bounds for a class of oblivious permutation routing algorithms.

2.1 1-1 Routing

Theoretically the most interesting variant is the **permutation-routing problem** , in which every PU is the source and destination of exactly one packet. It has attracted a considerable amount of attention. Ideally a permutation should be routed in a time close to the distance bound, and with maximum queue size equal to a small constant.

SIMD Meshes. In the SIMD model, $4 \cdot n - 4$ steps is a lower bound for almost any problem. For permutation routing, this lower-bound is matched by the simple **greedy routing strategy** , in which all packets are first sent along the rows to their destination columns and then along the columns to their destinations. The routing time is optimal, but if all packets from a certain row have destination in the same column, they are all wiped into a single PU, yielding queues of size $\Omega(n)$.

For the general routing problem in the SIMD model there is not much known except for simulating the uni-axial or MIMD algorithms with constant delay. The best results are obtained by simulating the algorithms of Kunde [47] (deterministic) or the one by Rajasekaran and Tsantillas [89] (randomized):

Lemma 1 *Permutation routing on an SIMD mesh can be performed in $4 \cdot n + \mathcal{O}(n/q)$ steps and with maximum queue size q [47, 89].*

There is an interesting subset of the permutations which can be routed faster and with smaller queues: the **bit-oriented permutations**. These are the permutations like the transition from a shuffled row-major indexing to a column-major indexing. Generally, these are the permutations under which the packet from P_i has to be routed to some $P_{i'}$, with i' obtained by permuting and inverting the bits of i.

For a formal definition see [76]. The set of permutations can be extended to the **affine permutations**, under which i', the index of the PU to which P_i routes its packet, is given by $\overline{i'} = A \cdot \overline{i} + \overline{b}$. Here $\overline{i}$ and $\overline{i'}$ denote the binary expansion of i and i', respectively, A is an invertible 0-1 matrix and $\overline{b}$ a 0-1 vector.

Theorem 1 *Affine permutations can be routed on an SIMD mesh in* $4 \cdot n - 4$ *steps and with maximum queue size 2 [76, 97].*

Notice that here we have an instance of an off-line routing problem. For affine permutations it is not unrealistic to assume that the permutation is globally known as they typically arise in the context of standard transformations, and because they can be encoded with just $\mathcal{O}(\log^2 n)$ bits. The precomputation (essentially matrix inversion) takes $\mathcal{O}(\log^3 n)$ computation time.

MIMD Meshes. Now we consider permutation routing on MIMD meshes. $2 \cdot n - 2$ steps is a lower bound because of the diameter of the mesh.

Leighton has shown that for random inputs the greedy algorithm performs very well, even the queue sizes remain small [56]. Achieving good results for all permutations is harder. The first near-optimal algorithms were presented in [47, 89]. Then Leighton, Makedon and Tollis [58] proved 'the impossible': a deterministic algorithm running in $2 \cdot n - 2$ steps with constant size queues. Though constant, the queue size was still very large (about 1000). In a number of later papers it was reduced to almost practical values [87, 12, 96]. The essential idea in all papers since [89] is that **critical packets**, those that move between opposite corner regions of the mesh, are treated in a special way so that they do not incur any delay.

Theorem 2 *Permutation routing on an MIMD mesh can be performed in* $2 \cdot n - 2$ *steps and with maximum queue size* 32 *[58, 87, 87, 12, 96].*

Accepting some extra steps, the queue size can be reduced to a much smaller value:

Lemma 2 *Permutation routing on an MIMD mesh can be performed in* $2 \cdot n + \mathcal{O}(1)$ *steps and with maximum queue size* 2 *[12, 96, 35].*

Theoretically this is an interesting result, but the constant hidden in the $\mathcal{O}(1)$ is large. More practical are algorithms with routing time below $3 \cdot n$ and queue size 5 or 6 [35].

In the light of Theorem 1 and Theorem 2 it is interesting to consider

Problem 1 *Can all permutations be routed on an SIMD mesh with* $4 \cdot n - 4$ *steps and constant queue size?*

It would be desirable to have algorithms that are almost as simple as greedy routers, but have a much better worst-case performance with respect to time and queue size. An interesting candidate family of such algorithms are the so-called **minimal-adaptive algorithms**, in which packets are allowed to choose a suitable path depending on the presence of other packets, but are still routed along a shortest path. Chinn, Leighton and Tompa analyze minimal-adaptive routing of permutations on $n \times n$ meshes, with maximal queue size Q [11]. They prove a lower bound of $\mathcal{O}(n^2/Q^2)$ for a large class of such algorithms. On the other hand, they present a minimal-adaptive algorithm that solves the problem in $\mathcal{O}(n)$ steps with constant Q.

The off-line permutation-routing problem on MIMD meshes has been considered by Kaklamanis, Krizanc and Rao [32]. They present a $2 \cdot n - 1$ algorithm with queue size 4. Smaller queues are obtained by considering the $n \times n$ mesh as a Cartesian product of two n-node 1-dimensional arrays:

Theorem 3 *Any permutation can be off-line routed on a uni-axial mesh in* $3 \cdot n - 3$ *steps with maximal queue size one [1].*

Proof: The optimal schedule can be found by solving the edge coloring problem of an n-regular bipartite graph with $2 \cdot n$ nodes. According to Hall's matching theorem (see [10, Ch. 12]) a coloring with n colors exists, and using the algorithm of Cole and Hopcroft [15], it can be constructed in $\mathcal{O}(n^2 \log n)$. □

For bit-oriented or affine permutations one can apply the algorithms of [76] or [97], running in $2 \cdot n - 2$ steps, with queue size one or two, respectively. These routing schedules can be computed much faster.

Hot-potato routing is a paradigm in which packets are never stored, but continue to be sent (possibly in a wrong direction) until they ultimately reach their destinations. Several researchers have considered the problem. For average-case instances $2 \cdot n + \mathcal{O}(\log n)$ steps or fewer suffice, as is shown in [98] (incomplete analysis) and [17] (complete analysis). In the same paper Feige and Raghavan also present a randomized algorithm with linear running time. Schuster and Newman [77] were the first to present a deterministic linear-time algorithm. This algorithm was improved by Kaufmann, Lauer and Schröder [34].

Lemma 3 *Hot-potato permutation routing on an MIMD mesh can be performed in* $7/2 \cdot n + o(n)$ *steps [77, 34].*

In [8] Ben-Aroya and Schuster analyze greedy hot-potato routing. Their goal is to route packets along the shortest possible path. It is shown that when there are at most k packets to route and no packet has to go further than D steps, then the routing can be performed in $2 \cdot (k-1) + D$ steps. Notice that this result is only interesting for a k which is very small in comparison to the number of PUs of the network. For the special case in which $d = 3$, the routing is performed in 7 steps. This gives a partial answer to the following question:

Problem 2 *Can hot-potato routing problems in which all packets have to go at most D steps be solved in $\mathcal{O}(D)$ steps?*

2.2 k-k Routing

Practically more relevant than 1-1 routing is k-k **routing** . Here every PU is the source and destination of at most k packets. It requires at least $k \cdot n/2$ steps, because of the bisection bound of the mesh. On a torus the lower bound is $k \cdot n/4$. For average-case inputs the lower bounds are $k \cdot n/4$ and $k \cdot n/8$, respectively.

The first non-trivial algorithm for k-k routing was presented by Kunde and Tensi [51]. It requires $5/4 \cdot k \cdot n + o(k \cdot n)$ steps. Several improvements gradually reduced the routing time to almost optimal [48, 88, 36, 38]. One of the most important ideas is the **coloring** of packets [48]: the packets are colored deterministically or randomly white and black, and at all times the white packets are routed orthogonally to the black packets. In this way the routing capacity of the MIMD mesh can be fully exploited. Another idea is to route the packets first to (pseudo-) random positions and from there to their destinations. This idea goes back on Valliant and Brebner [108]. Near-optimal results were first achieved with randomized algorithms [36]. Then it was discovered that randomization is superfluous [50, 41]: it can just as well be replaced by sorting the packets in submeshes and handing them out in a regular way (unshuffling).

Theorem 4 *k-k routing on MIMD meshes can be performed deterministically in $k \cdot n/2 + \mathcal{O}(k^{5/6} \cdot n^{2/3})$ steps. The maximum queue size is k [50, 41].*

It is not hard to perform k-k routing twice as fast for average-case inputs: just omit the (pseudo-) randomization phase. An analysis is given in [52]. In [42], several ideas are combined to construct an algorithm with routing time close to $k \cdot n/4$ for average-case inputs while never exceeding $k \cdot n/2$.

2.3 k-l Routing

General k-l **routing** has attracted less attention [24, 72, 65, 103, 66] than the previous problems. Here every Pu initially holds at most k packets, and no PU is the destination of more than l packets. Of course one can perform an m-m routing, for $m = \max\{k, l\}$, but when k and l are not almost equal, this implies a substantial loss. Considering the maximum of the bisection bound and the number of packets that may have to move into or out of a corner, it follows that

Lemma 4 *The k-l routing problem on MIMD meshes requires at least* $\max\{\frac{k}{2} \cdot (\frac{l}{l+k})^{1/2},\ \frac{l \cdot k}{l+k},\ \frac{l}{2} \cdot (\frac{k}{l+k})^{1/2}\} \cdot n$ *steps [103].*

At first glance it is not obvious how to achieve better than $\Omega(\max\{k,l\} \cdot n)$. For the case $l > k$, the central idea is to somehow color the packets so that in the final routing phase, in which the packets are routed within their rows or columns towards their destinations, the maximum number of packets in a row or column remains bounded. For $k > l$ one proceeds analogously, by coloring the packets so that during the first phase the number of packets moving in any row or column is as small as possible. Optimal results have been established for all $k = o(l)$ and $l = o(k)$:

Theorem 5 *k-l routing on MIMD meshes can be performed in $\sqrt{k \cdot l} \cdot n/2 + \mathcal{O}(\min\{k,l\} \cdot n)$ [65, 103].*

Problem 3 *Can k-l routing be performed in the time given by the lower bound of Lemma 4 plus some lower-order terms?*

A nice feature of the algorithm of [103] is that it works without knowing the values of k and l. Furthermore, though it is hard to prove something in general, the routing time is close to optimal for most inputs. The performance-ratio of k-l routing problems is addressed in [72, 66]. In these papers, algorithms are presented that perform within some logarithmic factor from optimal for *all* inputs. For applications such a factor is still much too large:

Problem 4 *Can on-line k-l routing be performed in the time required by the best off-line schedule plus some lower-order terms?*

Most results presented so far in Section 1 immediately generalize to higher dimensional meshes (except for the optimal permutation-routing algorithms). For k-l routing this is not the case, the performance of all algorithms mentioned strongly deteriorates with the dimension.

Problem 5 *Are there algorithms, randomized or deterministic, for k-l routing that perform well for higher dimensional meshes?*

A partial positive answer to this question is given in [72], but much better results are desirable.

2.4 Cut-through and Wormhole Routing

Cut-through routing is a routing paradigm under which the packets consists of k flits. During the routing the consecutive flits of a packet have to reside in the same PU or an adjacent one. The study of cut-through routing was initiated by Makedon and Symvonis [63]. Subsequent improvements brought the time required for cut-through routing down close to the lower bound.

Lemma 5 *For routing a permutation on an MIMD mesh, with packets consisting of k flits each, $k \cdot n/2 + n/k + 3/2 \cdot n + o(k \cdot n)$ steps are sufficient [63, 88, 36].*

To achieve this one can apply the same algorithm as for k-k routing. It is interesting to consider whether cut-through routing is essentially harder than k-k routing:

Problem 6 *Can cut-through routing as specified in Lemma 5 be solved in $k \cdot n/2 + o(k \cdot n)$ steps?*

Wormhole routing is the same as cut-through routing with one important difference: the packets (worms) can be expanded only once. That is, during the routing, the flits of a packet should not reside in the same PU. For wormhole routing, theoretical analysis of the congestion is more difficult, since packets which have partially crossed an edge can block other packets for an arbitrary time. Without special provisions deadlock is possible. Presently wormhole routing appears to be practically the most important paradigm (see [26, 81] for more details concerning the model).

In [18] Felperin, Raghavan and Upfal apply a greedy algorithm in which the packets are delayed by a randomly chosen number of steps. In this way they obtain good performance for long messages:

Theorem 6 *For worms of length k, delayed greedy wormhole routing can be performed on an $n \times n$ mesh in $\mathcal{O}(k \cdot n \cdot \log n + n^2/\log n)$ steps, with high probability [18].*

Practically even more important is that they also show that wormhole routing is suited for dynamic routing: if, in every step, each PU generates with sufficiently small probability a packet, then all packets reach their destinations without much delay.

Theorem 7 *There is a constant $c > 0$, so that if in every step all PUs generate with probability at most $c/(k \cdot n + n^2)$ a packet with random destination, then any packet reaches its destination within $\mathcal{O}(\log n \cdot (k + n))$ steps, with high probability.*

Notice that the expected number of packets generated in every step is at most c. This means that at most $k \cdot c$ flits pass the bisection. For $k << n$ this is very few in comparison with the possible n flits that might pass.

Newman and Schuster combined the requirements of hot-potato and wormhole routing. They achieve a bound of $\mathcal{O}(k^{2.5} \cdot n)$ steps for routing worms of length k on an $n \times n$ mesh [78]. In [103] it is shown that under a realistic condition the time consumption is only $\mathcal{O}(k^{1.5} \cdot n)$. The idea behind [78] is ingenious: worms that are temporarily blocked are kept cycling around over 'parking lots', and as soon as the 'highway' running along the parking lot is free, they take off. Bar-Noy ea. presented a randomized algorithm running in $\mathcal{O}(k \cdot n)$ steps [5]. Recently Roberts and Symvonis showed how the same performance can be achieved by an off-line algorithm [90]. These results make it likely that the following question can be answered affirmatively (applying either pseudo-randomization [41] or techniques to turn off-line algorithms into on-line algorithms [106]):

Problem 7 *Is there a deterministic on-line algorithm for hot-potato wormhole routing of worms with maximal length k running in $\mathcal{O}(k \cdot n)$ time?*

2.5 Dynamic Routing Problems

In **dynamic routing** algorithms, it is assumed that in every step each node generates a packet with a fixed probability p. A generated packet is assumed to have a random destination. Whether these assumptions are justified depends on the application.

There is a trivial upper bound on the generation probability p: approximately one half of the packets generated in the left half of the mesh have their destinations in the right half. As at most n packets can pass the bisection in every step (we consider a MIMD mesh), this implies that the system will get more and more congested, eventually resulting in infinite delays, for $p > 4/n$. Leighton has shown that for all $p < 4/n$ routing the packets greedily along row and column to their destinations normally works fine [56]. If several packets are competing for the use of a connection, then the one that has to move farthest in this direction is given priority.

Theorem 8 *If the packet arrival rate in an $n \times n$ MIMD mesh is less than the network capacity $4/n$, then the maximum delay incurred by any packet in any window of T steps is bounded by $\mathcal{O}(\log T + \log n)$, with high probability. The maximum occurring queue size is $\mathcal{O}(1 + \log T / \log n)$ [56].*

A somewhat weaker result holds for dynamic cut-through routing:

Lemma 6 *We consider cut-through routing on an $n \times n$ MIMD mesh with packets consisting of at most k flits. If the packet arrival rate is less than $1/(k \cdot n)$, then the maximum delay incurred by any packet in any window of T steps is bounded by $\mathcal{O}(k \cdot (\log T + \log n))$, with high probability. The maximum occurring queue size is $\mathcal{O}(k + k \cdot \log T / \log n)$ [56].*

A recent paper by Mitzenmacher [73] provides additional results on greedy dynamic routing. He analyzes lower and upper bounds for the expected time that a packet spends in the network.

The situation for dynamic wormhole routing remains largely unclear:

Problem 8 *Is there a dynamic wormhole routing algorithm that assures that a constant fraction of the network capacity can be used while all packets reach their destination with minimal delay?*

Such an analogue of Lemma 6 would have considerable practical importance. Trivial greedy routers eventually come to a deadlock: a certain degree of adaptiveness is required. Ngai and Seitz give an analysis of adaptive dynamic routing [80].

2.6 Fault Tolerant Routing

So far we have considered meshes whose PUs and connections operate correctly at all times. In actual systems several types of failures may arise. It is important that a correct computation can nevertheless be sustained by the functioning PUs. Faults may occur in the connections and in the PUs. If we are interested only in asymptotical results, then only faulty PUs need to be considered (exclude both PUs connected by a faulty link). Two types of faults are generally distinguished: random and worst-case. In the case of random faults, it is assumed that PUs or connections fail with a certain fixed probability p, independently of each other. In [29] (and elsewhere) a **p-faulty** network is a network in which the PUs fail with probability at most p. There is one more major distinction: on the one hand there are algorithms that give on-line adaptations to dynamicly occurring faults, on the other there are algorithms that cope with static faults.

Dynamic Faults. In the case of dynamic faults a typical goal is to minimize the expected lifetime of a packet. In order to realize this, we apply the **shortest-path strategy**: routing the packets as far as possible along shortest paths. The best shortest-path route is the one with the most alternatives. The **Z^2-policy** [4] states that propagation of messages towards the diagonal should be given precedence. The diagonal with respect to PU $P_{0,0}$ is the set of PUs $\{P_{i,i} | \ 0 \leq i \leq n-1\}$. Thus, a packet residing in $P_{i,j}$ with destination in $P_{0,0}$, tries to move to $P_{i,j-1}$ if $i < j$, to $P_{i-1,j}$ if $i > j$, and to either of them if $i = j$.

Theorem 9 *The Z^2-policy minimizes the expected lifetime of the packets [4].*

A negative result on the effectiveness of shortest-path routing is given by Gordon and Stout:

Lemma 7 *Shortest-path routing on the p-faulty $n \times n$ mesh has a probability of successful completion approaching zero for large n [20].*

Another approach, **sidetracking**, attempts to move a packet along a shortest path from its current position to its destination. If at any instant such a node cannot be found, the packet is sent to a randomly chosen non-faulty node from the set of non-faulty neighboring nodes. Thus, this approach corresponds to oblivious non-minimal path routing. Although the probability of successful routing on the p-faulty mesh still converges to zero, good empirical results were reported (also for the hypercube) [21].

In an algorithm by Raghavan [86], instead of sending a packet along its master path P as defined by Valiant and Brebner's randomized algorithm, a packet is broadcast within a routing region $R(P)$, defined to contain all PUs within distance $c \cdot \log n$ of P, where c is a constant. This region $R(P)$ has the property that if there is a live path between the packet's source and destination, the path will be contained in $R(P)$.

Theorem 10 *For all $p \leq 0.29$, on-line permutation routing can be performed on a p-faulty $n \times n$ mesh in $\mathcal{O}(n \cdot \log n)$ steps, with high probability [86]. The maximum queue size is $\mathcal{O}(\log^2 n)$.*

Mathies even improved the bound on p to about 0.40 [69].

Static Faults. The above approaches give on-line adaptations. For static faults, for example those that arise as a result of production errors, stronger results can be obtained. The idea is to emulate a fault-free network on the faulty one. In principle it would suffice to find a suitable embedding, but a larger number of faults can be tolerated if in addition we perform **redundant computation**. This means that operations of a PU of the fault-free network are emulated on several nodes of the faulty network. Of course this introduces possible consistency problems that must be handled carefully.

Kaklamanis e.a. [29] achieve the following: "in many cases the running time is the same as if there were no faults in the array, up to constant factors". Among other things they prove

Theorem 11 *Permutation routing among all live processors on an $n \times n$ mesh with $f \leq n/3$ faulty nodes can be performed in $\mathcal{O}(n + f^2)$ steps using constant size queues with high probability [29].*

Further improvements are given by Cole, Maggs and Sitaraman [16]. They prove that $n^{1-\epsilon}$, for some constant $\epsilon > 0$, worst-case faults can be handled so that the computation is slowed down by only a constant factor. The most surprising result is

Theorem 12 *A p-faulty $n \times n$ mesh, for some constant $p > 0$, can emulate a fault-free $n \times n$ mesh with constant slowdown, with high probability [16].*

For practical purposes the described static approaches are insufficient: we would lose a considerable constant factor when only a small constant fraction of the PUs is not functioning. Furthermore, we would also like to have algorithms that tolerate small number of faults with small delays:

Problem 9 *Can an $n \times n$ mesh with x random faults emulate a fault-free mesh with delay factor $1/(1 - \mathcal{O}(x/n^2))$, for all $x = 1, 2, \ldots$?*

3 Sorting on Meshes

In the **sorting problem**, all packets have a key from a totally ordered set, and the PUs are indexed with some indexing scheme. The packets must be rearranged so that they acquire a sorted order with respect to the indexing. Depending on the details of the mesh and the comparison model, the sorting time can depend more or less on the chosen indexing scheme.

3.1 1-1 Sorting

One-Packet Model. In the earliest papers on sorting on meshes, it was assumed that at all times every PU could hold only one packet and that the connections acted as comparators: the **one-packet model**. Under this assumption one can derive interesting lower bounds using so-called **joker-zone arguments**, which exploit that information that has a considerable impact on the sorting may remain unknown to the PUs in a corner of the mesh for almost $2 \cdot n$ steps:

Lemma 8 *Sorting on a MIMD mesh in the one-packet model requires at least $3 \cdot n - \mathcal{O}(\sqrt{n})$ steps for sorting in (snake-like) row-major order [94, 46].*

Whereas these results are almost trivial, much deeper methods are required to prove lower bounds for all indexing schemes:

Lemma 9 *There is no indexing scheme with respect to which sorting in the one-packet model can be performed in fewer than $2.27 \cdot n$ steps [23].*

Thompson and Kung [107] presented the first $\mathcal{O}(n)$ sorting algorithm. Schnorr and Shamir [94] then developed the first 'optimal' algorithm.

Theorem 13 *Sorting on MIMD meshes in the one-packet model with respect to the snake-like row-major indexing can be performed in $3 \cdot n + \mathcal{O}(n^{3/4})$ steps [107, 62, 94]. On a $m \times n$ mesh the algorithm takes $m + 2 \cdot n + o(n + m)$ steps.*

Multi-Packet Model. If we get away from the theoretically interesting but practically meaningless restriction that PUs can hold only one packet, much better results are possible. A further distinction can be made between algorithms in which packets are copied, and those that do not make copies. Just by concentrating all the packets first in the central $n/2$ columns and then applying the algorithm of [94], Kunde achieved a sorting time of $2\frac{1}{2} \cdot n + o(n)$ steps [48]. The first near-optimal algorithm, almost matching the distance bound of $2 \cdot n$, was designed by Kaklamanis and Krizanc [31]. This randomized algorithm was successfully derandomized in [41]. These latter algorithms sort the packets with respect to a blocked snake-like row-major indexing scheme. An important impulse towards considering more natural indexing schemes was given by Krizanc and Narayanan [43]. They analyzed the case in which the keys of the packets are only zero or one. Then in [100] it was shown that the general problem of 1-1 sorting with respect to a row-major indexing scheme can be solved optimally.

Theorem 14 *Sorting on MIMD meshes in row-major order can be performed in $2 \cdot n + o(n)$ steps with maximal queue size 5 by a deterministic algorithm [48, 30, 31, 41, 43, 100].*

In many recursive algorithms for meshes, sorting has to be performed on submeshes of decreasing size. For the last stages of such a recursion it is important to have an algorithm that does not only perform well asymptotically, but even for small meshes. For example, the estimates of the queue sizes of the permutation routing algorithms in [87, 12, 96] essentially depend on the existence of such an algorithm. Such algorithms are also important because existing meshes are of limited size ($n \leq 256$).

Problem 10 *What is the best sorting algorithm for small values of n, say $n \leq 256$?*

Though asymptotically optimal, all algorithms cited in Theorem 14 have lower-order terms that tend to dominate the overall sorting time for small meshes. The reason is that they are splitter-based. Selecting the splitters, and comparing them with the packets requires that submeshes of fairly large size, e.g. $n^{3/4} \times n^{3/4}$, are sorted repeatedly. Much better in this respect are merge-sort algorithms: they show a larger leading term, but no 'lower-order terms'. Valuable contributions have been made by Thompson and Kung [107], Nassimi and Sahni [75], Kumar and Hirschberg [45], and Lang e.a. [54]. Some of them are implementations of Batchers **bitonic sort** [6] , which is also an effective technique for sorting on hypercubes ($\mathcal{O}(\log^2 n)$ time). Lately several ideas have been combined to reduce the leading term by almost a factor of two:

Lemma 10 *For all n, sorting in row-major order can be performed in $4.75 \cdot n$ steps with maximal queue size 9. The routing model is uni-axial. For $n = 2^l$, for some $l > 0$ sorting can be performed in $4\frac{1}{2} \cdot n$ with queue size 6, and for $n = 3^l$ in $4\frac{1}{3} \cdot n$ with queue size 5 [100].*

Limited Instruction Sets. All these algorithms require rather elaborate programs. Many attempts have been made to compose a sorting algorithm containing as few as possible basic operations which are alternated. We consider $m \times n$ meshes.

The simplest idea is to alternately sort rows and columns: **shearsort** . All columns and rows with even index are sorted in ascending order, while odd rows are sorted in descending order. $\log m + 1$ iterations are sufficient:

Lemma 11 *Shearsort sorts $m \cdot n$ numbers on a $m \times n$ mesh in snake-like order in $(m+n) \cdot (\lceil \log m \rceil + 1)$ steps [91, 93].*

An extension of this algorithm by Schnorr and Shamir [94] includes an extra phase in each iteration, after the two basic sorting phases. This phase corresponds to a shuffle permutation applied to row i, $1 \leq i \leq m$ for $rev(i) = i \bmod n^{1/2}$ times. The resulting **revsort** algorithm iterates these three phases $\lceil \log \log m \rceil$ times.

Lemma 12 *Revsort sorts $m \cdot n$ numbers on a $m \times n$ mesh in snake-like order in $(m + n) \cdot \lceil \log \log m \rceil$ steps [94].*

A generalization of the shearsort idea to higher-dimensional meshes is given in [110].

The next natural question is whether it is possible to sort in a constant number of phases. Marberg and Gafni achieve this by including more sorting and shuffle steps in each iteration. Their algorithm is called **rotatesort**.

Theorem 15 *Rotatesort sorts on a mesh in snake-like or row-major order with 16 row and column phases [68].*

They also prove a lower bound on the number of required phases:

Theorem 16 *Sorting on a $m \times n$ mesh, $m, n \geq 8$, into any sorting order requires at least 5 phases consisting of column/row sorts and shuffles [68]. This takes a total of $\mathcal{O}(n + m)$ steps.*

Problem 11 *Is it possible to bridge the gap between the lower bound, 5, and the upper bound, 16, on the number of phases consisting of row/column sorts and shuffles necessary to sort on a mesh?*

Better upper bounds are known only for special shapes of the mesh: Like the previous algorithms, Leighton's **columnsort** algorithm [55] consists of an alternation of sorting columns and rearrangements within rows. But, for $N = m \cdot n$, the number of elements to be sorted, it is assumed that there are at least $N^{2/3}$ numbers stand in every column.

Theorem 17 *For $m \geq n^2$, columnsort sorts $m \cdot n$ numbers on a $m \times n$ mesh into column major order using a sequence of 8 phases of row/column sorts and transpose/shifts [55]. In total this takes $\mathcal{O}(n+m)$ steps.*

Even simpler than the mentioned algorithms, which operate with a small number of different operations like row and column sorts, are **bubble-sort** algorithms. Such algorithms cycle through a small set of local comparisons in a fixed order. Clearly such algorithms are excellently suited for implementations on meshes which are build of very elementary processing units (fine-grain model), such as can be found in the MassPar, for example. Unfortunately, several straight-forward approaches fail [92], and result in an $\Omega(n^2)$ sorting time, even on the average. Recently a positive result has been reported by Ierardi. In [27], he presents a bubble-sort algorithm with almost linear-time consumption on the average:

Theorem 18 *On an $n \times n$ mesh, bubble sort can be performed in $\mathcal{O}(n \cdot \log^{1/2} n)$ time on the average [27].*

3.2 k-k Sorting

The theory of k-k sorting was developed hand in hand with the theory of k-k routing. This is not surprising: using splitters for estimating the ranks of the packets, k-k sorting is in most cases hardly more difficult than k-k routing. In some recent deterministic algorithms the splitters are not explicitly selected and broadcast, but the packets use the other packets that stand in the same submesh to estimate their ranks.

Kunde [48] gave a deterministic $k \cdot n + o(k \cdot n)$ sorting algorithm. The first almost optimal algorithm was developed by Rajasekaran in [36], and improved to $k \cdot n/2 + o(k \cdot n)$ in [40]. These algorithms are randomized. In [50, 41] it is shown how to derandomize the algorithm. In retrospect, these algorithms can be viewed as suitable combinations of the hypercube-routing algorithm by Valliant and Brebner [108] and Leighton's column-sort [55]. In [100] it was shown that the indexing scheme does not need to be blocked. One can actually sort with respect to any **piece-wise continuous indexing** . That is, an indexing scheme in which subsets of n^ϵ PUs, for some $\epsilon > 0$, are indexed consecutively. Most important, the ordinary row-major indexing scheme is piece-wise continuous.

Theorem 19 *k-k sorting on MIMD meshes can be performed in $k \cdot n/2 + o(k \cdot n)$ steps with maximal queue size $k + o(k)$ by a deterministic algorithm [48, 36, 40, 50, 41, 100].*

In [42] several 'easier variants of the sorting problem' are analyzed. For example, it is shown that $k \cdot n/4 + o(k \cdot n)$ steps are sufficient (and necessary) to determine the rank of all packets without requiring that they are arranged with respect to some particular indexing scheme.

4 Mesh-Like Networks

We provide some references to known results for variants of the mesh architecture: one-dimensional arrays, higher dimensional meshes, tori, meshes with diagonals and other shortcuts, meshes with buses and reconfigurable meshes.

4.1 One-Dimensional Arrays

Routing on **linear arrays** (one-dimensional meshes) is trivial: using the farthest-first priority strategy, a packet distribution is routed in exactly the number of steps required. If copying is allowed, then a

bubble-sort-like algorithm performs sorting in the same number of steps. Otherwise the last steps must be performed with some care and require a few extra steps.

For **rings** (one-dimensional tori) it is not obvious how to exploit the additional connection so as to reduce routing and sorting times by a factor of two [67, 64, 39, 99]. The paper by Mansour and Schulman [67] proves an interesting lower bound, and shows how to perform 1-1 sort in $\lfloor n/2 \rfloor + 1$ steps. For k-k sorting one can just apply a one-dimensional version of the algorithms from [50, 41] (though this reverses the order of the developments).

Interestingly, whereas for two-dimensional meshes the best deterministic routing algorithm at present is actually a sorting algorithm, for rings an independent routing algorithm exists. This algorithm has almost no lower order terms and is thereby competitive with greedy strategies:

Theorem 20 *k-k routing on an MIMD ring can be performed in $k \cdot n/4 + \sqrt{n}$ steps [64, 39, 99].*

A modification is suited for dynamic routing.

Problem 12 *Can the technique of [99] be generalized for k-k routing on meshes of dimension $d \geq 2$?*

4.2 Higher Dimensional Meshes and Tori

The routing and sorting algorithms of [36, 38, 50, 41, 100] are applicable for meshes and tori of arbitrary dimension. The dimension may even increase with n to a certain extent [41]. Optimal time, almost matching the bisection bound, is assured as long as there are at least $k \geq 4 \cdot d$ packets in every PU. Only the complexity of one-one problems remains largely unclear:

Problem 13 *What are the lower and upper bounds for one-one routing and sorting on meshes of dimension $d \geq 3$ and tori of dimension $d \geq 2$?*

Kunde [49] provides results for routing, but even there a considerable gap between lower and upper bounds remains. Kaklamanis, Krizanc and Rao have proven that the one-one off-line routing problem can be solved in $3 \cdot n - 1$ steps [32]. It is likely that applying techniques from [100] most results on routing can immediately be turned into results for sorting as well.

For off-line routing we can use the following generalization of Theorem 3:

Theorem 21 *Given permutation routing algorithms for graphs G and H, running in $T(G)$ and $T(H)$ steps, respectively, an off-line algorithm exists that routes any permutation on $G \times H$ in $T(G) + 2 \cdot T(H)$ steps [1].*

Greedy routing to independently chosen random destinations is analyzed in [56]. The fault-tolerant Z^2-policy (see Section 2.6) can be generalized to higher dimensional meshes [4].

Hardly any theoretical papers deal with **meshes with diagonals**, in which the PUs have up to 8 connections each. Kunde, Niedermeier and Rossmanith show that most routing and sorting times can be reduced by almost a factor of two [53]. The edge bisection of such a network is surprisingly more than twice as large (for $n \times n$ meshes $3 \cdot n$ instead of n), and indeed k-k routing and sorting can be solved more than twice as fast

Lemma 13 *On an $n \times n$ mesh with diagonals, k-k routing and sorting can be performed in $2/9 \cdot k \cdot n + o(k \cdot n)$ steps [53].*

In [7] Beivide ea. introduce a mesh-like graph with degree 4 which has a smaller diameter than a mesh, and which gives smaller average distances between the PUs.

4.3 Meshes with Fixed Buses

Meshes with fixed buses are meshes in which every row and column is equipped with an additional bus. Such a bus can be used by one of the PUs connected to it to transfer a packet in a single step to any other connected PU. There are numerous variants. One could also assume that the bus can be used by

several sending PUs as long as the required sections of the bus do not overlap. Sometimes it is assumed that there are two buses. Another possibility is the **mesh of buses**. In this model there are only buses and no standard mesh connections.

Having buses is most interesting for 'sparse problems', problems that involve little communication. For example, the maximum of n^2 numbers spread out over an $n \times n$ mesh with row and column buses can be solved in $\Theta(n^{1/3})$ steps, a tremendous improvement over the $2 \cdot n - 2$ steps required on an ordinary mesh. On the other hand, for routing k-k relations one still needs $k \cdot n/3$ steps, because the buses increase the capacity of the bisection by only 50%. On meshes with buses it is always trivial to perform sorting in only slightly more steps than routing: the buses are excellent for rapidly broadcasting and gathering a set of splitters.

The basics of routing on meshes with buses have been investigated by Leung and Shende [61]. Considerable improvements and extensions were achieved in three further papers [105, 85, 106].

4.4 Meshes with Reconfigurable Buses

Reconfigurable meshes offer many more possibilities. These are meshes in which all PUs are connected to one big bus. Every PU has a switch with which it can connect or disconnect the bus at its position. In this way the bus can be broken up into many smaller buses.

A reconfigurable bus has unbelievable capacities for certain problems:

Theorem 22 *On a reconfigurable $n \times n$ mesh, n numbers can be sorted in a constant number of steps [109, 9].*

Applying column-sort (see Theorem 17) the number of steps can actually be reduced to below 40. Extensions for sorting n^{d-1} numbers on a d-dimensional reconfigurable mesh are immediate [82]. Theoretically interesting is

Problem 14 *How many steps are required for sorting n numbers on an $n \times n$ reconfigurable mesh?*

Another feature of reconfigurable meshes is that they allow the routing and sorting of extremely sparse packet distributions in a time given by the bisection bound:

Lemma 14 *On a reconfigurable $n \times n$ mesh $k \cdot n^2$ well-repartitioned numbers can be sorted in $k \cdot n + o(k \cdot n)$ steps for all $k > n^{-4/7}$ [37].*

The theory of reconfigurable meshes is very rich and we have only mentioned a few results. More references can be found in the referenced papers. Nakano gives an extensive and up-to-date list of publications [79].

5 Other Problems

Meshes are one of the fundamental networks, and are likely to dominate the scene of parallel computation in one form or another. Therefore, it is natural to analyze the possibility of using them as general purpose computers, and to develop a set of algorithms for basic problems on them. The alternative would be to simulate a suitable PRAM (Parallel Random Access Machine) on them and to apply PRAM algorithms. However, this inevitably leads to a considerable loss in performance. Sometimes the loss is a constant factor, mostly one loses a polylog term, and sometimes even more. In this section we treat several main problems. Rather than striving for completeness, we provide some pointers to important work. More text and more references can be found in Chapter 1 of Leigton's book [57].

Numerical Computations. In a natural way, meshes are suited for matrix operations. Several operations that sequentially take $\mathcal{O}(n^3)$ time can be solved on an $n \times n$ mesh in $\mathcal{O}(n)$ steps.

One of the simplest mesh algorithms is the one for multiplying two $n \times n$ matrices: by shifting them in from the top and the left in a suitable way, this problem is solved in $3 \cdot n - 2$ steps. More intriguing is that Warshall's transitive-closure algorithm can be implemented in $\mathcal{O}(n)$ steps [14, 95]. Atallah and Kosaraju give the following generalization:

Theorem 23 *Initially numbers $f_0(i,j)$ are stored in $P_{i,j}$. $f_k(i,j)$ is defined by a recurrence of the type*

$$f_k(i,j) = g(f_{k-1}(i,j), f_{k-1}(i,k), f_{k-1}(k,j)), \text{ for } 1 \le i,j \le n, 1 \le k \le n.$$

In $\mathcal{O}(n)$ steps the numbers $f_n(i,j)$ can be computed and stored in $P_{i,j}$ [3].

Graph Problems. The most analyzed graph problem is probably the connected-components problem. Inspired by Hirschberg's algorithm [25] many algorithms were developed for PRAMs and networks. For meshes the problem has been considered by Nassimi and Sahni [74], Atallah and Kosaraju [3] and Hambrusch [22]. Recently, better results have been obtained. The time consumption involves both the number of vertices and the number of edges of the graph.

Theorem 24 *On an $n \times n$ mesh, the connected components of a graph with v nodes and e edges can be determined in $\mathcal{O}((v \cdot e)^{1/2}/n + \log v \cdot (n + e/n^2))$ steps [104].*

Similar results are achieved for other problems that can be solved sequentially in linear time by breadth-first search: minimum spanning trees, and the construction of a path between a pair of vertices.

List Ranking. In parallel algorithms the list-ranking problem is important for the conversion of lists into arrays. It is used as a subroutine in many graph problems: by the Euler-tour technique problems on trees can be reduced to ranking lists (see [28]).

Suitable implementations of PRAM algorithms achieve a $\mathcal{O}(n)$ time for ranking a list of length n^2 on an $n \times n$ mesh [2, 19]. The hidden constant is so large that none of these algorithms appear practical. A considerable reduction of the leading constant is achieved in [101]. When every PU holds sufficiently many elements of the list, then a randomized algorithm achieves much better results:

Theorem 25 *If every PU holds $k = \omega(1)$ nodes of a set of singly linked lists, and all first and last nodes of the lists know their status, then the list-ranking problem can be solved by a randomized algorithm in $(1/2 + o(1)) \cdot k \cdot n$ steps, with high probability [102].*

Pattern Matching. In [13] Chlebus and Gąsieniec analyze the problem of finding all occurrencies of a given text pattern in a text. If on an $n \times n$ mesh all PUs hold one symbol of the text, then the problem is solved in $\mathcal{O}(n)$ time.

PRAM Simulation. Instead of developing a full set of algorithms for the mesh, one could also design an efficient PRAM simulator for the mesh and optimized PRAM algorithms. This has the great advantage of an increased portability, but the loss in performance is considerable. A good compromise is offered by a hybrid approach, in which a library of optimized mesh algorithms is available for the most applied basic problems, and which allows the user to program in a convenient PRAM-like style. In this way the PRAM simulator is applied mainly to connect library subroutines, and the overall performance hardly depends on its performance.

PRAM simulation is not limited to meshes, and there is a rich literature (see any of the papers mentioned for more references). An essential distinction exists between randomized algorithms, using the randomization to choose a hash function from a certain universal class [33]; and deterministic algorithms, distributing copies of every item in order to minimize the access time for all batches of requests [70].

For meshes there are only a few results. Leppänen and Penttonen report simulation results on randomized PRAM simulation [59]. The conclusion from this work is that memory requests can be answered almost as fast as is predictable by analyzing the traffic over a bisection. In a new paper [60] they obtain a higher speed-up per PU by considering PRAM simulations on a **coated mesh**. A coated mesh is an intermediate between a cross-bar switch and a mesh: the grid itself consists of routing nodes, while the actual PUs lie as a 'coating' along the perimeter. Thus, an $n \times n$ coated mesh has only $4 \cdot n$ PUs. On the theoretical side we find an adaptation for meshes of the deterministic algorithm by Pietracaprina and Preparata [83], which was designed for completely connected networks [84]. Here it is shown that deterministic PRAM simulation can be performed with only slightly more than a constant factor loss in comparison with the randomized approach. Simulation results of this algorithm are given in [71].

References

[1] Annexstein, F., M. Baumslag, 'A Unified Approach to Off-line Permutation Routing on Parallel Networks,' *Proc. 2nd ACM Symposium Parallel Algorithms and Architectures*, pp. 398–406, ACM, 1990.

[2] Atallah, M.J., S.E. Hambrusch, 'Solving Tree Problems on a Mesh-Connected Processor Array,' *Information and Control*, 69, pp. 168–187, 1986.

[3] Atallah M.J., S.R. Kosaraju, 'Graph Problems on a Mesh-Connected Processor Array,' *Journal of the ACM*, 31(3), pp. 649–667, 1984.

[4] Badr, H.G., S. Podar, 'An Optimal Shortest-Path Routing Policy for Network Computers with Regular Mesh-Connected Topologies,' *IEEE Trans. Comput.* C-38(10), pp. 1362–1371, 1989.

[5] Bar-Noy, A., B. Schieber, P. Raghavan, H. Tamaki, 'Fast Deflection Routing for Packets and Worms,' *Proc. 12th Symp. on Principles of Distributed Computing*, pp. 75–86, ACM, 1993.

[6] Batcher, K.E., 'Sorting Networks and their Applications,' *Proc. AFIPS Spring Joint Computer Conference*, 32, pp. 307–314, 1968.

[7] Beivide, R., E. Herrada, J.L. Balcázar, A. Arruabarrena, 'Optimal Distance Networks of Low Degree for Parallel Computers,' *IEEE Transactions on Computers*, pp. 1–16, 40(10), 1991.

[8] Ben-Aroya, I., A. Schuster, 'Greedy Hot-Potato Routing on the Mesh,' *Proc. 2nd European Symposium on Algorithms*, pp. 365–376, Springer-Verlag, 1994.

[9] Ben-Asher, Y., D. Peleg, R. Ramaswami, A. Schuster, 'The Power of Reconfiguration,' *Journal of Parallel and Distributed Computing*, 13, pp. 139–153, 1991.

[10] Berge, C., *Graphs and Hypergraphs*, North-Holland, Amsterdam, 1973.

[11] Chinn, D.D., T. Leighton, M. Tompa, 'Minimal Adaptive Routing on the Mesh with Bounded Queue Size,' *Proc. 6th Symposium on Parallel Algorithms and Architectures*, pp. 354–363, ACM, 1994.

[12] Chlebus, B.S., M. Kaufmann, J.F. Sibeyn, 'Deterministic Permutation Routing on Meshes,' *Proc. 5th Symposium on Parallel and Distributed Processing*, pp. 814–821, IEEE, 1993.

[13] Chlebus, B.S., L. Gąsieniec 'Optimal Pattern Matching on Meshes,' *Proc. 11th Symposium on the Theoretical Aspects of Computer Science*, LNCS 775, pp. 213–224, Springer-Verlag, 1994.

[14] Christopher, T.W., An Implementation of Warshall's Algorithm for Transitive Closure on a Cellular Computer,' *Rep. No. 36*, Institute for Computer Research, University of Chicago, Chicago, 1973.

[15] Cole, R., J. Hopcroft, 'On Edge Coloring Bipartite Graphs,' *Siam Journal on Computing*, 11, pp. 540–546, 1982.

[16] Cole, R., B. Maggs, R. Sitaraman, 'Multi-Scale Self-Simulation: A Technique for Reconfiguring Arrays with Faults,' *Proc. 25th Symposium on Theory of Computing*, pp. 561–572, ACM, 1993.

[17] Feige, U., P. Raghavan, 'Exact Analysis of Hot-Potato Routing,' *Proc. 33rd Symposium on Foundations of Computer Science*, pp. 553–562, IEEE, 1992.

[18] Felperin, S., P. Raghavan, E. Upfal, 'A Theory of Wormhole Routing in Parallel Computers,' *Proc. 33rd Symposium on Foundations of Computer Science*, pp. 563–572, IEEE, 1992.

[19] Gibbons, A.M., Y. N. Srikant, 'A Class of Problems Efficiently Solvable on Mesh-Connected Computers Including Dynamic Expression Evaluation,' *International Processing Letters*, 32, pp. 305–311, 1989.

[20] Gordon, J.M., Q.F. Stout, 'Fault Tolerant Message Routing on Large Parallel Systems,' *Proc. 2nd Symp. Frontiers Massively Parallel Computation*, pp. 155–158, IEEE, 1988.

[21] Gordon, J.M., Q.F. Stout, 'Hypercube Message Routing in the Presence of Faults,' *Proc. 3rd Conf. Hypercube Concurrent Computing*, pp. 318–327, 1988.

[22] Hambrusch, S.E., 'VLSI Algorithms for the Connected Components Problem,' SIAM Journal on Computing, 12, pp. 354–365, 1983.

[23] Han, Y., Y. Igarashi, M. Truszczynski, 'Indexing Functions and Time Lower Bounds for Sorting on a Mesh-Connected Computer,' *Discrete Applied Mathematics*, 36, pp. 141–152, 1992.

[24] Han, T., D.F. Stanat, ' "Move and Smooth" Routing Algorithms on Mesh Connected Computers,' *Proc. 28th Allerton Conference*, pp. 236–245, 1990.

[25] Hirschberg, D.S., A.K. Chandra, D.V. Sarwate, 'Computing Connected Components on Parallel Computers,' *Communications of the ACM,* 22(8), pp. 461–464, 1979.

[26] Hwang, K., *Advanced Computer Architecture; Parallelism, Scalability, Programmability*, McGraw-Hill, Inc., 1993.

[27] Ierardi, D., '2d-Bubblesorting in Average Time $\mathcal{O}(\sqrt{N \lg N})$,' *Proc. 6th Symposium on Parallel Algorithms and Architectures*, pp. 36–45, ACM, 1994.

[28] JáJá, J., *An Introduction to Parallel Algorithms*, Addison-Wesley Publishing Company, Inc., 1992.

[29] Kaklamanis, C., A.R. Karlin, F.T. Leighton, V. Milenkovic, P. Raghavan, S. Rao, C. Thomborson, A. Tsantilas, 'Asymptotically Tight Bounds for Computing with Faulty Arrays of Processors,' *Proc. 31st Symposium on Foundations of Computer Science*, pp. 285–296, IEEE, 1990.

[30] 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.

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

[32] Kaklamanis, C., D. Krizanc, S. Rao, 'Simple Path Selection for Optimal Routing on Processor Arrays,' *Proc. 4th Symposium on Parallel Algorithms and Architectures*, pp. 23–30, ACM, 1992.

[33] Karlin, A.R., E. Upfal, 'Parallel Hashing - an Efficient Implementation of Shared Memory,' *Proc. 18th Symposium on Theory of Computing*, pp. 160–168, 1986.

[34] Kaufmann, M., H. Lauer, H. Schröder, 'Fast Deterministic Hot-Potato Routing on Meshes,' *Proc. 5th Int. Symposium on Algorithms and Computation*, LNCS 834, pp. 333–341, Springer-Verlag, 1994.

[35] Kaufmann, M., U. Meyer, J.F. Sibeyn, 'Towards Practical Permutation Routing on Meshes,' *Techn. Rep. MPI-I-94-153*, Max-Planck Institut für Informatik, Saarbrücken, Germany, 1994.

[36] Kaufmann, M., S. Rajasekaran, J.F. Sibeyn, 'Matching the Bisection Bound for Routing and Sorting on the Mesh,' *Proc. 4th Symposium on Parallel Algorithms and Architectures*, pp. 31–40, ACM, 1992.

[37] Kaufmann, M., H. Schröder, J.F. Sibeyn, 'Asymptotically Optimal and Practical Routing on the Reconfigurable Mesh,' *Parallel Processing Letters*, 5(1), pp. 81–95, 1995.

[38] Kaufmann, M., J.F. Sibeyn, 'Optimal Multi-Packet Routing on the Torus,' *Proc. 3rd Scandinavian Workshop on Algorithm Theory*, LNCS 621, pp. 118–129, Springer-Verlag, 1992.

[39] Kaufmann, M., J.F. Sibeyn, 'Deterministic Routing on Circular Arrays,' *Proc. 4th Symposium on Parallel and Distributed Processing*, IEEE, pp. 376–383, 1992.

[40] Kaufmann, M., J.F. Sibeyn, 'Randomized k-k Routing and Sorting on Meshes,' submitted to *Algorithmica*, 1993.

[41] Kaufmann, M., J.F. Sibeyn, T. Suel, 'Derandomizing Routing and Sorting Algorithms for Meshes,' *Proc. 5th Symposium on Discrete Algorithms*, pp. 669–679, ACM-SIAM, 1994.

[42] Kaufmann, M., J.F. Sibeyn, T. Suel, 'Beyond the Bisection Bound: Fast Ranking and Counting on Meshes,' *Proc. 3rd European Symposium on Algorithms*, LNCS, Springer-Verlag, 1995, to appear.

[43] Krizanc, D., L. Narayanan, 'Zero-One Sorting on the Mesh,' *Proc. 5th Symposium on Parallel and Distributed Processing*, IEEE, pp. 641–647, 1993.

[44] Krizanc, D., S. Rajasekaran, Th. Tsantilas, 'Optimal Routing Algorithms for Mesh Connected Processor Arrays,' Proc. VLSI Algorithms and Architectures, 1988, LNCS 319, pp. 411–422.

[45] Kumar, M., D. Hirschberg, 'An Efficient Implementation of Batcher's Odd-Even Merge Algorithm and its Application to Parallel Sorting Schemes,' *IEEE Transactions on Computers*, 32, pp. 141–150, 1983.

[46] Kunde, M., 'Lower Bounds for Sorting on Mesh-Connected Architectures,' *Acta Informatica*, 24, pp. 121–130, 1987.

[47] Kunde, M., 'Routing and Sorting on Mesh Connected Processor Arrays,' *Proc. VLSI Algorithms and Architectures*, LNCS 319, pp. 423–433, Springer-Verlag, 1988.

[48] Kunde, M., 'Concentrated Regular Data Streams on Grids: Sorting and Routing Near to the Bisection Bound', *Proc 31th Symposium on Foundations of Computer Science*, pp. 141–150, IEEE, 1991.

[49] Kunde, M., 'Balanced Routing: Towards the Distance Bound on Grids,' *Proc. 3rd ACM Symposium on Parallel Algorithms and Architectures*, pp. 260–271, 1991.

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

[51] Kunde, M., T. Tensi, 'Multi-Packet Routing on Mesh Connected Processor Arrays,' *Proc. Symposium on Parallel Algorithms and Architectures*, pp. 336–343, ACM, 1989.

[52] Kunde, M., R. Niedermeier, K. Reinhardt, and P. Rossmanith, 'Optimal Average Case Sorting on Arrays ,' *Proc. 12th Symposium on Theoretical Aspects of Computer Science*, pp. 503–513, Springer, 1995.

[53] Kunde, M., R. Niedermeier, P. Rossmanith, 'Faster Sorting and Routing on Grids with Diagonals,' *Proc. 11th Symposium on Theoretical Aspects of Computer Science*, LNCS 775, pp. 225–236, Springer Verlag, 1994.

[54] Lang, H.W., M. Schimmler, H. Schmeck, H. Schröder, 'Systolic Sorting on a Mesh-Connected Network,' *IEEE Transactions on Computers*, 34, pp. 652–658, 1985.

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

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

[57] Leighton, T., *Introduction to Parallel Algorithms and Architectures: Arrays-Trees-Hypercubes*, Morgan-Kaufmann Publishers, San Mateo, California, 1992.

[58] Leighton, T., F. Makedon, Y. Tollis, 'A $2n-2$ Step Algorithm for Routing in an $n \times n$ Array with Constant Size Queues,' *Proc. Symposium on Parallel Algorithms and Architectures*, pp. 328–335, ACM, 1989.

[59] Leppänen V., M. Penttonen, 'Simulation of PRAM Models on Meshes,' *Proc. 6th Parallel Languages and Architectures Europe*, LNCS 817, pp. 146–158, Springer-Verlag, 1994.

[60] Leppänen V., M. Penttonen, 'Work-Optimal Simulation of PRAM Models on Meshes,' *Nordic Journal of Computing*, 2, pp. 51–69, 1995.

[61] Leung, J., S.M. Shende, 'Packet Routing on Square Meshes with Row and Column Buses,' *Proc. 3rd Symposium on Parallel and Distributed Processing*, pp. 834–837, IEEE, 1991.

[62] Ma, Y., S. Sen, D. Scherson, 'The Distance Bound for Sorting on Mesh Connected Processor Arrays is Tight,' *Proc. 27th Symposium on Foundations of Computer Science*, pp. 255–263, IEEE, 1986.

[63] Makedon, F., A. Simvonis, 'On Bit Serial Packet Routing for the Mesh and the Torus,' *Proc. 3rd Symposium on Frontiers of Massively Parallel Computation*, pp. 294–302, 1990.

[64] Makedon, F., A. Simvonis, 'Optimal Algorithms for Multipacket Routing Problems on Rings,' *Journal of Parallel and Distributed Computing*, 22, pp. 37–43, 1994.

[65] Makedon, F., A. Simvonis, 'Optimal Algorithms for the Many-to-One Routing Problem on Two-Dimensional Meshes,' *Microprocessors and Microsystems*, 17(6), pp. 361–367, 1993.

[66] Mansour Y., B. Patt-Shamir, 'Many-to-One Packet Routing on Grids,' *Proc. 27th Symposium on Theory of Computing*, pp. 258–267, ACM, 1995.

[67] Mansour, Y., L. Schulman, 'Sorting on a Ring of Processors,' *Journal of Algorithms*, 11, pp. 622–630, 1990.

[68] Marberg, J.M., E. Gafni, 'Sorting in Constant Number of Row and Column Phases on a Mesh,' *Algorithmica*, 3, pp. 561–572, 1988.

[69] Mathies, T.R., 'Percolation Theory and Computing with Faulty Arrays of Processors,' *Proc. 3rd Symposium on Discrete Algorithms*, pp. 100–103, ACM-SIAM, 1992.

[70] Mehlhorn, K., U. Vishkin, 'Randomized and Deterministic Simulations of PRAMs by Parallel Machines with Restricted Granularity of Parallel Memories,' *Acta Informatica*, 9(1), pp. 29–59, 1984.

[71] Meyer, U., J.F. Sibeyn, 'Simulating the Simulator: Deterministic PRAM Simulation on a Mesh Simulator,' *Proc. Eurosim '95*, F. Breitenecker and I. Husinsky (Eds), Elsevier, 1995, to appear.

[72] Meyer auf der Heide, F., B. Oesterdiekhoff, R. Wanka, 'Strongly Adaptive Token Distribution,' *Proc. 20th ICALP*, pp. 398–409, EATCS, 1993.

[73] Mitzenmacher, M., 'Bounds on the Greedy Routing Algorithm for Array Networks,' *Proc. 6th Symposium on Algorithms and Architectures*, pp. 346–353, ACM, 1994.

[74] Nassimi, D., S. Sahni, 'Finding Connected Components and Connected Ones On a Mesh-Connected Parallel Computer,' *SIAM Journal on Computing*, Vol. 9, No. 4, 1980.

[75] Nassimi, D., S. Sahni, 'Bitonic Sort on a Mesh-Connected Parallel Computer,' *IEEE Transactions on Computers*, C-28, pp. 2–7, 1979.

[76] Nassimi, D., S. Sahni, 'An Optimal Routing Algorithm for Mesh-Connected Parallel Computers,' *Journal of the ACM*, 27, pp. 6–29, 1980.

[77] Newman, I., A. Schuster, 'Hot-Potato Algorithms for Permutation Routing,' *Proc. Israeli Symposium on Theory of Computing Systems*, 1993.

[78] Newman, I., A. Schuster, 'Worm Routing is almost as easy as Packet Routing,' *Proc. of the Israeli Symposium on Theory of Computing Systems*, pp. 202–211, IEEE, 1993. Journal of Parallel and Distributed Computing, to appear.

[79] Nakano, K., 'A Bibliography of Published Papers on Dynamically Reconfigurable Architectures,' *Parallel Processing Letters*, 5(1), pp. 111–124, 1995.

[80] Ngai, J.Y., C.L. Seitz, 'A Framework for Adaptive Routing in Multicomputer Networks,' *Proc. ACM Symposium on Parallel Algorithms and Architectures*, pp. 1–9, 1989.

[81] Ni, L.M., Ph.K. McKinley, 'A Survey of Wormhole Routing Techniques in Direct Networks,' *IEEE Computer*, 26(2), pp. 62–76, 1993.

[82] Nigam, M., S. Sahni, 'Sorting n Numbers on $n \times n$ Reconfigurable Meshes with Buses,' *Journal of Parallel and Distributed Computing*, 23, pp. 37–48, 1994.

[83] Pietracaprina, A., F.P. Preparata, 'An $\mathcal{O}(\sqrt{n})$-Worst-Case-Time Solution to the Granularity Problem,' *Proc. 10th Symposium on Theoretical Aspects of Computer Science*, LNCS 665, pp. 110–119, 1993.

[84] Pietracaprina, A., G. Pucci, J.F. Sibeyn, 'Constructive Deterministic PRAM Simulation on a Mesh-Connected Computer,' *Proc. 6th Symposium on Parallel Algorithms and Architectures*, pp. 248–256, ACM, 1994.

[85] Rajasekaran, S., 'Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing, Sorting, and Selection,' *Proc. European Symposium on Algorithms*, LNCS 726, pp. 309–320, Springer-Verlag, 1993.

[86] Raghavan, P., 'Robust Algorithms for Packet Routing in a Mesh,' *Proc. Symposium on Parallel Algorithms and Architectures*, pp. 344–350, ACM, 1989.

[87] Rajasekaran, S., R. Overholt, 'Constant Queue Routing on a Mesh,' *Journal of Parallel and Distributed Computing*, 15, pp. 160–166, 1992.

[88] Rajasekaran, S., M. Raghavachari, 'Optimal Randomized Algorithms for Multipacket and Cut Through Routing on the Mesh', *Journal of Parallel and Distributed Computing*, 26(2), pp. 257–260, 1995.

[89] Rajasekaran, S., Th. Tsantilas, 'Optimal Routing Algorithms for Mesh-Connected Processor Arrays', *Algorithmica*, 8, pp. 21–38, 1992.

[90] Roberts, A., A. Symvonis, 'On Deflection Worm Routing on Meshes,' *Techn. Rep. 490*, Dep. of Computer Science, University of Sydney, Sydney, 1994.

[91] Sado, K., Y. Igarashi, 'Some Parallel Sorts on a Mesh-Connected Processor Array and their Time Efficiency,' *Journal of Parallel and Distributed Computing*, 3, pp. 398–410, 1986.

[92] Savari, S.A., 'Average Case Analysis of Five Two-Dimensional Bubble Sorting Algorithms,' *5th Symposium on Parallel Algorithms and Architectures*, pp. 336–345, ACM, 1993.

[93] Scherson, I.D., S. Sen, 'Parallel Sorting in Two-Dimensional VLSI Models of Computation,' *IEEE Trans. Comput.*, 38, pp. 238–249, 1989.

[94] 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.

[95] Van Scoy, F.L., 'The Parallel Recognition of Classes of Graphs,' *IEEE Transactions on Computers*, 29, pp. 563–570, 1980.

[96] Sibeyn, J.F., B.S. Chlebus, M. Kaufmann, 'Shorter Queues for Permutation Routing on Meshes,' *Proc. 19th Symposium on the Mathematical Foundations of Computer Science*, LNCS 841, pp. 597–607, Springer-Verlag, 1994.

[97] Sibeyn, J.F., 'Matrix Techniques for Faster Routing of Affine Permutations on a Mesh Interconnection Network,' *Techn. Rep. RUU-CS-90-17*, Dep. of Comp. Sc., Utrecht University, Utrecht, the Netherlands, 1990.

[98] Sibeyn, J.F., 'Routing Permutations on Mesh Interconnection Networks,' *Proc. 2nd Symposium on Parallel and Distributed Processing*, pp. 94–97, IEEE, 1990.

[99] Sibeyn, J.F., 'Deterministic Routing and Sorting on Rings,' *Proc. 8th International Parallel Processing Symposium*, pp. 406–410, IEEE, 1994.

[100] Sibeyn, J.F., 'Desnakification of Mesh Sorting Algorithms,' *Proc. 2nd European Symposium on Algorithms*, LNCS 855, pp. 377–390, Springer-Verlag, 1994.

[101] Sibeyn, J.F., 'Independent Sets and List Ranking on Meshes,' *Proc. Computing Science in the Netherlands*, pp. 271–280, SION, Amsterdam, 1994.

[102] Sibeyn, J.F., 'List Ranking on Interconnection Networks,' Submitted to *FST & TCS '95*.

[103] Sibeyn, J.F., M. Kaufmann, 'Deterministic 1-k Routing on Meshes,' *Proc. 11th Symposium on Theoretical Aspects of Computer Science*, LNCS 775, pp. 237–248, Springer-Verlag, 1994.

[104] Sibeyn, J.F., M. Kaufmann, 'Solving Cheap Graph Problems on Meshes', *Proc. 20th Symposium on the Mathematical Foundations of Computer Science*, LNCS, Springer-Verlag, 1995, to appear. Full version in *Techn. Rep. WSI-95-9*, Universität Tübingen, Tübingen, Germany, 1995.

[105] Sibeyn, J.F., M. Kaufmann, R. Raman, 'Randomized Routing on Meshes with Buses,' *Proc. 1st European Symposium on Algorithms*, LNCS 726, pp. 333–344, Springer-Verlag, 1993.

[106] Suel, T., 'Routing and Sorting on Meshes with Row and Column Buses,' *Proc. 8th International Parallel Processing Symposium*, pp. 411–417, IEEE, 1994.

[107] Thompson, C.D., H.T. Kung, 'Sorting on a Mesh Connected Parallel Computer,' *Communications of the ACM*, 20, pp. 263–270, 1977.

[108] Valiant, L.G., G.J. Brebner, 'Universal Schemes for Parallel Communication,' *Proc. 13th Symposium on Theory of Computing*, pp. 263–277, ACM, 1981.

[109] Wang B., G. Chen, F. Lin, 'Constant Time Sorting on a Processor Array with a Reconfigurable Bus-System,' *Information Processing Letters*, 34(4), pp. 187–192, 1990.

[110] Wanka, R., 'Fast General Sorting on Meshes of Arbitrary Dimension without Routing,' *Techn. Rep. Nr. 87, Reihe Informatik*, Fachbereich Mathematik-Informatik, Universität GH Paderborn, Paderborn, Germany, 1991.

Index