

Randomized Algorithms on the Mesh

Lata Narayanan

Department of Computer Science, Concordia University, Montreal, Quebec H3G
1M8, Canada, email: lata@cs.concordia.ca, FAX (514) 848-2830

Abstract. The mesh-connected array is an attractive architecture for parallel machines, and a number of existing parallel machines are based on a mesh topology. In this paper, we survey the results on three fundamental problems in parallel computation: routing, sorting, and selection, and demonstrate that randomized algorithms for these problems on the mesh are often the most natural and efficient algorithms available.

1 Introduction

The mesh-connected array has been the object of a great deal of theoretical study as well as the basis for a number of proposed and implemented parallel computers. The simplicity and regularity of its interconnection pattern make it ideal for VLSI implementation. Despite the fact that its diameter is large in comparison to other well-studied networks (e.g., hypercube, butterfly, shuffle-exchange networks), work by Dally [7] suggests that high diameter networks such as the mesh may provide a more efficient communication medium for VLSI-based parallel computers. A 2D mesh is used by Maspar machines [1], the Intel Touchstone Delta [5], Paragon [5], and the Simult 2010 [40], and a 3D mesh in the J-machine [32]. Furthermore, a large number of efficient algorithms have been designed to run on this architecture.

In this paper, we discuss the fundamental problems of routing, sorting and selection on mesh-connected parallel computers. Algorithms for these problems have been extensively studied for nearly twenty years now. Since all such algorithms need data movement on the mesh, distance can be seen easily to provide a lower bound on their performance. Another limiting factor could be link bandwidth. When an algorithm matches the distance lower bound for the problem, we say that it is *distance-optimal*. A series of algorithmic refinements has given rise to distance-optimal algorithms for routing and sorting, as well as efficient algorithms for selection. At the same time, finding efficient implementations of algorithms for these problems, especially sorting, continues to be an active area of research.

Randomization is typically used in two ways in these problems. For routing, randomization is used to reduce the likelihood of hot spots, by “scattering” data in a limited but random fashion. For comparison-based problems, randomization is additionally used to select a small set of “splitters”, which enable the computation of approximate ranks of elements, at a much lower cost than computing their exact ranks.

For almost all variants of these problems, optimal or best-available solutions are given by simple randomized algorithms. While in many cases, such as for permutation routing and sorting, deterministic algorithms with matching performance have been discovered subsequently, these do not diminish the value of the randomized algorithms. First, the deterministic algorithms are more complicated, and thus less efficient in practice. Second, they were often directly derived by “derandomizing” the randomized algorithms. In the case of selection, the best-known randomized algorithm has better performance than the best-known deterministic algorithm. However, it is likely that the techniques of derandomization mentioned above could be applied here as well.

On the practical side, there are numerous research results attesting to the fact that implementations of randomized algorithms are typically faster in practice. This is because they are usually simpler, and the constants hidden in lower order terms are often smaller than in deterministic algorithms.

Section 2 formally defines the computational model of the mesh and the problems we consider. The following three sections briefly survey the results, especially randomized algorithms, for the problems of routing, sorting and selection on the mesh respectively.

2 Preliminaries

The $n \times n$ mesh-connected array of processors (or two-dimensional mesh) contains $N = n^2$ processors arranged in a two-dimensional grid without wrap-around edges. More precisely, it corresponds to the graph, $G = (V, E)$, with $V = \{(x, y) \mid x, y \in \langle n \rangle\}$ and $E = \{((x, y), (x, y+1)) \mid x \in \langle n \rangle, y \in \langle n-1 \rangle\} \cup \{((x, y), (x+1, y)) \mid x \in \langle n-1 \rangle, y \in \langle n \rangle\}$, where $\langle n \rangle = \{1, \dots, n\}$. Nodes in the graph correspond to processors and edges to bidirectional communication links. The two dimensional torus-connected network (or torus) is a mesh with wrap-around edges. The d -dimensional mesh and torus are the logical extensions of their respective two-dimensional versions to higher dimensions.

Computations are to be performed using the following model of the mesh: During a single parallel communication step, each processor can send and receive a single *packet* along each of its incident edges, where a packet consists of at most a single data element along with $O(\log N)$ bits of header information used for routing and counting purposes. Between communication steps, processors can store packets in their local queues, which are of bounded size. Furthermore, they can perform a constant number of simple operations (e.g., copying, addition, comparison) on the elements and the header information of packets. The measures of performance commonly used are the number of parallel communication steps, and the maximum queue size at any node. This model is often referred to as a *store-and-forward* model for routing; other models for routing packets are *wormhole* and *cut-through* models [10, 34]. For comparison-based problems, the model given in [39, 24] has been used to prove many lower bounds. However, this model is unnecessarily restrictive, and recent work on these problems is based on the model described here.

Given a set of messages or *packets*, one at every processor in the network, and a permutation P of the nodes in a $n \times n$ mesh, the *permutation routing* problem is: for each $(o, d) \in P$, to find a connected path for the packet initially at o (the packet's origin) to go to d (the packet's destination). In the $k - k$ routing problem, there are k permutation routing problems to be solved simultaneously. The input to both the sorting and selection problems is a set $X = \{x_1, \dots, x_N\}$, the elements of which may be linearly ordered. An *indexing scheme* is a bijection from $\langle N \rangle$ to $\langle n \rangle \times \langle n \rangle$. Commonly studied indexing schemes are the row-major indexing, snake-like row-major indexing, and block-wise snake-like row-major indexing. In the last ordering, the mesh is divided into square blocks of side n^δ for $\delta < 1$, and consecutively ordered blocks are physically adjacent on the mesh, e.g. snake-like; the processors inside each block B_i are indexed with indices in the interval $[(i-1)N^\delta, iN^\delta]$ in some arbitrary way. The *sorting* problem on the mesh is: Given a set X , stored with one element per processor, and an indexing scheme I , move the element of rank k in X to the processor labeled $I(k)$. In the $k - k$ sorting problem, each processor initially holds k elements, and the indexing scheme is such that in the final sorted order, each processor holds k consecutive elements. The *selection* problem is: Given a set X , stored with one element per processor, an integer $1 \leq k \leq N$, and a specified processor (i, j) , move the element of rank k in X to the processor labeled (i, j) . In what follows we will consider only the case where the specified processor is labeled $(\lceil \frac{n}{2} \rceil, \lceil \frac{n}{2} \rceil)$, referred to as the middle processor below. It is generally straightforward how to modify the algorithm for the case of another designated processor. In the *multi-packet* selection problem, each processor initially holds ℓ packets.

The algorithms we discuss are generally randomized (Las Vegas) and therefore have some probability of failure. In this paper, *with high probability* means with probability at least $1 - n^{-\beta}$ for some appropriate constant β . The tools that are commonly used to analyze such probabilities are *Chernoff bounds*, which are bounds on the tails of the binomial distribution [3].

Proposition 1. (*Bernstein-Chernoff bounds*) *Let $S_{N,p}$ be a random variable having binomial distribution with parameters N and p . Then, for any h such that $0 \leq h \leq 1.8Np$,*

$$P(S_{N,p} \geq Np + h) \leq \exp(-h^2/3Np).$$

For any $h \geq 0$,

$$P(S_{N,p} \leq Np - h) \leq \exp(-h^2/2Np).$$

Central to many randomized sorting and selection algorithms is the use of a random sample of the keys in order to determine approximately the rank of each key. More specifically, given N keys and a small constant δ , consider the problem of choosing $N^\delta - 1$ elements which split the keys into buckets of size between $N^{1-\delta}(1 - N^{-2\delta})$ and $N^{1-\delta}(1 + N^{-2\delta})$. Using ideas from [38, 37] we describe the following randomized algorithm to select these splitters:

SELECT-SPLITTERS(N)

Phase A Select a sample of keys by having each key toss a coin with bias $\alpha N^{5\delta-1} \ln N$, for some constant α .

Phase B Count the actual size S of the sample. Then select $N^\delta - 1$ splitters by picking every $\lceil S/N^\delta \rceil$ -th element from the sample to be a splitter.

Notice that the average size of the sample will be $\alpha N^{5\delta} \ln N$ and with high probability the size will not differ from its average value by more than $\alpha N^{5\delta/2} \ln N$. Furthermore, the actual rank of the j -th splitter will be $jN^{1-\delta}(1 \pm N^{-\delta})$. This can be shown with the following lemma:

Lemma 2. *For any sufficiently small constant δ , there exists $\alpha > 0$, such that, given N keys, the above algorithm produces $N^\delta - 1$ splitters which split the keys into buckets of size $N^{1-\delta}(1 \pm N^{-2\delta})$; the probability that the algorithm fails is smaller than $N^{-\alpha/5}$.*

3 Routing on the mesh

Routing is well-known to be a problem where randomization makes a provable difference in the time needed to solve the problem using an oblivious algorithm [42, 2, 18]. A good survey on strategies for routing can be found in [29], and the book by Leighton is an invaluable and more detailed resource [28]. In this section, we will focus on routing algorithms on the mesh.

An *oblivious* algorithm for routing is one in which the routing path for each packet is chosen without knowledge of the origins and destinations of other packets. Oblivious algorithms are interesting because they are clearly easy to implement: the originating processor of a packet need not consult with other processors in order to determine the path of the packet. Additionally, oblivious algorithms are also typically easier to analyze. The so-called greedy algorithm is a good example of an oblivious routing algorithm on the mesh. A packet starting at node (i, j) and destined for node (i', j') would travel first through the i^{th} row to the node (i, j') and then “turn” into the j'^{th} column to reach its destination by traveling only in the j'^{th} column. It is well-known that the greedy algorithm can perform very poorly on the mesh: on certain permutations, queues may build up to have $\Omega(n)$ packets.

The greedy algorithm does perform well on *average*. To analyze average-case performance of routing algorithms, we generally assume that each packet has a random destination. Thus there is a possibility that several packets are destined for the same processor. In this situation, it can be shown that with high probability, the greedy algorithm routes all packets in $2n + o(n)$ time steps [27, 28]. Furthermore, the total size of all queues at any node does not exceed 7 with high probability. Unfortunately, the worst-case behavior of the algorithm occurs in routing problems that do occur commonly in practice.

The good average-case behavior of the greedy algorithm allows us to make use of a common technique to design an algorithm with good worst-case performance. In particular, a worst-case problem can often be converted into average

case problems with a small amount of overhead. This leads to the concept of randomized routing, proposed by Valiant [42, 43]. In randomized routing, each packet is greedily sent to a *randomly chosen intermediate destination*, and then greedily sent to the actual destination. Thus each routing problem is converted into two average-case routing problems, which can each be solved efficiently using the simple greedy algorithm. Using this technique, we immediately obtain a routing algorithm that runs in $4n + o(n)$ time steps and uses constant size queues, with high probability. Note that in this case, the high probability is owing to the random choice of intermediate destination made by the algorithm, and has nothing to do with the input received by the algorithm. Thus this algorithm works well on *any* problem with high probability.

By reducing the amount of randomization, for example, by moving to a random intermediate destination within a small distance in the same column as the origin, as in [36], one can obtain a randomized algorithm that runs in $2n + O(\log n)$ steps using constant size queues with high probability. In [16], a simple randomized algorithm that routes in $2n + O(\log n)$ steps using only constant size queues is described. In [20], derandomizing techniques are applied to convert this to a deterministic algorithm with the same performance.

Randomized algorithms are also the best possible in several variants of the routing model such as cut-through routing [34], $k-k$ routing [34], and deflection routing [9, 17, 30, 14].

4 Sorting on the mesh

Sorting is one of the most important and well-studied problems in computer science. The problem of sorting on a mesh has a long history starting with Thompson and Kung [41], who gave an algorithm which sorts $N = n^2$ inputs into snake-like row major order in $6n + o(n)$ parallel communication steps on a $n \times n$ synchronous SIMD mesh; their algorithm may be adapted to run in $3n + o(n)$ time on a MIMD mesh. Schnorr and Shamir [39] gave a second algorithm for sorting into snake-like row major order on a mesh running in $3n + o(n)$ time. This algorithm has the nice property that it is an *oblivious comparison-exchange* algorithm, *i.e.* it consists solely of prespecified operations of the form, compare the contents of cell i with cell j , and place the smaller in cell i and the larger in cell j . A number of other sorting algorithms on the mesh are described in [28].

There is an obvious lower bound of $2n - 2$ steps for sorting due to distance constraints. For example, if the largest element is at the left top corner, then the number of steps necessary just to move the element to its final destination is $2n - 2$. It is interesting to see if the sorting can be performed distance-optimally, *i.e.* in $2n + o(n)$ steps. Kaklamani *et al.* [15] gave a $2.5n + o(n)$ randomized algorithm for sorting on the mesh. This algorithm is not an oblivious comparison-exchange algorithm; in particular, this algorithm duplicates packets, and every processor is assumed to have a constant (> 1) size buffer. This was subsequently matched by a deterministic algorithm given by Kunde [25]. The randomized

algorithm for sorting was then improved in [13] to a *distance-optimal* $2n + o(n)$ algorithm on the $n \times n$ mesh.

We will briefly describe the key ideas behind these algorithms. First, a random sample of the elements, and then splitters are extracted from this as described in Section 2. These splitters are broadcast in each quadrant, in order to determine approximately the rank of each element. In the meantime, a copy of each element is routed to a random location in each quadrant close to the middle of the mesh; this is so that the elements do not have as far to go after computing their approximate ranks. Using these approximate ranks, the copy of each element that is in the *same* quadrant as its destination is then routed to an *approximate destination* which is provably very likely to be quite close to its final destination in the sorted mesh. The remaining copies do not survive, so for each element only one of the four copies will survive. Meanwhile, in each quadrant the exact ranks of the splitters are computed and broadcast. Using these and local prefix operations, the exact rank of each of the elements can be computed, and every element can be finally routed to its correct destinations.

In [20], some derandomizing techniques were presented, and the authors used these techniques to derive from the above randomized algorithm a distance-optimal deterministic algorithm for sorting on the mesh. The derandomizing techniques essentially consist of a sort-and-unshuffle operation in place of randomized routing, to eliminate hot-spots, and using deterministically rather than randomly chosen splitters. From a theoretical standpoint, then, every known randomized algorithm for sorting on the mesh can be matched by a deterministic algorithm with the same performance. What then is the utility of randomization for developing sorting algorithms on the mesh? It is clear that the development of randomized algorithms preceded and guided the development of deterministic algorithms. Randomized algorithms are simpler to conceptualize and describe. This has the additional effect that the constants in the lower order terms are smaller in the randomized algorithms, thus making them more efficient in practice. The same is the case for the problem of $k - k$ sorting: see [19, 34, 25, 26] for results. In [21], similar experiences are reported for algorithms for sorting-like problems such as ranking and excess counting. Finally, randomized algorithms for sorting are also the best-known for meshes with reconfigurable or fixed buses [33].

Finding a sorting algorithm that performs well in practice on specific parallel machines continues to be an active area of research. In several papers, [12, 8, 6] it can be seen that randomized algorithms such a version of Samplesort are the best algorithms in practice, when there are large numbers of elements at nodes.

5 Selection on the mesh

Any selection algorithm requires at least $n - 1$ steps, since this is the distance from the corners to the center of the mesh. This is the only known lower bound for selection on the standard model of the mesh (Kunde's lower bound of $2n - o(n)$ steps [24] applies only to a very restricted model of the mesh). Clearly any

sorting algorithm would suffice to perform selection, thus there are algorithms to place the median at the center processor in $2n + o(n)$ steps. In [15, 31], a new selection algorithm that took $1.22n$ steps was described. This algorithm was based on the well-known randomized linear-time sequential algorithm for selection given by Floyd and Rivest [11]. The main open question is thus whether there exists a distance-optimal, or $n + o(n)$ -time algorithm for selection on the mesh. This question is especially interesting in light of the fact that distance-optimal algorithms for the related problems of sorting and routing have been discovered, as discussed in the previous sections.

In [4], Condon and Narayanan show that the techniques used in the best previous selection algorithms cannot yield a distance-optimal algorithm. To explain this, they define a notion of adaptiveness for comparison-based algorithms on the mesh, and show that “weakly-adaptive” algorithms cannot be distance-optimal. Intuitively, median-finding algorithms adapt over time, based on new information learned from comparisons. For example, packets that appear likely to be the median, based on comparisons with a sample of elements, may be routed towards the center early on. However, in all known median-finding algorithms, such adaptive routing decisions are made only once or twice. This is in part because gathering large samples of elements is expensive.

To precisely limit the degree of adaptiveness of an algorithm, they limit the set of comparison results on which a processor’s computation can depend. In their model, at fixed steps called *knowledge steps*, each processor learns the results of comparisons between every pair of elements that could possibly have reached the processor at that step. Between knowledge steps, processors may not perform new comparisons. However, at a step which is not a knowledge step, a processor may still learn new comparison results in the following way: it learns the comparison results of its neighbors at the previous step. An algorithm is *weakly-adaptive* if it has $O(1)$ knowledge steps; otherwise it is called *highly-adaptive*. A *maximally adaptive* algorithm is one where every step is a knowledge step. Using this notion, they explain why the best previous algorithms for selection on the mesh were weakly adaptive. More importantly, they show that there can be no distance-optimal weakly adaptive algorithm for selection on the mesh. To prove this, they exploit the non-adaptiveness of the algorithm to show that there must be many possible candidates for the median at some time t . Since each processor can only store a constant number of packets, some candidate must be of distance much greater than $n - t$ from the center. Furthermore, on a possibly different input, this candidate is the true median, and is also sufficiently far from the center that it cannot reach the center in $(1 + \epsilon)n$ steps for some constant $\epsilon > 0$. They also show several other lower bounds for selection for highly adaptive algorithms that are restricted in other ways, such as in the number of packets that can be at one processor at any given time.

The best algorithm for selection is also given by [4]. Their $1.15n$ randomized algorithm is obtained in part by increasing the number of knowledge steps, and hence the adaptiveness, of the algorithm. Their algorithm, as well as previous algorithms, including the method of [15], can be described as “filtering” methods:

initially all elements are considered to be possible candidates for the median and are routed towards the center; then, over time, unlikely candidates are filtered out by a set of filtering processors to reduce the routing bottleneck close to the center. Processors that are equidistant from the center form a diamond-shaped filter at a given time. Each filtering processor uses sampling techniques to compute, at this time, a restricted range that is likely to contain the median; henceforth, elements routed to that processor which lie outside this range are discarded. Thus, the routes of packets are adapted at the filtering steps.

The success of the filtering method depends on the routing scheme, the locations of the filters and the times that filtering is done. In the algorithm of Kalamanis *et al.* [15], filtering is only done once, based on a single sample which is collected at the center. Condon and Narayanan's scheme uses three filters, and uses a new distributed sampling method to enable the filtering processors to filter elements earlier in the algorithm. They also use a different randomized routing method in order to spread packets uniformly over the mesh, which increases the effectiveness of the filtering method. They claim that their techniques push the filtering method to its limits, in that little further improvement to the running time can be obtained by increasing the number of filters.

It remains an open question whether a distance-optimal algorithm for selection on the mesh exists. The lower bound of [4] shows that, in order to achieve such a bound, the number of knowledge steps of an algorithm must grow with the input size. They also argue, though they do not prove, that even with an arbitrary number of knowledge steps, it is unlikely that a distance optimal algorithm can be constructed using a filtering approach, where filtering diamonds periodically eliminate all elements which are no longer live. An alternative approach might be to construct a highly dynamic scheme to route elements towards the center, where the route of an element at a given time depends not just on whether it is live or not. Instead, the route of an element may depend on such parameters as its relative order among the live elements that are accessible to the processor at which it currently resides. Analyzing or implementing an algorithm based on this approach would likely be difficult, however.

The best deterministic algorithm for this problem also employs a filtering approach, and takes $1.44n$ time steps [23]. The derandomizing techniques of Kaufmann *et al.* [20] would likely yield a deterministic $1.15n$ algorithm, but as far as the author is aware, such a result has not been published. Distance-optimal randomized algorithms for selection exist in some limited circumstances [22]. Furthermore, the best multi-packet selection algorithms are randomized (see [35]).

References

1. T. Blank. The Maspar MP-1 architecture. In *Proceedings of IEEE Comcon*, 1990.
2. A. Borodin and J. Hopcroft. Routing, merging and sorting on parallel models of computation. *Journal of Computer and System Science*, 30:130–145, 1985.
3. H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *Annals of Mathematics and Statistics*, 23:493–507, 1952.

4. A. Condon and L. Narayanan. Upper and lower bounds for selection on the mesh. In *Symposium on Parallel and Distributed Processing*, pages 497–504. IEEE, 1994.
5. I. Corporation. Paragon xp/s product overview. Technical report, Intel, 1991.
6. T. Dachraoui and L. Narayanan. Fast deterministic sorting on large parallel machines. In *Symposium on Parallel and Distributed Processing*. IEEE, 1996. To appear.
7. W. Dally. Wire efficient VLSI multi-processor communication networks. In *Advanced Research in VLSI*, pages 391–415, 1987.
8. R. Diekmann, J. Gehring, R. Luling, B. Monien, M. Nubel, and R. Wanka. Sorting large data sets on a massively parallel system. In *Proceedings of the Symposium on Parallel and Distributed Processing*, pages 2–9. IEEE, 1994.
9. U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *Symposium on the Foundations of Computer Science*, pages 553–562. IEEE, 1992.
10. S. Felperin, P. Raghavan, and E. Upfal. A theory of wormhole routing in parallel computers. In *Symposium on the Foundations of Computer Science*, pages 563–572. IEEE, 1992.
11. R. Floyd and R. Rivest. Expected time bounds for selection. *Communications of the ACM*, 18:165–172, 1975.
12. W. Hightower, J. Prins, and J. Reif. Implementations of randomized sorting on large parallel machines. In *Symposium on Parallel Algorithms and Architecture*, pages 158–167. ACM, 1992.
13. C. Kaklamanis and D. Krizanc. Optimal sorting on mesh-connected processor arrays. In *Symposium on Parallel Algorithms and Architecture*, pages 50–59, 1992.
14. C. Kaklamanis and D. Krizanc. Multipacket hot-potato routing on processor arrays. In *Euro-Par*, pages 270–277, 1996.
15. C. Kaklamanis, D. Krizanc, L. Narayanan, and A. Tsantilas. Randomized sorting and selection on mesh-connected processor arrays. In *Symposium on Parallel Algorithms and Architecture*, pages 17–28, 1991.
16. C. Kaklamanis, D. Krizanc, and S. Rao. Simple path selection for optimal routing on processor arrays. In *Symposium on Parallel Algorithms and Architecture*, pages 23–30, 1992.
17. C. Kaklamanis, D. Krizanc, and S. Rao. Hot-potato routing on processor arrays. In *Symposium on Parallel Algorithms and Architecture*, pages 273–282, 1993.
18. C. Kaklamanis, D. Krizanc, and A. Tsantilas. Tight bounds for oblivious routing in the hypercube. In *Symposium on Parallel Algorithms and Architecture*, pages 31–36, 1990.
19. M. Kaufmann, S. Rajasekaran, and J. Sibeyn. Matching the bisection bounds for routing and sorting on the mesh. In *Proceedings of SPAA 92*, pages 31–40, 1992.
20. M. Kaufmann, J. Sibeyn, and T. Suel. Derandomizing algorithms for routing and sorting on meshes. In *Symposium on Discrete Algorithms*, pages 669–679. ACM-SIAM, 1994.
21. M. Kaufmann, J. Sibeyn, and T. Suel. Beyond the bisection bound: ranking and counting on meshes. In *Proceedings of the European Symposium on Algorithms*, volume LNCS 979, 1995.
22. D. Krizanc and L. Narayanan. Optimal algorithms for selection on a mesh-connected processor array. In *Symposium on Parallel and Distributed Processing*, pages 70–76. IEEE, 1992.
23. D. Krizanc, L. Narayanan, and R. Raman. Fast deterministic selection on mesh-connected processor arrays. *Algorithmica*, 15:319–332, 1996.

24. M. Kunde. 1-selection and related problems on grids of processors. *Journal of New Generation Computer Systems*, 2:129–143, 1989.
25. M. Kunde. Concentrated regular data streams on grids: Sorting and routing near to the bisection bound. In *Symposium on the Foundations of Computer Science*, pages 141–150. IEEE, 1991.
26. M. Kunde. Block gossiping on grids and tori: Sorting and routing match the bisection bound deterministically. In *Proceedings of the 1st European Symposium on Algorithms*, 1993.
27. F. T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Symposium on Parallel Algorithms and Architecture*, pages 2–10, 1990.
28. F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*. Morgan Kaufmann, 1992.
29. F. T. Leighton. Methods for message routing in parallel computers. In *Symposium on the Theory of Computation*, pages 77–96, 1992.
30. F. Meyer auf der Heide and M. Westermann. Hot-potato routing on multi-dimensional tori. In *Work-shop on Graph-Theoretic Concepts in Computer Science*, pages 275–287, 1995.
31. L. Narayanan. *Selection, Sorting, and Routing on Mesh-Connected Processor Arrays*. PhD thesis, University of Rochester, 1992.
32. M. Noakes, D. Wallach, and W. Dally. The J-machine multicomputer: an architectural evaluation. In *International Symposium on Computer Architecture*, 1993.
33. S. Rajasekaran. Mesh-connected computers with fixed and reconfigurable buses: packet routing, sorting and selection. In *European Symposium on Algorithms*, 1993.
34. S. Rajasekaran. $k - k$ routing, $k - k$ sorting and cut-through routing on the mesh. *Journal of Algorithms*, 19:361–382, 1995.
35. S. Rajasekaran, W. Chen, and S. Yooseph. Unifying themes for network selection. In *Proceedings of ISSAC*, 1995.
36. S. Rajasekaran and T. Tsantilas. Optimal algorithms for routing on the mesh. *Algorithmica*, 8:21–38, 1992.
37. J. Reif and L. Valiant. A logarithmic time sort for linear size networks. *Journal of the ACM*, 34(1):60–76, 1987.
38. R. Reischuk. Probabilistic parallel algorithms for sorting and selection. *SIAM Journal of Computing*, 14(2):396–411, May 1985.
39. C. Schnorr and A. Shamir. An optimal sorting algorithm for mesh-connected computers. In *Symposium on the Theory of Computation*, pages 255–263, 1986.
40. W. Seitz, C. amd Athas, F. C., A. Martin, J. Siezovic, C. Steele, and W. Su. The architecture and programming of the ametek series 20101 multicomputer. In *Proceedings of the Third Conference on Hypercube Concurrent Computers and Applications*, pages 33–36, 1988.
41. C. Thompson and H. Kung. Sorting on a mesh connected parallel computer. *Communications of the ACM*, 20:263–270, 1977.
42. L. Valiant. A scheme for fast parallel communication. *SIAM Journal of Computing*, 11:350–361, 1982.
43. L. Valiant and G. Brebner. Universal schemes for parallel communication. In *Symposium on the Theory of Computation*, pages 263–277, 1981.