

# PACKET ROUTING IN FIXED-CONNECTION NETWORKS: A SURVEY

Miltos D. Grammatikakis  
Institut für Informatik  
Universität Hildesheim  
31141 Hildesheim, GERMANY

D. Frank Hsu  
Department of Computer and Information Science  
Fordham University  
Bronx, NY 10458, USA

Miro Kraetzl  
Communications Division  
Defence Science and Technology Organisation  
Salisbury, SA 5108, AUSTRALIA

Jop F. Sibeyn  
Max-Planck-Institut für Informatik  
Im Stadtwald, 66123 Saarbrücken, GERMANY

**Proposed running head:**

Grammatikakis et al. Packet Routing in Fixed-Connection Networks: A Survey

**The address for all correspondence:**

Miro Kraetzl  
Communications Division, DSTO  
PO Box 1500  
Salisbury, SA 5108, AUSTRALIA  
Telephone: +61 8 8259 6757  
Fax: +61 8 8259 6549  
Email: miro.kraetzl@dsto.defence.gov.au

**Abstract**

We survey routing problems on fixed-connection networks. We consider many aspects of the routing problem and provide known theoretical results for various communication models. We focus on (partial) permutation,  $k$ -relation routing, routing to random destinations, dynamic routing, isotonic routing, fault tolerant routing, and related sorting results. We also provide a list of unsolved problems and numerous references.

**List of symbols:**

$$x \equiv x_1 x_2 \dots x_n$$

$$x_i \in \{0, 1\}$$

$$\bar{x}_i = 1 - x_i$$

$$\pi(x) = x' \equiv x'_1 x'_2 \dots x'_n$$

$$|x - y|$$

$$x \mapsto \pi(x)$$

$$x : (x_j \leftarrow \alpha)$$

$$\{1, 2, \dots, N^\varepsilon\}$$

$$\Delta$$

$$d$$

$$\lambda$$

$$b$$

$$\Theta()$$

$$\Omega()$$

$$\mathcal{O}()$$

$$o()$$

$$N!$$

$$\log N$$

$$\log_b N$$

$$x_1 \implies x_3 \implies x_2 \implies \bar{x}_1$$

$$\lfloor N \rfloor$$

$$\lceil N \rceil$$

$$x_i \in \langle d \rangle = \{0, 1, 2, \dots, d - 1\}$$

# 1 Introduction

With advances in the VLSI technology it has become feasible to build multicomputers consisting of hundreds, or even thousands of processor nodes with local memory, which communicate with each other over a fixed interconnection network. An essential condition for the efficient use of such machines are routines for exchanging data between the processors. In view of the many network topologies and the multitude of communication patterns, it is not surprising that a rich body of theoretical and practical studies has been developed around the theme of communication. In this introduction we will first probe into several of the topics that are covered in more detail in later sections.

## 1.1 Communication Models

Communication in multicomputers is usually performed in *packet routing* mode; a message along with its destination address is placed in a packet that moves through the nodes of the interconnection network without establishing a physical connection path between the packet's source and destination. In contrast to packet routing, *circuit switching* allows the data to move through the network only after establishing a physical connection path between the source and destination. Routing along a network of buses is somewhat in between these two extremes. Here a connection is not switched, but it is up to the algorithm designer to guarantee that no two packets are using a bus at the same time.

In the *SIMD communication model* every routing step is restricted to a single dimension; every node sends and receives at most one packet along a single direction. In the *MIMD model* (also called *all-port model*) every node can communicate with all its neighbors in a single step.

The simplest packet routing model is *store-forward*, also called *node to neighbor communication*. In this model, each node has a store/send capability, it sends a packet and waits for an acknowledge message from the receiver. In the *constant* model, transmission of a packet from one network node to any of its neighbors takes one time step. In the *linear* model, this transmission takes  $\alpha + \beta b$  time, where  $\alpha$  is the start-up time,  $b$  is the size of the packet transmitted, and  $\beta$  is the transmission rate. With this strategy, a packet incurs a delay which is at least proportional to the distance it has to travel.

In order to reduce the distance dependency, alternative packet-routing models have been proposed. With *wormhole routing*, every node has a queue associated with each of its adjacent links which can hold a small number of bits. A packet may be spread over many nodes on the path from its source to its destination and progresses only if there is space in the queue at the next node. When the head moves, or when bits are consumed by the destination node, the entire packet can move by moving to the free space created [61]. The unit chunks in which packets are divided are called *flits*. If there are no other packets interfering, then the transfer of a packet consisting of  $b$  flits over distance  $d$  takes  $\alpha + \beta b + \gamma d$  time. Here  $\alpha$ ,  $\beta$  and  $\gamma$  are constants. As mostly  $\beta$  and  $\gamma$  are of the same order of magnitude, while  $b \gg d$ , the transfer time is almost distance independent. *Virtual cut-through* is similar to wormhole routing, except that the entire packet can pile up at a node, since sufficient queuing is provided [147].

For efficient packet routing, the total communication delay (propagation and queuing) must be small, while the system maintains high reliability, i.e. multiple disjoint paths between nodes exist, and transmission errors are rare. Also, the queue size, defined as the maximum number of packets stored at any network node during communication, should be kept as small as possible. This is an important design consideration because of physical constraints arising from buffer space limitations and required fast access to buffer elements. Various models have been developed to specify more precisely these criteria.

## 1.2 Communication Patterns

**General Routing.** Many communication problems are special instances of the following  $(N, p, k_1, k_2)$ -routing problem.  $N$  packets, each with its own source and destination, must be routed such that at most  $k_1$  packets are initially at any node, and at most  $k_2$  packets are finally at any node. The  $N$  packets reside on  $p$  nodes. Regular topologies offer the advantage that all nodes have a global knowledge of the network, allowing for simple routing and scheduling decisions.

Special algorithms are also interesting if many of the packets that are sent or received by a node are the same. Of particular importance are the following basic operations: (a) A *single node broadcast* involves the transfer of a message from a particular node to all other network nodes; (b) A *single node scatter* is similar to single node broadcast except that  $N$  different messages are broadcasted; (c) A *multinode broadcast* involves the simultaneous single node broadcast from all network nodes (there are  $N$  different messages); (d) A *total exchange* (also called *gossiping*) is similar to multinode broadcast, except that all the packets sent are different; (e) A *single node accumulate* (also called *gather*) is the dual operation to single node scatter; and (f) A *multinode accumulate* is the dual operation to multinode broadcasting.

Multinode broadcast can be implemented by performing a single-node gather followed by a single node scatter. On the other hand, single-node gather, which has the same complexity as single-node scatter, is a subproblem of multinode broadcast. Therefore, all three problems have the same complexity up to constant factors. The same is true for the multinode accumulate. For problems (a)-(f) above, pipelining techniques and edge-disjoint spanning trees effectively reduce the delay time [82, 104, 109, 122, 155, 295].

**Isotonic Routing.** *Isotonic routing* refers to any of the following routing schemes. A *semi-contraction* is a special case of partial permutation routing; the absolute distance  $|x - y|$  between the initial locations of any two packets  $x, y$  is always greater than or equal to the absolute distance  $|\pi(x) - \pi(y)|$  between their final destinations. Distances refer to integer differences between node representations, and not Hamming distances. A *semi-expansion* is the dual operation to a *semi-contraction*. *Packing* corresponds to partial permutation routing of  $M \leq N$  packets onto a single interval (e.g. the first  $M$  locations), while preserving their relative order. *Unpacking* is the dual operation to packing. Thus, packings and unpackings are special cases of semi-contractions and semi-expansions. A *semi-broadcast* refers to routing  $M \leq N$  packets, when the final destinations of packets from different nodes are ordered, i.e. packets at nodes  $x$  and  $y$  ( $x < y$ ) are headed to a set of destinations  $S_x$  and  $S_y$ , such that any member of  $S_x$  is smaller than any member of  $S_y$ . A *semi-gather* is the dual operation to a semi-broadcast.

**Permutation Routing.** Another special case of  $(N, p, k_1, k_2)$ -routing is *permutation routing*. In this problem, there are initially  $N$  packets, one at each of the  $N$  processor nodes. The packet at node  $x$ ,  $0 \leq x \leq N - 1$  is destined to node  $\pi(x)$ , where the mapping  $x \mapsto \pi(x)$  is a *permutation*. The aim is to *route* the packets in *parallel*, given that at most one packet can traverse each link at each time step. If the permutation is known in advance an optimal routing schedule can be precomputed and the routing is *off-line*, while if the permutation is known on the fly the routing is *on-line*. Frequently used permutations belong to the *BPC*, *Omega*, *Inverse Omega*, and *Ascend/Descend* classes [171, 186, 223, 299]. Off-line routing is useful when data dependencies are known before run time, as in scientific applications [85].

On-line routing may be further classified into *oblivious* and *adaptive* strategies. A routing algorithm is said to be *deterministic oblivious* if the route of any packet depends only on its origin and destination, while it is *randomized oblivious* if the route of any packet is independently chosen according to a probability distribution (which is a function of its origin and destination) [2]. An *adaptive* strategy makes routing decisions for a packet depending on its current node and destination of packets encountered on its way. In a faulty system,

adaptive routing may also check the status of adjacent links and nodes. Randomized and adaptive routing has been implemented in hardware, initially on the Chaos router [149], and more recently on commercial systems, such as the CM-5 data network [185], and the STC104  $32 \times 32$  routing chip [310].

### 1.3 Optimality and Lower Bounds

Many papers in our list of references consider the “optimality” of algorithms which may be a more or less strong notion. Sometimes it means that the time consumption of the algorithm exceeds some lower bound by a constant factor only [31, 32, 49]; it may mean that the leading constants of a lower bound and the algorithm match [131, 133, 289]; and sometimes it means that the time consumptions are identical [182, 262, 293]. The latter two types of optimality are mainly applied to meshes (because it is mostly trivial to achieve the first optimality on meshes).

The most commonly applied lower bounds for communication algorithms working under the store-forward assumptions are the following:

**Distance bound.** A packet travels at most over distance 1 during a single routing step. Thus, a routing algorithm takes at least the maximum distance a packet has to go.

**Throughput bound.** At most one packet can go over a connection during one routing step. Thus, a routing algorithm takes at least the maximum number of packets which have to pass through a single connection.

**Bisection bound.** A subset of  $c$  network connections can obviously transfer at most  $c$  packets during one routing step. When these connections “bisect” the interconnection network into two parts, with  $m$  packets moving from one side to the other for a certain routing problem, then any routing algorithm will take at least  $m/c$  routing steps. We say that  $c$  forms a *network bisection*.

**Connection availability bound.** Any network connection can be used for the transmission of at most one packet during one routing step. If the total number of network connections is  $c$ , then the routing of a certain distribution of packets requires at least  $D/c$  steps, where  $D$  denotes the sum, taken over all packets, of distances to travel.

In most cases where it is possible to prove a lower bound at all, the argument relies on one of these four bounds. Only rarely, for example in some routing problems on rings and sorting problems on meshes, a higher bound can be proven by an argument based on the initial lack of global knowledge [156, 288].

### 1.4 Routing Strategies

In standard routing strategy packets are sent along a more or less cleverly chosen path. In commercial architectures, the router is usually part of the hardware and very simple. A very common strategy (on grids of various dimensions and hypercubes) is the so called *dimension-order routing*, in which a packet is routed along direction 0 until it has reached its correct position (along this dimension), then along dimension 1, and so on. Such non-adaptive strategies are extremely vulnerable to faults and hot spots in the network.

Therefore, several adaptive strategies have been proposed. In *hot-potato routing* (also called deflection routing) there are no buffers at intermediate nodes [16]. Packets continue to move, possibly in a wrong direction, until they ultimately reach their destinations.

With the *mad-postman* strategy, a packet is sent to the next node in the virtual network, even before compiling its address information [120, 121]. Packets moving in virtual networks

ensure deadlock-free communication (see Section 1.5).

**Randomization.** In the design of efficient routing algorithms randomization plays a key role. *Probabilistic routing* uses a sequence of random bits for making routing decisions. The routing algorithm utilizes randomization to convert the input distribution, in which the deterministic algorithm suffers from unexpected worst case instances, into a manageable and predictable input distribution, usually the uniform distribution. This technique is useful when a deterministic algorithm for a certain problem runs on average much faster than in the worst case. Recently, Brassard and Bratley defined this type of randomization as *Sherwood-type* [37]. Ben-David et al. [23] conjectured that every randomized algorithm has the same performance as an appropriately designed on-line adaptive algorithm.

*Routing to random destinations* corresponds to routing  $N$  packets from distinct nodes to independently chosen random destinations. *Partial permutation routing* corresponds to routing with at most one packet originating from any node and at most one packet destined to any node. A  $k$ -*relation*, called also  $k$ - $k$  routing, corresponds to  $(kN, N, k, k)$ -routing. In *dynamic routing* packets are generated at every node with a fixed rate and head to independently chosen random destinations. Whether this assumption is justified depends on the particular application.

**Reduction of Problems.** Some reductions of the routing problems are possible. Permutation routing can be reduced to sorting, when sorting is on the unique final destination of the packets. Thus, if we can sort  $N$  data elements in a given amount of time, we can perform any permutation in the same time. Similarly, partial permutation routing can be reduced to sorting and unpacking as follows. First, nodes without messages send a dummy packet to destination  $N + 1$ . After sorting the packets according to their destination address,  $M$  genuine packets move to the nodes indexed 0 through  $M - 1$ . The dummy packets can now be discarded. Since the remaining packets are ordered according to their final destination, their routing corresponds to performing an unpacking operation. Integer sorting is equivalent to partial permutation routing if the integers to be sorted are consecutive numbers, i.e.  $\{1, 2, \dots, N^\varepsilon\}$ ,  $0 < \varepsilon < 1$  [152].

Clearly, routing-by-sorting imposes some conditions on the problem and the hardware. For example, a sorting problem should be solved statically, and this almost excludes that such an algorithm can be extended to dynamic inputs. In reaction to this Borodin [34] gives a detailed analysis of the so-called *pure packet routing*, a notion which he tries to define so that impractical tricks are excluded. Similar limitations are considered in [150, 199].

## 1.5 Deadlock, Livelock and Starvation

To ensure correct functionality the parallel system communication facility must be free of deadlock, livelock, and starvation.

A deadlock may be defined as a cyclic dependency of ungranted packet requests for buffer or channel resources. Deadlocks may occur when a common buffer pool is shared by incoming and outgoing traffic. A livelock refers to packets circulating the network without making any progress towards their destination. Livelock does not occur with shortest-path or randomized routing; it may be avoided with adaptive routing strategies by implementing age-based priority schemes [232]. Both livelock, and starvation, which refers to packets arriving at their destination but never been consumed by the processor, reflect problems of fairness in network routing or processor scheduling policies.

Special care has been taken to address network deadlock. While for non-adaptive store-forward and virtual cut-through routing, deadlocks can be avoided by using the concept of structured transmission buffer pools and route assignment to avoid cycles, special care must be taken for wormhole routing [70]. The usual way to provide deadlock-free routing for wormhole routing, is to define a partial order on the buffer classes which result in cycle-free (virtual) channel dependency graphs [61]. Although this works fine for deterministic routing,

in case of adaptive routing it is possible that a cycle exists but the packets still do not deadlock. A theory has been developed for designing maximally-adaptive, deadlock-free (but not necessarily livelock-free) routing algorithms in networks with nodes implementing either central buffering, or input/output buffering [67]. Minimizing the number of buffers needed, while providing for maximal adaptivity, has also been considered for tori [53].

For wormhole routing several other techniques to achieve deadlock-free routing are aimed at breaking up cycles in the channel dependency graphs. For example, this is done by using *initially adaptive followed by deterministic dimension-order routing* (after a certain number of dimension reversals) [60]. The other possibility is to divide packets into different *virtual networks* depending on their destination and assigning one buffer for each virtual network at each node [59], or by avoiding certain turns which cause cycles (e.g. using west first, north last, or negative first routing on meshes, tori, and other sparse networks) [86]. More efficient techniques could allow for more routing adaptivity; there exists a necessary and sufficient condition for deadlock-free wormhole routing on any network [68]. The condition has been extended to wormhole multicasting [69]. For deadlock-free multicasting on meshes using virtual networks see [191, 192].

More recent techniques for achieving deadlock-free routing consider the sparsity of deadlocks in asynchronous networks. Thus, instead of avoiding deadlocks, precautions are taken to efficiently recover from them. In the compressionless approach, any established virtual channels which have not reached their destination within a given time delay are “torn off” [148]; this method is implemented using special test packets which follow flow control paths. Alternatively, a new better approach called Disha<sup>1</sup> provides alternative paths on demand to deliver deadlocked packets [11, 247].

## 1.6 Covered Topics and Further Reading

In this study, we survey multicomputer routing, and provide known results and open problems. We concentrate on *permutation routing* and the related problems of *sorting*, *partial permutation routing*, *dynamic routing*, *routing to random destinations*, *k-relations*, *isotonic routing*, and *fault tolerant routing*. These problems are important in many parallel applications, such as computing multidimensional FFT, finite elements, matrix problems, and divide and conquer strategies. Furthermore, while many to one routing is used for simulating the CRCW and CREW PRAM (parallel random access memory) models on interconnection networks, permutation routing is employed for EREW PRAM simulations.

We consider general undirected graphs, especially meshes and hypercubes, but also star graphs, and related networks. Although architectural issues, such as virtual shared memory, cache coherence, and multithreading, play a prominent role in the design of massively parallel systems, numerous platform-independent parallel solutions are derived directly from either mesh or hypercube based algorithms. Unless otherwise specified, we assume full duplex (bidirectional) links, and the constant model for node to neighbor communication. For detailed description and analysis of some important communication algorithms contained here, the reader is referred to [176]. Parallel algorithms, mainly for meshes and hypercubes, are covered in [75, 118, 155, 269]. Sorting algorithms are examined in [6, 45, 167, 271].

In our survey, we concentrate on packet routing algorithms on fixed-connection networks, for which theoretical results are richer. We do not consider communication properties of multistage interconnection networks [74, 248], parallel models of computation and simulations among them [198, 316], analytical approaches based on queuing theory [30], network reliability and dependability [62, 101], optical networks and high-performance broadband networks [15, 325], and issues concerning the design of communication networks in actual parallel systems [114, 177, 185, 259].

---

<sup>1</sup>Disha means “direction” in Hindi.

## 2 Results for General Graphs

### 2.1 Common Permutations

Suppose that an arbitrary packet initially located at node  $x \equiv x_1 x_2 \dots x_n$  is destined to the node  $\pi(x) = x' \equiv x'_1 x'_2 \dots x'_n$ . For the *Bit Permute and Complement (BPC)* class of permutations  $\pi$  is uniformly defined; for any  $x$ ,  $0 \leq x \leq N - 1$ ,  $\pi(x)$  is a permutation of  $w \equiv w_1 w_2 \dots w_n$ , where either  $w_i = x_i$ , or  $w_i = \bar{x}_i = 1 - x_i$  (complement of  $x_i$ ). Examples of *BPC* permutations are:

1. Bit reversal:

$$\pi_1(x) = x_n x_{n-1} \dots x_2 x_1$$

2. Matrix transposition:

$$\pi_2(x) = \begin{cases} x_{l+1}x_{l+2} \dots x_{2l}x_1x_2 \dots x_l, & \text{if } n = 2l \\ x_{l+1}x_{l+2} \dots x_{2l+1}x_1x_2 \dots x_l, & \text{if } n = 2l + 1 \end{cases}$$

3. Shuffle:

$$\pi_3(x) = x_2 x_3 \dots x_n x_1$$

4. Vector reversal:

$$\pi_4(x) = \bar{x}_1 \bar{x}_2 \dots \bar{x}_n$$

5. Bit shuffle:

$$\pi_5(x) = \begin{cases} x_1x_3 \dots x_{n-1}x_2x_4 \dots x_n, & \text{if } n = 2l \\ x_1x_3 \dots x_nx_2x_4 \dots x_{n-1}, & \text{if } n = 2l + 1 \end{cases}$$

6. Unshuffle:

$$\pi_6(x) = x_n x_1 x_2 \dots x_{n-2} x_{n-1}$$

7. Shuffle row major:

$$\pi_7(x) = \begin{cases} x_1x_{l+1}x_2x_{l+2} \dots x_lx_{2l}, & \text{if } n = 2l \\ x_1x_{l+2}x_2x_{l+3} \dots x_lx_{2l+1}x_{l+1}, & \text{if } n = 2l + 1 \end{cases}$$

8. Butterfly:

$$\pi_8(x) = x_n x_2 x_3 \dots x_{n-1} x_1$$

9. Exchange:

$$\pi_9(x) = x_1 x_2 \dots x_{i-1} \bar{x}_i x_{i+1} x_{i+2} \dots x_n$$

The *BPC* permutations also include the *s sub-class* and *s super-class* of permutations (an *s* sub-bit reversal corresponds to a bit reversal permutation applied to the *s* least significant digits). *BPC* permutations are common in scientific applications [125]. The task of computing the transpose of a matrix arises often in matrix algorithms like in multidimensional FFT, computing congruence matrix transformations  $UAU^T$ , or executing a step of the *QR* algorithm for finding eigenvalues of a matrix. The matrix transposition permutation is also used for conversion of a matrix stored with one row per processor (called consecutive row) to the storage of one column per processor (called consecutive column), and for changing the allocation of an array from consecutive to cyclic storage representation (and vice-versa). Other *BPC* permutations are useful for conversions between: (a) consecutive row matrix representation, (b) consecutive column storage, (c) cyclic row, (d) cyclic column, and (e) combinations of cyclic/consecutive row (or column) representation. They are also useful for conversion between different matrix embeddings, such as binary and binary-reflected Gray

code on hypercubes. *BPC* permutations have been extended to *affine* permutations, under which  $i'$ , the index of the node to which the  $i^{\text{th}}$  node routes its packet, is given by  $\bar{i}' = A\bar{i} + \bar{b}$ . Here  $\bar{i}$  and  $\bar{i}'$  denote the binary expansion of  $i$  and  $i'$ , respectively,  $A$  is an invertible 0-1 matrix and  $\bar{b}$  is a 0-1 vector.

Omega (or Inverse Omega) permutations are realizable by a  $\log N$ -stage<sup>2</sup> Omega network (respectively, Inverse Omega) without conflict at any stage. More precisely, consider  $N = 2^n$  nodes represented by binary numbers from 0 to  $N - 1$ . For any two nodes  $x$  and  $y$ , let  $L(x, y)$  (and  $M(x, y)$ ) represent the number of similar least significant bits, up to the first different bit (respectively, similar most significant bits). Notice that,  $0 \leq L(x, y), M(x, y) \leq \log N$ . A permutation  $\pi$  is an *Omega permutation* if and only if for any nodes  $x$  and  $y$ , we have  $L(x, y) + M(\pi(x), \pi(y)) < \log N$ . We call  $\pi$  an *Inverse Omega permutation* if for any  $x$  and  $y$ , we have  $M(x, y) + L(\pi(x), \pi(y)) < \log N$ . Omega and Inverse Omega permutations are useful in parallel applications, such as parallel load and store on matrices, simultaneous conflict-free access to all rows, columns, diagonals, and blocks of arrays, and divide and conquer strategies [171]. Some examples of Omega and Inverse Omega permutations, which do not belong to the *BPC* class, are:

10. Cyclic shift of amplitude  $k$ ,  $1 \leq k \leq N$ :

$$\pi_{10}(x) = (jx + k) \bmod N,$$

where  $j$  is odd.

11. Cyclic shift within segments:

$$\pi_{11}(x) = \delta_2(x + k) \bmod 2^{n-j},$$

where  $\delta_2$  is the (decimal) number equivalent to the  $j$  most significant bits in the binary representation of  $x$ , and  $1 \leq k \leq N$ .

12. Unscrambling  $j$ -ordered vectors:

$$\pi_{12}(x) = (jx) \bmod 2^k + (x_n, x_{n-1}, \dots, x_{k+1}) 2^k,$$

$j$  is odd,  $1 \leq k \leq N$ .

With *normal Ascend* (and *Descend*) *permutations*, node  $i$  ( $0 \leq i < N = 2^n$ ) communicates with nodes  $(i \text{ XOR } 2^b)$  for all  $b = \{0, 1, \dots, n-1\}$  (respectively,  $b = \{n-1, n-2, \dots, 0\}$ ) [255]. In contrast, in the superclass of  $2^b$  *Ascend* (and  $2^b$  *Descend*) *computations*, node  $i$  communicates with nodes  $(i + 2^b) \bmod N$  and  $(i - 2^b) \bmod N$ , for all  $b = \{0, 1, \dots, n-1\}$  (respectively,  $b = \{n-1, n-2, \dots, 0\}$ ) [223]. Both normal Ascend/Descend permutations, and  $2^b$  Ascend/Descend computations arise in efficient parallel algorithms based on the divide and conquer primitive for many problems, including polynomial evaluation, inner product, linear recurrence, triangular system evaluation, Batcher's odd-even mergesort and bitonic sorting [115, 211, 220, 221, 223]. These patterns also occur in software implementations of scan/prefix operations and logarithmic complexity dissemination barriers [211, 220].

## 2.2 Routing and Sorting - Performance Bounds

Borodin and Hopcroft considered the possibility that certain communication patterns cause node congestion, leading to large queuing delays [35]. Assuming a graph with  $N$  nodes, and degree  $\Delta$ , equipped with  $n$ -port communication they proved a lower bound of  $\Omega(N^{1/2}/\Delta^{3/2})$  for the class of deterministic oblivious permutation routing algorithms. More recently, Kalamanis et al. and Borodin et al. improved this lower bound by considering link congestion:

---

<sup>2</sup>All logarithms in this paper are to the base 2, unless otherwise indicated.

**Theorem 2.1** *Deterministic permutation routing on a graph with  $N$  nodes and degree  $\Delta$  requires:*

1.  $\Theta(N^{1/2}/\Delta^{1/2})$  steps for oblivious algorithms with SIMD model [35, 36]; or
2.  $\mathcal{O}(\log N)$  steps for adaptive algorithms with SIMD model [36].
3.  $\Theta(N^{1/2}/\Delta)$  steps for oblivious algorithms with MIMD model [36, 134]; or
4.  $\Theta(\log_{\Delta} N)$  steps for adaptive algorithms with MIMD model [36].

Given a set of packet routes on a given network, we define the *dilation* ( $W$ ) as the maximum distance traveled by any packet, and the *congestion* ( $U$ ) as the largest number of packets that must traverse a single link. Leighton et al., using a nonconstructive proof, which can be viewed as an off-line algorithm, proved:

**Theorem 2.2** *There exists a schedule for routing a set of packets whose paths have congestion  $U$  and dilation  $W$  on an MIMD network in  $\mathcal{O}(U + W)$  steps using constant size queues [179].*

Recently, a constructive proof and improvements have been provided [180, 256]. Leighton et al. provided an algorithm for routing packets on a leveled network of constant degree  $\Delta$  and depth  $L$ . All links in such a network are directed, from a level  $i$  node to a level  $i + 1$  node, where  $0 \leq i \leq L - 1$ . Considering permutations, from the  $N$  inputs at level 0 to the  $N$  outputs at level  $L$ , they proved the following:

**Theorem 2.3** *There is an on-line scheduling algorithm which routes  $N$  packets whose paths have congestion  $U$  on any bounded-degree leveled MIMD network in  $\mathcal{O}(U + L + \log N)$  steps using constant size queues, with high probability [178].*

The algorithm is a generalization to Ranade's permutation routing strategy on the hypercube [267], which was originally intended for implementation of an efficient simulation of shared memory on a distributed memory machine.

It is based on ghost packets and random rankings assigned to packets, and uses only  $\mathcal{O}(\log^2 N)$  random bits. For messages  $b$ -bits long this algorithm can be pipelined, achieving  $L + cb(U + L + \log N)$  time delay, with high probability ( $c$  is a constant). It is assumed that each queue can hold at least  $b$  packets. This result was later used for permutation routing on the hypercube, mesh,  $n$ -dimensional mesh, and shuffle/exchange graph in asymptotically optimal time. However, the multiplicative constant of the dominant term was ignored. Furthermore, the above result provides the best known approximation ratio for the general job-shop scheduling problem.

Recently, Palis et al. [236] considered nonconstant degree leveled networks. By selecting a shortest path for all (source, destination) pairs on the original network, a new leveled network is constructed with  $L + 1$  levels,  $N$  nodes per level, and links between: (a) node  $u$  at level  $i$  and node  $u$  at level  $i + 1$ , and (b) node  $u$  at level  $i$  and node  $v$  at level  $i + 1$ , if there is a shortest path whose  $i^{\text{th}}$  link connects  $u$  and  $v$  in the original network. A two-phase routing algorithm first forwards the packets to independently chosen random destinations, and then sends them to their final destinations along a unique shortest-path determined by (b)-type links first, and then, if the path is shorter than  $L$ , by (a)-type links. The two-phase routing strategy is optimal, provided that the leveled network has degree  $\Delta \geq 2$ , and it is *nonrepeating*, i.e. any two paths that share a link and then diverge never share a link again.

**Theorem 2.4** *Any permutation of  $N$  packets can be routed on a leveled MIMD network, with  $LN$  nodes and  $L + 1$  levels, in  $\mathcal{O}(L)$  steps, with high probability [236].*

Although applying the above algorithm to the mesh,  $n$ -dimensional mesh, hypercube, shuffle/exchange, or star graph yields an asymptotically optimal time delay, the two-phase routing algorithm does not guarantee constant queue size.

Valiant pioneered the work on probabilistic routing. Let a symmetric scheme refer to traffic being uniformly distributed to all network links. We have:

**Theorem 2.5** *In any oblivious, nonrepeating, and symmetric routing scheme, an arbitrary  $k$ -relation can be routed on an MIMD  $N$  node, degree  $\Delta$  network, with maximal route length  $\mu$ , and expected route length  $\eta$  in just  $T$  steps (provided that  $T \leq (k \mu \eta)/\Delta$ ), with high probability ( $\geq 1 - k N ((e \eta \mu k)/(T \Delta))^T$ , where  $e = 2.718281\dots$ ) [314].*

Valiant also showed that any oblivious routing algorithm on a graph with an asymptotically optimal diameter (or mean route length), either requires  $\Omega(N^\varepsilon)$  steps (for some constant  $\varepsilon > 0$ ), or makes packets travel a path of length at least  $2 \log_\Delta N$  [315]. This, in turn, implies the asymptotic optimality of the probabilistic two-phase routing strategy for the shuffle/exchange, the  $n$ -dimensional  $n$ -way shuffle network and other sparse graphs.

Many graphs, including the mesh and the hypercube, can be written as a Cartesian product graph  $G \times H$ . By factoring permutations into elementary transpositions (called matchings), routing on the  $G \times H$  graph can be performed without any packet congestion, beyond that caused by permutation routing on  $G$  and  $H$ .

**Theorem 2.6** *Any permutation can be off-line routed on the MIMD  $G \times H$  network in  $T(G) + T(H) + \min\{T(G), T(H)\}$  steps, where  $T(G)$  ( $T(H)$ ) is the time required for off-line permutation routing on graph  $G$  (respectively, on graph  $H$ ) [10, 20].*

Theorem 2.6 also holds if each  $i^{\text{th}}$  copy of  $G$  in  $G \times H$ ,  $1 \leq i \leq N_G$  is permuted, where  $N_G$  is the number of nodes in graph  $G$ . A similar theorem applies when, each copy of  $G$  in  $G \times H$  contains a number  $f$  of faulty nodes [20]. Recently Youssef proved:

**Theorem 2.7** *Any Omega or Inverse Omega permutation can be off-line routed on the MIMD  $G \times H$  network in  $T(G) + T(H)$  steps [330].*

Expander graphs became increasingly important in search for asymptotically optimal permutation routing and sorting networks [127]. The AKS sorting network [3, 4] and the multibutterfly are the only two known bounded-degree networks which can sort in  $\mathcal{O}(\log N)$  bit steps with deterministic algorithms [200, 248]. Recent proof that multibutterflies can sort is based on a constant load, congestion and dilation embedding of an  $N$ -node AKS network onto a  $\frac{3n}{2}$ -node degree-8 multibutterfly [200]. Hence, both the AKS, and the multibutterfly network can solve the  $(N, N, k_1, k_2)$ -routing problem in optimal  $\mathcal{O}(k_1 + k_2 + \log N)$  time.

The multibutterfly discovered by Bassalygo and Pinsker [18] has a simple randomly-wired multistage structure; it consists of butterfly networks merged together after randomly permuting switches at each level. Upfal discovered that this network can deterministically permute its inputs in  $\mathcal{O}(\log N)$  time [312]. Furthermore, Arora, Leighton and Maggs [12] proved that back-to-back multibutterflies (called also Beneš networks) are strict-sense nonblocking, i.e. any unused connections between any input-output network pairs can be self-routed in  $\mathcal{O}(\log N)$  time using the circuit switching model. The multibutterfly has sparked an interest in randomly-wired networks for packet routing in parallel systems [193, 197].

AKS networks are point-to-point networks with an irregular structure which does not admit good embeddings on nonexpander, regular topologies [52]. Although significant reductions on the size and depth of the AKS sorting networks have been made [4, 50, 239], the AKS sorters remain impractical compared to classical  $\mathcal{O}(\log^2 N)$  delay, bitonic sorters [19]. For example, there exists a class of the AKS sorting networks with depth  $1830 \log_2 N - 58657$  and size  $N \geq 2^{78}$  [50]; the smallest such network still has hundreds of times the depth of a  $2^{78}$ -node bitonic sorter. New bounds on the minimum size and depth of (comparison-based)

sorting networks are provided by Kahale et al. in [129]. Small sorting networks of optimal depth are known for  $n \leq 10$ , and for optimal size for  $n \leq 8$  [238].

For randomized sorting, first Reif and Valiant obtained optimal randomized sorting on a nonexpander graph, called the cube-connected cycle (see Section 4.7) [270]. Leighton and Plaxton provided efficient randomized sorting algorithms on the butterfly (the constant is only 7.45).

**Theorem 2.8**  *$N \log N$  packets can be sorted on the MIMD  $N \log N$  node butterfly in  $\mathcal{O}(\log N)$  steps using constant size queues, with high probability [183].*

For messages  $b$ -bits long, pipelining this algorithm achieves  $b + \mathcal{O}(\log N)$  bit steps, with high probability [183]. Leighton et al. using Theorem 2.8 and efficient embeddings of butterfly networks, proved that probabilistic sorting on the  $n$ -dimensional binary hypercube, and  $d$ -dimensional mesh ( $n \geq 2$ ) can be performed in asymptotically optimal time [178].

### 2.3 Fault Tolerant Routing

Multicomputer systems are more susceptible to failure than conventional uniprocessor machines. As the size of parallel systems increases, the probability of a component failure (either node, or link) also increases. We are interested in routing a packet from its source to its destination, if there is a live path between these two nodes, i.e. the nodes belong to the same connected component.

One solution consists of reconfiguring the faulty network, by embedding a fault-free network [102, 103, 130, 213, 306]. After the reconfiguration, classical communication algorithms can be used.

Rabin proposed the information dispersal approach (IDA) for reliable communication when messages are at least  $\Omega(\log^2 N)$  bits long [257]. The information contained in a packet is dispersed to  $m_1 = \mathcal{O}(\log N)$  copies, which are routed to the same destination using parallel paths. Using error-correcting codes the message contained in a packet can be reconstructed at its destination from any  $m_2 < m_1$  copies, where  $m_2 \approx m_1$  [257].

Another interesting possibility consists of routing in a fault tolerant fashion directly on the faulty network. We assume that the fault pattern remains fixed for the duration of routing. It is much harder to model algorithmic fault-tolerance in the presence of transmission errors, or temporarily down links. In the *p-faulty* model, component failures occur independently with a fixed probability ( $p$ ) and may correspond to node failures, link failures, or both. These dynamic faults may correspond to a channel being unavailable, or to a full buffer. In the *worst case* model, a set ( $F$ ) of faulty components (links and nodes) is chosen by an adversary. Such static faults could arise during fabrication runs. Notice that, when a node fails, all its adjacent links also fail.

A fault-tolerant communication algorithm is optimal if it finds a minimal feasible path for every packet, whenever such a path exists. A path is *feasible* if it contains no faulty nodes. A path is a *minimal* feasible path if it is the shortest feasible path. When faults are placed maliciously, we are interested in certain graph properties [110, 111]. In particular, good routing ( $\rho$ ) on network  $G$  corresponds to bounding the diameter ( $D_R$ ) of the *surviving route graph*  $R$  [66]. The directed graph  $R$  has the same number of vertices as graph  $G$ , and any two nonfaulty nodes  $x$  and  $y$  are joined by a link if there are no faults on the route  $\rho(x, y)$  between them. The surviving graph diameter is a measure of the worst case performance degradation caused by faults, for a given routing algorithm. Recall that a graph  $G$  is *k-node connected* if there exist  $k$  node-disjoint paths from any node  $x$  to any node  $y$  in  $G$ .

**Theorem 2.9** *If  $|F| \leq k$  (node or link faults) and  $G$  is  $(k + 1)$ -node connected, then there exists a routing algorithm  $\rho$  such that the diameter of the surviving route graph  $R$  is  $D_R \leq \max\{2k, 4\}$  [66]. In addition, if  $|F| \leq k/2$ , this bound becomes  $D_R \leq 4$  [241].*

**Theorem 2.10** *If  $\rho$  is a shortest-path routing on network  $G$ , and the set  $F$  consists of  $f$  faulty links and  $k$  faulty nodes with degrees  $\Delta_1, \Delta_2, \dots, \Delta_k$ , then the diameter of the surviving graph is  $D_R \leq 2f + 1 + \sum_{i=1}^k (\Delta_i - 1)$  [72].*

**Problem 1** *Find an upper bound on  $D_R$  if routing is of almost minimal length?*

**Problem 2** *Find an upper bound on  $D_R$  if randomized routing is used?*

A shortest-path routing  $\rho$  is  $(d, f)$ -tolerant, if for every choice of  $f$  (node or link) faults in  $G$  the surviving route graph has diameter  $D_R \leq d$ . A graph  $G$  is  $(d, f)$ -tolerant, if there exists a shortest-path routing  $\rho$  on  $G$  which is  $(d, f)$ -tolerant.

**Theorem 2.11** *If  $G$  is  $(d_G, f_G)$ -tolerant and  $H$  is  $(d_H, f_H)$ -tolerant, then graph  $G \times H$  is  $(\max\{d_G, d_H, 2\}, f_G + f_H + 1)$ -tolerant [38].*

Peleg and Simmons studied graph properties which lead to good bounds on the diameter  $D_R$  [241]. We can also pad graphs to build larger fault tolerant graphs [38].

## 2.4 PRAM Simulation on Distributed Memory Computers

Out of the observation that PRAM algorithms are (relatively) easy to design, but that most existing parallel computers have a distributed memory (except for parallel machines with a small number of nodes), a natural need arises to efficiently simulate PRAMs on distributed-memory machines (DMM). A rich theory has been developed, and here we want to provide pointers to the most relevant recent literature.

A PRAM simulation on a DMM is performed in rounds. In every round each node performs a step for all the PRAM processors it is simulating (possibly just one), and then a routing is performed to write results away and to read new data. If the access pattern is assumed to be randomly distributed, and if there is a lot of parallel slackness (implying that each node of the DMM simulates a large number of PRAM processors), one can hope that things work out fine. Because of Chernoff bounds, one can estimate that with high probability the random access pattern is actually highly balanced. The maximum number  $k_{\max}$  of packets that any node sends to any other node can easily be estimated<sup>3</sup> or determined. Therefore, one has a simple routing pattern in which each node sends at most  $k_{\max}$  packets to each other node.

However, in case that one cannot assume a random access pattern, or when there is insufficient parallel slackness, PRAM simulation implies more than just a routing problem. The real problem is the distribution of data over the memories of the DMM so that the necessary routing can be performed fast for all or most of the accesses.

One approach, which eliminates the assumption of random inputs, is to distribute the data according to some hash-function which is chosen uniformly from a universal class of hash functions [267]. Even sharper upper-bounds can be obtained if in addition to some clever distribution technique data are copied [57, 64, 87, 136, 209, 217]. In these papers, it is assumed that the network is completely connected (by an optical crossbar switch). Other authors have dealt with bounded degree networks [105, 244], or even with simulations on meshes and mesh-like networks [106, 187, 245, 246]. More recently the focus appears to be no longer on achieving even faster simulations (in [57], a step of a PRAM with  $N$  processors is simulated with a delay of  $\mathcal{O}(\log \log \log N \cdot \log^* N)$  only), but on more diverse aspects such as fault tolerant simulations [25], and simulations on reconfigurable networks [58].

---

<sup>3</sup>If in a given round of a PRAM simulation on a DMM with  $N$  nodes,  $k$  is the expected number of packets, then with high probability,  $k_{\max} \leq k + 2 \cdot (k \cdot \log(T \cdot N^2))^{1/2}$ , where  $T$  is the total number of rounds that are performed.

### 3 Mesh-Connected Multicomputers

A *two-dimensional*  $n \times n$  mesh consists of  $N = n^2$  processor nodes arranged in a two-dimensional  $n \times n$  grid. Each node is connected to its (at most) four neighbors. Nodes are identified by their mesh coordinates; the node at position  $(i, j)$ ,  $0 \leq i, j < n$ , is denoted by  $P_{i,j}$ , where position  $(0, 0)$  lies in the upper-left corner. All definitions carry on to nonsquare  $m \times n$  meshes.

Meshes and their direct generalizations are attractive because of regularity, scalability and conceptual simplicity. They are easy to program and are suitable for VLSI implementation. The major draw-back is large diameter and small bisection width ( $2n - 2$  and  $n$ , resp., for two-dimensional meshes). As a consequence, the maximal speedup for problems without much “locality” is  $n$ , with  $n^2$  nodes.

In addition to the SIMD and MIMD models, we distinguish the *uni-axial model*; in every routing step there is one axis along which communication can be performed (either horizontal, or vertical). Along this axis each node can send information to both of its neighbors, and receive from both of them.

A *torus* is defined as a mesh with *wrap-around links*;  $P_{i,0}$  is connected with  $P_{i,n-1}$ , and  $P_{0,j}$  is connected with  $P_{n-1,j}$ . Tori have the advantage that they are uniform, i.e. there are no corners that have to be treated in a special way. The diameter of a torus is only half the diameter of the corresponding mesh and the bisection width is twice as large.



Figure 1: 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 onto a mesh with dilation 2 is realized analogously, by “folding” in both directions.

Since a torus can be embedded onto a mesh with dilation 2 (see Figure 1), any torus algorithm can be simulated on a mesh with delay factor 2 (except for the SIMD model). Therefore, any optimal torus algorithm, that is an algorithm matching the distance or bisection bound (see Section 1.3), yields automatically an optimal mesh algorithm. However, the opposite is not true. There are problems which cannot be solved twice as fast on a torus. The simplest example is 1-1 sorting on one-dimensional arrays [202].

#### 3.1 On-Line Routing

In this section we consider variants of the routing problem on meshes. 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.

**Permutation Routing.** In the SIMD model,  $4n - 4$  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 in the case that all packets from a certain row have destination in the same column, they are all wiped into a single node, yielding queues of size  $\Omega(n)$ .

In the SIMD model, the best results for permutation routing are obtained by simulating the MIMD algorithms of Kunde [157] (deterministic) and Rajasekaran and Tsantilas [264] (randomized).

**Lemma 3.1** *Permutation routing on an SIMD mesh can be performed in  $4n + \mathcal{O}(n/q)$  steps using queues of size  $q$  [157, 264].*

In the MIMD model, the mesh diameter  $2n - 2$  is a lower bound for permutation routing. Leighton has shown that for random destinations the greedy algorithm performs very well, even the queue size remains small [175]. The first near-optimal permutation routing algorithms were presented in [157, 264]. Leighton, Makedon and Tollis [182] proved “the impossible”; a deterministic algorithm running in  $2n - 2$  steps with constant size queues (about 1000). In later papers the queue size was reduced to more practical values [262, 293]. The essential idea in all papers since [264] is that “critical packets” (packets moving between opposite corner regions of the mesh) are given priority, so that they do not incur any delay. Queue size estimates essentially depend on sorting algorithms of sub-meshes of decreasing size.

**Theorem 3.1** *Permutation routing on an MIMD mesh can be performed in  $2n - 2$  steps using queues of size 32 [182, 262, 293].*

Another randomized  $2n + \mathcal{O}(\log N)$  algorithm is given in [133]. This algorithm does not use critical packets, which makes it much simpler. It served as a source of inspiration for several later algorithms. In [138] it was shown how this algorithm can be made deterministic with queue size 2. Applying critical packets again the loss of time can be reduced from  $\mathcal{O}(n^{4/5})$  to  $\mathcal{O}(1)$ :

**Lemma 3.2** *Permutation routing on an MIMD mesh can be performed in  $2n + \mathcal{O}(1)$  steps using queues of size two [138, 293].*

Although this is an interesting result, the constant hidden in the  $\mathcal{O}(1)$  is large. More practical algorithms have routing time below  $3n$  and queue size five or six [138].

**$k$ - $k$  Routing.** Practically, more relevant than 1-1 routing are the  $k$ - $k$  routing problems. Because of the mesh bisection bound,  $k$ - $k$  routing requires at least  $kn/2$  steps. The first nontrivial algorithm for  $k$ - $k$  routing was presented by Kunde and Tensi [163]. It requires  $5kn/4 + o(kn)$  steps. Several improvements gradually reduced the routing time to almost optimal [141, 158, 263]. One of the most important ideas is *packet coloring* [158]; the packets are colored deterministically or at random, white or black. 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 random positions, and from there to their destinations. This idea goes back to Valliant and Brebner [317]. Near-optimal results were first achieved with randomized algorithms [141, 263]. It was then discovered that randomization is superfluous, and can be replaced by sorting packets in sub-meshes and performing unshuffling [144, 160].

**Theorem 3.2**  *$k$ - $k$  routing on MIMD meshes can be performed deterministically in  $kn/2 + \mathcal{O}(k^{5/6}n^{2/3})$  steps. The maximum queue size is  $k$  [144, 160].*

It is not hard to perform  $k$ - $k$  routing twice as fast for average case inputs; just omit the randomization phase. An analysis is given in [161]. In [145], several ideas are combined to obtain an algorithm with routing time close to  $kn/4$  for average case inputs while never exceeding  $kn/2$ .

**Locality Preserving Routing.** It may happen that all packets which have to be routed actually have to travel only a short distance. For such cases it would be a pity to apply the sophisticated algorithms which inevitably lead to a routing time of at least  $\max\{2n, kn/2\}$ . However, if  $d$  is the maximum distance any packet has to go, it is not hard to obtain a routing time of  $\mathcal{O}(kd)$ . This is called *routing with locality* or *locality preserving routing*. If  $d$  is known globally, one can apply a variant of the basic algorithm by Rajasekaran and Tsantillas in [264]. If  $d$  is not known, then each packet may use its own distance  $d'$  as an estimate. Therefore it performs a randomization in a range of size  $d'$  along the columns, followed by a greedy routing; first along the rows, then along the columns. A detailed description of such

an algorithm is given in Chapter 6 of [287]. Other variants are possible too. To make such an algorithm deterministic requires more effort, but at the cost of extra routing time.

It is not true that every individual packet arrives within a constant factor from the optimal time (the applied farthest-first strategy may cause that a packet which has to travel a short distance is delayed many steps), but globally these algorithms achieve a constant competitive ratio.

***k-l Routing.*** The general *k-l* routing problem has attracted less attention [98, 203, 204, 214, 294]. Considering the bisection bound and the number of packets that may have to move into or out of a corner, it follows that:

**Lemma 3.3** *k-l routing on an MIMD mesh requires at least*

$$\left[ \max \left\{ \frac{k}{2} \left( \frac{l}{l+k} \right)^{1/2}, \frac{lk}{l+k}, \frac{l}{2} \left( \frac{k}{l+k} \right)^{1/2} \right\} \right] n$$

*steps [294].*

At first it is not obvious how to achieve better than  $\Omega([\max\{k, l\}] n)$ . For case  $l > k$ , the central idea is to color packets such that in the final routing phase, in which packets are routed within their rows or columns towards their destinations, the maximum number of packets in a row or column remains bounded. For  $k > l$  the procedure is analogous. Optimal results have been established for  $k = o(l)$  and  $l = o(k)$ .

**Theorem 3.3** *k-l routing on an MIMD mesh can be performed in  $(kl)^{1/2}n/2 + \mathcal{O}(\min\{k, l\} n)$  time. [203, 294].*

**Problem 3** *Can k-l routing be performed in time given by Lemma 3.3?*

Not considering constants, Lemma 3.3 and Theorem 3.3 were already suggested in Ex. 1.246 and 1.247 in [176]. Unlike the algorithm in [203], the algorithm of [294] works without knowing the values of  $k$  and  $l$ . Furthermore, the routing time seems close to optimal for most inputs. In [204, 214], algorithms are presented with performance within a logarithmic factor from optimal for *all* inputs. This property is called *strong adaptivity* in [214]. We say that algorithms with this property are *competitive*.

**Token Distribution vs. *k-l* Routing.** [214] actually deals with *token distribution* [242]. Token distribution refers to rearranging of “tokens”, which initially are distributed inhomogeneously over a network, so that afterwards all nodes hold approximately the same number of them. The main difference compared to routing is in the final position of the tokens which is *not* prescribed.

*k-l* routing and token distribution are closely related. After counting the number of tokens and attribution of a rank, a *k-m* routing algorithm can be used for smoothing the packets out. Here,  $m$  is the average number of packets. On the other hand, if one has a token distribution algorithm, then one can first smooth the packets out, and then perform an *m-m* routing, with  $m \leq \min\{k, l\}$  as before, and then perform the inverse of a token distribution.

This approach need not be competitive for arbitrary *k-l* routing problems (which may be much cheaper than the token distribution), but for *k-1* routing it is. In that case, we need only perform a token-distribution and a 1 – 1 routing. This gives the following:

**Lemma 3.4** *If 1 – 1 routing can be performed while preserving the locality, and if there is a competitive token distribution algorithm, then also the k-1 routing problem can be solved competitively.*

### 3.2 Off-Line Permutation Routing

**Theorem 3.4** *Affine permutations can be routed on an SIMD mesh in  $4n - 4$  steps using queues of size two [226, 285].*

By encoding the permutation in only  $\mathcal{O}(\log^2 n)$  bits, precomputation (essentially a matrix inversion) takes  $\mathcal{O}(\log^3 n)$  time.

**Lemma 3.5** *Normal Ascend/Descend permutations can be performed in  $2n - 2$  steps on the SIMD  $n \times n$  mesh [220].*

For MIMD meshes, Krizanc proved a  $2.5n$  steps off-line permutation routing algorithm with queue size two [151]. Kaklamani, Krizanc and Rao presented a  $2n - 1$  steps MIMD algorithm with queue size four [133]. Smaller queues are obtained by considering the  $n \times n$  mesh as a Cartesian product of two  $n$  node, 1-dimensional arrays. Using Theorem 2.6 we obtain:

**Corollary 3.1** *Any permutation can be off-line routed on an uni-axial mesh in  $3n - 3$  steps using queues of size one [20].*

**Corollary 3.2** *Any Omega or Inverse Omega permutation can be off-line routed on the MIMD  $n \times n$  mesh in  $2n - 2$  steps using queues of size one [330].*

Precomputing the schedule for Corollary 3.1 takes  $\mathcal{O}(n^2 \log n)$  steps [47]. Affine permutations can be routed on the MIMD mesh using the algorithms in [226] or [285], with routing time  $2n - 2$  steps and queue size two. These routing schedules can be computed much faster.

Another important class of off-line routing problems are the transpositions. In a transposition, the node at position  $(x, y)$ , has to exchange its data with the node at position  $(y, x)$ . If trivial greedy routing is applied, then many packets have to move through two of the corner nodes, but with some care one can show that:

**Theorem 3.5** *If every node of an  $n \times n$  mesh holds  $k$  packets, then these packets can be transposed in  $(1 - 1/\sqrt{2})kn + o(kn)$  steps [139]. This bound is sharp [65].*

### 3.3 Fault Tolerant Routing

**$P$ -faulty model.** For dynamic faults we want to minimize the expected lifetime of a packet and maximize the probability that the packet reaches its destination. To achieve these goals, packets are routed as long as possible via shortest paths. The best shortest-path route is the one with the most alternatives. The  $Z^2$ -policy states that precedence should be given to propagation of messages towards the diagonal [13]. The diagonal with respect to node  $P_{0,0}$  is the set of nodes  $\{P_{i,i} | 0 \leq i \leq n - 1\}$ . Thus, a packet residing at  $P_{i,j}$  with destination at  $P_{0,0}$ , 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 3.6** *The  $Z^2$ -policy is optimal [13].*

A negative result on the effectiveness of shortest-path routing is:

**Lemma 3.6** *For large  $n$ , the probability of successful completion of shortest-path routing on the  $p$ -faulty  $n \times n$  mesh approaches zero [89].*

Another approach, *sidetracking*, attempts to move a packet along a shortest path from its current position to its destination. If at any instant all nodes along a shortest path are faulty, the packet is sent to a randomly chosen nonfaulty node from the remaining neighboring nodes. Thus, this approach corresponds to oblivious nonminimal path routing. Although the

probability of successful routing still converges to zero, good empirical results were reported (also for the hypercube) [88].

In an algorithm by Raghavan [258], instead of sending a packet along its master path  $P$  (defined by Valiant and Brebner's randomized algorithm), the packet is broadcast within a routing region  $R(P)$ , defined to contain all nodes within distance  $(c \log n)$  from  $P$ , where  $c$  is a constant. This region is such that if there is a live path between the packet's source and destination, the path will likely be contained in  $R(P)$ .

**Theorem 3.7** *For all  $p \leq 0.29$ , on-line permutation routing can be performed on a  $p$ -faulty  $n \times n$  mesh in  $\mathcal{O}(n \log n)$  steps, with high probability [258]. The maximum queue size is  $\mathcal{O}(\log^2 n)$ .*

Mathies improved the above bound on  $p$  to about 0.40 [207]. The most surprising result is:

**Theorem 3.8** *There exists a constant  $p > 0$ , such that  $p$ -faulty  $n \times n$  mesh can emulate a fault-free  $n \times n$  mesh with constant slowdown, with high probability [48].*

**Worst Case Model.** The above algorithms give on-line adaptations. For static faults, stronger results can be obtained by embedding a fault-free network. Kaklamanis et al. stated that "in many cases the routing time is the same as if there were no faults in the array, up to constant factors" [130]. Among other things they prove:

**Theorem 3.9** *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 [130].*

Further improvements were given by Cole, Maggs and Sitaraman [48]. They prove that, for some constant  $\varepsilon > 0$ ,  $n^{1-\varepsilon}$  worst case faults can be handled in such way that the computation is slowed down only by a constant factor. In practice, the above algorithms are insufficient, since constant factors are large. We would like to have algorithms which tolerate a small number of faults with small delays.

**Problem 4** *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  $x = 1, 2, \dots$ ?*

### 3.4 Dynamic Routing Problems

In dynamic routing each node generates packets with a fixed rate  $\lambda$ . Each generated packet is assumed to have a random destination. There is a trivial upper bound on the generating rate  $\lambda$ ; 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 an MIMD mesh), this implies that the system will get more and more congested, eventually resulting in infinite delays, for  $\lambda > 4/n$ . Leighton proved that, for all  $\lambda < 4/n$ , greedy routing along row and column to their destinations normally works fine [175]. This results has been further improved by Kahale and Leighton [128]. If several packets compete for a link, then the one that has to move farthest in this direction is given priority.

**Theorem 3.10** *If the generating 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 queue size is  $\mathcal{O}(1 + \log T / \log n)$  [175].*

In [218] this result is generalized and an alternative, more powerful analysis technique is introduced. Even more general, not limited to fixed interconnection networks like meshes or hypercubes, are the results by Harchol-Balter et al. in [99, 100]. In [99], it is shown how to simplify the computation of upper bounds on the expected packet delay in a dynamic situation for the class of “Markovian queuing networks”.

### 3.5 Cut-Through - Wormhole - Hot-Potato Routing

The study of *cut-through routing* on meshes was initiated by Makedon and Symvonis in [201]. Subsequent improvements reduced the time complexity close to the lower bound.

**Lemma 3.7** *For routing a permutation on an MIMD mesh, with packets consisting of  $b$  flits each,  $bn/2 + n/b + 3n/2 + o(bn)$  steps are sufficient [141, 201, 263].*

This is obtained by applying the same algorithm as in  $k$ - $k$  routing. It is interesting to consider whether cut-through routing is essentially harder than  $k$ - $k$  routing (when  $k = b$ ).

**Problem 5** *Can cut-through permutation routing be solved in  $bn/2 + o(bn)$  steps?*

A weaker result than Theorem 3.10 holds for dynamic cut-through routing.

**Lemma 3.8** *We consider cut-through routing on an  $n \times n$  MIMD mesh with packets of  $b$  flits. If the generating rate is less than  $1/(bn)$ , then the maximum delay incurred by any packet in any window of  $T$  steps is bounded by  $\mathcal{O}(b(\log T + \log n))$ , with high probability. The maximum queue size is  $\mathcal{O}(b + b \log T / \log n)$  [175].*

For *wormhole routing*, theoretical analysis of congestion is more difficult, since packets which have partially crossed a link can block other packets for arbitrary time. However, *delayed greedy* wormhole permutation routing is within a logarithmic factor of being optimal [73]. In delayed greedy routing each message waits at its source for a random initial delay before moving greedily towards its destination.

**Theorem 3.11** *For worms of length  $b$ , the delayed greedy wormhole routing can be performed on an  $n \times n$  MIMD mesh in  $\mathcal{O}(bn \log n + n^2 / \log n)$  steps, with high probability [73].*

The situation remains largely unclear for dynamic wormhole routing:

**Problem 6** *Is there a dynamic wormhole routing algorithm that assures a constant fraction of the network capacity can be used while all packets reach their destination with minimal delay?*

For average case instances, *hot-potato routing* can be performed in  $2n + \mathcal{O}(\log n)$  steps (or less), as shown in [286] (incomplete analysis) and [71] (complete analysis). In the same paper Feige and Raghavan presented a randomized algorithm with linear time complexity. Schuster and Newman [230] were the first to give a deterministic linear-time algorithm. This algorithm was improved by Kaufmann, Lauer and Schröder [137].

**Lemma 3.9** *Hot-potato permutation routing on an MIMD mesh can be performed in  $7n/2 + o(n)$  steps [137, 230].*

Newman and Schuster combined the requirements of hot-potato and wormhole routing. They achieve a bound of  $\mathcal{O}(b^{2.5}n)$  steps for routing worms of length  $b$  on an  $n \times n$  mesh [231]. In [294] it is shown that under a realistic condition the time delay is only  $\mathcal{O}(b^{1.5}n)$ . The idea behind [231] is ingenious; temporarily blocked worms are kept cycling around “parking lots”, and as soon as the “highway” running along the parking lot is free they take off. Bar-Noy et al. presented a randomized algorithm running in  $\mathcal{O}(bn)$  steps [17]. Roberts and Symvonis achieved the same result with an off-line algorithm [272]. These results make it likely that the following question can be answered affirmatively, either by applying derandomization [144], or techniques for converting off-line into on-line algorithms [301].

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

In all hot-potato routing algorithms, packets may take a detour. The algorithms are such that packets which have to go further make fewer detours, and therefore the overall completion time is close to optimal. However, individual packets may get delayed considerably. In [42] it is shown that such a flexibility is essential; lower bounds are constructed for adaptive routing algorithms which use routing of packets along shortest paths. Such algorithms are called *minimal adaptive routing algorithms*. For an  $n \times n$  mesh, with maximal storage capacity  $k$  in each node, it is shown that for a large class of minimal adaptive routing algorithms there are inputs which require at least  $\Omega(n^2/k^2)$  steps.

The topics of wormhole and hot-potato routing have been considered for more general networks in [54, 93, 215, 216, 268]. Particular attention is devoted to vertex-symmetric networks (among them tori and butterflies). More practical aspects of wormhole routing, taking into account that not only the connections of the network have finite speed, but that also the rate with which the nodes can supply data to the network is finite, are considered in [96, 292].

### 3.6 Sorting on Meshes

In sorting problems data elements must be rearranged so that they stand in sorted order with respect to the indexing [6]. An *indexing scheme* is a bijection  $I : \{0, 1, \dots, n-1\}^2 \rightarrow \{0, 1, \dots, n^2-1\}$ , which attributes to every node a unique index. In *row-major* indexing,  $P_{i,j}$  has index  $in + j$ . In *column-major* indexing,  $P_{i,j}$  has index  $i + jn$ . In *snake-like row-major* indexing, the indexing of the odd rows is reversed. In *shuffled row-major* indexing, the shuffle row major permutation ( $\pi_7$ ) is applied to the row major ordering scheme. These indexing schemes are illustrated in Figure 2.

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 2: Indexings of a  $4 \times 4$  mesh. From left to right: row-major, column-major, snake-like row-major, shuffled row-major indexing.

**One-Packet Model.** In the earliest papers on mesh sorting it was assumed that at all times every node can hold only one packet. Under this assumption one can derive interesting lower bounds using the so-called “joker-zone arguments”.

**Lemma 3.10** *Sorting on an MIMD mesh in the one-packet model requires at least  $3n - \mathcal{O}(n^{1/2})$  steps for sorting in (snake-like) row-major order [156, 278].*

Deeper methods are required for finding lower bounds for all indexing schemes.

**Lemma 3.11** *There is no indexing scheme, with respect to which sorting in the one-packet model can be performed in less than  $2.27n$  steps [97].*

Thompson and Kung [309] presented the first  $\mathcal{O}(n)$  sorting algorithm. Schnorr and Shamir [278] designed the first optimal algorithm.

**Theorem 3.12** *Sorting on an MIMD mesh in one-packet model with respect to the snake-like row-major indexing can be performed in  $3n + \mathcal{O}(n^{3/4})$  steps [196, 278, 309]. On a  $m \times n$  mesh the algorithm takes  $m + 2n + o(n + m)$  steps.*

**Multi-Packet Model.** If we drop the restriction that nodes can hold only one packet, much better results are achievable. By first concentrating all packets in the central  $n/2$  columns and then applying the algorithm of [278], Kunde achieved a sorting time of  $2.5n + o(n)$  steps [158]. The first algorithm which almost matches the diameter bound with delay  $2n + o(n)$ , was discovered by Kaklamanis and Krizanc [131]. This probabilistic algorithm was successfully derandomized in [144]. The latter algorithms use blocked snake-like row-major indexing. Krizanc and Narayanan considered more natural indexing schemes when all keys are either zero, or one [153]. In [289] it was shown that sorting with respect to row-major indexing can be solved optimally.

**Theorem 3.13** *Sorting on an MIMD mesh in row-major order can be performed in  $2n + o(n)$  steps using queues of size five [131, 132, 144, 152, 153, 158, 289].*

Although asymptotically optimal, all algorithms cited in Theorem 3.13 have lower-order terms that tend to dominate the overall sorting time for small meshes. In this perspective, valuable contributions have been given by Thompson and Kung [309], Nassimi and Sahni [225], Kumar and Hirschberg [154], and Lang et al. [169]. Lately several ideas were combined to reduce the dominant term by almost a factor of two:

**Lemma 3.12** *For all  $n$ , sorting in row-major order can be performed in  $4.75n$  steps using queues of size nine. The routing model is uni-axial. For  $n = 2^l$  sorting can be performed in  $4.5n$  with queue size six, and for  $n = 3^l$  in  $4\frac{1}{3}n$  with queue size five [289].*

**Problem 8** *What is the best sorting algorithm for small values of  $n$ , say  $n \leq 256$ ?*

The theory of  $k$ - $k$  sorting has been developed hand in hand with that of  $k$ - $k$  routing. Kunde [158] gave a deterministic  $kn + o(kn)$  sorting algorithm. The first almost optimal algorithm was developed by Rajasekaran in [141], improved to  $kn/2 + o(kn)$  in [143, 260]. These algorithms are randomized. In [144, 160] it is shown how to achieve  $kn/2 + o(kn)$  deterministically.

**Theorem 3.14**  *$k$ - $k$  sorting on MIMD meshes can be performed in  $kn/2 + o(kn)$  steps using queues of size  $k + o(k)$ , by a deterministic algorithm [141, 143, 144, 158, 160, 289, 291].*

**Limited Instruction Sets.** Many attempts have been made to design sorting algorithms on  $m \times n$  meshes in which few basic operations are alternated. The simplest algorithm performs rounds of the following type: first the columns are sorted, then the rows (the odd rows one leftwards, the even rows rightwards). This algorithm is called *shearsort*. Applying the 0-1 principle (which states that a comparison-based sorting algorithm that sorts an arbitrary 0-1 distribution is correct, see [176, p. 141]), it is easy to prove that  $\lceil \log m \rceil + 1$  rounds are enough (here  $m$  is the number of rows):

**Lemma 3.13** *Shearsort sorts  $mn$  numbers on an  $m \times n$  mesh in snake-like order in  $(m + n)(\lceil \log m \rceil + 1)$  steps [274, 277].*

Kutyłowski and Wanka have analyzed the complexity of this algorithm more precisely. They show that the above bound is sharp when the number of columns  $n$  is not a power of 2, but that  $\min\{\log n, \lceil \log m \rceil + 1\}$  rounds are sufficient when  $n$  is a power of 2 [165].

An extension of this algorithm by Schnorr and Shamir includes an extra phase in each iteration, after the two basic sorting phases [278]. This phase corresponds to a shuffle permutation applied to  $i^{\text{th}}$  row,  $1 \leq i \leq m$  for  $\text{rev}(i) = i \bmod n^{1/2}$  times. The resulting *revsort* algorithm iterates these three phases  $\lceil \log \log m \rceil$  times.

**Lemma 3.14** *Revsort sorts  $mn$  numbers on a  $m \times n$  mesh in snake-like order in  $(m + n)\lceil \log \log m \rceil$  steps [278].*

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

**Theorem 3.15** *Rotatesort sorts  $mn$  numbers on a  $m \times n$  mesh in snake-like or row-major order with 16 row and column phases [206]. This takes  $\mathcal{O}(n + m)$  steps in total.*

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

**Theorem 3.16** *Sorting on a  $m \times n$  mesh,  $m, n \geq 8$ , into any sorting order requires at least five phases consisting of column/row sorts and shuffles [206].*

**Problem 9** *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?*

For special shapes of a mesh, better upper bounds are known. Leighton's *columnsort* algorithm consists of alternating column sorting and row rearrangements [174].

**Theorem 3.17** *For  $m \geq n^2$ , columnsort sorts  $mn$  numbers on a  $m \times n$  mesh into column major order using a sequence of eight phases of row/column sorts and transpose/shifts [174]. In total this takes  $\mathcal{O}(n + m)$  steps.*

More abstractly, the basic idea of columnsort is that in order to sort  $N$  elements, one must first sort all  $N^{1/3}$  subsets of size  $N^{2/3}$  and perform an unshuffle. Then all subsets are sorted again, and another rearrangement is performed. Two more sortings of pairs of consecutive subsets complete the sorting. This fundamental idea has appeared (sometimes in disguise) several times. Of course, the algorithm can also be applied recursively: in order to sort the subsets of size  $N^{2/3}$ , it is sufficient to sort subsets of size  $N^{4/9}$  or larger. As on a square  $n \times n$  mesh, the number of nodes in a row  $n \geq (n^2)^{4/9}$ , columnsort can be applied to sort with a constant number of phases on meshes of arbitrary shape.

*Bubblesort* algorithms cycle through a small set of local comparisons in a fixed order. Unfortunately, straight-forward approaches result in  $\Omega(n^2)$  sorting time, even on average [276]. A positive result has been reported by Ierardi:

**Theorem 3.18** *On an  $n \times n$  mesh, bubblesort can, on average, be performed in  $\mathcal{O}(n \log^{1/2} n)$  time [117].*

Related results have been obtained by Schwiegelsohn and Kutylowski ea. [117, 164, 280]. In [164], it is shown how to obtain periodic sorting networks of depth only 5 out of comparator-based sorting networks. In doing so, a factor of  $\mathcal{O}(\log n)$  is lost in time consumption. This is a very strong result, but for meshes, the algorithm of Ierardi, which is away from optimal by only a factor  $\mathcal{O}(\log^{1/2} n)$ , is still slightly better.

### 3.7 Mesh-Like Networks

We provide some references for variants of the mesh architecture.

**One-Dimensional Arrays.** Greedy routing on *linear arrays* (one-dimensional meshes) using farthest-first priority yields minimum delay. A bubblesort-like algorithm works in the minimum number of steps.

For *rings* (one-dimensional tori) Mansour and Schulman derived an interesting lower bound:

**Lemma 3.15** *If every node of a ring of length  $n$  can hold only one packet at a time, then  $1 - 1$  sorting requires at least  $2\lfloor n/2 \rfloor - 1$  steps.*

On the other hand, they show that if the nodes can hold six packets at a time, then  $1 - 1$  sorting can be performed in  $\lfloor n/2 \rfloor + 1$  steps [205].

For  $k$ - $k$  sorting one can apply a one-dimensional version of the algorithms from [144, 160] (the result is cited in Theorem 3.14), but there is also a routing algorithm which is independent of sorting:

**Theorem 3.19**  *$k$ - $k$  routing on a MIMD ring can be performed in  $kn/4 + n^{1/2}$  steps [142, 202, 288].*

**Higher Dimensional Arrays.** A  $d$ -dimensional  $n \times \dots \times n$  mesh (or torus) consists of  $N = n^d$  processors arranged in a  $d$ -dimensional cube. On a  $d$ -dimensional torus the lower bound for  $k$ - $k$  routing is  $kn/4$ . The routing and sorting algorithms of [141, 144, 160, 163, 178, 289, 327] are applicable to higher dimensional meshes and tori. Optimal time, matching the bisection bound within lower-order terms, is assured as long as there are at least  $k \geq 4d$  packets at every node.

For permutation routing, optimal randomized results for two-dimensional tori and three-dimensional meshes are established in [133]. Similar results can be obtained by applying a deterministic algorithm to results from [144] and [289]. For higher dimensional meshes, Kunde [159] gave some preliminary results, which were improved by Suel. In [302] it is shown how to sort in  $5/4dn + o(n)$  on a  $d$ -dimensional mesh, and in  $3/4dn + o(n)$  on a  $d$ -dimensional torus. Still there remains a considerable gap with lower bounds.

**Problem 10** *Find lower and upper bounds for permutation routing and sorting on meshes of dimension  $d > 3$  and tori of dimension  $d > 2$ ?*

For  $k$ - $l$  routing on  $d$ -dimensional meshes, one can apply algorithms from [164, 203, 294] but considering the constant in the leading term, these are not even close to optimal.

**Problem 11** *Find algorithms, randomized or deterministic, for  $k$ - $l$  routing which perform well for higher dimensional meshes?*

Greedy routing to independently chosen random destinations is analyzed in [175]. The fault-tolerant  $Z^2$ -policy can be generalized to higher dimensional meshes [13]. Theorem 2.6 can be used to off-line routing of any permutation in optimal time. Normal Ascend/Descend permutations can also be performed in optimal time [219].

**Meshes with Diagonals.** Hardly any theoretical papers deal with *meshes with diagonals*, in which each node has up to eight neighbors. Surprisingly, routing and sorting times on such meshes can be reduced by more than a factor of two.

**Lemma 3.16** *On an  $n \times n$  mesh with diagonals,  $k$ - $k$  routing and sorting can be performed in  $2kn/9 + o(kn)$  steps [162]. On a torus with diagonals, this problem can be solved in  $k n/12 + o(kn)$  steps [290].*

**Meshes with Buses.** In meshes with *fixed buses* every row and column is equipped with a bus. The bus can be used to transfer a packet from one node to all connected nodes. There exist numerous variants. Sometimes the bus can be used by several sending nodes if the required sections of the bus do not overlap. In other models there are two buses, or buses without any mesh connections. For meshes with buses, sorting is only slightly more complex than routing, since the buses are excellent for rapid broadcasting and gathering. Routing has been investigated in [140, 188, 261, 301].

Another variant are networks with the computing nodes on the outside, while the mesh makes an interconnection network. The motivation to consider such a network is that on

a traditional mesh with  $n^2$  nodes the bisection width is only  $n$ . This implies that for most problems the best speed-up which can be achieved is much smaller than the number of nodes. By putting nodes on the outside one gets the *coated mesh*; one can save substantially on the costs of the hardware, without losing much of the computation power. Such architectures are also particularly suited for PRAM simulations [43, 187]. Various routing problems are considered in [44].

**Reconfigurable Meshes.** In a *reconfigurable mesh* all nodes are connected to one big bus, which can be decomposed into many smaller buses. It is surprising that the following holds.

**Theorem 3.20** *On a reconfigurable  $n \times n$  mesh  $n$  numbers can be sorted in a constant number of steps [22, 326].*

The algorithms are based on rotate- and column sort. Pushing a column sort based algorithm to the limit, the number of steps can be decreased below 40. An extension to sorting  $n^{d-1}$  numbers on a  $d$ -dimensional reconfigurable mesh is simple [233]. Theoretically interesting is the following:

**Problem 12** *How many steps are required for sorting  $n$  numbers on a reconfigurable  $n \times n$  mesh?*

Another feature of reconfigurable meshes is that they allow routing and sorting extremely sparse packet distributions in time given by the bisection bound [146]. Also [21] deals with this problem, and in [301] sparse dynamic problems are treated. The theory of reconfigurable meshes is rich and we could only mention a few results. Nakano gives an extensive and up-to-date list of publications [222].

## 4 The Binary Hypercube

The  $n$ -dimensional binary hypercube has  $N = 2^n$  processors, numbered  $0, 1, \dots, N - 1$ . A processor is addressed by a binary string  $x = x_1 x_2 \dots x_n$ , where  $x_i \in \{0, 1\}$ , for all  $1 \leq i \leq n$ . Two processors with addresses  $x, y$  are connected if their binary representations differ in exactly one bit. Both the degree, and diameter of the  $n$ -dimensional binary hypercube equal  $\log N$ .

### 4.1 On-Line Routing

The *greedy routing* algorithm sends packets from source to destinations along shortest paths. Dimensions are examined in a left to right order. Packets (not in their final destination) move by correcting each dimension for which the corresponding source and destination bits are different. Packet conflicts are commonly resolved with a farthest-first or random queuing discipline.

We proceed with a description of the greedy algorithm. The notation  $x : (x_j \leftarrow \alpha)$  defines the node obtained by changing the  $j$ th digit in the binary representation of  $x$  to  $\alpha$ . Let an arbitrary packet  $\Psi$  originate at source node  $x = x_1 x_2 \dots x_n$  and head to destination node  $x' = x'_1 x'_2 \dots x'_n$ .

*Greedy routing* (“cobegin ... coend” refers to all packets).

```

cobegin
while  $x \neq x'$  do
    transmit  $\Psi$  from  $x$  to  $x : x_i \leftarrow x'_i$ , where  $x_i \neq x'_i$ 
coend

```

**Permutation Routing.**

Theorem 2.1 specializes to an  $\Omega(N^{1/2}/n)$  lower bound for oblivious permutation routing on the MIMD  $n$ -dimensional binary hypercube. By examining the bit reversal and matrix transposition permutations it is simple to show that greedy routing on the binary hypercube is not optimal within the class of deterministic oblivious algorithms, for MIMD communication.

**Theorem 4.1** *Greedy permutation routing on the  $N$  node,  $n$ -dimensional binary hypercube takes  $\mathcal{O}(N^{1/2})$  steps, for both SIMD, and MIMD communications.*

However, the  $\Omega(N^{1/2}/n)$  lower bound for MIMD communication is tight since a deterministic oblivious algorithm, based on a Hamiltonian cycle decomposition of the hypercube, can route any permutation in  $2N^{1/2}/(\log N - 2 \log \log N + 2) + \log N/2 = \mathcal{O}(N^{1/2}/\log N)$  time [134]. For hypercubes of dimensions  $n \leq 14$ , this was considered as the most efficient algorithm. Small improvements have been made by adapting optimal routing methods (many to one and one to many) for small cubes. For  $n \leq 7$ , an arbitrary permutation can be on-line routed on the binary  $n$ -dimensional hypercube in just  $n$  moves [116]. The proposed adaptive routing is *local* since a message does not need to know the other packet routes, only assuming that a packet at a node can detect whether an adjacent link is already occupied. For  $n \leq 6$ , optimal oblivious permutation routing algorithms, based on uniformly constructed destination graphs, have been proposed in [91]. Dasgupta, Hwang and Yao have recently extended the destination graph idea to incorporate delay schedules, proving optimal oblivious permutation routing for  $n = 7, 8$  and providing the most efficient permutation algorithms known for  $n \leq 14$  [63].

**Problem 13** *Is  $n$ -move local permutation routing on the MIMD  $n$ -dimensional hypercube possible for arbitrary  $n$ ?*

Kuzmaul has shown:

**Lemma 4.1** *Semi-contractions (or semi-expansions) can be routed on the SIMD  $n$ -dimensional binary hypercube in  $n$  steps using synchronous greedy routing, correcting dimensions from right to left (resp., left to right), one at a time [166].*

Since packings (unpackings) are also semi-contractions (resp., semi-expansions), we have:

**Corollary 4.1** *Packings and unpackings can be routed on the SIMD  $n$ -dimensional binary hypercube in just  $n$  steps.*

Lemma 4.1 also applies to the MIMD model, since this model is stronger. The methods of greedy routing (correcting dimensions either from right to left, or left to right) are also used to perform optimally general vector operations, such as merge, split, rotation, reversal, compression, and expansion [190]. Furthermore, for messages of size  $\mathcal{O}(n)$ , the greedy algorithm can be pipelined:

**Lemma 4.2** *Any semi-contraction, semi-expansion, or semi-broadcast can be routed on a pipelined MIMD  $n$ -dimensional binary hypercube in  $\mathcal{O}(h + n)$  steps using queues of size  $h$  [318].*

Partial permutation routing can use Batcher's sorting algorithm followed by an unpacking. Load balancing can also be based on simple operations of broadcast, concentrate and distribute (semi-expansions) [119, 250, 269]. By using synchronous greedy routing algorithm, load balancing after each routing step, Jaja and Ryu proved:

**Theorem 4.2**  *$(N, N, k_1, k_2)$ -routing can be solved on the pipelined MIMD  $n$ -dimensional binary hypercube in  $\mathcal{O}(k_1 + k_2n + n^2)$  time with  $\mathcal{O}(k_2)$  queue size [119].*

Up to this point we have considered deterministic hypercube routing. A major theoretical breakthrough was the probabilistic permutation routing algorithm on the binary hypercube, proposed by Valiant [314] and extended to general graphs by Valiant and Brebner [317]. The two-phase algorithm consists of randomization (Phase I) and deterministic routing (Phase II). In Phase I, packets are sent to independent, randomly selected nodes through the network. The role of this phase is to reduce the difference between the average and the worst case performance. In Phase II, packets follow a shortest path to their final destinations. Dimensions are corrected either in an orderly [317], or random fashion [314].

**Theorem 4.3** *Probabilistic permutation routing on the MIMD  $n$ -dimensional binary hypercube runs in  $\mathcal{O}(n)$  time using queues of size  $\mathcal{O}(n)$ , with high probability [314, 317].*

Probabilistic routing has been applied to multistage networks. Upfal showed how to route on the  $\log N$ -stage butterfly in  $\mathcal{O}(\log N)$  steps using queues of size  $\mathcal{O}(\log N)$  [311]. Pippenger was the first to show how to reduce the queue size to a constant, by probabilistic routing on a variant of the butterfly network in  $\mathcal{O}(\log N)$  steps using constant size queues [249]. Valiant and Brebner extended the above theorem to routing  $k$ -relations on an arbitrary graph (Theorem 2.5) and proved that the two-phase strategy is optimal within the class of oblivious routing algorithms for graphs of asymptotically optimal diameter [315]. By introducing more randomness at each node (and some precomputation) Valiant reduced the propagation delay, effectively reducing the time delay of the randomized routing algorithm to  $n + \mathcal{O}(n/\log n)$  steps using constant size queues, with high probability [316]. Ferreira and Grammatikakis improved the probability of routing all packets in asymptotically optimal time, by routing packets in Phase I according to a random permutation (and not random destinations) [76].

Valiant's routing algorithm on the binary hypercube requires  $\mathcal{O}(N \log N)$  bits of randomness. Ranade showed how to route on the hypercube in  $\mathcal{O}(\log N)$  time using only  $\mathcal{O}(\log^2 N)$  random bits [267]. The effects of limited randomness and pseudorandomness on routing time were studied in [135, 243, 313].

**Global Communications.** Global communication schemes on hypercubes have been extensively considered. Broadcasting can be easily implemented, since the hypercube belongs to the class of broadcasting graphs, i.e. in a broadcasting graph, the number of nodes possessing the necessary information can be doubled after each communication step. Thus, hypercube broadcasting is implemented based on an embedded binary spanning tree. The algorithm can be used on an SIMD hypercube, since dimensions are considered in sequential order [122, 300]. Assuming the constant delay model, broadcasting takes an optimal  $n$  steps on the SIMD  $n$ -dimensional binary hypercube.

Many optimal global communication algorithms for the hypercube are based on binomial spanning trees. A one-node binomial tree is the node itself. A  $k$ -node binomial tree is obtained from two  $k/2$ -node binomial trees by joining (with an edge) the roots of the two trees. Binomial trees can be easily constructed for the hypercube. Assuming the linear model, the time complexity can be improved by splitting the packet (taken as a no-cost operation) into  $n$  smaller sub-packets which are first routed to nodes:  $0 \dots 01$ ,  $0 \dots 010$ ,  $\dots$ ,  $10 \dots 0$ . From these nodes packets are routed farther using  $n$  edge-disjoint symmetric binomial spanning trees which are constructed using circular shifts [80, 122]. Optimal time can be achieved by further splitting of sub-packets into groups and exploiting pipelining on the binomial spanning trees graph [300].

Multinode broadcasting can be implemented using simultaneous single-node broadcasts from each hypercube node; in this case, pipelining can not be used. The time complexity of each step approximately doubles, since each node must now forward twice the information. The algorithm achieves  $\mathcal{O}(\lceil (N-1)/n \rceil)$  delay which is optimal for the MIMD  $n$ -dimensional hypercube [31, 122]. A similar algorithm is optimal for the SIMD hypercube [122].

Single node scatter may use spanning trees used for broadcasting; by giving priority to nodes farthest away the algorithm achieves optimal  $\mathcal{O}(\lceil (N-1)/n \rceil)$  time on the MIMD  $n$ -

dimensional hypercube [31]. For an optimal number of packet transmissions a new perfectly balanced spanning tree has been introduced [31]. For the SIMD hypercube, similar techniques achieve optimal complexity [122, 273]. Varvarigos and Bertsekas have also considered the effect of a random packet size on the performance of multinode broadcasting. While for the hypercube only small latency variations were shown, the situation is much different for a ring [322].

Total exchange algorithms are similar to multinode broadcasting ones; only the packet contents and sizes really differ [49]. Furthermore, since a lot of information must be exchanged, simpler techniques based on routing packets along hamiltonian circuits on the  $n$ -cube are also competitive. A simple algorithm which achieves an optimal  $\mathcal{O}(\lceil N/2 \rceil)$  delay for the MIMD  $n$ -dimensional hypercube is given in [31]. The recursive algorithm is based on three overlapped phases. The total exchange on each  $(n - 1)$ -sub-cube, packet exchange between corresponding  $(n - 1)$ -dimensional sub-cube nodes, and total exchange of the packets received in phase 2, on each  $(n - 1)$ -dimensional sub-cube. Optimal SIMD algorithms for this problem are much simpler [273].

The topic of fault-tolerant global communication algorithms on the hypercube has attracted an immense interest.

## 4.2 Off-Line Permutation Routing

A permutation can be decomposed off-line into a sequence of elementary transpositions, corresponding to the  $2 \times 2$  crossbar settings of an appropriate size Beneš permutation network [24]. These transpositions can then be simulated in optimal time, by embedding the Beneš network onto the binary hypercube.

**Theorem 4.4** *Any permutation can be off-line routed on the  $n$ -dimensional binary hypercube in:*

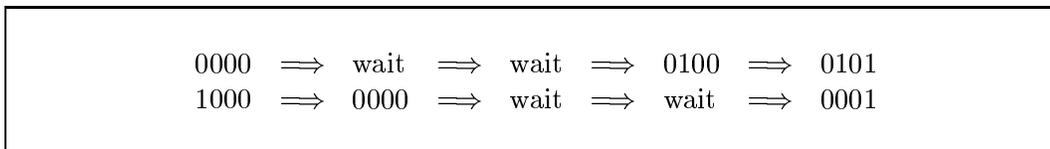
1.  $2n - 1$  steps using queues of size two, with SIMD model [308]; or
2.  $2n - 3$  steps using queues of size two, with MIMD model [283].

The latter algorithm provides a recursive construction based on Szymanski's result on realizing permutations on a circuit switched 3-dimensional binary hypercube (see Lemma 4.10). No parallel algorithm for setting up switches in this model has been provided, while a sequential algorithm seems to require  $\Omega(N)$  steps.

Serial setting of the switches takes  $\mathcal{O}(N \log N)$  time, which is prohibitive [324]. The best parallel algorithm for computing the switch settings on the binary hypercube uses Cypher and Plaxton's sorting algorithm, improving the results of Nassimi and Sahni and Lev, Pippenger and Valiant [189, 227] as follows:

**Lemma 4.3** *Computing the switch settings for the  $2^n$  node Beneš permutation network takes  $\mathcal{O}(n^3 \log n)$  steps on the MIMD  $n$ -dimensional binary hypercube [51].*

All  $(N \log N!)$  BPC permutations can be realized on the  $n$ -dimensional binary hypercube in  $n$  steps, using the routing algorithm of Nassimi and Sahni [228]. The algorithm is synchronous and follows individual cycles in the permutation. As an example, let  $N = 16 = 2^4$ , and  $\pi(x) = x_3 \bar{x}_1 x_2 \bar{x}_4$ . Since  $x_1 \implies x_3 \implies x_2 \implies \bar{x}_1$ , and  $x_4 \implies \bar{x}_4$ , this permutation consists of a 3-cycle  $(1, 3, 2)$ , and a singleton cycle  $(4)$ . For each such cycle, and for packets not in their final destinations, the corresponding source digits are changed one at a time. The BPC algorithm routes packets as follows:



**Lemma 4.4** *Any BPC permutation can be routed on the MIMD  $n$ -dimensional binary hypercube in just  $n$  steps [228].*

Assuming the linear model, Johnsson and Ho showed how to route some *BPC* permutations (including the shuffle and matrix transposition) in optimal time for both the 1-port and  $n$ -port pipelined hypercube [123, 124]. The class of all permutations which can be routed optimally on the hypercube using the above cyclic scheme is wider than the *BPC* class [237]. Furthermore, it has been shown that cyclic shifts (a subclass of *BPC*) can be routed in  $4n/3$  time on the  $n$ -dimensional binary hypercube without any local message buffers [235].

**Lemma 4.5** *Any Omega (or Inverse Omega) permutation can be routed in  $n$  steps using synchronous greedy routing on the SIMD  $n$ -dimensional binary hypercube, correcting dimensions from left to right (respectively, right to left) [166].*

**Lemma 4.6** *Normal Ascend/Descend permutations can be performed in  $\mathcal{O}(n)$  time with SIMD model [220]. Furthermore,  $2^b$  Ascend/Descend computations on the  $n$ -dimensional binary hypercube take  $\mathcal{O}(1)$  steps (using Gray-code) with MIMD model [223]. With SIMD model,  $2^b$  Descend computations take  $\mathcal{O}(n)$  steps, while  $2^b$  Ascend computations take  $\mathcal{O}(n^{1.5})$  steps with  $\mathcal{O}(1)$  space per node [223, 224].*

Varvarigos and Bertsekas studied isotropic communication tasks which are symmetric with respect to any origin node. By considering also symmetric routing algorithms, one instance of isotropic tasks, the total exchange was reduced to a matrix decomposition problem, minimizing both completion time and average packet delay on the MIMD hypercube (and wraparound mesh) [319, 320]. While the hypercube definitions of isotropic tasks were based on XOR operations, those for the wraparound mesh were based on modular addition.

**Problem 14** *Classify isotropic and nearly isotropic tasks in symmetric topologies.*

### 4.3 Fault Tolerant Routing

***P*-faulty model.** Assuming dynamic node failures and focusing on a single packet route, ignoring packet congestion, Gordon and Stout proved:

**Lemma 4.7** *If processors have no local knowledge, the probability that a single message is successfully routed to its antipodal destination on the  $n$ -dimensional hypercube, using shortest-path routing, is  $(1 - p)^{n-1}$ , and converges to zero as  $n$  approaches infinity. If processors have local knowledge, the corresponding probability is  $\prod_{i=2}^n (1 - p^i)$ , and converges to nonzero as  $n$  approaches infinity [89].*

**Worst Case Model.** Bounds on the hypercube diameter for a fixed number of faults are provided in [234, 307]. Other known results for this model are given in:

**Lemma 4.8** *By maliciously placing  $\mathcal{O}(n)$  faulty nodes on the  $n$ -dimensional binary hypercube, the hypercube can be disconnected into components of size  $\mathcal{O}(N/n^{1/2})$  [89].*

**Theorem 4.5** *Any shortest-path routing on the  $n$ -dimensional binary hypercube is  $(3, n - 1)$ -tolerant [38].*

**Lemma 4.9** *Greedy shortest-path routing on the  $n$ -dimensional binary hypercube is  $(2, n - 1)$ -tolerant [38].*

Also, if each node has at least one nonfaulty neighbor, there is a routing on the  $n$ -dimensional binary hypercube which is  $(4, 2n - 3)$ -tolerant [281].

Chen and Shin first considered routing directly on the faulty hypercube. Let's first assume that each node can determine the status of its own communication links (live or faulty). Algorithm  $A_1$  tries to move a packet closer to its destination. If all possible moves are blocked, then the packet is sent along spare dimension  $i$ , chosen so that dimension  $i$  has not been used before as a spare dimension.

**Theorem 4.6** *Algorithm  $A_1$  routes a packet on the  $n$ -dimensional binary hypercube, with  $f < n$  faulty links (or  $f < n$  faulty nodes), via an optimal path, with high probability. The expected path length is  $n + \mathcal{O}(n/N)$  (respectively,  $n + \mathcal{O}(n^3/N)$ ) [41].*

Now, we assume that each node can determine the status not only of its own communication links, but also of its neighboring components. Algorithm  $A_2$  works exactly as  $A_1$ , except that it checks if a move is blocked one more hop away.

**Theorem 4.7** *Algorithm  $A_2$  routes a packet, from its source  $u$  to its destination  $v$ , on the  $n$ -dimensional binary hypercube with  $f < n$  faulty links (or  $f < n$  faulty nodes) via a path of length at most  $H(u, v) + 2$ , where  $H$  represents the Hamming distance [41].*

In the presence of more than  $n - 1$  faults, shortest-path routing can be obtained using network delay tables. Since these tables are sometimes difficult to maintain and update, an alternative technique based on depth-first search (*DFS*) routing has been proposed. The *DFS* algorithm tries to move a packet closer to its destination. If all such moves are blocked, then an alternate path is considered. If no alternate path exists, backtracking is enforced. To avoid message looping, the *DFS* algorithm never visits a node twice, unless backtracking is enforced.

**Theorem 4.8** **DFS* routes a packet on the  $n$ -dimensional binary hypercube via an optimal path, with high probability, provided that a live path exists [40].*

Hayes and Lee considered fault tolerant routing and broadcasting schemes, when each node has no more than  $k$  faulty nodes in its  $k$ -neighborhood [172]. They proposed a routing algorithm  $A'_1$  which works as follows. A packet currently at distance  $r$  from its destination examines  $r$  node-disjoint shortest-paths, formed by use of cyclical shifts. The first path without any faults in its first  $k$  nodes is taken. If no such path is found, the  $n - r$  nonminimal paths of length  $r + 2$  are checked, and one of the remaining dimensions is shifted.

**Theorem 4.9** *If every nonfaulty node has at most  $k$  faulty nodes in its  $k$ -neighborhood ( $k \leq 2$ ), then algorithm  $A'_1$  routes a packet on the  $n$ -dimensional binary hypercube via an optimal path [172].*

Algorithm  $A'_1$  is not optimal for  $k \geq 3$ . Suppose that another algorithm  $A'_2$  works as  $A'_1$ , but searches for a feasible path, not only within the  $r$  node-disjoint paths, but within all shortest paths. Then, we have:

**Theorem 4.10** *Algorithm  $A'_2$  routes a packet on the  $n$ -dimensional binary hypercube via an optimal path, provided that every nonfaulty node has at most  $k$  faulty nodes in its  $k$ -neighborhood ( $k \leq n - 1$ ) [172].*

Finally, algorithm  $A'_3$  works as  $A'_2$ , but prefers to route along the shortest path to safe nodes, which by definition are nonfaulty nodes with at most one faulty or unsafe nearest neighbor. All other nodes are unsafe or faulty.

**Theorem 4.11** *If the number of faulty nodes is  $f < \lceil n/2 \rceil$ , algorithm  $A'_3$  routes a packet on the  $n$ -dimensional binary hypercube, from its source  $u$  to its destination  $v$ , via a path of length at most  $H(u, v) + 2$  [172].*

**Problem 15** *Compare algorithms  $A_1$  through  $A'_3$  in terms of their performance and resources needed for their implementation.*

Leighton, Maggs, and Sitaraman considered the fault tolerance of *normal hypercube algorithms*. Communication in these algorithms is always performed by correcting dimensions in consecutive order (1-port communication); furthermore, wrap-around dimensions 0 and  $n$  is allowed.

**Theorem 4.12** *An  $N$ -node faulty hypercube, with  $N^{1-\varepsilon}$  worst case faults (for any constant  $\varepsilon > 0$ ), can emulate any normal algorithm running on the  $N$ -node hypercube with only constant slowdown [181].*

Similar results have been obtained for any algorithm (not necessarily normal) running on the butterfly and the shuffle-exchange graph [181].

**IDA Model.** Rabin's information dispersal approach can be applied to the  $N$  node,  $n$ -dimensional binary hypercube to tolerate  $N/n$  link failures [195, 257]. Hastad et al. improved this result by employing a better choice of the parallel routing paths. The  $n$  pieces of a message are first sent to the neighbors of the node which generated the packet. Then, these pieces are routed along parallel paths to the  $n$  neighbors of a random intermediate node. From there, they are routed along parallel paths to the neighbors of the intended destination, and finally to the destination itself.

**Theorem 4.13** *If each component (either node, or link) of the MIMD  $n$ -dimensional hypercube fails independently with probability  $1/cn$ , where  $c$  is a sufficient large constant, then permutation routing can be performed in  $\mathcal{O}(n)$  steps using queues of size  $\mathcal{O}(n)$ , with high probability [103].*

**Problem 16** *Do any constant degree networks share the above property?*

Many fault-tolerant algorithms have been proposed for global hypercube communication and especially broadcasting [26, 39, 81, 83, 107, 108, 173, 240, 266, 329]. Most algorithms exploit the multiplicity of hypercube paths by using extensively packet replication.

#### 4.4 Dynamic Routing Problems

Every node of the  $n$ -dimensional hypercube independently generates packets with rate  $\lambda$ . Each packet's destination is chosen randomly, with each node at distance  $r$  being assigned an a priori probability  $p^r(1-p)^{n-r}$ . Thus, if  $p = 0.5$  the distribution of the destinations is uniform, while if  $p < 0.5$  the distribution favors shorter (localized) distances. Greedy routing (section 4.2) is stable if the load factor  $\lambda p < 1$ , which is the best possible.

**Theorem 4.14** *If packets are generated at each node with rate  $\lambda$ , and destinations are selected randomly with each node at distance  $r$  being assigned a probability  $p^r(1-p)^{n-r}$ , the average packet delay  $T$  on the MIMD  $n$ -dimensional binary hypercube satisfies*

$$np + \frac{n\lambda p}{2(1-\lambda p)} \leq T \leq \frac{np}{1-p},$$

for any  $\lambda p < 1$  [296].

Approximate models for dynamic routing aim at giving intuitive support for simulation results. Abraham and Padmanabhan’s analysis for the buffered/unbuffered binary hypercube is based on the approximately correct assumption that each packet moves independently of other packets [1]. This model was later modified to consider locality of references [282]. Stability conditions and approximate delay models, when packaging several processors at each hypercube node, have also been investigated [46]. Dynamic routing on the hypercube outperforms the star graph and alternating-group graph [92]. For vertex-transitive graphs, the maximum generating rate prior to saturation depends on the degree and average node distance.

Dynamic broadcasting has also been extensively considered. In this problem, packet broadcasts are generated at random instants at each node. The problem also arises in practice, when overlapping global communication with local computation, e.g. when solving first-order recurrence equations,  $X_{i+1} = A \cdot X_i$ , where  $X$  is distributed to different nodes. Although dynamic broadcasting may be based on spanning trees [297, 298], an approach based on developing efficient partial multinode broadcasting algorithms (used as subroutines) is more general, and achieves the widest range of stability with asymptotically optimal latency [321, 323].

Routing  $N$  packets from distinct nodes to independently chosen random destinations corresponds exactly to Phase I of the probabilistic routing algorithm. Therefore, greedy routing can be used to achieve  $\mathcal{O}(n)$  delay, with high probability. In this scenario, deflection routing also achieves  $\mathcal{O}(n)$  delay, with high probability [71]. Another interesting problem concerns the maximum distance  $h_{max}$  over all (source, random destination) pairs. While if  $h_{max} \leq 2$ , all  $N = 2^n$  packets with distinct source nodes can be optimally routed on the MIMD  $n$ -dimensional hypercube, there is only one general (non-tight) bound on the time complexity for all  $h_{max} = 3, 4, \dots, n$ .

#### 4.5 Cut-through - Wormhole - Circuit Switching - Hot-Potato Routing

Fast bit-serial algorithms are very difficult to construct. This is particularly true for the  $n$ -dimensional binary hypercube since the head of the packet may be far from its tail, preventing up to  $\mathcal{O}(n)$  links to be used, and raising the issue of deadlock [61]. Bit-serial algorithms on the hypercube are often based on properties of more general routing schemes, e.g. edge-disjoint routing properties of sorting networks imply efficient schemes for wormhole, circuit-switched and hot-potato routing. In particular, Aiello et al. provided a randomized algorithm for bit serial routing on the  $n$ -dimensional binary hypercube. The algorithm consists of embedding a probabilistic two-phase routing algorithm, which runs on an  $n$ -dilated butterfly of  $n2^n$  nodes, onto the  $n$ -dimensional hypercube. The embedding is based on 1-error correcting codes and achieves constant load and congestion and  $\mathcal{O}(\log N)$  dilation. Leading address bits are deleted from the header as the packet moves on. The algorithm works in the wormhole, cut-through, or circuit-switching mode.

**Theorem 4.15** *For packets  $b$  bits long, any permutation can be routed on the  $n$ -dimensional binary hypercube in  $\mathcal{O}(b + \log N)$  bit steps using constant size queues, with high probability [2].*

An  $n$ -dimensional binary hypercube is *circuit-switching rearrangeable* if for any arbitrary permutation there exist  $2^n$  edge-disjoint paths connecting all (source, destination) pairs.

**Lemma 4.10** *The  $n$ -dimensional hypercube ( $n \leq 3$ ) is circuit-switching rearrangeable [303]. If all paths are restricted to minimal lengths, the  $n$ -dimensional binary hypercube is not circuit-switching rearrangeable [194].*

Dynamic hot-potato routing has good empirical performance. Simulations and approximate models have been considered in [94, 95, 113].

## 4.6 Sorting on Hypercubes

Batcher's bitonic sorter can be embedded onto the MIMD  $n$ -dimensional binary hypercube with  $\mathcal{O}(\log^2 N)$  time delay and queue size one [19]. Bitonic sorting can be implemented more efficiently on the  $n$ -port hypercube; it takes  $\mathcal{O}(\log^2 N/\log\log N)$  time, with high probability [208]. Quicksort can also be implemented on the  $n$ -port hypercube in  $\mathcal{O}(\log^2 N/\log\log N)$  time, with high probability [328].

Furthermore, Cypher has shown that any embedding of the AKS  $\mathcal{O}(\log N)$  depth sorting network requires  $\Omega(\log^2 N)$  time on the shuffle-exchange and cube-connected cycles [52]. Furthermore, many efforts have produced lower bounds on the depth of shellsort and hypercubic sorting networks [252, 253, 254].

Recently, Cypher and Plaxton have given an involved  $\mathcal{O}(\log N \log\log N)$  deterministic sorting algorithm (called sharesort) [55]. Since the constant involved is in the thousands, this algorithm is not practical for small dimension hypercubes ( $n \leq 16$ ).

**Problem 17** *Is it possible to perform deterministic sorting on the MIMD  $n$ -dimensional binary hypercube in  $\mathcal{O}(n)$  steps?*

We have concentrated here on the saturated case, where the number of data elements matches the number of processors. For sorting techniques when  $p < N$  refer to [56, 90, 208, 318].

## 4.7 Hypercubic and Related Networks

We provide references to known results on the MIMD cube-connected cycle, de Bruijn, shuffle/exchange, hypermesh, and generalized hypercube.

In general, communication on an  $N$ -node bounded-degree network can be simulated on any hypercubic network with only  $\mathcal{O}(\log N)$  slowdown. Simulation of an  $N$ -node square mesh on the  $N$ -node butterfly requires  $\Omega(\log N)$  time, while the same simulation on the  $N$ -node hypercube can be done in  $\mathcal{O}(1)$  time [32]. However, most common communication algorithms on the SIMD hypercube are normal, e.g., Ascend/Descend communication patterns, omega permutations, etc. Besides their simplicity, all normal hypercube algorithms can be simulated on any hypercubic network efficiently, with only constant slowdown [255, 279]. Most algorithms on hypercubic networks are normal.

A *cube-connected cycle* is obtained by replacing each node in the  $n$ -dimensional binary hypercube with a ring of  $n$  nodes. As it is shown in [255], the number of nodes is  $N = n2^n$ , the degree is 3, and the diameter is  $2n + \lfloor n/2 \rfloor - 2$ . Many communication algorithms require asymptotically the same time on the cube-connected cycle as on the binary hypercube. Single node broadcast/gather run in  $\mathcal{O}(\log N)$  steps, while all other patterns (total exchange, multi-node broadcast, and single node scatter) take  $\mathcal{O}(N)$  steps [210]. Reif and Valiant have given an optimal randomized sorting algorithm on the cube-connected cycle [270], and Leighton has showed how to sort on the butterfly in  $\mathcal{O}(\log N)$  time on the butterfly with constant queues. For deterministic sorting, Plaxton has defined a nearly logarithmic depth hypercubic network based on the shuffle/unshuffle permutations [251]. These algorithms also imply the best upper bounds for wormhole and hot-potato routing on hypercubic networks. Also, Cypher and Plaxton's sharesort algorithm is a normal algorithm. Hence, it can be implemented on any hypercubic network with only constant slowdown [250].

The undirected  $n$ -dimensional  $d$ -way shuffle (also called generalized de Bruijn network) has  $N = d^n$  nodes, diameter  $n$  and degree  $2d$ . A node represented as  $d_1 d_2 \dots d_n$ ,  $0 \leq d_i \leq d - 1$ ,  $1 \leq i \leq n$  is connected to nodes  $d_2 d_3 \dots d_n p$ , and  $p d_1 d_2 \dots d_{n-1}$ , where  $0 \leq p \leq d - 1$ . Greedy routing corresponds to correcting dimensions from left to right, but it does not produce a shortest-path route [28]. Aleliunas extended Valiant's probabilistic algorithm to dynamic routing on the  $d$ -way shuffle [9]. On the  $n$ -way shuffle ( $d = n$ ), deterministic

sorting, by embedding Batcher's sorting network, achieves  $\mathcal{O}(n^2)$  delay [229], probabilistic two-phase routing is asymptotically optimal [236], and off-line *BPC*, *Omega*, and *Inverse Omega* routing is optimal [275]. Wei and Hsu have provided upper and lower bounds for deterministic oblivious permutation routing and sorting on the directed version of the de Bruijn network [112].

The undirected *n-dimensional shuffle/exchange* has  $N = 2^n$  nodes, diameter  $2n - 1$ , and degree 3. A node represented by a binary string  $d_1 d_2 \dots d_n$ ,  $d_i \in \{0, 1\}$ ,  $1 \leq i \leq n$  is connected to node  $d_1 d_2 \dots d_{n-1} \bar{d}_n$  (via an exchange link), and nodes  $d_n d_1 d_2 \dots d_{n-1}$ , and  $d_2 d_3 \dots d_n d_1$  (via shuffle and unshuffle links). By embedding a leveled network on the *n-dimensional shuffle/exchange*, and applying the scheduling algorithm of Theorem 2.3, we obtain an asymptotically optimal probabilistic routing algorithm [178]. Using Theorem 2.6, optimal off-line permutation routing can be designed for the *n-dimensional shuffle/exchange*, de Bruijn, and cube-connected cycle with single node failures.

An *n-dimensional hypermesh* has  $N = d^n$  nodes arranged in *n-dimensional space*. All nodes which differ in exactly one base-*d* digit are connected with a network, which can realize a permutation of data between all its members in one time unit. The degree of the *n-dimensional hypermesh* with  $d^n$  nodes is *d*, and the diameter is *n*. Notice that for  $d = 2$ , the hypermesh degenerates to the binary hypercube. A greedy hypermesh routing algorithm corrects dimensions in an ordered fashion, until the packet reaches its destination. Furthermore *BPC*, *Omega* and *Inverse Omega* permutations can be performed in optimal time, with similar algorithms as those previously given for the *n-dimensional binary hypercube*. For hypermeshes with  $d = n$ , probabilistic two-phase *k*-relation routing, dynamic routing, and routing of *N* packets to independently chosen random destinations can all be performed optimally [304, 305].

An *n-dimensional base-d generalized hypercube* is an undirected graph with  $N = d^k$  nodes, degree  $(d - 1) \log_d N$ , and diameter  $\log_d N$  [33]. A node of the generalized hypercube is represented as  $x = x_1 x_2 \dots x_n$ , where  $x_i \in \langle d \rangle = \{0, 1, \dots, d - 1\}$ , for  $1 \leq i \leq n$ . Two nodes  $x = x_1 x_2 \dots x_n$  and  $y = y_1 y_2 \dots y_n$  of the hypercube are connected by a link, if  $x_i \neq y_i$ , and  $x_j = y_j$  for all  $j \neq i$  and  $1 \leq i, j \leq n$ . By applying Theorem 2.5, probabilistic two-phase permutation routing runs in optimal  $\mathcal{O}(\log_b N)$  time, with high probability. Routing *BPC*, *Omega* and *Inverse Omega* permutations can be performed in  $\log_d N$  steps, with algorithms similar to those given for the *n-dimensional binary hypercube*. Furthermore, by embedding an  $N = nd^n$  node, base-*d* generalized Beneš permutation network onto *n-dimensional base-d generalized hypercube* (with congestion and dilation one), any permutation can be off-line routed in  $2 \log_d N - 1$  steps [308]. Optimal algorithms for multinode broadcasting, single node scatter, and total exchange on the generalized hypercube are again based on multiple edge-disjoint spanning trees [79].

## 5 The Star Graph

Let  $X = \{1, 2, \dots, n\}$  where  $n \geq 2$ . Let  $S_n$  be the set of all possible permutations over  $X$ . Then,  $S_n$  forms a permutation group, known as *symmetric group*. A set  $\Omega$  is said to be a *set of generators* for group  $\Gamma$ , if by applying the elements of  $\Omega$  repeatedly, we obtain group  $\Gamma$ . Let  $T_{ij}$  be the permutation which swaps the *i*-th element and the *j*-th element. It can be verified that the set  $\Omega_1 = \{T_{1j} \mid 2 \leq j \leq n\}$  is a set of generators for  $S_n$ .

A *Cayley graph*  $G(V, E)$  obtained from group  $\Gamma$  and set of generators  $\Omega$ , is defined with  $V = \Gamma$  and  $E = \{(p, q) \mid q = p \cdot g, \text{ where } p, q \in \Gamma \text{ and } g \in \Omega\}$ . An *n-dimensional star graph* is a Cayley graph obtained from symmetric group  $S_n$ , and the set of generators  $\Omega_1$  [5]. The star graph has  $N = n!$  nodes, degree  $n - 1$ , and diameter  $\lfloor 3(n - 1)/2 \rfloor$ . Although both the star-graph, and the hypercube are Cayley graphs, they are topologically different [284].

## 5.1 On-Line Routing

Let an arbitrary packet  $\Psi$  originate at source node  $x = x_1 x_2 \dots x_{n-1} x_n$  and heads to destination node  $x' = x'_1 x'_2 \dots x'_{n-1} x'_n$ . Greedy routing corrects dimensions from left to right by applying the generator  $\Omega_1$ .

```

Greedy routing (“cobegin .. coend” refers to all packets).
cobegin
while  $x \neq x'$  do
  if  $x_1 = x'_1$  then
    transmit  $\Psi$  from  $x$  to  $x \cdot T_{1i}$ , where  $x_i \neq x'_i$ 
  else transmit  $\Psi$  from  $x$  to  $x \cdot T_{1i}$ , where  $x_1 = x'_i$ 
coend

```

If this algorithm is implemented synchronously (all packets correct the  $i^{\text{th}}$  dimension before they consider the  $(i+1)$ th dimension, or vice-versa) using FIFO queues, it has  $\mathcal{O}((n!)^{1/2})$  worst case time complexity.

**Lemma 5.1** *Greedy permutation routing on the MIMD  $n$ -dimensional star graph runs in  $8(n!)^{1/2} + o((n!)^{1/2})$  steps using queues of size  $\mathcal{O}((n!)^{1/2})$  [236].*

If packets execute the greedy algorithm asynchronously, the time complexity is slightly better, but with the same asymptotics. Greedy routing on the star graph can be used to perform isotonic routing [265]. The algorithm for packings first uses prefix computation to determine the correct destination of each packet. Prefix can be computed with a group-copy procedure (similar to Ascend/Descend) in  $\mathcal{O}(n \log n)$  steps [7, 8]. After that, greedy routing is used. In this phase there are at most  $i - 1$ ,  $2 \leq i \leq n$  link conflicts for a packet correcting its  $i^{\text{th}}$  dimension.

**Lemma 5.2** *A packing of  $M \leq n!$  packets can be routed on the MIMD  $n$ -dimensional star graph in  $\mathcal{O}(n^2)$  steps [265].*

**Lemma 5.3** *A sequence of  $n$  packings can be routed on a pipelined MIMD  $n$ -dimensional star graph in  $\mathcal{O}(n^2)$  steps [265].*

An adaptive permutation routing algorithm repeats  $n$  phases. Each phase consists of greedy routing followed by a packing that distributes one packet per each star node. The  $i^{\text{th}}$  phase requires  $n - i$  greedy routing steps and  $(n - i)^2$  packing steps. Summing over all  $n$  steps of the algorithm, the following result is obtained:

**Theorem 5.1** *Any permutation can be routed on the MIMD  $n$ -dimensional star graph in  $\mathcal{O}(n^3)$  steps [265].*

A two-phase randomized permutation routing algorithm for the star graph first sends packets to random destinations and then to their final destinations. It is optimal in terms of its time complexity.

**Theorem 5.2** *Permutation routing on the MIMD  $n$ -dimensional star graph can be completed in  $cn$  steps, with high probability ( $\geq 1 - 1/(n!)^c$ ), where  $c$  is a constant [236].*

**Claim 5.1** *Probabilistic permutation routing on the  $n$ -dimensional star graph can be performed in  $\mathcal{O}(n)$  steps using constant size queues.*

## 5.2 Off-Line Permutation Routing

Using Theorem 2.6 one can prove:

**Corollary 5.1** *Any permutation can be off-line routed on the  $n$ -dimensional star graph in  $4n - 7$  steps using queues of size one [20].*

**Claim 5.2** *Any BPC, Omega, or Inverse Omega permutation can be routed in optimal time (close to the diameter bound) on the  $n$ -dimensional star graph.*

## 5.3 Fault Tolerant Routing

Bounds on the star graph diameter for a given number of faults are derived in [170]. Some results for fault tolerant routing are:

**Lemma 5.4** *In general, any shortest-path routing on the  $n$ -dimensional star graph is  $(3, n - 1)$ -tolerant [84].*

**Theorem 5.3** *A depth-first search algorithm implementing backtracking can route a packet on the  $n$ -dimensional star graph, with  $f \leq n - 2$  faulty links, via a path of length at most  $2n^{1/2} + o(n^{1/2})$  from optimal [14].*

Results for cut-through, wormhole routing, and circuit switching on the MIMD  $n$ -dimensional star graph are rare. Also, it is not known how to route optimally on star graphs of small dimension.

## 5.4 Global Communication and Sorting on Star Graph

Fragopoulou and Akl have provided optimal communication algorithms for multinode broadcasting, single node scatter, and total exchange on the star graph for either SIMD, and MIMD communication [77]. The algorithms are based on multiple edge-disjoint spanning trees which also admit fault-tolerant communication on the star graph [78].

Shannon's information theoretic bound provides a lower-bound for comparison-based sorting on the star graph, assuming a constant number of local comparisons after each communication step.

**Theorem 5.4** *Sorting of  $N = n!$  data on the MIMD  $n$ -dimensional star graph requires  $\Omega(n \log n)$  steps.*

Also, Corollary 5.1 implies that the  $n$ -dimensional star graph can simulate any bounded-degree network with  $\mathcal{O}(n)$  slowdown. Since there are bounded-degree networks which sort in logarithmic time (to the number of nodes), the best upper bound for sorting on the star graph is at most  $\mathcal{O}(n^2 \log N)$ .

Sorting algorithms on the  $n$ -dimensional star graph are based on shearsort [212] and bitonic sort [265]. The bitonic sort on the star graph ( $ST_n$ ) can be briefly described as follows. Each subgroup  $ST_{n-1}$  of the star graph is recursively sorted in ascending or descending order depending on whether the subgraph is odd or even numbered. Then, each pair is merged into a new bitonic sequence, which is again used to obtain a larger bitonic sequence. This is repeated  $\lceil \log n \rceil$  times, each time doubling the size of the sorted sequence.

**Theorem 5.5**  *$N = n!$  data can be sorted on the MIMD  $n$ -dimensional star graph in  $\mathcal{O}(n^3 \log n)$  steps [212, 265].*

**Theorem 5.6**  *$N = n!$  data can be sorted on the MIMD  $n$ -dimensional star graph in  $\mathcal{O}(n^3)$  steps, with high probability [265].*

**Problem 18** *Is it possible to bridge the gap between the lower bound and the upper bound for sorting on the MIMD  $n$ -dimensional star graph?*

## 5.5 Other proposed graphs

Many graphs with good topological properties have been proposed, e.g. distributed loop networks [27], Kautz graph [28], Cayley graphs [29, 126, 168], and expanders, such as fat-trees [184]. Although these graphs have asymptotically optimal diameter, very little is known about their routing properties. Another interesting graph is a Cartesian product of two fully connected graphs, called the  $k$ -ary  $n$ -cube [61]. In this context, the binary hypercube is a 2-ary  $n$ -cube.

## 6 Conclusion

In this paper we have attempted to provide a complete overview over a subfield of a subfield of parallel computing. Even within such a restricted area we have made omissions, for which we apologize. Still we believe that by using the results of the referenced papers, particularly the more recent ones, one can access the covered topics. We hope that together with the short introductions we have provided, this may be of use to the interested.

## 7 Acknowledgements

At this place we want to acknowledge the effort of one of the referees, who has shown a great deal of expertise and has suggested many changes, most of them improvements.

The research of the third author was supported by the Co-operative Centre for Robust and Adaptive Systems (CRASys).

## References

- [1] Abraham, S., and Padmanabhan, K. Performance of the direct binary  $n$ -cube network for multiprocessors. *IEEE Trans. Comput.* **38** (7), 1989, pp. 1000–1011.
- [2] Aiello, W. A., Leighton, F. T., Maggs, B. M., and Newman, M. Fast algorithms for bit serial routing on a hypercube. *Math. Syst. Theory.* **24**, 1991, pp. 253–271.
- [3] Ajtai, M., Komlós, J., and Szemerédi, E. Sorting in  $c \log n$  parallel steps. *Combinatorica.* **3**, 1983, pp. 1–19.
- [4] Ajtai, M., Komlós, J., and Szemerédi, E. Halvers and expanders. *Proc. 33rd IEEE Symp. Foundations Comput. Sci.* 1992, pp. 686–692.
- [5] Akers, S. B., Harel, D., and Krishnamurthy, B. The star graph, an attractive alternative to the  $n$ -cube. *Proc. Int. Conf. Parallel Proc.* 1987, pp. 393–400.
- [6] Akl, S. G. *Parallel sorting algorithms*. Academic Press, 1985.
- [7] Akl, S. G., and Qiu, K. *A novel routing scheme on the star and pancake networks and its applications*. *Parallel Comput.* **19**, 1993, pp. 95–101.
- [8] Akl, S. G., Qiu, K., and Stojmenovic, I. *Fundamental algorithms for the star and pancake interconnection networks with applications to computational geometry*. *Networks, special issue on “Interconnection Networks and Algorithms”*. **23** (4), 1993, pp. 215–226.
- [9] Aleliunas, R. Randomized parallel communication. *Proc. ACM SIGACT-SIGOPS Symp. Princ. Distrib. Comput.* 1982, pp. 60–72.
- [10] Alon, N., Chung, F. R. K., and Graham, R. L. Routing permutations on graphs via matchings *SIAM J. Disc. Math.*, **23** (7), 1994, pp. 513–530.
- [11] Anjan, K. V., and Pinkston, T. M. An efficient, fully adaptive deadlock recovery scheme: DISHA *Proc. 24th IEEE Int. Symp. Comput. Arch.*, 1995, pp. 210–219.
- [12] Arora, S., Leighton, F. T., and Maggs, B. M. On-line algorithms for path selection in a nonblocking network. *SIAM J. Comput.* **25** (3), 1996, pp. 600–625.
- [13] Badr, H. G., and Podar, S. An optimal shortest-path routing policy for network computers with regular mesh-connected topologies. *IEEE Trans. Comput.* **C-38** (10), 1989, pp. 1362–1371.
- [14] Bagherzadeh, N., Nassif, N., and Latifi, S. A routing and broadcasting scheme on faulty star graph. *IEEE Trans. Comput.* **C-42** (11), 1993, pp. 1398–1403.

- [15] Bannister, J., Gerla, M., and Kovačević, M. Routing in optical networks. *in M. Steenstrup (ed.), Routing in Comm. Networks*. Prentice-Hall, **7**, 1995.
- [16] Baran, P. On distributed communication networks. *IEEE Trans. Comm.* **C-12** (1), 1964, pp. 1–9.
- [17] Bar-Noy, A., Schieber, B., Raghavan, P., and Tamaki, H. Fast deflection routing for packets and worms. *Proc. 12th ACM SIGACT-SIGOPS Symp. Princ. Distrib. Comput.* 1993, pp. 75–86.
- [18] Bassalygo, L. A., and Pinsker, M. S. Complexity of an optimum nonblocking switching network without reconstructions. *Problems of Inform. Transm.* **9**, 1974, pp. 64–66.
- [19] Batchner, K. E. Sorting networks and their applications. *Proc. AFIPS Spring Joint Comput. Conf.* **32**, 1968, pp. 307–314.
- [20] Baumslag, M., and Annexstein, F. A unified framework for off-line permutation routing in parallel networks. *Math. Syst. Theory.* **24**, 1991, pp. 233–251.
- [21] Ben-Asher, Y., Newman, I. Geometric approach for optimal routing on mesh with buses. *Proc. 7th IEEE Symp. Parallel Distrib. Comput.*, 1995, pp. 145–153.
- [22] Ben-Asher, Y., Peleg, D., Ramaswami, R., and Schuster, A. The power of reconfiguration. *J. Parallel Distrib. Comput.* **13**, 1991, pp. 139–153.
- [23] Ben-David, S., Borodin, A., Karp, R., Tardos, G., and Wigderson, A. On the power of randomization in on-line algorithms. *Algorithmica.* **11** (1), 1994, pp. 2–14.
- [24] Beneš, V. E. On rearrangeable three-stage connecting networks. *Bell Syst. Tech. J.* **16** (5), 1962, pp. 1481–1491.
- [25] Berenbrink, P., Meyer auf der Heide, F. and Stemmann, V. Fault-tolerant shared memory simulations. *Proc. 13th Symp. Theoret. Asp. Comput. Sci., Lect. Notes Comput. Sci.* **1046**, 1996, pp. 181–192.
- [26] Berman P., Diks, K., and Pelc A. Reliable broadcasting in logarithmic time with byzantine link failures. *J. Algorithms*, **22** (2), 1997, pp. 199–211.
- [27] Bermond, J. C., Comellas, F., and Hsu, D. F. Distributed loop networks: a survey. *J. Parallel Distrib. Comput.* **24**, 1995, pp. 2–10.
- [28] Bermond, J. C., and Peyrat, C. De Bruijn and Kautz networks: A competitor for the hypercube? *Proc. 1st Euro Workshop Hypercube Distrib. Comput.* 1989, pp. 279–293.
- [29] Berthomé, P., Ferreira, A., Perennes S. Optimal information dissemination in star and pancake networks. *IEEE Trans. Parallel Distrib. Syst.* **C-7** (12), 1996, pp. 1292–1300.
- [30] Bertsekas, D. P., and Gallager, R. G. *Data networks*. Prentice Hall, 1992.
- [31] Bertsekas, D. P., Ozveren, C., Stamoulis, G. D., et al. Optimal communication algorithms for hypercubes *J. Parallel Distrib. Comput.* **11** (4), 1991, pp. 263–275.
- [32] Bhatt, S. N., Chung, F. R. K., Hong, J. W., et al. Optimal emulations by butterfly-like networks. *J. ACM.* **43** (2), 1996, pp. 293–330.
- [33] Bhuyan, L. N., and Agrawal, D. P. Generalized hypercube and hyperbus structures for a computer network. *IEEE Trans. Comput.* **C-33** (4), 1984, pp. 323–334.
- [34] Borodin, A. Towards a better understanding of pure packet routing. *Proc. 3rd Workshop Alg. and Data Struct., Lect. Notes Comput. Sci.* **709**, 1993, pp. 14–25.
- [35] Borodin, A., and Hopcroft, J. E. Routing, merging, and sorting on parallel models of computation. *J. Comput. Syst. Sci.* **30** (1), 1985, pp. 130–145.
- [36] Borodin, A., Raghavan, P., Schieber, B., and Upfal, E. How much can hardware help routing. *J. ACM.* **44** (5), 1997, pp. 726–741.
- [37] Brassard, G., and Bratley, P. *Algorithmics: theory and practice*. Prentice-Hall, Englewood Cliffs, 1988.
- [38] Broder, A., Fischer, M., Dolev, D., and Simmons, B. Efficient fault-tolerant routings in networks. *Information and Computation.* **75**, 1987, pp. 52–64.
- [39] Bruck, J., Cypher, R., and Ho, C. T. Wildcard dimensions, coding theory and fault-tolerant meshes and hypercubes. *IEEE Trans. Parallel Distrib. Syst.* **C-44** (1), 1995, pp. 150–155.
- [40] Chen, M. S., and Shin, K. G. Depth-first search approach for fault-tolerant routing in hypercube multicomputers. *IEEE Trans. Parallel Distrib. Syst.* **C-1** (2), 1990, pp. 152–159.
- [41] Chen, M. S., and Shin, K. G. Adaptive fault-tolerant routing in hypercube multiprocessors. *IEEE Trans. Comput.* **C-39** (12), 1990, pp. 1406–1416.
- [42] Chinn, D. D., Leighton, T. and Tompa, M. Minimal adaptive routing on the mesh with bounded queue size. *Proc. 6th ACM Symp. Parallel Alg. Arch.* 1994, pp. 354–363.
- [43] Chlebus, B. S., Czumaj, A., and Gąsieniec, L., Kowaluk, M., Plandowski, W. Parallel alternating-direction access machine. *Proc. 21st Math. Foundations Comput. Sci., Lect. Notes Comput. Sci.* **1113**, 1996, pp. 267–278.

- [44] Chlebus, B. S., Czumaj, A., and Sibeyn, J. F. Routing on the PADAM: Degrees of Optimality. *Proc. 3rd Euro-Par Conf., Lect. Notes Comput. Sci.* **1300**, 1997, pp. 272–279.
- [45] Chlebus, B. S., and Kukawka, M. A guide to sorting on the mesh-connected processor array. *Comput. Art. Intel.* **9**, 1990, pp. 599–610.
- [46] Chowdhury, S., and Holliday, A. M. Stability and performance of alternative two-level interconnection networks. *Proc. Int. Conf. Parallel Proc.* 1991, **I**, pp. 720–721. Also, in *Tech. Rep. CS-1991-12, Dept. Comput. Sci., Duke University.* 1991.
- [47] Cole, R., and Hopcroft, J. On edge coloring bipartite graphs. *SIAM J. Comput.* **11**, 1982, pp. 540–546.
- [48] Cole, R., Maggs, B., and Sitaraman, R. Multi-scale self-simulation: a technique for reconfiguring arrays with faults. *Proc. 25th ACM Symp. Theory Comput.* 1993, pp. 561–572. Also, part-I in *SIAM J. Comput.* **26** (6), 1997, pp. 1581–1611.
- [49] Coolsaet, K., Meyer auf der Heide, F., and Fack, V. Optimal algorithms for total exchange without buffering on the hypercube. *Bit.* **32**, 1992, pp. 559–569.
- [50] Chvátal, V. Lecture notes on the new AKS sorting network. *Tech. Rep. TR-DCS-294, Comput. Sci., Rutgers University.* 1992.
- [51] Cypher, R. Efficient communication in massively parallel computers. *PhD Thesis, Dept. Comput. Sci., University of Washington.* 1989.
- [52] Cypher, R. Theoretical aspects of VLSI pin limitations. *SIAM J. Comput.* **22** (2), 1993, pp. 356–378.
- [53] Cypher, R., and Gravano, L. Storage-efficient, deadlock-free packet routing algorithms for torus networks. *IEEE Trans. Comput.* **43** (12), 1994, pp. 1376–1385.
- [54] Cypher, R., Meyer auf der Heide, F., Scheideler, C. and Vöking, B. Universal algorithms for store-and-forward and wormhole routing. *Proc. 28th ACM Symp. on Theory of Computing*, 1996, pp. 356–365.
- [55] Cypher, R., and Plaxton, C. G. Deterministic sorting in nearly logarithmic time on the hypercube and related computers. *J. Comput. Syst. Sci.* **47**, 1993, pp. 501–548.
- [56] Cypher, R., and Sanz, J. L. C. Cubesort: a parallel algorithm for sorting  $N$  data items with  $S$ -sorters. *J. Alg.* **13** (2), 1992, pp. 211–234.
- [57] Czumaj, A., Meyer auf der Heide, F., and Stemann, V. Shared memory simulations with triple-logarithmic delay. *Proc. 3rd European Symp. Alg., Lect. Notes Comput. Sci.* **979**, 1995, pp. 46–59.
- [58] Czumaj, A., Meyer auf der Heide, F., and Stemann, V. Simulating shared memory in real time: on the computation power of reconfigurable architectures. *Information and Computation*, **137** (2), 1997, pp. 103–120.
- [59] Dally, W. J. Virtual-channel flow control. *IEEE Trans. Parallel Distrib. Syst.* **C-2** (2), 1992, pp. 194–205.
- [60] Dally, W. J., and Aoki, H. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Trans. Parallel Distrib. Syst.* **C-4** (4), 1993, pp. 466–474.
- [61] Dally, W. J., and Seitz, C. Deadlock free message routing in multiprocessor interconnection networks. *IEEE Trans. Comput.* **C-36** (5), 1987, pp. 547–553.
- [62] Das, C. R., Kreulen, J. T., Thazhuthaveetil, M., et al. Dependability modeling for multiprocessors. *IEEE Computer.* **23** (10), 1990, pp. 7–19.
- [63] Dasgupta, B., Hwang, F.K., and Yao, Y.C. Permutation routing in  $t$ -nary hypercubes. *J. Parallel Distrib. Comput.* 1995, submitted.
- [64] Dietzfelbinger, M., and Meyer auf der Heide, F. Simple, efficient shared memory simulations. *Proc. 5th ACM Symp. Parallel Alg. Arch.* 1993, pp. 110–118.
- [65] Ding, K.-S., Ho, C.-T., and Tsay, J.-J. Matrix transpose on meshes with wormhole and XY-routing. *Proc. 6th IEEE Symp. Parallel Distrib. Comput.* 1994, pp. 656–663.
- [66] Dolev, D., Halpern, J., Simmons, B., and Strong, R. A new look at fault tolerant network routing. *Information and Computation.* **72**, 1987, pp. 180–196.
- [67] Duato, J. A necessary and sufficient condition for deadlock-free routing in cut-through and store-and-forward networks. *IEEE Trans. Parallel Distrib. Syst.* **C-7** (8), 1996, pp. 841–854.
- [68] Duato, J. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.* **C-6** (10), 1995, pp. 1055–1067.
- [69] Duato, J. Theory of deadlock-free adaptive multicast routing in wormhole networks. *IEEE Trans. Parallel Distrib. Syst.* **C-6** (9), 1995, pp. 976–987.
- [70] Duato, J., Yalamanchili, S., and Ni, L. *Interconnection networks: an engineering approach.* IEEE Computer Society Press, 1997.

- [71] Feige, U., and Raghavan, P. Exact analysis of hot-potato routing. *Proc. 33rd IEEE Symp. Foundations Comput. Sci.* 1992, pp. 553–562.
- [72] Feldman, P. Fault tolerance of minimal path routing in a network. *Proc. 17th ACM Symp. Theory Comput.* 1985, pp. 327–334.
- [73] Felperin, S., Raghavan, P., and Upfal, E. A theory of wormhole routing in parallel computers. *IEEE Trans. Comm.* **C-45** (6), 1996, pp. 704–713.
- [74] Feng T. Y. A survey of interconnection networks. *IEEE Computer.* **14**, 1981, pp. 12–27.
- [75] Ferreira, A. Hypercubes. *Handbook Parallel Distrib. Comput., A. Zomaya (ed.)* Mc Graw Hill, 1995.
- [76] Ferreira, A., and Grammatikakis, M. D. Improved probabilistic routing on generalized hypercubes. *Theoret. Comput. Sci., North-Holland.* **158** (1-2), 1995, pp. 53–64.
- [77] Fragopoulou, P., and Akl, S. G. Optimal communication algorithms on star graphs using spanning tree constructions. *J. Parallel Distrib. Comput.* **24** (1), 1995, pp. 55–71.
- [78] Fragopoulou, P., and Akl, S. G. Edge-disjoint spanning trees on the star network with applications to fault tolerance. *IEEE Trans. Comput.* **45** (2), 1996, pp. 174–185.
- [79] Fragopoulou, P., Akl, S. G., and Meijer, H. Optimal communication primitives on the generalized hypercube network. *J. Parallel Distrib. Comput.* **32** (2), 1996, pp. 173–187.
- [80] Fraigniaud, P. Performance analysis of broadcasting in hypercubes with restricted communication capabilities. *J. Parallel Distrib. Comput.* **16** (1), 1992, pp. 15–26.
- [81] Fraigniaud, P. Asymptotically optimal broadcasting and gossiping in faulty hypercubes multicomputers. *IEEE Trans. Comput.* **C-41** (11), 1992, pp. 1410–1419.
- [82] Fraigniaud, P., and Lazard, E. Methods and problems of communication in usual networks. *Disc. Appl. Math., special issue on “Broadcasting and Gossiping”.* **53**, 1994, pp. 79–133.
- [83] Fraigniaud, P., and Peyrat, C. Broadcasting in a hypercube when some calls fail. *Inf. Proc. Letters.* **39** (3), 1991, pp. 115–119.
- [84] Gargano, L., Vacarro, U., and Vozella, A. Fault tolerant routing in the star and pancake interconnection networks. *Inf. Proc. Letters.* **45**, 1993, pp. 315–320.
- [85] Gautier de Lahaut, D., and Germain, C. Static communications in parallel scientific programs. *Proc. Parallel Arch. Lang., Europe Conf., Lect. Notes Comput. Sci.* **817**, 1994, pp. 262–276.
- [86] Glass, C. J., and Ni, L. M. The turn model for adaptive routing. *J. ACM.* **41** (5), 1994, pp. 874–902.
- [87] Goldberg, L. A., Matias, Y., and Rao, S. An Optical simulation of shared memory. *Proc. 6th ACM Symp. Parallel Alg. Arch.* 1994, pp. 257–267.
- [88] Gordon, J. M., and Stout, Q. F. Hypercube message routing in the presence of faults. *Proc. 3rd Conf. Hypercube Concurrent Comput. Appl.* 1988, pp. 318–327.
- [89] Gordon, J. M., and Stout, Q. F. Fault tolerant message routing on large parallel systems. *Proc. 2nd Symp. Frontiers Massively Parallel Comput.* 1988, pp. 155–158.
- [90] Gottlieb, A., and Kruskal, C. P. Complexity results for permuting data and other computations on parallel processors. *J. ACM.* **31** (2), 1984, pp. 193–209.
- [91] Grammatikakis, M. D., Hsu, D. F., and Hwang, F. K. Adaptive and oblivious routing algorithms on the d-cube. *Proc. 4th ACM Int. Symp. Alg. Comput., Lect. Notes Comput. Sci.* **762**, 1993, pp. 167–176.
- [92] Grammatikakis, M. D., Jwo, J. S., Kraetzl, M., and Wang, S. H. Dynamic and static packet routing on symmetric communication networks. *Proc. IEEE Conf. Global Comm., San Francisco, Cal., 1994*, pp. 1501–1505.
- [93] Grammatikakis, M. D., Kraetzl, M., and Fleury, E. Shortest-path and hot-potato routing on unbuffered toroidal networks. *Proc. IEEE Conf. Global Comm., Phoenix, AZ, 1997*, pp. 165–169.
- [94] Greenberg, A. G., and Goodman, J. Sharp approximate models of deflection routing in mesh networks. *IEEE Trans. Comm.* **C-41** (1), 1993, pp. 210–223.
- [95] Greenberg, A. G., and Hajek, B. Deflection routing in hypercube networks. *IEEE Trans. Comm.* **C-40** (6), 1992, pp. 1070–1081.
- [96] Hambrusch, S.E., Hameed, F., and Khokhar, A. Communication operations on coarse-grained architectures. *Parallel Computing*, **21**, 1995, pp. 731–751.
- [97] Han, Y., Igarashi, Y., and Truszczynski, M. Indexing functions and time lower bounds for sorting on a mesh-connected computer. *Disc. Appl. Math.* **36**, 1992, pp. 141–152.
- [98] Han, T., and Stanat, D. F. “Move and smooth” routing algorithms on mesh connected computers. *Proc. 28th Allerton Conf. Comm. Control Comput.* 1990, pp. 236–245.

- [99] Harchol-Balter, M., and Wolfe, D. Bounding delays in packet-routing networks. *Proc. 27th ACM Symp. on Theory of Comput.* 1995, pp. 248–257.
- [100] Harchol-Balter, M., and Black, P. E. Queuing analysis of oblivious packet-routing networks. *Proc. 5th ACM-SIAM Symp. Disc. Alg.*, 1994, pp. 583–592.
- [101] Harms, D. D., Kraetzl, M., Colbourn, C. J., and Devitt, J. S. *Network reliability: experiments with a symbolic algebra environment*. CRC Press, 1995.
- [102] Hastad, J., Leighton, F. T., and Newman, M. Reconfiguring a hypercube in the presence of faults. *Proc. 19th ACM Symp. Theory Comput.* 1987, pp. 274–284.
- [103] Hastad, J., Leighton, F. T., and Newman, M. Fast computation using faulty hypercubes. *Proc. 21st ACM Symp. Theory Comput.* 1989, pp. 251–263.
- [104] Hedetniemi, S. M., Hedetniemi, S. T., and Liestman, A. L., A survey of gossiping and broadcasting in communication networks. *Networks*. 1987, **18**, pp. 319–349.
- [105] Herley, K. T., and Bilardi, G. Deterministic simulations of PRAMs on bounded-degree networks. *SIAM J. Computing*, **23** (2), 1994, pp. 276–292.
- [106] Herley, K. T., Pietracaprina, A., and Pucci, G. Implementing shared memory on multi-dimensional meshes and on the fat-tree. *Proc. 3rd European Symp. Alg., Lect. Notes Comput. Sci.* **979**, 1995, pp. 60–74.
- [107] Ho, C. T., and Kao, M. Y. Optimal broadcast in all-port wormhole-routed hypercubes. *IEEE Trans. Parallel Distrib. Syst.* **C-6** (2), 1995, pp. 200–204.
- [108] Ho, C. T., and Kao, M. Y. Efficient broadcast on hypercubes with wormhole and  $e$ -cube routings. *Parallel Processing Letters*, **5** (2), 1995, pp. 213–222.
- [109] Hromkovič, J., Klasing, R., Monien, B., and Peine, R. Dissemination of information in interconnection networks (broadcasting and gossiping). in *Hsu, D. F., Du, D. Z. (eds.), Comb. Network Theory, Kluwer Academic Publishers*. 1995.
- [110] Hsu, D. F. On container width and length in graphs, groups and networks. *IEEE Trans. Fundamentals of Electronics, Comm., and Comput. Sci.* **E77-A** (4), 1994, pp. 668–680.
- [111] Hsu, D. F., and Tomasz, L. Note on the  $k$ -diameter of  $k$ -regular  $k$ -connected graphs. *Disc. Math.* 1995, **132**, 1994, pp. 291–296.
- [112] Hsu, D. F., and Wei, D.S. Efficient routing and sorting schemes for de Bruijn networks. *IEEE Trans. Parallel Distrib. Syst.* **8** (11), 1997, pp. 1157–1170.
- [113] Hsu, J. M., and Banerjee, P. Performance evaluation of hardware support for message passing in distributed memory multicomputers. *Proc. Int. Conf. Parallel Proc.* 1991, **I**, pp. 604–607.
- [114] Hsu, W. T.-Y., and Yew P.-C. The impact of wiring constraints on hierarchical network performance. *Proc. 6th IEEE Int. Parallel Proc. Symp.* 1992, pp. 580–588.
- [115] Huang, A. J., and Mou, Z. G. Parallel partition expansion for the solution of arbitrary recurrences. *Proc. Int. Conf. Parallel Proc.* 1992, **III**, pp. 109–114.
- [116] Hwang, F. K., Yao, Y. C., and Grammatikakis, M. D.  $d$ -move local permutation routing for the  $d$ -cube. *Disc. Appl. Math.* **72** (3), 1997, pp. 199–207.
- [117] Ierardi, D. 2D-bubblesorting in average time  $\mathcal{O}(\sqrt{N \lg N})$ . *Proc. 6th ACM Symp. Parallel Alg. Arch.* 1994, pp. 36–45. Also, in *J. Comput. Syst. Sci.* 1998, to appear.
- [118] Jaja, J. *An introduction to parallel algorithms*. Addison-Wesley Publishing Company, 1992.
- [119] Jaja, J., and Ryu, K. W. Load balancing and routing on the hypercube and related networks. *J. Parallel Distrib. Comput.* **14**, 1992, pp. 431–435.
- [120] Jesshope, C. R., Miller, P. R., and Yantchev, Y. S. High performance communications in processor networks. *Proc. 16th IEEE Int. Symp. Comput. Arch.* 1989, pp. 150–157.
- [121] Jesshope, C. R., Miller, P. R., and Yantchev, Y. S. The mad-postman network chip. in *Proc. Transputing Conf., IOS Press* 1991, **II**, pp. 517–536.
- [122] Johnsson, S. L., and Ho, C. T. Optimum broadcasting and personalized communication in hypercubes. *IEEE Trans. Comput.* **C-38** (12), 1989, pp. 1249–1267.
- [123] Johnsson, S. L., and Ho, C. T. Generalized shuffle permutations on boolean cubes. *J. Parallel Distrib. Comput.* **16**, 1992, pp. 1–14.
- [124] Johnsson, S. L., and Ho, C. T. Optimal communication channel utilization for matrix transpose and related permutations on boolean cubes. *Disc. Appl. Math.* **53** (1-3), 1994, pp. 251–274.
- [125] Johnsson, S. L., and Ho, C. T. Algorithms for matrix transposition for boolean  $n$ -cube configured ensemble architectures. *SIAM J. Matrix Appl.* **9** (3), 1988, pp. 419–454.

- [126] Jwo, J. S., Lakshminarayanan, S., and Dhall, S. K. A new class of interconnection networks based on the alternating group. *Networks*. **23**, 1993, pp. 315–326.
- [127] Kahale, N. Eigenvalues and expansion of regular graphs. *J. ACM*. **42** (5), 1995, pp. 1091–1106.
- [128] Kahale, N., and Leighton, T. Greedy dynamic routing on arrays. *Proc. 6th ACM-SIAM Symp. Disc. Alg.*, 1995, pp. 558–566.
- [129] Kahale, N., Leighton, F. T., Ma, Y., Plaxton, C. G., Suel, T., and Szemerédi, E. Lower bounds for sorting networks. *Proc. 27th ACM Symp. Theory Comput.* 1995, pp. 548–558.
- [130] Kaklamanis, C., Karlin, A. R., Leighton, F. T., Milenkovic, V., Raghavan, P., Rao, S., Thomborson, C., and Tsantilas, A. Asymptotically tight bounds for computing with faulty arrays of processors. *Proc. 31st IEEE Symp. Foundations Comput. Sci.* 1990, pp. 285–296.
- [131] Kaklamanis, C., and Krizanc, D. Optimal sorting on mesh-connected processor arrays. *Proc. 4th ACM Symp. Parallel Alg. Arch.* 1992, pp. 50–59.
- [132] Kaklamanis, C., Krizanc, D., Narayanan, L., and Tsantilas, T. Randomized sorting and selection on mesh-connected processor arrays. *Proc. 3rd ACM Symp. Parallel Alg. Arch.* 1991, pp. 17–28.
- [133] Kaklamanis, C., Krizanc, D., and Rao, S. Simple path selection for optimal routing on processor arrays. *Proc. 4th ACM Symp. Parallel Alg. Arch.* 1992, pp. 23–30.
- [134] Kaklamanis, C., Krizanc, D., and Tsantilas, A. Tight bounds for oblivious routing on the hypercube. *Math. Syst. Theory*. **24** (4), 1991, pp. 223–232.
- [135] Karloff, H., and Raghavan, P. Randomized algorithms and pseudorandom numbers. *J. ACM*. **40** (3), 1993, pp. 454–476.
- [136] Karp, R., Luby, M., and Meyer auf der Heide, F. Efficient PRAM simulation on a distributed memory machine. *Algorithmica*, **16** (4/5), 1996, pp. 517–542.
- [137] Kaufmann, M., Lauer, H., and Schröder, H. Fast deterministic hot-potato routing on meshes. *Proc. 5th ACM Int. Symp. Alg. Comput., Lect. Notes Comput. Sci.* **834**, 1994, pp. 333–341.
- [138] Kaufmann, M., Meyer, U., and Sibeyn, J. F. Towards practical permutation routing on meshes. *Proc. 6th IEEE Symp. Parallel Distrib. Comput.* 1994, pp. 664–671.
- [139] Kaufmann, M., Meyer, U., and Sibeyn, J. F. Matrix transposes on meshes: theory and practice. *Computers and Artificial Intelligence*, **16** (2), 1997, pp. 107–140.
- [140] Kaufmann, M., Raman, R., and Sibeyn, J. F. Randomized routing on meshes with buses. *Algorithmica*, **18**, 1997, pp. 417–444.
- [141] Kaufmann, M., Rajasekaran, S., and Sibeyn, J. F. Matching the bisection bound for routing and sorting on the mesh. *Proc. 4th ACM Symp. Parallel Alg. Arch.* 1992, pp. 31–40. Full versions in [260, 143].
- [142] Kaufmann, M., and Sibeyn, J. F. Deterministic routing on circular arrays. *Proc. 4th IEEE Symp. Parallel Distrib. Comput.* 1992, pp. 376–383.
- [143] Kaufmann, M., and Sibeyn, J. F. Randomized multipacket routing and sorting on meshes. *Algorithmica*, **17**, 1997, pp. 224–244.
- [144] Kaufmann, M., Sibeyn, J. F., and Suel, T. Derandomizing routing and sorting algorithms for meshes. *Proc. 5th ACM-SIAM Symp. Disc. Alg.* 1994, pp. 669–679.
- [145] Kaufmann, M., Sibeyn, J. F., and Suel, T. Beyond the bisection bound: fast ranking and counting on meshes. *Proc. 3rd European Symp. Alg., Lect. Notes Comput. Sci.* **979**, 1995, pp. 75–88.
- [146] Kaufmann, M., Schröder, H., and Sibeyn, J. F. Asymptotically optimal and practical routing on the reconfigurable mesh. *Parallel Proc. Letters*. **5**, 1995, pp. 81–95.
- [147] Kermani, P., and Kleinrock, L. Virtual cut through: a new computer communication switching technique. *Computer Networks*. **3** (4), 1979, pp. 267–286.
- [148] Kim, J. H., Liu, Z., and Chien, A. A. Compressionless routing: a framework for adaptive and fault-tolerant routing. *IEEE Trans. Parallel Distrib. Syst.* **C-8** (3), 1997, pp. 229–244.
- [149] Konstantinidou, S., and Snyder, L. The Chaos router. *IEEE Trans. Comput.* **C-43** (12), 1994, pp. 1386–1397.
- [150] Krizanc, D. Oblivious routing with limited buffer capacity. *J. Computer and System Sciences*, **43**, 1991, pp. 317–327.
- [151] Krizanc, D. A note on off-line permutation routing on a mesh-connected processor array. *Parallel Proc. Letters*. **1**, 1991, pp. 67–70.
- [152] Krizanc, D. Integer sorting on a mesh-connected array of processors. *Inf. Proc. Letters*. **47**, 1993, pp. 283–289.
- [153] Krizanc, D., and Narayanan, L. Zero-one sorting on the mesh. *Parallel Proc. Letters*. **5** (2), 1995, pp. 149–155.

- [154] Kumar, M., and Hirschberg, D. S. An efficient implementation of Batcher's odd-even merge algorithm and its application in parallel sorting schemes. *IEEE Trans. Comput.* **C-32** (3), 1983, pp. 254–264.
- [155] Kumar, V., Grama, A., Gupta, A., and Karypis, G. *Introduction to parallel computing: design and analysis of algorithms*. Benjamin Cummings Publishing Company, 1994.
- [156] Kunde, M. Lower bounds for sorting on mesh-connected architectures. *Acta Informatica.* **24**, 1987, pp. 121–130.
- [157] Kunde, M. Routing and sorting on mesh connected processor arrays. *Proc. VLSI Alg. Arch. Lect. Notes Comput. Sci.* **319**, 1988, pp. 423–433.
- [158] Kunde, M. Concentrated regular data streams on grids: sorting and routing near to the bisection bound. *Proc. 32nd IEEE Symp. Foundations Comput. Sci.* 1991, pp. 141–150.
- [159] Kunde, M. Balanced routing: towards the distance bound on grids. *Proc. 3rd ACM Symp. Parallel Alg. Arch.* 1991, pp. 260–271.
- [160] Kunde, M. Block gossiping on grids and tori: deterministic sorting and routing match the bisection bound. *Proc. European Symp. Alg., Lect. Notes Comput. Sci.* **726**, 1993, pp. 272–283.
- [161] Kunde, M., Niedermeier, R., Reinhardt, K., and Rossmannith, P. Optimal average case sorting on arrays. *Proc. 12th Symp. Theoret. Asp. Comput. Sci., Lect. Notes Comput. Sci.* , 1995, pp. 503–513.
- [162] Kunde, M., Niedermeier, R., and Rossmannith, P. Faster sorting and routing on grids with diagonals. *Proc. 11th Symp. Theoret. Asp. Comput. Sci., Lect. Notes Comput. Sci.* **775**, 1994, pp. 225–236.
- [163] Kunde, M., and Tensi, T.  $k$ - $k$  routing on multidimensional mesh-connected arrays. *J. Parallel Distrib. Comput.* **11**, 1991, pp. 146–155.
- [164] Kutylowski, M., Loryś, K., Oesterdiekhoff, B., and Wanka, R. Fast and feasible periodic sorting networks of constant depth. *Proc. 35th IEEE Symp. Foundations Comput. Sci.*, 1994, pp. 369–380.
- [165] Kutylowski, M., and Wanka, R. Periodic sorting on two-dimensional meshes. *Parallel Processing Letters*, **2**, 1992, pp. 213–220.
- [166] Kuzmaul, B. C. Fast deterministic routing on hypercubes using small buffers. *IEEE Trans. Comput.* **C-39** (11), 1990, pp. 1388–1393.
- [167] Lakshmivarahan, S., Dhall, S. K., and Miller L. Parallel sorting algorithms. *Adv. Comput.* 1984, pp. 295–354.
- [168] Lakshmivarahan, S., Jwo, J. S., and Dhall, S. K. Analysis of symmetry in interconnection networks based on Cayley graph of permutation graphs: a survey. *Parallel Comput.* 1993, **19**, pp. 361–407.
- [169] Lang, H. W., Schimmler, M., Schneck, H., and Schröder, H. Systolic sorting on a mesh connected network. *IEEE Trans. Comput.* **C-34** (7), 1985, pp. 652–658.
- [170] Latifi, S. On the fault-diameter of the star graph. *Inf. Proc. Letters.* **46**, 1992, pp. 143–150.
- [171] Lawrie, D. H. Access and alignment of data in an array processor. *IEEE Trans. Comput.* **C-24** (12), 1975, pp. 1145–1155.
- [172] Lee, T. C., and Hayes, J. P. A fault tolerant communication scheme for hypercube computers. *IEEE Trans. Comput.* **C-41** (10), 1992, pp. 1242–1255.
- [173] Lee, S., and Shin, K. G. Interleaved all-to-all reliable broadcast on meshes and hypercubes. *IEEE Trans. Parallel Distrib. Syst.* **C-5** (5), 1994, pp. 449–458.
- [174] Leighton, F. T. Tight bounds on the complexity of parallel sorting. *IEEE Trans. Comput.* **C-34** (4), 1985, pp. 344–354.
- [175] Leighton, F. T. Average case analysis of greedy routing algorithms. *Proc. 2nd ACM Symp. Parallel Alg. Arch.* 1990, pp. 2–8.
- [176] Leighton, F. T. *Introduction to parallel algorithms and architectures: arrays • trees • hypercubes*. Morgan-Kaufman, 1992.
- [177] Leighton, F. T. Methods for message routing in parallel machines. *Theoret. Comput. Sci., North-Holland.* **128** (1-2), 1994, pp. 31–62.
- [178] Leighton, F. T., Maggs, B. M., Ranade, A., and Rao, S. Randomized routing and sorting on fixed-connection networks. *J. Alg.* **17**, 1994, pp. 157–205.
- [179] Leighton, F. T., Maggs, B. M., and Rao, S. Universal packet routing algorithms. *Combinatorica.* **14** (2), 1994, pp. 167–186.
- [180] Leighton, F. T., Maggs, B. M., and Richa, A. W. Fast algorithms for finding  $O(\text{congestion} + \text{dilation})$  packet routing schedules. *Proc. 28th Hawaii Int. Conf. Syst. Sci.* 1995, **II**, pp. 555–563. Also, in *Combinatorica*. 1998, to appear.

- [181] Leighton, F. T., Maggs, B. M., and Sitaraman, R. On the fault tolerance of some popular bounded-degree networks. *Proc. 33rd IEEE Symp. Foundations Comput. Sci.* 1992, pp. 542–552. Also, in *SIAM J. Comput.* 1998, to appear.
- [182] Leighton, F. T., Makedon, F., and Tollis, I. G. A  $2n - 2$  step algorithm for routing in an  $n \times n$  array with constant size queues. *Algorithmica*. **14**, 1995, pp. 561–572.
- [183] Leighton, F. T., and Plaxton, C. G. Hypercubic sorting networks. *SIAM J. Computing*, **27** (1), 1998, pp. 1–47.
- [184] Leiserson, C. E. Fat-trees: universal networks for hardware-efficient supercomputing. *IEEE Trans. Comput.* **C-34** (10), 1985, pp. 892–901.
- [185] Leiserson, C. E., Abuhamdeh, Z. S., Douglas, D. C., et al. The network architecture of the connection machine CM-5. *J. Parallel Distrib. Comput.* **33** (2), 1996, pp. 145–158.
- [186] Lenfant, J. Permutations of data: A Beneš network control algorithm for frequently used permutations. *IEEE Trans. Comput.* **C-27** (7), 1978, pp. 637–647.
- [187] Leppänen V., and Penttonen, M. Work-optimal simulation of PRAM models on meshes. *Nordic J. Computing*, **2**, 1995, pp. 51–69.
- [188] Leung, J., and Shende, S. M. On multi-dimensional packet routing on meshes with buses. *J. Parallel Distrib. Comput.* **20**, 1994, pp. 187–197.
- [189] Lev, G., Pippenger, N., and Valiant, L. G. A fast parallel algorithm for routing in permutation networks. *IEEE Trans. Comput.* **C-30** (2), 1981, pp. 93–100.
- [190] Lin, W. Manipulating general vectors on synchronous binary  $n$ -cube. *IEEE Trans. Comput.* **C-42** (7), 1993, pp. 863–870.
- [191] Lin, X., McKinley, P. K., and Ni, L. M. Deadlock-free multicast wormhole routing in 2D mesh multicomputers. *IEEE Trans. Parallel Distrib. Syst.* **C-5** (8), 1994, pp. 793–804.
- [192] Lin, X., and Ni, L. M. Multicast communication in multicomputer networks. *IEEE Trans. Parallel Distrib. Syst.* **C-4** (10), 1993, pp. 1105–1117.
- [193] Lisinski, D., Leighton, F. T., and Maggs B. M. Empirical evaluation of randomly-wired multistage networks. *Proc. IEEE Int. Conf. Comput. Design.* 1990, pp. 380–385.
- [194] Lubiw, A. Counterexample to a conjecture of Szymanski on hypercube routing. *Inf. Proc. Letters*. **35**, 1990, pp. 57–61.
- [195] Lyuu, Y. D. Fast fault-tolerant parallel communication and on-line maintenance for hypercubes using information dispersal. *Math. Syst. Theory*. **24** (4), 1991, pp. 273–294.
- [196] Ma, Y., Sen, S., and Scherson, I. D. The distance bound for sorting on mesh-connected processor arrays is tight. *Proc. 27th IEEE Symp. Foundations Comput. Sci.* 1986, pp. 255–263.
- [197] Maggs, B. M. Randomly wired multistage networks. *Statistical Sci.* **8** (1), 1993, pp. 70–75.
- [198] Maggs, B. M., Matheson, L. R., and Tarjan, R. E. Models of parallel computation: a survey and synthesis. *Proc. 28th Hawaii Int. Conf. Syst. Sci.* 1995, **II**, pp. 61–70.
- [199] Maggs, B., and Sitaraman, R. Simple algorithms for routing on butterfly networks with bounded queues. *Proc. 24th ACM Symp. on Theory of Computing*, 1992, pp. 150–161, ACM.
- [200] Maggs, B. M., and Vöcking, B. Improved routing and sorting on multibutterflies. *Proc. 29th ACM Symp. Theory Comput.* 1997, pp. 517–530.
- [201] Makedon, F., and Symvonis, A. Flit-serial packet routing on meshes and tori. *Math. Syst. Theory*. **29** (4), 1996, pp. 311–330.
- [202] Makedon, F., and Symvonis, A. Optimal algorithms for multipacket routing problems on rings. *J. Parallel Distrib. Comput.* **22**, 1994, pp. 37–43.
- [203] Makedon, F., and Symvonis, A. Optimal algorithms for the many-to-one routing problem on two-dimensional meshes. *Microprocessors and Microsystems*. **17**, 1993, pp. 361–367.
- [204] Mansour Y., and Patt-Shamir, B. Many-to-one packet routing on grids. *Proc. 27th ACM Symp. Theory Comput.* 1995, pp. 258–267.
- [205] Mansour, Y., and Schulman, L. Sorting on a ring of processors. *J. Alg.* **11**, 1990, pp. 622–630, 1990.
- [206] Marberg, J. M., and Gafni, E. Sorting in constant number of row and column phases on a mesh. *Algorithmica*. **3**, 1988, pp. 561–572.
- [207] Mathies, T. R. Percolation theory and computing with faulty arrays of processors. *Proc. 3rd ACM-SIAM Symp. Disc. Alg.* 1992, pp. 100–103.
- [208] Mayr, E. W., and Plaxton, C. G. Pipelined parallel prefix computations, and sorting on a pipelined hypercube. *J. Parallel Distrib. Comput.* **17**, 1993, pp. 374–380.

- [209] Mehlhorn, K., and Vishkin, U. Randomized and deterministic simulations of PRAMs by parallel machines with restricted granularity of parallel memories. *Acta Informatica*, **9** (1), 1984, pp. 29–59.
- [210] Meliksetian, D. S., and Chien, C. Y. R. Optimal routing algorithms and other communication issues of the cube-connected cycle. *IEEE Trans. Parallel Distrib. Syst.* **C-4** (10), 1993, pp. 1172–1178.
- [211] Mellor-Crummey, J. M., and Scott, M. L. Algorithms for scalable synchronization on shared memory multiprocessors. *ACM Trans. Comput. Syst.* **C-9** (1), 1991, pp. 21–65.
- [212] Menn, A., and Somani, A. An efficient sorting algorithm for the star graph. *Proc. Int. Conf. Parallel Proc.* 1990, **III**, pp. 1–8.
- [213] Meyer auf der Heide, F. Efficient simulations among several models of parallel computers. *SIAM J. Comput.* **15** (1), 1986, pp. 106–119.
- [214] Meyer auf der Heide, F., Oesterdiekhoff, B., and Wanka, R. Strongly adaptive token distribution. *Algorithmica*. **15** (5), 1996, pp. 413–427.
- [215] Meyer auf der Heide, F. and Scheideler, C. Space-efficient routing in vertex-symmetric networks. *Proc. 7th ACM Symp. Parallel Alg. Arch.* 1995, pp. 137–146.
- [216] Meyer auf der Heide, F. and Scheideler, C. Routing with bounded buffers and hot-potato routing in vertex-symmetric networks. *Proc. 3rd European Symp. Alg., Lect. Notes Comput. Sci.* **979**, 1995, pp. 341–354.
- [217] Meyer auf der Heide, F., Scheideler, C., Stemann, V. Exploiting storage redundancy to speed up randomized shared memory simulations. *Theoret. Comput. Sci., North-Holland*. **162** (2), 1996, pp. 245–281.
- [218] Mitzenmacher, M. Bounds on the greedy routing algorithm for array networks. *Proc. 6th ACM Symp. Parallel Alg. Arch.* 1994, pp. 346–353.
- [219] Mou, Z. G., Constantinescu, C., and Hickey, T. J. Optimal mappings of divide and conquer algorithms to mesh connected parallel architectures. *Proc. Int. Comput. Symp.* Taichung, Taiwan, 1992, pp. 273–274.
- [220] Mou, Z. G., and Goodman, M. A comparison of communication costs for three parallel program paradigms on hypercube and mesh architectures. *Proc. 5th SIAM Conf. Parallel Proc. Sci. Comput.* 1992, pp. 491–500.
- [221] Mou, Z. G., and Wang, X. A divide and conquer method of solving tridiagonal systems on hypercube massively parallel computers. *Proc. 3rd IEEE Symp. Parallel Distrib. Comput.* 1991, pp. 810–817.
- [222] Nakano, K. A bibliography of published papers on dynamically reconfigurable architectures. *Parallel Proc. Letters*. **5**, 1995, pp. 111–124.
- [223] Nassimi, D. Parallel algorithms for the classes of  $2^b$  descend and ascend computations on a SIMD hypercube. *IEEE Trans. Parallel Distrib. Syst.* **C-4** (12), 1993, pp. 1372–1381.
- [224] Nassimi, D. Nearly logarithmic-time parallel algorithm for the class of  $2^b$  ascend computations on a SIMD hypercube. *J. Parallel Distrib. Comput.* **20**, 1994, pp. 289–302.
- [225] Nassimi, D., and Sahni, S. Bitonic sorting on a mesh-connected parallel computer. *IEEE Trans. Comput.* **C-28** (1), 1979, pp. 2–7.
- [226] Nassimi, D., and Sahni, S. An optimal routing algorithm for mesh connected parallel computers. *J. ACM*. **27** (1), 1980, pp. 6–29.
- [227] Nassimi, D., and Sahni, S. Parallel algorithms to set-up the Beneš permutation network. *IEEE Trans. Comput.* **C-31** (2), 1982, pp. 148–154.
- [228] Nassimi, D., and Sahni, S. Optimal BPC permutations on a cube connected SIMD computer. *IEEE Trans. Comput.* **C-31** (4), 1982, pp. 338–341.
- [229] Nassimi, D., and Sahni, S. Parallel permutation and sorting algorithms and a new generalized connection network. *J. ACM*. **29** (3), 1982, pp. 642–667.
- [230] Newman, I., and Schuster, A. Hot-potato algorithms for permutation routing. *IEEE Trans. Parallel Distrib. Syst.* **C-6** (11), 1995, pp. 1168–1176.
- [231] Newman, I., and Schuster, A. Hot-potato worm routing via store-and-forward packet routing. *J. Parallel Distrib. Comput.* **30**, 1995, pp. 76–84.
- [232] Ngai, J. Y., and Seitz, C. L. A framework for adaptive routing in multicomputer networks. *Proc. 1st ACM Symp. Parallel Alg. Arch.* 1989, pp. 2–10.
- [233] Nigam, M., and Sahni, S. Sorting  $n$  numbers on  $n \times n$  reconfigurable meshes with buses. *J. Parallel Distrib. Comput.* **23**, 1994, pp. 37–48.
- [234] Oh, A. D., and Choi, H. A. Generalized measure of fault tolerance in  $n$ -cube networks. *IEEE Trans. Parallel Distrib. Syst.* **C-4** (6), 1993, pp. 702–703.

- [235] Ouyang, P., and Palem K. V. Very efficient cyclic shifts on hypercubes. *J. Parallel Distrib. Comput.* **39** (1), 1996, pp. 79–86.
- [236] Palis, M. A., Rajasekaran, S., and Wei, D. S. L. Packet routing and PRAM emulations on star graphs and leveled networks. *J. Parallel Distrib. Comput.* **20**, 1994, pp. 145–157.
- [237] Panaite, P. Hypercube permutations routable under dimension orderings. *Inf. Proc. Letters.* **59** (4), 1996, pp. 185–189.
- [238] Parberry, I. A computer assisted optimal depth lower bound for nine-input sorting networks. *Math. Syst. Theory.* **24**, 1991, pp. 101–116. *Mathematical Systems Theory, Vol. 24*, pp. 101–116, 1991.
- [239] Paterson, M.S. Improved sorting networks with  $\mathcal{O}(\log N)$  depth. *Algorithmica.* **5**, 1990, pp. 65–92.
- [240] Peleg, D. A note on optimal time broadcast in faulty hypercubes. *J. Parallel Distrib. Comput.* **26** (1), 1995, pp. 132–135.
- [241] Peleg, D., and Simmons, B. On fault tolerant routings in general networks. *Information and Computation.* **74** (1), 1987, pp. 33–44.
- [242] Peleg, D., and Upfal, E. The Token distribution problem. *SIAM J. Computing.* **18** (2), 1989, pp. 229–243.
- [243] Peleg, D., and Upfal, E. A time randomness tradeoff for oblivious routing. *SIAM J. Comput.* **19** (2), 1990, pp. 256–266.
- [244] Pietracaprina, A., and Pucci, G. Tight bounds on deterministic PRAM emulations with constant redundancy. *Proc. 2nd European Symp. Alg., Lect. Notes Comput. Sci.* **855**, 1994, pp. 391–400.
- [245] Pietracaprina, A., and Pucci, G. Improved deterministic PRAM simulation on the mesh. *Proc. 22nd Int. Colloquium on Automata Lang. and Programming, Lect. Notes Comput. Sci.* **944**, 1995, pp. 372–383.
- [246] Pietracaprina, A., Pucci, G., and Sibeyn, J. F. Constructive deterministic PRAM simulation on a mesh-connected computer. *Proc. 6th ACM Symp. Parallel Alg. Arch.* 1994, pp. 248–256.
- [247] Pinkston, T. M., and Warnakulasuriya, S. On deadlocks in interconnection networks. *Proc. 24th IEEE Int. Symp. Comput. Arch.* 1997, pp. 38–49.
- [248] Pippenger, N. Communication networks. in *J. van Leeuwen ed., Handbook Theoret. Comput. Sci.* North-Holland, **A**, 1990, pp. 807–833.
- [249] Pippenger, N. Parallel communication with limited buffers. *Proc. 25th IEEE Symp. Foundations Comput. Sci.* 1984, pp. 127–136.
- [250] Plaxton, C. G. Load balancing, selection, and sorting on the hypercube. *Proc. ACM Symp. Parallel Alg. Arch.* 1989, pp. 64–73.
- [251] Plaxton, C. G. A hypercubic sorting network with nearly logarithmic depth. *Proc. 24th ACM Symp. Theory Comput.* 1992, pp. 405–416.
- [252] Plaxton, C. G., and Suel, T. A lower bound for sorting networks based on the shuffle permutation. *Math. Syst. Theory.* **27**, 1994, pp. 491–508.
- [253] Plaxton, C. G., and Suel, T. A super-logarithmic lower bound for hypercubic sorting networks. *Proc. 21st EATCS Int. Coll. Aut. Lang. Progr.*, 1994, pp. 618–629.
- [254] Plaxton, C. G., and Suel, T. Lower bounds for Shellsort. *J. Alg.*, **23** (2), 1997, pp. 221–240.
- [255] Preparata, F. P., and Vuillemin, J. The cube-connected cycle: A versatile network for parallel computation. *Comm. ACM.* **24** (5), 1981, pp. 300–309.
- [256] Rabani, Y., and Tardos, E. Distributed packet switching in arbitrary networks. *Proc. 28th ACM Symp. Theory Comput.* 1996, pp. 366–375.
- [257] Rabin, M. Efficient dispersal of information for security, load balancing, and fault tolerance. *J. ACM.* **36** (2), 1989, pp. 335–348.
- [258] Raghavan, P. Robust algorithms for packet routing in a mesh. *Math. Syst. Theory.* **28** (1), 1995, pp. 1–11.
- [259] Raghunath, M. T., and Ranade, A. Customizing interconnection networks to suit packaging hierarchies. *Tech. Rep. UCB/CSD-92-725, EECS Comput. Sci. Division, University of California, Berkeley.* 1992.
- [260] Rajasekaran, S.  $k$ - $k$  Routing,  $k$ - $k$  Sorting, and Cut-Through Routing on the Mesh. *J. Alg.*, **19** (3), 1995, pp. 361–382.
- [261] Rajasekaran, S. Mesh connected computers with fixed and reconfigurable buses: packet routing and sorting. *IEEE Trans. Comput.* **C-45** (5), 1996, pp. 529–539.
- [262] Rajasekaran, S., and Overholt, R. Constant queue routing on a mesh. *J. Parallel Distrib. Comput.* **15**, 1992, pp. 160–166.

- [263] Rajasekaran, S., and Raghavachari, M. Optimal randomized algorithms for multipacket and cut through routing on the mesh. *J. Parallel Distrib. Comput.* **26**, 1995, pp. 257–260.
- [264] Rajasekaran, S., and Tsantilas, T. Optimal routing algorithms for the mesh-connected processor arrays. *Algorithmica*. **8**, 1992, pp. 21–38.
- [265] Rajasekaran, S., and Wei, D. S. L. Selection, routing, and sorting on the star graph. *Proc. 7th IEEE Int. Parallel Proc. Symp.* 1993, pp. 661–665.
- [266] Ramanathan, P., and Shin, K. G. Reliable broadcast in hypercube multicomputers. *IEEE Trans. Comput.* **C-37** (12), 1988, pp. 1654–1657.
- [267] Ranade, A. How to emulate shared memory. *J. Comput. Syst. Sci.* **42** (3), 1991, pp. 307–326.
- [268] Ranade, A., Schleimer, S., and Wilkerson, D.S. Nearly Tight Bounds for Wormhole Routing. *Proc. 35th Symp. Foundations Comput. Sci.*, pp. 346–255, IEEE, 1994.
- [269] Ranka, R., and Sahni, S. *Hypercube algorithms with application to image processing and pattern recognition*. Springer-Verlag, 1990.
- [270] Reif, J. H., and Valiant, L. G. A logarithmic time sort for linear size networks. *J. ACM.* **34** (1), 1987, pp. 60–76.
- [271] Richards, D. Parallel sorting – a bibliography. *SIGACT News.* **18** (1), 1986, pp. 28–48.
- [272] Roberts, A., and Symvonis, A. A general method for deflection worm routing on meshes based on packet routing algorithms. *Tech. Rep. 490*, Dept. Comp. Sci., University of Sydney, Sydney, 1994.
- [273] Saad, Y., and Schultz, M. H. Data communication in hypercubes. *J. Parallel Distrib. Comput.* **6**, 1989, pp. 115–135.
- [274] Sado, K., and Igarashi, Y. Some parallel sorts on a mesh-connected processor array and their time efficiency. *J. Parallel Distrib. Comput.* **3**, 1986, pp. 398–410.
- [275] Samatham, M. R., and Pradhan, D. K. The de Bruijn multiprocessor network: A versatile parallel processing and sorting network for VLSI. *IEEE Trans. Comput.* **C-38** (4), 1989, pp. 567–581.
- [276] Savari, S. A. Average case analysis of five two-dimensional bubblesorting algorithms. *Proc. 5th ACM Symp. Parallel Alg. Arch.* 1993, pp. 336–345.
- [277] Scherson, I. D., and Sen, S. Parallel sorting in two-dimensional VLSI models of computation. *IEEE Trans. Comput.* **C-38** (2), 1989, pp. 238–249.
- [278] Schnorr, C., and Shamir, A. An optimal sorting algorithm for mesh connected computers. *Proc. 18th ACM Symp. Theory Comput.* 1986, pp. 255–263.
- [279] Schwabe, E. J. Constant-slowdown simulations of normal hypercube algorithms on the butterfly network. *Inf. Proc. Letters.* **45**, 1993, pp. 295–301.
- [280] Schwiegelsohn, U. A short periodic two-dimensional sorting technique for VLSI networks. *Proc. Conf. Systolic Arrays*, 1988, pp. 257–264.
- [281] Sengupta, A., Viswanathan, S. On fault-tolerant fixed routing in hypercube. *Inf. Proc. Letters.* **51**, 1994, pp. 93–99.
- [282] Shaout, A., and Smyth, D. Data flow models for a hypercube multiprocessor network. *Proc. Int. Conf. Parallel Proc.* 1991, **I**, pp. 351–354.
- [283] Shen, X., Hu, Q., Liang, W. Realization of an arbitrary permutation on a hypercube. *Inf. Proc. Letters.* **51**, 1994, pp. 237–243.
- [284] Shen, X., Hu, Q., Cong, B., et al. The 4-star graph is not a subgraph of any hypercube. *Inf. Proc. Letters.* **45** (4), 1993, pp. 199–203.
- [285] Sibeyn, J. F. Matrix techniques for faster routing of affine permutations on a mesh interconnection network. *Tech. Rep. RUU-CS-90-17*, Dept. Comp. Sci., Utrecht University, the Netherlands, 1990.
- [286] Sibeyn, J. F. Routing permutations on mesh interconnection networks. *Proc. 2nd IEEE Symp. Parallel Distrib. Comput.* 1990, pp. 94–97.
- [287] Sibeyn, J. F. *Algorithms for routing on meshes*, Ph.D. thesis, Utrecht University, the Netherlands, 1992.
- [288] Sibeyn, J. F. Deterministic routing and sorting on rings. *Proc. 8th IEEE Int. Parallel Proc. Symp.* 1994, pp. 406–410.
- [289] Sibeyn, J. F. Desnaking of mesh sorting algorithms. *Proc. 2nd Euro Symp. Alg., Lect. Notes Comput. Sci.* **855**, 1994, pp. 377–390. Also, in Row-major sorting on meshes. *SIAM J. Comput.* 1998, to appear.
- [290] Sibeyn, J. F. Routing on triangles, tori and honeycombs. *Int. J. Foundations of Comput. Sci.* **8**(3), 1997, 269–287.

- [291] Sibeyn, J. F. Sample sort on meshes. *Proc. 3rd Euro-Par Conf., Lect. Notes Comput. Sci.* **1300**, 1997, pp. 389–398.
- [292] Sibeyn, J. F. Routing with finite speeds of memory and network. *Proc. 22nd Symp. Math. Foundations Comput. Sci., Lect. Notes Comput. Sci.* **1295**, 1997, pp. 488–497.
- [293] Sibeyn, J. F., Chlebus, B. S., and Kaufmann, M. Deterministic permutation routing on meshes. *J. Alg.*, **22**, 1997, pp. 111–141.
- [294] Sibeyn, J. F., and Kaufmann, M. Deterministic  $1-k$  routing on meshes. *Proc. 11th Symp. Theoret. Asp. Comput. Sci., Lect. Notes Comput. Sci.* **775**, 1994, pp. 237–248.
- [295] Šoch, M., and Tvrdík, P., Optimal gossip in store-and-forward noncombining 2-D Tori. *Proc. 3rd Int. Euro-Par Conf., Lect. Notes Comput. Sci.* **1300**, 1997, pp. 234–241.
- [296] Stamoulis, G. D., and Tsitsiklis, J. N. Greedy routing in hypercubes and butterflies. *IEEE Trans. Comm.* **42** (11), 1994, pp. 3051–3061.
- [297] Stamoulis, G. D., and Tsitsiklis, J. N. Efficient routing schemes for multiple broadcasts in hypercubes. *IEEE Trans. Parallel Distrib. Syst.* **C-4** (7), 1993, pp. 725–739.
- [298] Stamoulis, G. D., and Tsitsiklis, J. N. An efficient algorithm for multiple simultaneous broadcasts in the hypercube. *Inf. Proc. Letters.* **46** (5), 1993, pp. 219–224.
- [299] Steinberg, D. Notes on invariant properties of the Shuffle/Exchange and a simplified cost-effective version of the Omega network. *IEEE Trans. Comput.* **C-32** (5), 1983, pp. 444–450.
- [300] Stout, Q. F. Intensive hypercube communication. *J. Parallel Distrib. Comput.* **10**, 1990, pp. 167–181.
- [301] Suel, T. Routing and sorting on meshes with row and column buses. *Proc. 8th IEEE Int. Parallel Proc. Symp.* 1994, pp. 411–417.
- [302] Suel, T. Improved bounds for routing and sorting on multi-dimensional meshes. *Proc. 6th ACM Symp. Parallel Alg. Arch.* 1994, pp. 26–35.
- [303] Szymanski, T. On the permutation capability of a circuit-switched hypercube. *Proc. Int. Conf. Parallel Proc.* 1989, **I**, pp. 103–110.
- [304] Szymanski, T.  $\mathcal{O}(\log N/\log\log N)$  randomized routing in degree- $\log N$  hyper-meshes. *Proc. Int. Conf. Parallel Proc.* 1991, **I**, pp. 443–450.
- [305] Szymanski, T. "Hypermeshes": optical interconnection networks for parallel computing. *J. Parallel Distrib. Comput.* **26**, 1995, pp. 1–23.
- [306] Tamaki, H. Efficient self-embedding of butterfly networks with random faults. *Proc. 33rd IEEE Symp. Foundations Comput. Sci.* 1992, pp. 533–541.
- [307] Tien, S. B., and Raghavendra, C. S. Algorithms and bounds for shortest paths and diameter in faulty hypercubes. *IEEE Trans. Parallel Distrib. Syst.* **C-4** (6), 1993, pp. 713–718.
- [308] Thompson, C. D. Generalized connection networks for parallel processor interconnection. *IEEE Trans. Comput.* **C-27** (2), 1978, pp. 1119–1125.
- [309] Thompson, C. D., and Kung, H. T. Sorting on a mesh-connected parallel computer. *Comm. ACM.* **20** (4), 1977, pp. 263–270.
- [310] Thompson, P. W., and Lewis, J. D. The STC104 packet routing chip. *VLSI Design.* **2** (4), 1995, pp. 305–314.
- [311] Upfal, E. Efficient schemes for parallel communication. *J. ACM.* **31** (2), 1984, pp. 507–517.
- [312] Upfal, E. An  $\mathcal{O}(\log N)$  deterministic packet-routing scheme. *J. ACM.* **39** (1), 1992, pp. 55–70.
- [313] Upfal, E., Felperin, S. and Snir, M. Randomized routing with shorter paths. *Proc. 5th ACM Symp. Parallel Alg. Arch.* 1993, pp. 283–292.
- [314] Valiant, L. G. A scheme for fast parallel communication. *SIAM J. Comput.* **11** (2), 1982, pp. 350–361.
- [315] Valiant, L. G. Optimality of a two phase strategy for routing in interconnection networks. *IEEE Trans. Comput.* **C-32** (9), 1983, pp. 861–863.
- [316] Valiant, L. G. General purpose parallel architectures. in *J. van Leeuwen ed., Handbook Theoret. Comput. Sci.* North-Holland, **A**, 1990, pp. 943–971.
- [317] Valiant, L. G., and Brebner, G. J. Universal schemes for parallel communication. *Proc. 13th ACM Symp. Theory Comput.* 1981, pp. 263–277.
- [318] Varman, P. J., and Doshi, K. Sorting with linear speedup on a pipelined hypercube. *IEEE Trans. Comput.* **C-41** (1), 1992, pp. 97–105.
- [319] Varvarigos, E. A., and Bertsekas, D. P. Communication algorithms for isotropic tasks in hypercubes and wraparound meshes. *Parallel Comput.* 1992, **18**, pp. 1233–1257.

- [320] Varvarigos, E. A., and Bertsekas, D. P. Transposition of banded matrices: a nearly isotropic task. *Parallel Comput.* 1995, **21**, pp. 243–264.
- [321] Varvarigos, E. A., and Bertsekas, D. P. Dynamic broadcasting in parallel computing. *IEEE Trans. Parallel Distrib. Syst.* **C-6** (2), 1995, pp. 120–131.
- [322] Varvarigos, E. A., and Bertsekas, D. P. Multinode broadcast in hypercubes and rings with randomly distributed length of packets. *IEEE Trans. Parallel Distrib. Syst.* **C-4** (2), 1993, pp. 144–154.
- [323] Varvarigos, E. A., and Banerjee, A. Routing schemes for multiple random broadcasts in arbitrary network topologies. *IEEE Trans. Parallel Distrib. Syst.* **C-7** (8), 1996, pp. 886–895.
- [324] Waksman, A. A permutation network. *J. ACM.* **15** (1), 1968, pp. 159–163.
- [325] Walrand, J., and Varaiya, P. *High-performance communication networks*. Morgan-Kaufman, 1996.
- [326] Wang B., Chen, G., and Lin, F. Constant time sorting on a processor array with a reconfigurable bus system. *Inf. Proc. Letters.* **34**, 1990, pp. 187–192.
- [327] Wanka, R. Fast general sorting on meshes of arbitrary dimension without routing. *Tech. Rep. 87, Reihe Informatik*, Fachbereich Mathematik-Informatik, Universität-GH Paderborn, Paderborn, Germany, 1991.
- [328] Wei, D. S. Quicksort and permutation routing on the hypercube and de Bruijn networks. *Proc. Int. Symp. Parallel Arch. Alg. and Networks.* 1994, pp. 238–245.
- [329] Wu, J. Safety levels - An efficient mechanism for achieving reliable broadcasting in hypercubes. *IEEE Trans. Parallel Distrib. Syst.* **C-44** (5), 1995, pp. 702–706.
- [330] Youssef, A. Cartesian product networks *Proc. Int. Conf. Parallel Proc.* 1991, **I**, pp. 684–685.

**Miltos D. Grammatikakis**, born in 1962 in Heraklion, Greece, received MSc (1985) and PhD (1991) in computer science from the University of Oklahoma, USA. He was a post-doctoral fellow in the parallel processing laboratory of ENS-Lyon (1991-1992), researcher in the Institute of Computer Science, F.O.R.T.H., and lecturer at TEI Technological College, both in Heraklion, Greece (1992-1995). In 1995 he joined the Institut für Informatik, Universität Hildesheim, Germany, as scientific assistant (C1) in theoretical computer science. His research interests include parallel communication systems and discrete-event simulation.

**D. Frank Hsu**, received his PhD degree from the University of Michigan, Ann Arbor, in 1979. Since then he has been a Professor in the Department of Computer and Information Science at Fordham University, Bronx, USA. His research interests include interconnection networks, massively parallel computation, combinatorial methods and algorithms, and graph problems arising from the design of parallel architectures, VLSI circuits, and communication networks. Dr. Hsu is a member of ACM, SIAM, and IEEE Computer Society and a Foundation Fellow of the Institute of Combinatorics and its Applications.

**Miro Kraetzl**, is a Senior Research Scientist in the Communications Division of Defence Science and Technology Organisation in Salisbury, South Australia. He received BSc (1975) and MSc (1978) degrees in pure mathematics from the University of Zagreb (Croatia), and a PhD (1993) from Curtin University of Technology in Perth, Australia. His current research interests include analysis and control of information networks, reliability of communications networks, routing and fault tolerance in computer architectures, and automatic analysis of large data bases. He is a senior member of IEEE, and a member of AFCEA, and ACM.

**Jop F. Sibeyn**, born in 1962 in the Netherlands, got a masters degree in mathematics (1987), and a PhD in computer science from the University of Utrecht (1992). His PhD thesis was on "Routing Algorithms for Meshes". Since 1992 he has been a researcher in the Max-Planck-Institut für Informatik in Saarbrücken, Germany. His main interests lie in parallel algorithms, particularly for interconnection networks. Furthermore he has done research on load-balancing and combinatorial problems. Presently he gets more interested in practical aspects of parallel and external computing. He is a member of the IEEE Computer Society.