Parallel Processing Letters

PERMUTATION ROUTING AND SORTING ON MESHES WITH ROW AND COLUMN BUSES*

TORSTEN SUEL†
Department of Computer Science
University of Texas at Austin
Austin, TX 78712

Received July 6, 1994
Revised November 15, 1994
Communicated by A. Schuster

ABSTRACT

We study the problems of permutation routing and sorting on several models of meshes with fixed and reconfigurable row and column buses. We describe two fast and fairly simple deterministic algorithms for permutation routing on two-dimensional networks, and a more complicated algorithm for multi-dimensional networks. The algorithms are obtained by converting two known off-line routing schemes into deterministic routing algorithms, and they can be implemented on a variety of different models of meshes with buses. We also give a deterministic algorithm for 1–1 sorting whose running time matches that for permutation routing, and another algorithm that matches the bisection lower bound on reconfigurable networks of arbitrary constant dimension.

Keywords: Parallel Algorithms, Routing, Sorting, Reconfigurable Mesh, Mesh, Buses.

1. Introduction

The mesh-connected array of processors is one of the most thoroughly investigated interconnection schemes for parallel processing. It is of great importance due to its simple structure and its good performance in practice. Consequently, a variety of algorithmic problems have been analyzed as to their complexity on theoretical models of the mesh; probably the two most extensively studied problems are those of routing and sorting. One of the main drawbacks of the mesh is its large diameter in comparison to many other networks, such as the hypercube and its bounded-degree variants [1]. An $n \times n$ mesh has a radius of $n-1$, and hence even computations that require only a very limited amount of communication, for example prefix computations, still require at least $n-1$ communication steps.

---

*A preliminary version of this paper was presented at the 8th International Parallel Processing Symposium, April 1994.

† Supported by the Texas Advanced Research Program under Grant Nos. #003658–480 and #003658–461, and by a Schlumberger Graduate Fellowship.

To remedy this situation, several authors [2, 3, 4] have proposed to augment the mesh architecture with high-speed buses that allow fast communication between processors located in different areas of the mesh. This has resulted in a large body of literature on various models of meshes with bus connections, and many important algorithmic problems have been studied under these models. Among the most frequently studied problems are Maximum [3, 5, 6, 7], Prefix Sums [4, 6, 8, 9, 10, 11], Selection [4, 11, 12, 13], as well as some problems in image processing and graph theory [7, 14, 15, 16, 17]. Additional literature can be found in [18].

Due to the low communication requirements of the above problems, significant speed-ups over the standard mesh can be achieved; the exact time complexities depend heavily on the properties of the bus system. For example, the maximum of n^2 elements can be computed in time $O(\lg \lg n)$ on an $n \times n$ mesh with a fully reconfigurable bus, while the same problem requires $\Theta(n^{1/3})$ steps on a mesh with fixed row and column buses. In the following, we briefly describe some of the main features of the bus system that determine the power of the model.

(1) Architecture of the bus system: A bus is called *global* if it is connected to all processors in the network. A bus that is connected to only a subset of the processors is called *local*. Examples of meshes with one or several global buses are given in [3, 4, 5, 7]. Most of the work on local buses has focused on meshes with row and column buses [6, 11, 17], although other architectures have been proposed [17, 19].

(2) Reconfigurability of the buses: A bus is called *reconfigurable* if it can be partitioned into subbuses, such that each subbus can be used as a separate, independent bus. A bus that is not reconfigurable is called *fixed*.

(3) Conflict resolution for bus access: Most papers assume that the buses have broadcast capability, that is, a value written on the bus by one processor can be read by all other processors connected to the bus in the next step. Another common assumption is that the result is undefined if several processors attempt to write on the same bus in a single step of the computation. Using the PRAM terminology, we refer to such a bus as being *Concurrent Read Exclusive Write*, or CREW for short. There is a close relationship between a shared memory cell in a CREW PRAM and a global bus of the same type [19].

Additional features that have been studied include buses with non-unit delay [7, 18], and buses that allow pipelining of messages under certain conditions [20]. Finally, the concept of a mesh with a reconfigurable bus system can also be generalized to reconfigurable networks of arbitrary topology [21].

The model of computation assumed in this paper is a mesh with row and column buses. We consider both fixed and reconfigurable buses. Of course, all algorithms designed for such a model also run on more powerful models, such as the *Polymorphic Torus* [18], the RMESH [7], or the PARBUS [22], whose bus systems can be reconfigured into row and column buses. On the other hand, it does not seem that these more powerful models offer any advantages with respect to permutation routing and 1–1 sorting, which are primarily restricted by the bisection width of the network. Unless explicitly stated otherwise, we assume the buses to be CREW.

An alternative way to overcome the diameter restriction of the standard mesh is to augment the network with a sparse system of bidirectional communication links connecting processors in different areas of the mesh. Examples for this approach are the *Mesh of Trees* [1], or the *Packed Exponential Connections* [23]. It turns out that many of the algorithms and techniques described in this paper can be adapted to these classes of networks, and we will point this out in a few instances.

1.1. Related Results

In this paper, we study the problems of permutation routing and 1–1 sorting on meshes with buses. The *routing problem* is the problem of rearranging a set of packets in a network, such that every packet ends up at the processor specified in its destination address. A routing problem in which each processor is the source and destination of at most k packets is called a k–k routing problem. The routing problem most extensively studied in the literature is the 1–1 routing problem, also called the *permutation routing problem*. In the 1–1 sorting problem, we assume that each processor initially holds a single packet, where each packet contains a key drawn from some totally ordered set. Our goal is to rearrange the packets such that the packet with the key of rank i is moved to the processor with index i, for all i. For both permutation routing and 1–1 sorting, at least $\Theta(n)$ steps are required on all proposed variants of meshes with buses, due to bisection width. However, the exact complexity of these problems has only recently been investigated.

The study of permutation routing on meshes with buses was initiated by Leung and Shende [24]. They assume a model of computation, hereinafter referred to as the *mesh with fixed buses*, that consists of a mesh with fixed row and column buses in addition to the mesh edges. For the one-dimensional case, they obtain a routing algorithm running in $2n/3$ steps, and show a matching lower bound that also extends to multi-dimensional networks. For the two-dimensional case, Leung and Shende show that every permutation can be routed off-line in $n+1$ steps. They also describe deterministic algorithms that run in time $(1+\epsilon)n + o(n)$ and queue size $O(1/\epsilon)$ on the two-dimensional mesh with fixed buses, and in time $(7(d-1)/6+\epsilon)n+o(n)$ and queue size $O(\epsilon^{1-d})$ on d-dimensional networks. (The queue size is the maximum number of packets any node has to store during the algorithms.)

Rajasekaran and McKendall [25, 26] describe randomized algorithms for routing and sorting on a network in which the mesh edges have been replaced by a global reconfigurable bus. This model is essentially the same as the PARBUS, but has the additional property that every subbus of length one can be used in the same way as a bidirectional edge in a standard mesh (that is, a message can be transmitted in either direction in a single step). There is an obvious lower bound of $n/2$ steps for permutation routing and sorting in this model, due to bisection width. Rajasekaran and McKendall describe a $3n/4$ time deterministic algorithm for permutation routing in the one-dimensional case, and a randomized algorithm for the two-dimensional case that achieves a running time of $(1+\epsilon)n$ and a queue size of

$O(1/\epsilon)$, with high probability. They also give randomized algorithms for sorting with the same bounds on running time and queue size.

While this assumption of bidirectional communication in subbuses of length one may be technologically feasible, it can also be seen as somewhat unsatisfactory from a theoretical point of view, since it adversely affects the simplicity of the model. In this context, we point out that many of their algorithms, including the $(1+\epsilon)n$ time routing algorithm, do not make use of this assumption. Similarly, their algorithms do not use any bus connections other than those along a single row or column. Thus, in the present paper, we consider a model with reconfigurable row and column buses, and we assume that only one processor can write on a subbus in any single step, regardless of the length of that subbus. In this model, hereinafter referred to as the *mesh with reconfigurable buses*, there is a trivial lower bound of n steps for permutation routing and sorting due to bisection width.

Comparing the two different models of meshes with buses described above, we observe that the mesh with reconfigurable buses can emulate the standard mesh with constant slowdown by partitioning the buses appropriately. In the case of the mesh with fixed buses, on the other hand, we cannot remove the standard mesh edges without losing the capability of efficiently performing local communication among groups of adjacent processors. In fact, several routing algorithms for such a network with fixed buses and no local connections have been proposed by Iwama, Miyano, and Kambayashi [27]. Due to the impossibility of efficient local communication, their algorithms have a queue size of $\Theta(n)$ in the worst case.

Very recently, and independent of our work, Sibeyn, Kaufmann, and Raman [28] have obtained a randomized routing algorithm for the two-dimensional mesh with fixed buses that runs in time $0.78n$, and an algorithm for d-dimensional networks that runs in time $(2-1/d)n+o(n)$. (The exact running time is actually slightly better than this bound.) By applying a new "derandomization technique" (in an informal sense) for routing and sorting on meshes recently described in [29], it is possible to obtain deterministic algorithms that match the running times of these randomized algorithms.

Sibeyn, Kaufmann, and Raman also show improved lower bounds for routing on meshes with fixed buses. In particular, they show lower bounds of $0.69n$ and $0.72n$ for the two-dimensional and three-dimensional cases, respectively, and a lower bound of approximately $\frac{d-1}{d}n$ for d-dimensional networks. (The lower bound for the two-dimensional case was also discovered by Cheung and Lau [30].)

In other independent work, Cogolludo and Rajasekaran [31] have given a $\frac{17n}{18}+o(n)$ time randomized routing algorithm for the two-dimensional mesh with reconfigurable buses, under the assumption that subbuses of length one can be used as bidirectional edges. They also give an algorithm with running time $\frac{2n}{3}-\frac{n}{64}+o(n)$ for a model with two unidirectional reconfigurable buses in each row and column.

For k–k routing and sorting on d-dimensional networks, there are lower bounds of $kn/3$ and kn for the mesh with fixed and reconfigurable buses, respectively, due to bisection width. For the mesh with fixed buses, Rajasekaran [25] and Sibeyn, Kaufmann, and Raman [28] describe randomized algorithms that match the lower bound, within a lower order additive term. An optimal randomized algorithm for k–k sorting on the mesh with reconfigurable buses can be obtained by a straightforward implementation of the algorithm for the standard mesh given in [32]. Very recent work by Kaufmann, Sibeyn, and Suel [29] and Kunde [33] implies that this bound can also be matched deterministically.

1.2. Overview of the Paper

In this paper, we study the problems of permutation routing and 1–1 sorting on meshes with row and column buses. We consider several variants of this model, with both fixed and reconfigurable buses.

In the first part of the paper, we describe deterministic algorithms for permutation routing. We give two fairly simple algorithms for the two-dimensional case that achieve a running time of $n + o(n)$ and very small queue size, and an algorithm for d-dimensional networks, $d \geq 3$, with a running time of $(2 - 1/d)n + o(n)$ and a queue size of two. An interesting feature of our algorithms is that they can be efficiently implemented on a variety of different classes of networks. The algorithms are obtained with a new technique that allows us to convert certain off-line routing schemes into deterministic on-line algorithms. We believe that this technique may have further applications.

In the second part of the paper, we present two algorithms for 1–1 sorting. The first algorithm is based on a deterministic sampling technique, and its running time matches that for permutation routing, within a lower order additive term. The second algorithm is based on a variation of Leighton's Columnsort [34], and runs in time $n + o(n)$ on meshes with reconfigurable buses of arbitrary constant dimension, thus nearly matching the bisection lower bound of n steps.

2. Permutation Routing

In this section, we describe a technique that allows us to convert certain off-line routing schemes into deterministic routing algorithms. We then use this technique to design new algorithms for permutation routing on meshes with buses. We begin by giving an alternative description of a simple $n + 1$ step off-line routing scheme proposed by Leung and Shende [24]. In Subsection 2.2 we show how this off-line routing scheme can be used to obtain a fast and fairly simple deterministic routing algorithm for two-dimensional meshes with buses. Subsection 2.3 applies the technique to multi-dimensional networks. Finally, Subsection 2.4 gives another algorithm for the two-dimensional case.

2.1. Off-line Routing

In the off-line routing scheme of Leung and Shende [24], every packet is routed to its destination by first routing it on a column bus to its destination row, and then routing it on a row bus to its destination column in the following step. Thus, the algorithm does not use the mesh edges at all. Leung and Shende show that, for any input permutation, a schedule for the above routing scheme can be computed in time $O(n^{7/2})$ by computing a sequence of n maximum matchings. Once the schedule has been computed, it can be executed in $n + 1$ steps.

Now consider the following interpretation of the above scheduling problem. The columns of the mesh are interpreted as *processes* $P_0, \ldots, P_{n-1}$. Every process P_i has exclusive ownership of its column bus, and has to route the n packets initially located in its column. To do so, a process needs to use the row buses, which are interpreted as *resources* $R_0, \ldots, R_{n-1}$. Before a packet can be transmitted on a row bus to its final destination, it has to be routed within its column to the correct row; this can be done in the preceding step using the column bus. If k packets in column i have a destination in row j, then process P_i needs to access resource R_j for k time steps. These k steps can be scheduled in any arbitrary order, provided that in any step, each resource is accessed by at most one process, and each process uses at most one resource. The problem of finding a minimum time schedule that satisfies these demands is known as the *Open Shop Scheduling Problem* [35].

For $0 \le i, j < n$, let $D_{i,j}$, the *demand* of process P_i for resource R_j, be the number of packets in column i that have a destination in row j. Note that

$$\sum_{i=0}^{n-1} D_{i,j} = n \tag{1}$$

holds for all j, since every row is the destination of n packets, and

$$\sum_{j=0}^{n-1} D_{i,j} = n \tag{2}$$

holds for all i, since every column is the origin of n packets. A simple algorithm for finding a minimum time schedule computes a sequence of maximum matchings in the bipartite graph $G = (U, V, E)$ defined by $U = \{P_0, \ldots, P_{n-1}\}$, $V = \{R_0, \ldots, R_{n-1}\}$, and $E = \{(P_i, R_j) \mid D_{i,j} > 0\}$. More precisely, the algorithm first computes a maximum matching M of G, and schedules each process with its matched resource for $D_{\min}$ time steps, where $D_{\min} = \min\{D_{i,j} \mid (P_i, R_j) \in M\}$. Next, we subtract $D_{\min}$ from all $D_{i,j}$ with $(P_i, R_j) \in M$, construct a new bipartite graph G' corresponding to the new values of the $D_{i,j}$, and compute a new matching M'. This procedure is repeated until all $D_{i,j}$ are equal to zero. Using Hall's Matching Theorem, it can be shown that Equations (1) and (2) guarantee that the resulting schedule has a length of n. This implies that at most n matchings have to be computed, since every matching increases the length of the schedule by at least one.

A maximum matching on a bipartite graph with $2n$ vertices can be computed in time $O(n^{5/2})$ using the algorithm of Hopcroft and Karp [36]. Thus, the entire schedule can be computed in time $O(n^{7/2})$. In the following, we show how to convert this off-line algorithm into an on-line algorithm that runs in time $n + o(n)$.

2.2. Routing on Two-Dimensional Networks

In order to get a running time of $n + o(n)$, we modify the above algorithm such that the routing schedule can be computed in time $o(n)$. Executing the computed schedule then takes another $n + o(n)$ steps. The key idea in our construction is to reduce the size of the scheduling problem, and thus the size and number of the matchings that have to be computed. Informally speaking, this is done by partitioning the mesh into a smaller number of processes and resources, and by treating sets of packets with similar sources and destinations as if they were a single packet. This is described more formally in the following.

We partition the network into blocks B_i, $0 \leq i < n^{2-2\alpha}$, of size $n^\alpha \times n^\alpha$, where α is some constant that is smaller, but sufficiently close to 1 (for example, $\alpha = 0.9$). We now interpret each of the $n^{1-\alpha}$ columns of blocks as a process, and each of the $n^{1-\alpha}$ rows of blocks as a resource. Each process P_i, $0 \leq i < n^{1-\alpha}$, has exclusive ownership of its n^α column buses, while each resource R_j, $0 \leq j < n^{1-\alpha}$, consists of n^α row buses. At most one process is allowed to access a single resource at any time. Thus, a process that has exclusive access to a resource can transmit up to n^α packets across the row buses of the resource in a single step.

We now have to arrange the packets inside the processes in such a way that we can make optimal use of this new configuration. To do so we have to slightly relax the goal of the routing schedule that has to be computed. Rather than requiring each packet to be at its final destination after execution of the schedule, we are content with routing each packet to some position in the $n^\alpha \times n^\alpha$ block that contains its destination. After completion of the schedule, we can then bring the packets to their final destinations by routing locally inside each block.

To arrange the packets for the routing schedule, we sort the blocks into row-major order, where the packets are sorted by their destination blocks. We say that a row of a block B_i is *clean* if all its packets have the same destination block. Otherwise, we say that the row is *dirty*. All n^α packets in a clean row of a block are transmitted across the row buses to their common destination block in a single step, after they have been routed to the correct row of blocks in the preceding step. If a row of a block is dirty, then the packets in the row are transmitted across the row buses in r separate steps, where r is the number of distinct destination blocks that occur among the packets in the row. In other words, such a row is treated in the same way as r separate rows; this increases the number of steps required to route this row by $r - 1$. Since there are only $n^{2-2\alpha}$ blocks, this increases the number of steps required to route the packets of a single block across the row buses by at most $n^{2-2\alpha} - 1$. Thus, the number of steps required to route all packets of a process P_i across the row buses is increased by less than $n^{3-3\alpha}$. Hence, if $D_{i,j}$ denotes the number of steps that process P_i needs resource R_j, then

$$\sum_{j=0}^{n^{1-\alpha}-1} D_{i,j} < n + n^{3-3\alpha} \tag{3}$$

holds for all processes P_i. Correspondingly, it can shown that

$$\sum_{i=0}^{n^{1-\alpha}-1} D_{i,j} < n + n^{3-3\alpha} \tag{4}$$

holds for all resources R_j, since for any two blocks B_k, B_l, there can be at most two dirty rows in B_k that contain packets destined for B_l. Equations (3) and (4) guarantee the existence of a schedule of length at most $n + n^{3-3\alpha} = n + o(n)$ that routes every packet to its destination block.

It remains to show that such a schedule can be computed in time $o(n)$. Since we only have $n^{1-\alpha}$ processes and resources, the graph G that is used in the construction of the schedule has only $2n^{1-\alpha}$ vertices. Hence, a maximum matching in this graph can be computed in time $O\left(\left(n^{1-\alpha}\right)^{5/2}\right)$. For each matching that is computed, at least one edge is removed from the graph. This implies that at most $n^{2-2\alpha}$ matchings have to be computed, and the total time to compute the schedule sequentially is bounded by $O\left(\left(n^{1-\alpha}\right)^{9/2}\right) = o(n)$. In order to implement this computation on a mesh with buses, all the data needed to construct the graph G is routed on the buses to a small area, say in the center of the mesh, where the schedule is computed and then broadcast to all blocks. It suffices if each block contributes the numbers m_i, $0 \le i < n^{2-2\alpha}$, where m_i is defined as the number of elements in the block that are destined to block B_i; this can be done in time $o(n)$. We do not elaborate any further on the implementation of the maximum matching algorithm on the mesh. Since we do not need an algorithm that is faster than the sequential one, this is an easy task. (In fact, we could even afford a straightforward simulation of a turing machine algorithm on the mesh; this could be done with a queue size of one.) All in all, we obtain the following algorithm:

(1) Partition the mesh into blocks of size $n^\alpha \times n^\alpha$. Sort the packets in each block into row-major order by destination blocks. This takes $O(n^\alpha)$ steps.

(2) In each block, compute the m_i, $0 \le i < n^{2-2\alpha}$ (m_i was defined as the number of packets with destination block B_i). Send the m_i to a block of side length $n^{2-2\alpha}$ in the center of the mesh. This takes $O(n^{2-2\alpha})$ steps.

(3) Compute the schedule and broadcast it to all blocks of the mesh. This takes $O\left((n^{1-\alpha})^{9/2}\right)$ steps.

(4) Execute the computed schedule of length $n + n^{3-3\alpha}$.

(5) Perform local routing inside each block to bring the packets to their final destinations. This takes time $O(n^\alpha)$.

It remains to show that the algorithm can be implemented with a small queue size. Consider any destination block B_i, and recall that up to n^α packets enter B_i across the row buses in a single step. Due to the sorting in Step (1) of the algorithm, every block in the mesh can have at most two dirty rows that contain elements with destination block B_i. This implies that B_i only receives packets in at most $n^\alpha + 2n^{2-2\alpha}$ steps of the schedule. If we require that the packets arriving in the ith such step are stored by the processors in the $(i \bmod n^\alpha)$th column of B_i, then most processors in B_i only get a single packet, while up to $2n^{2-\alpha}$ processors receive two packets. In addition, every processor in B_i can contain one packet with source in B_i that has not been sent out yet. Finally, some of the processors in B_i, say those on the diagonal of the block, also have to store the packets that can enter the block across the column buses in each step, and that are then routed on the row buses in the following step. This gives a total queue size of four.

We can decrease the queue size to three by assuming that the elements in the diagonal of B_i do not receive any of the packets entering the block across the row buses. To get a queue size of two, we require that every destination block stops accepting new packets from the row buses after it has received $n^\alpha - 1$ batches of packets. It can be shown that every block is still able to deliver the vast majority of its packets to their destination blocks. We can now rearrange the packets in each block, and then deliver the remaining packets; the details of this construction are omitted. This establishes the following result.

Theorem 1 *There exists a deterministic algorithm for permutation routing on the $n \times n$ mesh with buses that runs in time $n + o(n)$ with a queue size of two.*

The above algorithm does not assume any particular model of the mesh with row and column buses, and it can be efficiently implemented on a variety of different classes of networks. For the mesh with fixed buses, this improves upon the best previously known deterministic algorithm [24] in both running time and queue size. As an example, the algorithm in [24] requires a queue size of more than 200 to obtain a running time of $1.2n$. (However, the algorithm is not as fast as the independently discovered algorithms of Sibeyn, Kaufmann, and Raman [28].) On the mesh with reconfigurable buses, our algorithm improves upon the best previously known randomized algorithm of Rajasekaran and McKendall [26], and matches the bisection lower bound, within a lower order additive term.

The algorithm can also be easily adapted to the Polymorphic Torus described in [18]. (This network is essentially a mesh with reconfigurable row and column buses and wrap-around connections.) The resulting algorithm routes any permutation in time $n/2 + o(n)$, and thus nearly matches the lower bound of $n/2$.

For another example, consider a model of the mesh with fixed buses in which the buses have a non-unit propagation delay $\rho(n)$. It was observed by Cheung and Lau [30] that, for any non-constant delay function ρ, routing takes time $2n - o(n)$ in this model, assuming that no pipelining is allowed on the buses. However, if we lift this restriction and allow a processor that sends a packet on the bus to send another packet in the next step, then we can route in time $n + o(n)$, for any $\rho = o(n)$, using a variant of the above algorithm. (As a corollary, this also gives an $n + o(n)$ algorithm for permutation routing on the Mesh of Trees [1].)

The above result shows that even a fairly simple algorithm on the mesh with buses can achieve a speed-up by a factor of two over meshes without buses. Moreover, our algorithm has a queue size of two. In this context, we point out that the $3n - 3$ step off-line scheme for routing on the standard mesh described by Annexstein and Baumslag [37], as well as the $3n + o(n)$ sorting algorithm of Schnorr and Shamir [38], achieve a queue size of one because two packets can be exchanged across an edge in a single step. Since we do not allow two arbitrary processors connected to a common bus to exchange two packets in a single step, it seems difficult to design any algorithm with queue size one that uses the buses to transmit packets.

An even greater speed-up over the standard mesh can be achieved for certain restricted classes of permutations. Consider a partial permutation with only a small number of packets (say, at most ϵn^2). In the case of the standard mesh, this problem still requires a running time of $2n-2$ in the worst case. On the mesh with buses, our only restriction is the bisection bound, and hence we could hope for a speed-up of up to $1/\epsilon$ over full permutation routing. The above algorithm can be adapted in such a way that it achieves this bound for any constant ϵ, provided that the sources and destinations of the packets are approximately evenly distributed over the mesh.

In the following, a partial permutation with no more than ϵn^2 packets is called an ϵ-permutation. We say that an ϵ-permutation is δ-approximate if every $m \times m$ block of the mesh is the source and destination of at most $\epsilon m^2 + \delta$ packets, for all m with $1 \leq m \leq n$. Then the following holds for any constant $\epsilon > 0$.

Theorem 2 *For any $\delta = o(n)$, there exists a deterministic algorithm that routes every δ-approximate ϵ-permutation in time $\epsilon n + o(n)$ with a queue size of 2.*

2.3. Routing on Multi-Dimensional Networks

The ideas from the previous subsection can also be applied to multi-dimensional networks. In this subsection, we outline an algorithm for d-dimensional meshes with buses that runs in time $(2-1/d)n + o(n)$ with queue size two. A more detailed description of the algorithm can be found in [39].

Our algorithm is based on a well-known scheme for off-line routing on d-dimensional meshes described in [37]. The routing scheme consists of $2d-1$ phases. In phase i, $1 \leq i \leq d-1$, each packet is routed along dimension i to an appropriately chosen intermediate location. In phase i, $d \leq i \leq 2d-1$, each packet is greedily routed along dimension $2d-i$. Each phase involves a collection of routing problems on linear arrays of length n, and thus takes at most n steps. Hence, the entire routing takes $(2d-1)n$ steps on the standard mesh.

To route a given permutation with the above routing scheme, it is necessary to determine appropriate choices for the intermediate locations assumed by the packets in the first $d-1$ phases. The existence of such intermediate locations is implied by Hall's Matching Theorem, and they can be computed by constructing a sequence of perfect matchings in a graph; the details of this construction can be derived from Section 1.7.5 of [1]. Faster algorithms for computing appropriate intermediate locations are described in [40]. For our purposes, it suffices that the running time of these computations is polynomial in n^d, the number of packets in the network.

To convert the above off-line routing scheme into an on-line algorithm, we introduce the notion of a *super-packet.* Informally speaking, a *super-packet* consists of a collection of packets that have similar sources and destinations, and that move in lock step. By combining a large number of packets into a single super-packet, we can decrease the number of packets in the network in such a way that the intermediate locations can be computed in time $o(n)$.

The resulting on-line algorithm with running time $(2d-1)n + o(n)$ is *uni-axial*, that is, the algorithm communicates only across a single dimension in any given step of the computation. Thus, we can simultaneously run up to d "copies" of the algorithm without any contention for the buses. We now partition the set of packets into d sets of equal size, such that the sources and destinations of the packets in each set are approximately evenly distributed over the entire network. As each set contains only a $1/d$ fraction of all packets, it can be routed in time $(2-1/d)n+o(n)$ by the above algorithm. Altogether, we can show the following result.

Theorem 3 *There exists a deterministic algorithm for routing on d-dimensional meshes with buses that runs in time $(2-1/d)n+o(n)$ with a queue size of two.*

The exact lower order term depends on the algorithm used in the computation of the intermediate locations of the packets. The algorithm can be efficiently implemented on a variety of multi-dimensional networks. A faster algorithm for reconfigurable networks is presented in the next section in the context of sorting.

2.4. Fast Routing without Matching

While the routing algorithms described in the previous subsections are fast from a theoretical point of view, they are certainly not efficient in practice. One source of this inefficiency are the fairly large lower order terms in the running times. As an example, choosing $\alpha = 9/11$ results in a lower order term of $O(n^{9/11})$ in the case of the two-dimensional algorithm. As the constant hidden by the big-Oh notation is sufficiently large, this term would dominate the running time of the algorithm on networks of realistic size. Another source of inefficiency is the complicated control structure of the algorithm, especially in the computation of the matchings, which makes the algorithm unsuitable for any implementation in hardware.

In the following, we describe an $n+O(n^{2/3})$ time algorithm for two-dimensional networks that does not require any computation of matchings, and that uses only prefix computations and local sorting as subroutines. Like the algorithm in Subsection 2.2, it is based on the off-line algorithm of Leung and Shende, and assumes that the network is partitioned into blocks of side length n^α, for some α. However, instead of computing an optimal schedule for the usage of the buses, the algorithm computes an assignment of the row buses to the columns of blocks (and of the column buses to the rows of blocks) that stays fixed throughout most of the algorithm. In this assignment, each column of blocks receives in each row of blocks a number of row buses that is proportional to the number of its packets that have a destination in this row of blocks. (Alternatively, the algorithm can also be seen as an approximate solution for a special case of the *Open Shop Scheduling Problem.*)

Let $\alpha = 2/3$, let $s_{i,j}$ denote the number of packets in the ith column of blocks with destination in the jth row of blocks, and let $b_{i,j} = \lfloor s_{i,j}/n \rfloor$. We now assign $b_{i,j}$ row buses in the jth row of blocks to the ith column of blocks, and $b_{i,j}$ column buses in the ith column of blocks to the jth row of blocks. Note that $\sum_{i=0}^{n^{1/3}-1} b_{i,j} \leq n^{2/3}$ holds for all j, $0 \leq j < n^{1/3}$, and $\sum_{j=0}^{n^{1/3}-1} b_{i,j} \leq n^{2/3}$ holds for all i, $0 \leq i < n^{1/3}$. This assures that the total number of buses assigned in each row of blocks and each column of blocks does not exceed $n^{2/3}$. Such an assignment of the row buses to the columns of blocks, and of the column buses to the rows of blocks, can be easily computed from the $b_{i,j}$ using prefix computations.

After the assignment of the buses has been computed, we run the following protocol for $n+1$ steps. In each step, $b_{i,j}$ column buses in the ith column of blocks are used to transmit $b_{i,j}$ packets (with destination in the jth row of blocks) to the jth row of blocks. Also, in each step, $b_{i,j}$ row buses in the jth row of blocks are used to transmit $b_{i,j}$ packets to their destination blocks. Thus, all packets routed along the columns in step k are routed along the rows to their destination blocks in step $k+1$.

After $n+1$ steps of the above protocol, there are at most $s_{i,j} - n \cdot b_{i,j} < n$ untransmitted packets in the ith column of blocks that have a destination in the jth row of blocks. We can now transmit these remaining packets by setting $b_{i,j} = n^{1/3}$ for all i, j, and running the above protocol for another $n^{2/3}+1$ steps. Finally, local routing inside each block can be used to bring every element to its final destination. Altogether, we obtain the following algorithm.

(1) Partition the mesh into blocks of side length $n^{2/3}$. Use local sorting and prefix computations to compute the assignment of the buses in time $O(n^{2/3})$.

(2) Run the protocol described above for $n+1$ steps.

(3) Set $b_{i,j} = n^{1/3}$ for all i, j, and run the protocol for another $n^{2/3}+1$ steps.

(4) Perform local routing inside each block to bring the packets to their final destinations. This takes time $O(n^{2/3})$.

Some details have been omitted from this description. Before running the protocol in Steps (2) and (3), we have to arrange the packets inside the blocks such that, for all i, j, all $b_{i,j}$ row and column buses can be used in each step, and such that no write conflicts occur. This can be done in time $O(n^{2/3})$ using local sorting and prefix computations. In order to achieve a small, constant queue size, we also have to arrange the packets appropriately upon entering their destination block. The algorithm can again be implemented on a variety of different models of meshes with buses. Of course, it is still too complicated to be of immediate practical interest.

However, we believe that the result is interesting in that it indicates that simple global operations such as prefix computations might be useful in the design of efficient routing algorithms on meshes with buses. In contrast, previously described algorithms for these networks use the buses only for the transmission of the packets, and not for the computation of the routing schedule. While a restriction to local control is appropriate for networks that do not provide any fast global communication, it may be that some amount of global control is useful on networks that support fast (but low bandwidth) global primitives such as prefix computations.

3. Sorting

In this section we describe two algorithms for sorting on meshes with buses. The first algorithm makes use of the routing algorithms from the previous section, and its running time (nearly) matches that for permutation routing. The second algorithm assumes a mesh with reconfigurable buses, and its running time matches the bisection bound for networks of any dimension $d = o(n^{1/3})$. Thus, this algorithm also implies an improved bound for routing on reconfigurable networks with $d \geq 3$.

Recall that in the sorting problem we have to move the element of rank i to the processor with index i, for all i. Sorting algorithms on meshes and related networks are usually designed with a particular indexing of the processors in mind. In the following we assume a *blocked* indexing scheme, in which the network is partitioned into blocks of side length n^δ, $2/3 \leq \delta < 1$, and the processors in each block have consecutive indices, while the blocks are indexed in snake-like row-major order.

3.1. Sorting by Deterministic Sampling

The first algorithm uses a deterministic sampling technique that computes a set of splitters whose ranks are determined to within an additive lower order term. This reduces the problem of sorting to that of routing an appropriate permutation, plus some local operations. The structure of the algorithm is as follows.

(1) Sort each block of side length n^δ into row-major order. This takes time $O(n^\delta)$.

(2) Route copies of the elements in the first column of each block to a block B of side length $n^{1-\delta/2}$ in the center of the mesh. This takes time $O(n^{1-\delta/2})$.

(3) Sort the elements in B and select n^δ elements of equidistant ranks as splitter elements. This takes time $O(n^{1-\delta/2})$.

(4) Use prefix computations to compute the exact ranks of the splitters, and broadcast them to all blocks. This takes time $O(n^\delta)$.

(5) It can be shown that the ith splitter element has a rank between $(i-1) \cdot n^{2-\delta}$ and $i \cdot n^{2-\delta}$. Hence, every element can now determine its rank to within $n^{2-\delta} = O(n^{2\delta})$. Using prefix computations, we can assign each element a preliminary destination that is at most one block away from its final destination.

(6) Route every element to its preliminary destination using the routing algorithm in Subsection 2.2 (or any other routing algorithm).

(7) Perform local sorting between consecutive blocks. This takes time $O(n^\delta)$.

The computation of the splitters in Steps (1) to (3) is essentially a simplified version of a more sophisticated sampling technique used in the parallel selection algorithm of Cole and Yap [41] (see [39] for a more detailed description). Apart from Step (6), all steps take time $o(n)$. For $\delta = 2/3$, we get the following result.

Theorem 4 *For all models of meshes with buses, there exists a sorting algorithm whose running time matches that for permutation routing, within $O(n^{2/3})$ steps.*

3.2. Sorting on Meshes with Reconfigurable Buses

Our second algorithm is based on a variation of Leighton's Columnsort algorithm [34], similar to that described in [29, 33]. The algorithm can be efficiently implemented on several classes of meshes with reconfigurable buses, and also on the Mesh of Trees [1] and the Packed Exponential Connections [23], but it does not give improved bounds on meshes with fixed buses.

We start out by describing how a particular class of routing problems, called κ-way *unshuffle permutations*, can be efficiently solved on a linear array with a reconfigurable bus. We then give the sorting algorithm for networks of arbitrary dimension, and explain how it can be implemented through a sequence of κ-way unshuffle permutations on linear arrays.

Formally, for any $n, \kappa > 0$ with $n \bmod \kappa = 0$, the κ-way unshuffle permutation on n elements is defined as the permutation π_κ that moves the element in position i to position $\pi(i) = (i \bmod \kappa) \cdot n/\kappa + \lfloor i/\kappa \rfloor$, for all i in $[n]$. We observe that if $\kappa = n^{1-\delta}$ for some $\delta \geq 1/2$, then a κ-way unshuffle permutation on a linear array of length n has the effect of distributing the elements of each block of length n^δ evenly over all $n^{1-\delta}$ blocks of length n^δ.

Due to bisection arguments, at least $n/2$ steps are required to route an $n^{1-\delta}$-way unshuffle permutation on a linear array with a reconfigurable bus. The following routing scheme matches this bound, within a lower order term. The routing scheme consists of two parts. In the first part, we route all packets that have to move to the right; in the second part, we route all packets that have to move to the left. Since the two parts are symmetric, we only describe how to route the rightgoing packets.

The schedule for the rightgoing packets is divided into $n^{1-\delta}/2$ phases P_i, $0 \leq i < n^{1-\delta}/2$. Phase P_i of the schedule consists of $n^{1-\delta} - 2i$ subphases $S_{i,j}$, and each subphase takes $n^{2\delta-1}$ steps. Thus, the entire schedule has a length of

$$\sum_{i=0}^{(n^{1-\delta}/2)-1} \left(n^{1-\delta} - 2i\right) \cdot n^{2\delta-1} \;=\; \frac{n}{2} - 2n^{2\delta-1} \sum_{i=0}^{(n^{1-\delta}/2)-1} i \;=\; \frac{n}{4} + \frac{n^\delta}{2}.$$

Given a partition of the array into $n^{1-\delta}$ blocks of length n^δ, we say that block i sends to block j if all packets in block i that have a destination in block j are transmitted to this destination. Note that for all i and j, exactly $n^{2\delta-1}$ packets in block i have a destination in block j. Under our schedule, the rightgoing packets are transmitted according to the following rules:

(a) In any subphase $S_{i,0}$, block i sends to block $n^{1-\delta} - i - 1$.

(b) In any subphase $S_{i,j}$ with $1 \leq j < n^{1-\delta} - 2i$, block i sends to block $i + j - 1$, while block $i + j$ sends to block $n^{1-\delta} - i - 1$.

The following sorting algorithm for d-dimensional networks assumes a blocked indexing scheme with blocks of side length n^δ, $\delta = 2/3$. The algorithm alternates local sorting and communication steps. Each communication step performs a *total exchange* operation among the blocks. The *total exchange* operation, also often called *all-to-all personalized communication*, is a well-known communication problem that arises in a number of parallel applications (e.g., see Section 1.3 of [42]).

(1) Sort the elements inside each block. This takes time $O(d \cdot n^\delta)$ using, say, the k-k sorting algorithm for the standard mesh described in [29, 33].

(2) Perform a total exchange among the blocks, where block i sends the $n^{d(2\delta-1)}$ elements with a local rank of $j \bmod n^{d(1-\delta)}$ to block j, for all i, j.

(3) Sort the elements inside each block.

(4) Perform a total exchange among the blocks, where block i sends the $n^{d(2\delta-1)}$ elements with a local rank between $j \cdot n^{d(2\delta-1)}$ and $(j+1) \cdot n^{d(2\delta-1)} - 1$ to block j, for all i, j.

(5) Perform local sorting between consecutive blocks. This takes time $O(d \cdot n^{\delta})$.

After Step (4) of the algorithm, every element is at most one block away from its final destination (see [29] for a proof of this claim). Thus, the local sorting in Step (5) moves each element to its final destination. Steps (2) and (4) can be implemented by performing an appropriate local permutation in each block, followed by d consecutive $n^{1-\delta}$-way unshuffle permutations, where the ith unshuffle permutation is applied to all linear arrays in direction of the ith dimension. However, using this simple approach we only get a running time of $dn/2 + o(n)$ for each of Steps (2) and (4), since at any point in time only buses along a single dimension are being used.

To overcome this problem, we partition the mesh into d subnetworks, where the νth subnetwork consists of all processors with coordinates $(x_0, \ldots, x_{d-1})$ such that $\sum_{i=0}^{d-1} x_i = \nu \bmod d$. We also partition the set of elements into d subsets, such that each subset contains exactly $n^{d(2\delta-1)}/d$ elements that have to be sent from any block i to any block j.

Each linear array inside a subnetwork has a length of n/d, and can hence perform an unshuffle permutation in time $\frac{n}{2d} + o(n)$. We can now implement Steps (2) and (4) in $n/2 + o(n)$ steps each, by routing each subset within its corresponding subnetwork, where the ith unshuffle permutation is applied to the elements of the jth subset in direction of dimension $(i+j) \bmod d$. This gives the following result.

Theorem 5 *For any $d = o(n^{1/3})$, there exists a sorting algorithm for d-dimensional meshes with reconfigurable buses that runs in time $n + o(n)$ with queue size two.*

For the Polymorphic Torus, and the mesh with two unidirectional reconfigurable buses of [31], we can obtain a running time of $n/2 + o(n)$, by simultaneously routing the leftgoing and rightgoing elements in the unshuffle permutation. The same bound can be achieved on the Mesh of Trees. We can also adapt the algorithm to run in time $\frac{n}{2 \lg n} + o(\frac{n}{\lg n})$ on the Packed Exponential Connections [23] of arbitrary constant dimension. In all of these cases, the algorithm nearly matches the bisection bound.

Finally, we point out that it is straightforward to adapt the algorithm to the k–k sorting problem, in which each processor is the source and destination of k packets. (For $k < 1$, the k–k sorting problem can be defined in a similar way as the δ-approximate ϵ-permutations in Subsection 2.2.) The resulting algorithm matches the bisection lower bound within an additive lower order term for all k with $k = \Omega(1/n^{1-c})$ for some $c > 0$.

Theorem 6 *For any constant $c > 0$, and for any k, d with $k = \Omega(1/n^{1-c})$ and $d = o\left((n \cdot k^{1/d})^{1/3}\right)$, there exists a k–k sorting algorithm for d-dimensional meshes with reconfigurable buses that runs in time $kn + o(kn)$ with queue size two.*

4. Concluding Remarks

In this paper, we have described deterministic algorithms for permutation routing and sorting on meshes with fixed and reconfigurable buses. While the routing algorithms in Section 2 are based on fairly simple ideas, they are impractical due to the large lower order terms in the running times. It is an open question whether the ideas of this paper can be used in the design of more practical algorithms.

Another possible research direction is to find efficient algorithms for routing with locality, or for the routing of sparse or irregular communication patterns. In this context, the buses might be helpful in the design of algorithms that adapt to the degree of locality, sparseness, or irregularity of a problem. One possible approach would be to first design a good off-line routing scheme, and then try to convert this off-line scheme into an on-line algorithm using the ideas of this paper.

Acknowledgements

I would like to thank Phil MacKenzie, Greg Plaxton, Rajmohan Rajaraman, and Jop Sibeyn for helpful discussions.

References

1. F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, and Hypercubes* (Morgan-Kaufmann, San Mateo, CA, 1991).
2. H. F. Jordan, A special purpose architecture for finite element analysis, In *Proc. International Conference on Parallel Processing*, 1978, 263–266.
3. S. H. Bokhari, Finding maximum on an array processor with a global bus, *IEEE Trans. Comput.* **33** (1984) 133–139.
4. Q. F. Stout, Mesh-connected computers with broadcasting, *IEEE Trans. Comput.* **32** (1983) 826–830.
5. A. Aggarwal, Optimal bounds for finding maximum on arrays of processors with k global buses, *IEEE Trans. Comput.* **35** (1986) 62–64.
6. A. Condon, R. E. Ladner, J. Lampe, and R. Sinha, Complexity of sub-bus mesh computations, Technical Report # 93–10–02, Department of Computer Science and Engineering, University of Washington, 1993.
7. R. Miller, V. K. Prasanna Kumar, D. I. Reisis, and Q. F. Stout, Parallel computations on reconfigurable meshes, *IEEE Trans. Comput.* **42** (1993) 678–692.
8. A. Bar-Noy and D. Peleg, Square meshes are not always optimal, In *Proc. 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, July 1989, 138–147.
9. Y. C. Chen, W. T. Chen, G. H. Chen, and J. P. Sheu, Designing efficient parallel algorithms on mesh-connected computers with multiple broadcasting, *IEEE Trans. Parallel and Distrib. Systems* **1** (1990) 241–245.
10. R. E. Ladner, J. Lampe, and R. Rogers, Vector prefix addition on sub-bus mesh computers, In *Proc. 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1993, 387–396.

11. V. K. Prasanna Kumar and C. S. Raghavendra, Array processors with multiple broadcasting, *J. Parallel and Distrib. Comput.* **4** (1987) 173–190.

12. Y. C. Chen, W. T. Chen, and G. H. Chen, Efficient median finding and its application to two-variable linear programming on mesh-connected computers with multiple broadcasting, *J. Parallel and Distrib. Comput.* **15** (1992) 79–84.

13. E. Hao, P. D. MacKenzie, and Q. F. Stout, Selection on the reconfigurable mesh, In *Proc. 4th IEEE Symposium on the Frontiers of Massively Parallel Computations*, 1992, 38–45.

14. K. Iwama and Y. Kambayashi, An $O(\lg n)$ parallel connectivity algorithm on the mesh, In *Information Processing 89*, 1989, 305–310.

15. J. Jang, H. Park, and V. K. Prasanna-Kumar, A fast algorithm for computing histogram on reconfigurable mesh, Technical Report IRIS 290, Institute for Robotics and Intelligent Systems, University of Southern California, 1992.

16. V. K. Prasanna Kumar and C. S. Raghavendra, Image processing on enhanced mesh connected computers, In *Computer Architecture for Pattern Analysis and Image Database Management*, 1985, 243–247.

17. Q. F. Stout, Meshes with multiple buses, In *Proc. 27th Annual IEEE Symposium on Foundations of Computer Science*, 1986, 264–273.

18. H. Li and Q. F. Stout, *Reconfigurable Massively Parallel Computers* (Prentice Hall, Englewood Cliffs, NJ, 1991).

19. F. Meyer auf der Heide and H. T. Pham, On the performance of networks with multiple busses, In *Proc. 9th Symposium on Theoretical Aspects of Computer Science*, 1992, 98–108.

20. Z. Guo, R. G. Melhem, R. W. Hall, D. M. Chiarulli, and S. P. Levitan, Array processors with pipelined optical buses, In *Proc. 3rd IEEE Symposium on the Frontiers of Massively Parallel Computations*, 1990, 333–342.

21. Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, The power of reconfiguration, *J. Parallel and Distrib. Comput.* **13** (1991) 139–153.

22. B. Wang and G. Chen, Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems, *IEEE Trans. Parallel and Distrib. Systems* **1** (1990) 500–507.

23. W. W. Kirkman and D. Quammen, Packed exponential connections - a hierarchy of 2-D meshes, In *Proc. 5th International Parallel Processing Symposium*, 1991, 464–470.

24. J. Y. Leung and S. M. Shende, On multidimensional packet routing for meshes with buses, *J. Parallel and Distrib. Comput.* **20** (1994) 187–197.

25. S. Rajasekaran, Mesh-connected computers with fixed and reconfigurable buses: Packet routing, sorting, and selection, In *Proc. 1st Annual European Symposium on Algorithms*, September 1993, 309–320.

26. S. Rajasekaran and T. McKendall, Permutation routing and sorting on the reconfigurable mesh, Technical Report MS-CIS-92-36, Department of Computer and

Information Science, University of Pennsylvania, May 1992.

27. K. Iwama, E. Miyano, and Y. Kambayashi, Routing problems on the mesh of buses, In *Proc. 3rd International Symposium on Algorithms and Computation*, 1992, 155–164.

28. J. F. Sibeyn, M. Kaufmann, and R. Raman, Randomized routing on meshes with buses, In *Proc. 1st Annual European Symposium on Algorithms*, September 1993, 333–344.

29. M. Kaufmann, J. Sibeyn, and T. Suel, Derandomizing algorithms for routing and sorting on meshes, In *Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1994, 669–679.

30. S. Cheung and F. C. M. Lau, A lower bound for permutation routing on two-dimensional bused meshes, *IPL* **45** (1993) 225–228.

31. J. C. Cogolludo and S. Rajasekaran, Permutation routing on reconfigurable meshes, In *Proc. 4th International Symposium on Algorithms and Computation*, 1993, 157–166.

32. M. Kaufmann, S. Rajasekaran, and J. F. Sibeyn, Matching the bisection bound for routing and sorting on the mesh, In *Proc. 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, July 1992, 31–40.

33. M. Kunde, Block gossiping on grids and tori: Deterministic sorting and routing match the bisection bound, In *Proc. 1st Annual European Symposium on Algorithms*, September 1993, 272–283.

34. F. T. Leighton, Tight bounds on the complexity of parallel sorting, *IEEE Trans. Comput.* **34** (1985) 344–354.

35. T. Gonzalez and S. Sahni, Open shop scheduling to minimize finish time, *J. ACM* **23** (1976) 665–679.

36. J. E. Hopcroft and R. M. Karp, An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs, *SIAM J. Comput.* **2** (1973) 225–231.

37. F. Annexstein and M. Baumslag, A unified approach to off-line permutation routing on parallel networks, In *Proc. 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, July 1990, 398–406.

38. C. P. Schnorr and A. Shamir, An optimal sorting algorithm for mesh-connected computers, In *Proc. 18th ACM Symposium on Theory of Computing*, May 1986, 255–263.

39. T. Suel, Routing and sorting on meshes with row and column buses, Technical Report TR–94–09, University of Texas at Austin, Department of Computer Science, April 1994.

40. G. F. Lev and N. Pippenger and L. G. Valiant, A Fast Parallel Algorithm for Routing in Permutation Networks, *IEEE Trans. Comput.* **30** (1981) 93–100.

41. R. Cole and C. K. Yap, A parallel median algorithm, *IPL* **20** (1985) 137–139.

42. D. P. Bertsekas and J. N. Tsitsiklis, *Parallel and Distributed Computation: Numerical Methods* (Prentice-Hall, Englewood Cliffs, NJ, 1989).