Routing and Sorting on Meshes with Row and Column Buses

Torsten Suel*

Department of Computer Sciences
University of Texas at Austin

Abstract

We give improved deterministic algorithms for permutation routing and sorting on meshes with row and column buses. Among our results, we obtain a fairly simple algorithm for permutation routing on two-dimensional meshes with buses that achieves a running time of $n + o(n)$ and a queue size of 2. We also describe an algorithm for routing on r-dimensional networks with a running time of $(2 - 1/r)n + o(n)$ and a queue size of 2, and show how to obtain deterministic algorithms for sorting whose running times match those for permutation routing. An interesting feature of our algorithms is that they can be implemented on a wide variety of different models of meshes with buses within the same bounds on time and queue size. Finally, we also study the performance of meshes with buses on dynamic routing problems, and propose fast routing schemes under several different assumptions about the properties of the bus system.

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. The main drawback of the mesh is its large diameter in comparison to many other networks, such as the mesh of trees or the hypercubic networks [10]. An $n \times n$ mesh has a diameter of $2n - 2$, and hence even computations that require only a very limited amount of communication, for example prefix computations, require at least $n - 1$ communication steps.

To remedy this situation, it was proposed by several authors [2, 7, 19] 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 different models of meshes with bus connections, and a number of important algorithmic problems have been studied under these models. Among the most frequently studied problems on meshes with buses are Maximum, Prefix Sums, Selection, as well as various algorithmic problems in image processing and graph theory (e.g., see [13, 14, 20] for a list of references).

Due to the low communication requirements of these problems, significant speed-ups over the standard mesh can be achieved. The exact time complexities of the proposed algorithms heavily depend 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 will 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 of the network. Otherwise, it is called *local*.

(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 processors connected to the bus in the next step. Another common assumption is that the result is undefined if several processors simultaneously attempt to write on a bus. Using the PRAM terminology, we refer to such a bus as *Concurrent Read Exclusive Write*, or CREW for short. We also use the term CRCW for buses with broadcast capability and some form of write conflict resolution.

The model of computation assumed in this paper is a mesh with row and column buses. We will consider both fixed and reconfigurable buses. Of course, all algorithms designed for such a model will also run on more powerful models, such as the *Polymorphic Torus* [13] or the PARBUS [21], that can be reconfigured into a mesh with row and column buses. Unless explicitely stated otherwise, we assume the buses to be CREW. However, we will also discuss the impact of other conflict resolution schemes on the performance of the network.

1.1 Related Results

In this paper, we consider the problems of permutation routing, sorting, and dynamic routing on meshes with row and column 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 referred to as 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 in such a way that the packet with the key of rank i is moved to the processor with

*Supported by the Texas Advanced Research Program under Grant No. #003658480. Email: torsten@cs.utexas.edu.

index i, for all i. Finally, in a *dynamic routing problem*, packets are continuously generated at each processor according to some random process; the destinations of the generated packets are randomly chosen among the processors of the network. While dynamic routing problems have been studied on several other classes of networks, including the mesh [10, Section 1.7.2] and the hypercube [18], we are not aware of any previous attempt to analyze the performance of meshes with buses on these problems. In the case of permutation routing and sorting, it is easy to see that 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 row and column buses was initiated by Leung and Shende [11]. They assume a model of computation, hereafter referred to as the *mesh with fixed buses*, that consists of a mesh with non-reconfigurable row and column buses in addition to the standard mesh edges. For the one-dimensional case, they obtain a permutation routing algorithm with running time $2n/3$, and also a matching lower bound. For the two-dimensional case, Leung and Shende show that every permutation can be routed off-line in $n+1$ steps. They also describe a deterministic on-line algorithm that routes in time $(7/6+\epsilon)n+o(n)$ and queue size $O(1/\epsilon)$ on the two-dimensional mesh with fixed buses, and in time $(7(r-1)/6+\epsilon)n+o(n)$ and queue size $O(\epsilon^{1-r})$ on r-dimensional networks. In a subsequent paper [12], they obtain an improved algorithm for the two-dimensional case, running in time $(1+\epsilon)n+o(n)$ with queue size $O(1/\epsilon)$.

Rajasekaran and McKendall [16] and Rajasekaran [15] describe randomized algorithm for routing and sorting on a mesh in which the mesh edges have been replaced by a reconfigurable bus system. This model, hereafter referred to as the *mesh with reconfigurable buses*, is essentially the same as the PARBUS, but has the additional property that every subbus of length 1 can be used in the same way as a bidirectional edge in a standard mesh. There is an obvious lower bound of $n/2$ steps for permutation routing and sorting on this model, due to the bisection width of the network. Rajasekaran and McKendall describe a deterministic algorithm for routing on the one-dimensional mesh with reconfigurable buses with a running time of $3n/4$ and constant queue size, 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 obtain randomized algorithms for sorting with the same bounds on running time and queue size.

Very recently, Sibeyn, Kaufmann, and Raman [17] have shown improved lower bounds for routing on the d-dimensional mesh with fixed buses. In particular, they obtain lower bounds of $0.69n$ and $0.72n$ for the two-dimensional and three-dimensional case, respectively. For large d, their lower bounds are approximately $\frac{d-1}{d}n$. The lower bound for the two-dimensional case was also independently discovered by Cheung and Lau [3]. Sibeyn, Kaufmann, and Raman also give randomized algorithms for permutation routing on meshes with fixed buses that are significantly faster than the deterministic algorithms of Leung and Shende. For the two-dimensional case, they obtain an algorithm with running time $0.78n$. They also give an algorithm for d-dimensional networks that achieves a running time of $(2-1/r)n+o(n)$.

For the problem of k–k routing on r-dimensional networks, $r \geq 1$, there are obvious lower bounds of $kn/3$ and $kn/2$ for meshes with fixed and reconfigurable buses, respectively, due to bisection width. For the mesh with fixed buses, Rajasekaran [15] and Sibeyn, Kaufmann, and Raman [17] describe randomized algorithms that asymptotically match this lower bound. An optimal deterministic algorithm for k–k sorting on the mesh with reconfigurable buses can be obtained by a straightforward implementation of the optimal algorithms for the standard mesh in [8, 9].

1.2 Overview of the Paper

In this paper, we study the complexity of permutation routing, sorting, and dynamic routing on meshes with fixed and reconfigurable row and column buses.

We give a fairly simple deterministic algorithm for permutation routing on the $n \times n$ mesh with buses that achieves a running time of $n+o(n)$ and a queue size of 2. We also give an algorithm for d-dimensional meshes, $d \geq 3$, with a running time of $(2-1/r)n+o(n)$ and a queue size of 2, and show how to obtain algorithms for 1–1 sorting that match those for permutation routing. An interesting feature of all our algorithms is that they can be efficiently implemented on a variety of different classes of networks, including the PARBUS and the Mesh of Trees [10]. For the mesh with fixed buses, our algorithms offer a significant improvement over the best previously known deterministic algorithms [11, 12] with respect to both running time and queue size. For the mesh with reconfigurable buses, our algorithms even improve over the best known randomized algorithms [15]. Our algorithms are obtained with a novel 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 study dynamic routing problems on meshes with buses. In the case of permutation routing, we cannot hope to get a speed-up of more than a constant factor over the standard mesh, with any system of buses that can be laid out in $O(n^2)$ area. However, the situation is completely different in the case of dynamic routing. We will show an algorithm for the mesh with fixed buses that routes every packet in time $O(n^{1/3})$, with high probability. For the mesh with reconfigurable buses, we propose a very simple routing scheme that routes each packet in $O(\lg n)$ steps. For the mesh with fixed CRCW buses, a delivery time of $O(\lg n \lg\lg n)$ can be achieved. In contrast, the expected time for the completion of a routing request on the standard mesh is $\Theta(n)$.

Due to space constraints, we can only give a brief description of the main results. A more detailed presentation can be found in [20].

The remainder of the paper is organized as follows. Section 2 describes our algorithms for permutation routing and sorting. Section 3 contains the analysis of dynamic routing on various models of meshes with buses. Finally, Section 4 lists some open questions for future research.

2 Permutation Routing and Sorting

In this section, we introduce a new technique that allows us to convert certain off-line routing schemes into deterministic routing algorithms. We then use this technique to design improved algorithms for permutation routing and sorting on meshes with buses. We begin by giving a description of a simple $n+1$ step off-line routing scheme proposed by Leung and Shende [11, 12]. In Subsection 2.2 we show how this off-line routing scheme can be converted into a fast and fairly simple deterministic routing algorithm. In Subsection 2.3 we apply our technique to multi-dimensional meshes with buses. Finally, Subsection 2.4 contains our algorithms for 1–1 sorting.

2.1 Off-line Routing

In the off-line routing scheme of Leung and Shende [11, 12], every packet is first routed on a column bus to its destination row, and then routed on a row bus to its destination in the following step. Leung and Shende show that for any input permutation, a schedule for the above routing scheme can be computed in time $O(n^{3.5})$, 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 bused 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 transmit the n packets in its column to their destinations. To do so, a process needs to send packets on the row buses, which are interpreted as *resources* $R_0, \ldots, R_{n-1}$. Before a packet can be transmitted across 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 will need resource R_j for k time steps. These k steps can be scheduled in any arbitrary order, provided that in any given 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 all of these demands is known as the *Open Shop Scheduling Problem*, and was solved by Gonzalez and Sahni [5].

For $0 \leq i, j < n$, 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

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

holds, since every row is the destination, and every column the origin, of exactly 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 maximum matching M'. This procedure is repeated until all demands $D_{i,j}$ have been reduced to zero. Using Hall's Matching Theorem, it can be shown that the above equation (1) guarantee that the resulting schedule has a length of at most n. This in turn implies that at most n matchings have to be computed, since for every matching the length of the schedule is increased by at least one step.

A maximum matching on a bipartite graph with $2n$ vertices can be computed in time $O(n^{2.5})$ using the algorithm of Hopcroft and Karp [6]. Thus, the entire schedule can be computed in time $O(n^{3.5})$. Of course, this makes the algorithm inappropriate for use as an on-line algorithm.

2.2 Two-Dimensional Routing

In order to get a running time of $n + o(n)$, we will modify the above algorithm in such a way that the routing schedule can be computed on-line in time $o(n)$. Executing the computed schedule will then take another $n + o(n)$ steps. The key idea in our construction is a technique to reduce the size of the scheduling problem that has to be solved, and thus the size and number of the matchings that have to be computed. Informally speaking, this can be 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.

We partition the bused mesh 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 assume that the blocks B_i are indexed in row-major order (thus, B_0 and $B_{n^{2-2\alpha}-1}$ are the blocks in the upper left and lower right corner, respectively). 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 has exclusive ownership of its n^α column buses, while each resource R_j consists of n^α row buses. At most one process will be allowed to access a single resource at any point in 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 show how 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 will be computed. Rather than requiring each packet to be at its final destination after execution of the schedule, we will be content with routing each packet to some position in the $n^\alpha \times n^\alpha$ block that contains its destination. 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 the index of their destination block. 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 will be 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 will be transmitted across the row buses to their respective destination blocks in d separate steps, where d is the number of distinct destination blocks that occur among the packets in the row. In other words, such a row will be treated in the same way as d separate rows; this increases the number of steps required to route this row by $d-1$. Since there are only $n^{2-2\alpha}$ blocks, this will increase the number of steps required to route the elements of a single block across the row buses by at most $n^{2-2\alpha}-1$. Consequently, the number of steps required to route all the elements of a process P_i across the row buses will be 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

$$\sum_{j=0}^{n-1} D_{i,j} \leq n + n^{3-3\alpha} \quad \text{and} \quad \sum_{i=0}^{n-1} D_{i,j} \leq n + n^{3-3\alpha} \qquad (2)$$

hold for all i, j. Here, the second inequality holds 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. Equation (2) guarantees the existence of a schedule of length $n + n^{3-3\alpha} = n + o(n)$ 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 will have only $2n^{1-\alpha}$ vertices. Hence, a maximum matching in this graph can be computed in time $O\left(n^{2.5(1-\alpha)}\right)$. For each matching that is computed, at least one edge will be 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 $O\left(n^{4.5(1-\alpha)}\right) = o(n)$. In order to implement this computation on a bused mesh, all the data needed to construct the graph G is routed 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, $0 \leq i < n^{2-2\alpha}$, where m_i is defined as the number of elements in the block that are destined to block B_i. This can clearly be done in time $o(n)$, since not a lot of information has to be transmitted. We will not elaborate 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. Altogether, 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 $O(n^\alpha) = o(n)$ steps.

(2) In each block, use prefix computations to compute the m_i, $0 \leq i < n^{2-2\alpha}$ (m_i was defined as the number of packets with destination block B_i). Send the m_i to a small area in the center of the mesh. This takes $o(n)$ steps.

(3) Compute the schedule and broadcast it to all blocks of the mesh. This can be done in time $o(n)$, assuming that the constant α is chosen close enough to 1.

(4) Execute the computed schedule of length $n + o(n)$.

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

It remains to show that the above algorithm can be implemented with a small, constant queue size. Consider any destination block B_i inside the mesh, 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 will only receive 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 will only get a single packet, while up to $2n^{2-\alpha}$ processors will receive two packets. In addition, every processors can still contain one packet that originated from B_i and has not been sent out yet. Finally, some of the processors in B_i will also have to store the n^α packets that can enter the block across the column buses in each step, and that are then routed across the row buses in the following step. This gives a total queue size of 4. Using a slightly more complicated implementation [20], the following result can be stablished.

Theorem 2.1 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 2.

Note that the above algorithm does not assume any particular model of the mesh with row and column buses. In fact, the algorithm can be implemented on a variety of different models within the same bounds on running time and queue size. For the mesh with fixed buses, this improves upon the best previously known deterministic algorithm [11] in both running time and queue size (as an example, that algorithm required a queue size of more than 200 to obtain a running time of $1.2n$). On the mesh with reconfigurable buses, our algorithm even improves upon the best previously known randomized algorithms of Rajasekaran and McKendall [16]. On the PARBUS model, which does not allow bidirectional communication in subbuses of length one, our algorithm is optimal within an additive lower order term. For another example, consider a model of the bused mesh in which the buses have a non-unit propagation delay $\rho(n)$. It was observed by Cheung and Lau [3] that, for any nonconstant delay function ρ, routing will take 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 an interesting corollary, this implies an $n + o(n)$ algorithm for permutation routing on the *Mesh of Trees* [10]. Finally, our algorithm can be easily adapted to the *Polymorphic Torus* network described in [13] (this network is essentially a mesh with reconfigurable row and column buses and additional wrap-around connections). The resulting algorithm routes any permutation in time $n/2 + o(n)$, and thus nearly matches the bisection lower bound of $n/2$.

The above result shows that for the problem of permutation routing, even a fairly simple algorithm on the mesh with buses can achieve a speed-up by a factor of 2 over meshes without buses. For partial permutations with fewer packets, an even greater speed-up over the standard mesh can be achieved [20]. Moreover, our algorithm achieves a queue size of 2. 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 [1] achieves a queue size of 1 only because in the standard mesh model two packets can be exchanged across an edge in a single step. Since we do not allow two arbitrary processors that are connected to a common bus to exchange two packets in a single step, it seems unlikely that any algorithm that uses buses to transmit packets can achieve a queue size of 1.

2.3 Multi-Dimensional Routing

In this subsection, we apply the ideas from the previous subsection to obtain an improved deterministic algorithm for routing on multi-dimensional meshes with buses. On an r-dimensional network with side length n, our algorithm achieves a running time of $(2-\frac{1}{r})n+o(n)$ and a queue size of 2. This bound even holds if the dimension of the mesh is non-constant, provided that the side length n is sufficiently larger than the dimension r. In the following, we will only give a very brief description of the main ideas underlying our algorithm; more details can be found in [20].

The algorithm is based on a well-known scheme for off-line routing on r-dimensional meshes described by Annexstein and Baumslag [1]. The routing scheme consists of $2r-1$ phases. In phase i, $1 \leq i \leq r-1$, each packet is routed along dimension i to an appropriately chosen intermediate location. In phase i, $r \leq i \leq 2r-1$, each packet is greedily routed along dimension $2r-i$. Each phase of the routing scheme involves a collection of routing problems on linear arrays of length n, and will thus take at most n steps on the standard mesh. Hence, the entire routing will be completed after $(2r-1)n$ steps. Clearly, this bound can also be achieved on meshes with buses.

In order 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 $r-1$ phases. This can again be done by constructing a sequence of perfect matchings in a graph. The running time of these matching computations is polynomial in the number of packets in the network n^r. To convert this off-line routing scheme into a fast 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 $(2r-1)n+o(n)$ is *uni-axial*, that is, the algorithm communicates only across a single dimension in any given step of the computation. Thus, we could simultaneously run up to r "copies" of the algorithm without any contention for the buses. To make use of this observation, we partition the set of packets into r 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 $\frac{1}{r}$ fraction of all packets, it can be routed in time $(2-\frac{1}{r})n+o(n)$ by the above algorithm. By running r "copies" of the algorithm simultaneously, we obtain the following result.

Theorem 2.2 There exists a deterministic algorithm for routing on r-dimensional meshes with buses that runs in time $(2-\frac{1}{r})n+o(n)$ with a queue size of 2.

Like the algorithm in the previous subsection, this algorithm can be implemented on a variety of different models of meshes with buses. For the mesh with fixed buses, we obtain a significant improvement over the best previously known deterministic algorithm [11] with respect to both running time and queue size. Our algorithm matches the running time of the randomized algorithms of Sibeyn, Kaufmann, and Raman [17], while achieving a slightly better queue size. The algorithm can easily be adapted to the multi-dimensional variants of the other networks mentioned in the previous subsection.

2.4 Sorting on Meshes with Buses

The routing algorithms presented in the previous subsections can also be used to design fast deterministic algorithms for sorting on meshes with buses. The idea for the algorithms is very simple. First, we use a deterministic sampling technique to compute a set of splitters. After the splitter set has been broadcast throughout the network, each element can compute its approximate rank, and hence its approximate final location in the sorted order. Next, we use a routing algorithm to move each element to this location. Finally, local sorting can be used to bring each element to its final destination. A more detailed description of the algorithm and the deterministic sampling technique can be found in [20]. Ee get the following result.

Theorem 2.3 Deterministic sorting can be performed in time $n+o(n)$ on the two-dimensional mesh with buses, and in time $(2-\frac{1}{r})n+o(n)$ on the r-dimensional mesh with buses.

3 Dynamic Routing Problems

Previous work on routing algorithms for meshes with buses was restricted to the case of routing problems in which every packet is already present at the beginning of the routing, and the algorithm terminates after all packets have been delivered. However, in many real applications new packet routing requests are constantly generated by the processors throughout an ongoing computation. In this section, we study the performance of different models of meshes with buses on such *dynamic routing problems.*

Following the framework given by Leighton [10, Section 1.7.2], we assume that in a *dynamic routing problem*, each processor generates a new packet at each step with some fixed probability λ, also called the *arrival rate*. The destination addresses of these newly generated packets are chosen randomly from among the processors of the network. Our goal is to design routing algorithms that deliver each generated packet to its destination within some number τ of steps, with probability at least $1-O(1/n^2)$ (in the following referred to as *high probability*). We will assume that the performance of an algorithm on a dynamic routing problem is characterized by this time bound τ, and by the *network capacity* λ_0, that is, the maximal arrival rate λ that can be handled by the algorithm.

Note that for two-dimensional meshes and related networks, including the networks considered in this paper, there is an upper bound of $O(1/n)$ on the capacity of the network. The exact bound on the capacity depends on the specific properties of the network. As an example, on the standard mesh the maximal arrival rate is bounded by $4/n$ [10], while on meshes with fixed buses, λ can be at most $1/n$ if we want to make use of the buses for routing packets over long distances.

In [10, Section 1.7.2], Leighton investigates the dynamic routing problem on the standard mesh under the greedy routing scheme, in which each packet is first routed along the row to the correct column, and then along the column to its destination, and priority is given to the packet with the farthest distance to travel. It is shown that in any time interval of length T and for any arrival rate less than $4/n$, no packet is delayed by more than $O(\lg T + \lg n)$ steps, and no queue grows beyond size $O(1+\frac{\lg T}{\lg n})$, with high probability. While this result shows that greedy algorithms perform well on dynamic routing problems, the expected time for the completion of a routing request is, of course, still $\Theta(n)$, due to distance arguments. In the following, we will show that the maximal delivery time for a dynamic routing request can be significantly reduced by adding buses to the network.

In the greedy routing scheme considered by Leighton, a newly generated packet starts moving towards its destination as soon as it can do so under the given priority scheme. We will call such a routing scheme *dynamic*. Another possible way of solving a dynamic routing problem is to partition the computation time into intervals $[t_i, t_{i+1}]$, $i \geq 0$, of length k, and delay all packets generated in the interval $[t_i, t_{i+1}]$ until a new round of routing is started at time t_{i+1}. We will call such a routing scheme *static*. It is pointed out in [10, page 173] that such a scheme is not a good choice for networks with large diameter, such as the mesh. Since the delivery time of a packet in these networks is mainly determined by the distance it has to travel, it is a better idea to move the packet closer to its destination whenever this is possible. However, this situation is quite different on meshes with buses, where the delivery time is not determined by the distance. It will turn out that the main issue raised by dynamic routing problems on these networks is that of possible contention for the buses. In the following, we will propose static and dynamic algorithms for several different models of meshes with buses.

3.1 Meshes with Fixed Buses

We assume that new packets are generated by the processors with rate $\lambda = k/n$, for some $k < 1$. Thus, we expect about kn packets to be generated in any step. However, the generated packets will not be completely evenly distributed among the rows and columns of the mesh. This raises the problem of scheduling the buses in such a way that no two processors simultaneously attempt to write on the same bus.

We partition the mesh into blocks of size $n^{1/3} \times n^{1/3}$, and partition the computation time into intervals of length $\Theta(n^{1/3} \lg n)$. Then in each interval, and in each block, $\Theta(\lg n)$ packets will be generated, with high probability. On the other hand, during each interval, $\Theta(n^{2/3} \lg n)$ bus rides are available on the $n^{1/3}$ row buses passing through the block. Thus, we could use a fixed schedule that assigns to each of the $n^{2/3}$ blocks in a row $\Theta(\lg n)$ bus rides per interval. A newly generated packet can now walk in time $O(n^{1/3})$ to a unique "bus stop" located within its block, where it waits for a slot on the bus. The process is then repeated for the column buses. The resulting dynamic algorithm routes each packet in time $O(n^{1/3} \lg n)$, and no packet takes more than $O(\lg T + n^{1/3} \lg n)$ steps in any time interval of length T, with high probability.

We can improve the running time in the static case by using a prefix computation to determine the number of packets generated in each block. We can then assign an appropriate number of bus rides to each block. Since prefix computations can be performed in $O(n^{1/3})$ steps on a mesh with fixed CREW buses, we obtain a static routing algorithm that routes all packets in time $O(n^{1/3})$, with high probability.

3.2 Reconfigurable Meshes

We now consider the dynamic routing problem on meshes with reconfigurable CREW buses. It turns out that the problem of scheduling the buses becomes much simpler under this model, since we can avoid write conflicts by reconfiguring the buses appropriately. Consider a row of the mesh with a number of packets that want to move in a common direction, say towards the right. If any processor that contains such a packet disconnects the row bus between itself and its left neighbor, then no write conflict can occur when the packets are broadcast towards the right in the following step. This observation leads to the following simple routing scheme.

As before, every packet is first routed along the row to its destination column, and then along the column to its final destination. We divide the routing into odd and even steps. In an odd step, we route all packets that have to be moved to the right. To do this, we disconnect the row bus to the left of each packet and broadcast the packet to the right. Similarly, we route all packets that have to move downwards on the column buses. In an even step, we route the packets that have to move upwards or towards the left.

The resulting dynamic algorithm routes each packet in $O(\lg n)$ steps, and no packet takes more than $O(\lg T + \lg n)$ steps in any time interval of length T, with high probability. To prove this, observe that in every step, either at least one element in each row reaches its destination column, or no element at all is routed in that row. A corresponding statement holds for the columns.

Note that the above scheme is very similar to the greedy routing scheme on the standard mesh. We can also use the scheme in the case of buses with non-unit delay. Interestingly, as the delay function increases, the number of packets currently in the network will increase, and the resulting behavior of the algorithm will eventually become more and more similar to the greedy algorithm for the standard mesh.

3.3 Meshes with CRCW Buses

Up to this point, we have restricted our attention to meshes with CREW buses. The main reason for this restriction was that there was no apparent benefit in allowing concurrent write access to the buses in permutation routing and sorting. This situation appears to be different, however, in the case of dynamic routing problems on meshes with fixed buses. In the following, we assume that no packet will be transmitted at all if two or more processors try to write to the same bus in a single step. Instead, the bus will broadcast a special signal indicating that a write conflict has occurred.

A naive routing scheme would require every packet to flip a coin before each step, and attempt to write on the bus if and only if the outcome of the coin flip is a "1". If every row and every column contains only a constant number of packets, then this scheme will route a packet across each bus in a constant fraction of the steps. However, if a row or column receives a larger number of packets, then this routing scheme will very likely result in a write conflict in any given step, and eventually in a large backlog of undelivered packets. To avoid this problem, a more intricate coin-flipping strategy is needed.

We partition the computation time into intervals of length $O(\lg n \lg\lg n)$. Then in every such interval $\Theta(\lg n \lg\lg n)$ packets will be generated in each row. To route these packets across the row buses to their destination column, we can use an algorithm recently analyzed by Geréb-Graus and Tsantilas [4] in the context of routing h-relations on an optical computer model. The algorithm uses appropriately biased coin-flips to route $O(\lg n \lg\lg n)$ packets in time $O(\lg n \lg\lg n)$, with high probability. For a description of the algorithm, we refer the reader to the cited paper. The algorithm is then repeated for the routing across the columns.

The above algorithm can be implemented both as a static and as a dynamic scheme. However, it seems to have one serious drawback. In the unlikely event that too many packets are generated in the same row over some sufficiently long period of time, the network will not return to its "normal" state even after the arrival rate has gone back to normal.

3.4 Discussion

We have observed in this section that the delivery time for a dynamic routing request is considerably smaller on meshes with buses than on the standard mesh. Of course, it needs to be pointed out that an improvement in the expected routing time from $\Theta(n)$ to, say, $O(\lg n)$ will in general not result in a proportional speed-up of the underlying computation performed by the network. The maximal speed-up that can be achieved by adding a bus system will depend on the properties of the particular application. If on average only a small number of routing requests are generated in a single step, then a significant speed-up is possible. For larger arrival rates, some constant speed-up can still be obtained in many cases.

If most of the generated packets only have to travel a short distance, then the performance of a network with fixed buses can be improved by reserving the buses for the few packets that have to travel over a long distance, and routing the other packets on the mesh edges. Of course, this is already done in many algorithms, such as the $O(n^{1/3})$ prefix sums algorithm, but it might be interesting to investigate this idea under the more general framework of dynamic routing.

4 Summary and Open Problems

In this paper, we have given improved deterministic algorithms for routing and sorting on meshes with row and column buses. The algorithms can be implemented on a variety of different models of meshes with buses, and are based on a new technique that seems especially suitable for networks with a large diameter that have been augmented with a bus system or a small-bandwidth interconnection network. We have also investigated the performance of various models of meshes with buses on dynamic routing problems.

While our algorithms for permutation routing and sorting are optimal for some models of meshes with buses, for example the PARBUS or the *Polymorphic Torus*, there is still a gap between the best upper and lower bounds on the mesh with fixed and reconfigurable buses. It would certainly be an interesting improvement to close these gaps. For the two-dimensional mesh with reconfigurable buses, it is yet unclear whether routing can be done in less than n steps. For the mesh with fixed buses, there remains a small gap between the best lower bounds and the running times of the randomized algorithms of Sibeyn, Kaufmann, and Raman [17]. By applying a new and fairly general derandomization technique for routing and sorting on meshes [8], we have very recently obtained deterministic routing algorithms that match the running times of their randomized algorithms. The resulting algorithms will be described in a subsequent paper.

While the algorithms given in Section 2 are based on a fairly simple idea, they are not practical due to their complicated control structure and their fairly large lower order terms. It is an interesting question whether the ideas described in this paper can be used in the design of more practical algorithms.

Finally, we believe that the study of dynamic routing problems on meshes with buses deserves further attention. For example, one could try to show lower bounds for the cases of meshes with fixed CREW and CRCW buses. A more precise analysis of the proposed dynamic routing schemes would also be of interest.

Acknowledgements

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

References

[1] F. Annexstein and M. Baumslag. A unified approach to off-line permutation routing on parallel networks. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 398–406, July 1990.

[2] S. H. Bokhari. Finding maximum on an array processor with a global bus. *IEEE Transactions on Computers*, 33:133–139, 1984.

[3] S. Cheung and F. C. M. Lau. A lower bound for permutation routing on two-dimensional bused meshes. *Information Processing Letters*, 45:225–228, 1993.

[4] M. Geréb-Graus and T. Tsantilas. Efficient optical communication in parallel computers. In *Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 41–48, June 1992.

[5] T. Gonzalez and S. Sahni. Open shop scheduling to minimize finish time. *Journal of the ACM*, 23:665–679, 1976.

[6] J. E. Hopcroft and R. M. Karp. An $n^{5/2}$ algorithm for maximum matchings in bipartite graphs. *SIAM Journal on Computing*, 2:225–231, 1973.

[7] H. F. Jordan. A special purpose architecture for finite element analysis. In *International Conference on Parallel Processing*, pages 263–266, 1978.

[8] M. Kaufmann, J. Sibeyn, and T. Suel. Derandomizing algorithms for routing and sorting on meshes. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1994. To appear.

[9] M. Kunde. Block gossiping on grids and tori: Deterministic sorting and routing match the bisection bound. In *Proceedings of the 1st Annual European Symposium on Algorithms*, September 1993.

[10] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees and Hypercubes*. Morgan-Kaufmann, San Mateo, CA, 1991.

[11] J. Y. Leung and S. Shende. Packet routing on square meshes with row and column buses. In *Proceedings of the 3rd Annual IEEE Symposium on Parallel and Distributed Processing*, pages 834–837, December 1991.

[12] J. Y. Leung and S. Shende. On multi-dimensional packet routing for meshes with buses. Technical Report TR-150, Department of Computer Science and Engineering, University of Nebraska at Lincoln, 1992.

[13] H. Li and Q. F. Stout. *Reconfigurable Massively Parallel Computers*. Prentice Hall, Englewood Cliffs, New Jersey, 1991.

[14] R. Miller, V. K. Prasanna Kumar, D. I. Reisis, and Q. F. Stout. Parallel computations on reconfigurable meshes. *IEEE Transactions on Computers*, 42:678–692, June 1993.

[15] S. Rajasekaran. Mesh-connected computers with fixed and reconfigurable buses: Packet routing, sorting, and selection. In *Proceedings of the 1st Annual European Symposium on Algorithms*, September 1993.

[16] 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.

[17] J. Sibeyn, M. Kaufmann, and R. Raman. Randomized routing on meshes with buses. In *Proceedings of the 1st Annual European Symposium on Algorithms*, September 1993.

[18] G. D. Stamoulis and J. N. Tsitsiklis. The efficiency of greedy routing in hypercubes and butterflies. In *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 248–259, July 1991.

[19] Q. F. Stout. Mesh-connected computers with broadcasting. *IEEE Transactions on Computers*, 32:826–830, 1983.

[20] T. Suel. Routing and sorting on meshes with row and column buses. Technical report, Department of Computer Sciences, University of Texas at Austin, February 1994.

[21] 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 Transactions on Parallel and Distributed Systems*, 1:500–507, 1990.