[40] 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*, pages 243–247, 1985.

[41] V. K. Prasanna Kumar and C. S. Raghavendra. Array processors with multiple broadcasting. *Journal of Parallel and Distributed Computing*, 4:173–190, 1987.

[42] 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.

[43] 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.

[44] S. Rajasekaran and R. Overholt. Constant queue routing on a mesh. *Journal of Parallel and Distributed Computing*, 15:160–166, 1992.

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

[46] 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.

[47] 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.

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

[49] Q. F. Stout. Meshes with multiple buses. In *Proceedings of the 27th Annual IEEE Symposium on Foundations of Computer Science*, pages 264–273, 1986.

[50] T. Suel. Routing and sorting on meshes with row and column buses. In *Proceedings of the 8th International Parallel Processing Symposium*, April 1994.

[51] 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.

[27] 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.

[28] R. E. Ladner, J. Lampe, and R. Rogers. Vector prefix addition on sub-bus mesh computers. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 387–396, June 1993.

[29] F. T. Leighton. Tight bounds on the complexity of parallel sorting. *IEEE Transactions on Computers*, C–34:344–354, 1985.

[30] F. T. Leighton. Average case analysis of greedy routing algorithms on arrays. In *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 2–10, July 1990.

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

[32] F. T. Leighton, F. Makedon, and I. G. Tollis. A $2n-2$ step algorithm for routing in an $n \times n$ array with constant queue sizes. In *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 328–335, July 1989.

[33] 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.

[34] 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.

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

[36] C. U. Martel and T. P. Vayda. The complexity of selection resolution, conflict resolution, and maximum finding on multiple access channels. In *Proceedings of the 3rd International Workshop on Parallel Computation and VLSI Theory*, pages 401–410, 1988.

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

[38] 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.

[39] M. Mitzenmacher. Bounds on the greedy routing algorithm for array networks. In *Proceedings of the 6th Annual ACM Symposium on Parallel Algorithms and Architectures*, June 1994. To appear.

[14] A. G. Greenberg and S. Winograd. A lower bound on the time needed in the worst case to resolve conflicts deterministically in multiple access channels. *JACM*, 32:589–596, 1985.

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

[16] E. Hao, P. D. MacKenzie, and Q. F. Stout. Selection on the reconfigurable mesh. In *Proceedings of the 4th IEEE Symposium on the Frontiers of Massively Parallel Computations*, 1992.

[17] M. Harchol-Balter and P. E. Black. Queueing analysis of oblivious packet-routing networks. In *Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 583–592, January 1994.

[18] M. C. Herbordt, J. C. Corbett, C. C. Weems, and J. Spalding. Practical algorithms for online routing on fixed and reconfigurable meshes. *Journal of Parallel and Distributed Computing*, 20:341–356, 1994.

[19] 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.

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

[21] 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.

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

[23] N. Kahale and F. T. Leighton. Greedy routing on arrays. Unpublished manuscript, 1993.

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

[25] 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*, pages 669–679, January 1994.

[26] J. Komlós and A. G. Greenberg. An asymptotically nonadaptive algorithm for conflict resolution in multiple-access channels. *IEEE Transactions on Information Theory*, IT–31:302–306, 1985.

References

[1] A. Aggarwal. Optimal bounds for finding maximum on arrays of processors with k global buses. *IEEE Transactions on Computers*, 35:62–64, 1986.

[2] 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.

[3] A. Bar-Noy and D. Peleg. Square meshes are not always optimal. In *Proceedings of the 1st Annual ACM Symposium on Parallel Algorithms and Architectures*, pages 138–147, July 1989.

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

[5] 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. *Journal of Parallel and Distributed Computing*, 9:79–84, 1992.

[6] 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 Transactions on Parallel and Distributed Systems*, 1:241–245, 1990.

[7] 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.

[8] B. S. Chlebus, M. Kaufmann, and J. F. Sibeyn. Deterministic permutation routing on meshes. In *Proceedings of the 5th Annual IEEE Symposium on Parallel and Distributed Processing*, pages 614–621, December 1993.

[9] R. Cole and C. K. Yap. A parallel median algorithm. *IPL*, 20:137–139, 1985.

[10] 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.

[11] 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.

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

[13] A. G. Greenberg, P. Flajolet, and R. E. Ladner. Estimating the multiplicities of conflicts to speed their resolution in multiple access channels. *JACM*, 34:289–325, 1987.

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 still remains a gap between the known lower bounds and the upper bounds given by the randomized algorithms of Sibeyn, Kaufmann, and Raman [46]. (By applying a new "derandomization" technique for routing and sorting algorithms on meshes proposed in [25], it is possible to obtain deterministic routing algorithms that match the running times of these randomized algorithms, within a lower order additive term.)

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

Another possible research direction is to find efficient algorithms for local routing, or for the routing of sparse permutations. In this context, the buses might be very helpful in the design of algorithms that adapt to the degree of "locality" or "sparsity" of a problem.

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, or give upper bounds for the expected completion time of a routing request (as opposed to the high-probability bounds given in this paper). An analysis for other classes of cummunication patterns (e.g., random permutations with some degree of "locality") would also be of interest.

Acknowledgements

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

of the interval and apply the protocol to the remaining packets and the packets that arrived during the unsuccessful execution of the protocol. The same protocol is executed for each column bus. It can be shown that in any interval of length T, no packet takes more than $O(\lg T + \lg n \lg\lg n)$ steps, and no queue grows beyond $O(1 + \frac{\lg T}{\lg n})$, with high probability.

Using similar ideas we can obtain a dynamic algorithm with a delivery time of $O(\lg n \cdot (\lg T + \lg\lg n))$. The maximal possible arrival rate for these algorithms is $\Theta(1/n)$; the precise term depends on the performance of the protocol of Geréb-Graus and Tsantilas. In the case of reconfigurable buses, we do not know whether write conflict resolution can be used to improve on the bounds obtained in the previous subsection.

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. In particular, a faster delivery time may in many cases result in an increase in the arrival rate for new packets, which in turn will slow down the communication in the network. The maximal speed-up that can be achieved by adding a bus system depends 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 we can assume that 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. On meshes with reconfigurable buses, the routing scheme described in Subsection 3.2 will naturally take some advantage of locality.

4 Summary and Open Problems

In this paper, we have given improved deterministic algorithms for routing and sorting on meshes with buses. The algorithms can be implemented on a variety of different classes of networks, 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 low-bandwidth interconnection network of small diameter. 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 buses and the mesh with reconfigurable buses. It would certainly be an interesting improvement to close these by observing that in every step, either at least one element (the rightmost) in each row reaches its destination column, or no element at all is routed in that row. (A corresponding statement holds for the columns.) Using a more involved analysis, the result can be extended to the dynamic case and to all $\lambda < 2/n$.

The above routing algorithm was also recently studied by Herbordt, Corbett, Weems, and Spalding [18], who give experimental results for several important classes of permutations. While they do not perform a theoretical analysis in a dynamic setting, their results indicate that the algorithm performs well on sparse random permutations.

Note that the algorithm is quite similar to the greedy routing scheme on the standard mesh, in that we greedily move a packet along row-column paths towards its destination whenever possible. 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 increases, and the resulting behavior of the algorithm eventually becomes 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 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 is transmitted at all if two or more processors try to write to the same bus in a single step. Instead, the bus broadcasts a special signal indicating that a write conflict has occurred. We point out that this situation is similar to that encountered in the context of Ethernet-like multiple-access channels [13, 14, 26, 36].

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 routes 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 clever coin-flipping strategy is needed.

In our routing scheme, we will use a contention resolution protocol recently analyzed by Geréb-Graus and Tsantilas [11] in the context of routing h-relations on an optical computer model. The protocol uses appropriately biased coin-flips to route an h–h relation in time $\Theta(h + \lg n \lg h)$, with high probability (see [11] for a more detailed description). In the context of CRCW buses, the protocol can be used to route up to h packets in time $\Theta(h + \lg n \lg h)$ across a common bus.

To obtain a non-dynamic algorithm, 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 are generated in each row, and can be routed across the row bus using the above protocol, with high probability. In the case that not all packets are routed by the protocol, we double the length

3.1 Meshes with Fixed Buses

In the following, 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 and their destinations are not completely evenly distributed among the rows and columns of the mesh. In fact, it is likely that some rows and some columns receive up to $\Theta(\frac{\lg n}{\lg \lg n})$ packets. 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 $n^{1/3} \times n^{1/3}$. Then in each time interval of length $\Theta(t \cdot n^{1/3})$ and in each block, $O(t + \lg n)$ packets are generated, with high probability. On the other hand, during each interval, $\Theta(t \cdot n^{2/3})$ bus rides are available on the $n^{1/3}$ row buses passing through the block. Thus, we could "multiplex" the $n^{1/3}$ buses among the $n^{2/3}$ blocks in a row in such a way that each block "receives" one bus ride in any $n^{1/3}$ consecutive steps. Every 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. It can be shown that in any time interval of length T, no packet takes more than $O(\lg T + n^{1/3} \lg n)$ steps, and no queue grows beyond size $O(1 + \frac{\lg T}{\lg n})$, with high probability.

A non-dynamic algorithm with a delivery time of $O(\lg T + n^{1/3})$ can be obtained by using a prefix computation to determine the number of packets generated in each block within the last time interval. 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 can choose the intervals to be of length of $O(n^{1/3})$.

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.

For the resulting algorithm, it can be shown that in any time interval of length T, no packet takes more than $O(\lg T + \lg n)$ steps, and no queue grows beyond size $O(1 + \frac{\lg T}{\lg n})$, with high probability. For the non-dynamic algorithm and $\lambda < 1/n$, this can be easily seen have to cross the bisection of the network would be $\omega(n)$. Since only $O(n)$ packets can cross in any step, the expected time needed to deliver a packet would grow without bound as the computation proceeds. 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 $4/n$, while on meshes with fixed buses, λ can be at most $1/n$ if we want to make use of the buses to route packets over long distances.

In [30, 31], 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 packets with the longest distance to travel. He shows 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. (These results have recently been generalized and extended by Harchol-Balter and Black [17], Kahale and Leighton [23], and Mitzenmacher [39].) While this shows that greedy algorithms perform well on dynamic routing problems, it should be noted that the expected time for the completion of a routing request is, of course, still $\Theta(n)$, due to distance arguments.

In the following, we show that the maximal delivery time for a dynamic routing request can be significantly reduced by adding buses to the network, while the capacity of the network is only affected by at most a constant factor. Note that this is in contrast to the case of permutation routing, where adding buses to the network results in a speed-up by at most a constant factor.

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 call such a routing scheme *dynamic*. Another possible way of solving a dynamic routing problem is to partition the computation time into intervals $[t_i, t_{i+1}]$, $i \geq 0$, 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}. (Note that this is related to the problem of routing a sparse random permutation.) We call such a routing scheme *non-dynamic*. It is pointed out in [31, 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 a 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 so much determined by the distance between source and destination, but rather by the performance of the mechanism for resolving possible contention for the buses.

In the following, we propose dynamic and non-dynamic algorithms for several different models of meshes with buses. In the next subsection we give algorithms for dynamic routing on meshes with fixed row and column buses. In Subsection 3.2 we consider the case of meshes with reconfigurable buses. Subsection 3.3 studies meshes with CRCW buses. Finally, Subsection 3.4 contains a brief discussion of our results.

Lemma 2.1 Let S be the sample set of size $n^{2-\delta}$ chosen in Phase (ii). Then for any $s \in S$ with Rank $(s, S) = i$ we have

$$(i - n^{2-2\delta}) \cdot n^{\delta} < \text{Rank}\,(s, X) < i \cdot n^{\delta}.$$

The proof of the above lemma is omitted. The next lemma establishes that the splitter elements used in our algorithm have the desired properties. It can be proved by a simple application of Lemma 2.1.

Lemma 2.2 Let D be the splitter set of size n^{δ} selected in Step (iii). Then D is a set of "good" splitters, that is, it satisfies conditions (1) and (2) stated above.

Thus, we get the following result for sorting on two-dimensional meshes with buses.

Theorem 2.4 There exists a deterministic algorithm for sorting on the two-dimensional mesh with buses running in time $n + o(n)$ with queue size 2.

As before, the algorithm can be implemented on several different models of meshes with buses. It can also easily be adapted to networks of higher dimension. By using the routing algorithm for r-dimensional meshes with buses described in Subsection 2.3, we obtain a deterministic algorithm for sorting on the r-dimensional mesh with buses with a running time of $(2 - 1/r)n + o(n)$ and a queue size of 2.

3 Dynamic Routing Problems

Previous work on routing algorithms for meshes with buses was restricted to the case of static 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 [30, 31], we assume that in a *dynamic routing problem*, each processor generates a new packet at each step with some fixed probability λ called the *arrival rate*. The destination addresses of these newly generated packets are chosen randomly from the set of all processors. Our goal is to design routing algorithms that deliver every 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 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. The network capacity is the maximal arrival rate λ that can be handled by the algorithm.

Note that for two-dimensional meshes and related networks, including the classes of networks considered in this paper, there is an upper bound of $O(1/n)$ on the capacity of the network. If $\lambda = \omega(1/n)$, then the expected number of packets generated in a single step that

(4) Broadcast the splitter elements to all processors in the mesh. This can be done in time $O(n^\delta)$.

(5) It is shown in Lemma 2.2 that the ith splitter element has a rank of approximately $i \cdot n^{2-\delta}$, up to $O(n^{2-\delta})$. Hence, every element in the mesh can now determine its rank to within $O(n^{2-\delta})$ positions. We partition the mesh into blocks of size $n^\alpha \times n^\alpha$, for some α close to 1 and larger than $1 - \delta/2$. Every element selects as its destination block the block containing the processor corresponding to its approximate rank.

(6) Route every element to its destination block using the routing algorithm in Subsection 2.2.

(7) Compute the exact ranks of the splitter elements using prefix computations, and broadcast these ranks to all processors in the mesh. This can be done in time $O(n^\delta)$.

(8) Use local sorting and routing to move all elements to their final destinations. This takes time $O(n^\alpha)$.

Apart from Step (5), all steps of the above algorithm take time $o(n)$. Thus, the running time of the algorithm is determined by the running time of the routing algorithm used in Step (6), up to a lower order term. To establish the correctness of the algorithm, we have to show that the splitter elements selected in Step (3) have the properties claimed in Step (5).

The computation of the splitter elements in the above algorithm is essentially a simplified version of a more sophisticated sampling technique used in the parallel selection algorithm of Cole and Yap [9]. The goal is to deterministically select a set of "good" splitters from a set of keys X of cardinality n^2. More precisely, we are interested in selecting a set of $t \stackrel{\text{def}}{=} n^\delta$ splitter elements $D = \{d_0, \ldots, d_{t-1}\}$ with $d_{i+1} > d_i$, such that the following properties hold for all i:

(1) $\text{Rank}\,(d_{i+1}, X) - \text{Rank}\,(d_i, X) \leq \frac{2n^2}{t}$

(2) $\frac{(i-1)n^2}{t} + 1 \leq \text{Rank}\,(d_i, X) \leq \frac{in^2}{t} + 1$

Ignoring the details of the implementation on the mesh, the sampling technique used in Steps (1) to (3) of the above algorithm proceeds in the following three phases:

(i) Partition the set X into $n^{2-2\delta}$ subsets X_i of size $n^{2\delta}$, for some δ with $0 < \delta < 1$.

(ii) Sort each subset, and select a sample set S by choosing n^δ equidistant elements from each sorted block, starting with the smallest element and going up to the (n^δ)th largest element.

(iii) Sort the sample set, and select a splitter set of size n^δ by choosing every $s \in S$ with $\text{Rank}\,(s, S) = i \cdot n^{2-2\delta} + 1$, for some non-negative integer i.

One important detail has been omitted from the description so far. Before running the protocol in Steps (2) and (4), 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 any step, and such that no write conflicts occur. This can be done during Steps (1) and (3) using local sorting and prefix computations, in time $O(n^{2/3})$. This establishes the following result.

Theorem 2.3 There exists a deterministic algorithm for permutation routing on the $n \times n$ mesh with buses that runs in time $n + O(n^{2/3})$ with constant queue size and that uses only prefix computations and local sorting as subroutines.

The above algorithm achieves a queue size of 4; this can be reduced to 2 by a more careful (but also more involved) implementation. The algorithm can again be implemented on several different models of meshes with buses. Of course, it needs to be pointed out that the algorithm is still too complicated to be of immediate practical interest. However, we believe that the result is interesting in that it indicates that even very simple global operations such as prefix computations might be useful in the design of efficient routing algorithms on meshes with buses. In contrast, all 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 such 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.

2.5 Sorting on Meshes with Buses

In this subsection we give algorithms for sorting on meshes with buses that match the running time of our routing algorithms, within a lower order additive term. The algorithms are based on a deterministic sampling technique that computes a set of splitter elements whose ranks are determined to within an additive lower order term. The high level structure of our sorting algorithm is as follows:

Algorithm SORT:

(1) Partition the mesh into blocks of size $n^\delta \times n^\delta$, where $1/2 < \delta < 1$, and sort each block into row-major order. This takes time $O(n^\delta)$ using, for example, the algorithm by Schnorr and Shamir [45].

(2) Route copies of the elements in the first column of each block to a block B of size $n^{1-\delta/2} \times n^{1-\delta/2}$ in the center of the mesh. This can be done in time $o(n)$ using the buses.

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

Let $\alpha = 2/3$, let $s_{i,j}$ denote the number of packets in the ith column of blocks whose destination is in the jth row of blocks, and let $b_{i,j} = \left\lfloor \frac{s_{i,j}}{n} \right\rfloor$. 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 $k+1$. (We assume that the row buses are idle during the first step of the protocol, and the column buses are idle during the last step.)

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.

Algorithm ROUTE2:

(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. This takes time $O(n^{2/3})$.

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

(3) Compute a new assignment of the buses with $b_{i,j} = n^{1/3}$ for all i, j. This takes time $O(n^{2/3})$.

(4) Run the protocol for another $n^{2/3}+1$ steps.

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

We now route every set of packets by first routing it within its sub-network to the correct destination block, and then within the destination block to its final position. We can simultaneously perform the routing in each of the sub-networks by running the above uni-axial algorithm in each sub-network. Due to their special structure, each of the r disjoint sub-networks is connected to all rn^{r-1} buses, and can use all n^{r-1} buses associated with a particular dimension in a single step. Since every sub-network only contains n^r/r packets, each of the $2r-1$ phases of the uni-axial algorithm only requires n/r steps. The queue size of this algorithm is 4. Using ideas similar to those in the previous subsection, the queue size can be reduced to 2. This gives us the following result.

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

The algorithm can again be implemented on several different models of meshes with buses. For the mesh with fixed buses, we obtain a significant improvement over the best previously known deterministic algorithm [33] with respect to both running time and queue size. Our algorithm nearly matches the running time of the randomized algorithms of Sibeyn, Kaufmann, and Raman [46], 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, but we are not aware of any prior results for those models. We can also obtain faster algorithms for routing sparse permutations.

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 additive lower order terms in the running times of the algorithms. 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 lower order 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. In particular, this 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 described as computing an approximate solution for a special case of the *Open Shop Scheduling Problem.*)

By combining a large number of packets into a single super-packet, we are able to decrease the number of packets in the network (and thus the number and size of the matchings that have to be computed) in such a way that the intermediate locations can be computed in time $o(n)$.

Formally, partition the mesh into r-dimensional blocks of side length n^α, for some α close to 1 (say $\alpha = 0.99$). Then sort the packets in each block according to their destination blocks, and combine up to $n^{r\beta}$ packets with a common destination block into a single super-packet, say for $\beta = 0.9$. Thus, the packets in a super-packet can be arranged in an r-dimensional submesh of side length n^β. In each block, we obtain at most $n^{r(\alpha-\beta)} + n^{r(1-\alpha)}$ super-packets. Hence, the number of super-packets in the entire mesh is $n^{r(1-\beta)} + o(n^{r(1-\beta)})$. We can assign to each super-packet a unique block of side length n^β inside the correct destination block, by running an appropriate prefix computation. We can now interpret the remaining problem as a $(o(n^{r(1-\beta)})$-approximate) permutation routing problem on an r-dimensional mesh with side length $n^{1-\beta}$, where each communication step takes time n^β (since it takes n^β steps to move all packets of a super-packet using $n^{(r-1)\beta}$ buses). Due to the small number of super-packets in this new routing problem, we can now compute the intermediate locations for the above routing scheme in time $o(n)$. This directly implies an on-line algorithm for routing on r-dimensional buses with a running time of $(2r-1)n + o(n)$.

An issue we have ignored in the above description is that by combining the packets into super-packets, we only get an approximate permutation and not a permutation in the strict sense. This problem can be easily overcome by, for example, first routing a (partial) permutation containing the vast majority of the packets with the above algorithm; the few remaining packets can then be routed in $o(n)$.

In the remainder of this subsection, we show how this algorithm can be modified to run in time $(2 - 1/r)n$. Note that the above algorithm only uses a small part of the available bandwidth, since at any point in time all communication is performed across a single dimension. In order to obtain an algorithm whose running time does not grow linearly with the dimension r, we have to make simultaneous use of all the buses in the network. The basic idea to achieve this is to partition the packets of the routing problem into r sets of packets. Each set of packets can then be routed in time $(2r-1)n/r + o(n)$ using the above algorithm. Since that algorithm uses only a single dimension in each time step, we can route all r sets of packets simultaneously without increasing the running time.

Formally, we partition the mesh into r sub-networks by assigning the label j to each processor with coordinates $(i_0, \ldots, i_{r-1})$ and $i_0 + \cdots + i_{r-1} = j \bmod r$. Next, we partition the packets of the routing problem into r sets by first sorting the packets in each block of side length n^α by destination blocks, as before, and then assigning each packet with rank j in the block to set $j \bmod r$. Note that in this way, for any destination block B, we have an approximately equal number of packets with destination in B in each of the r sets of packets. There are n^r/r packets in each set, and hence in each set there are approximately $n^{r\alpha}/r$ packets with destination block B. Next, we move the packets in set i, $0 \leq j < r$, to the sub-network consisting of the processors with label i, such that each processor receives exactly one packet. Note that all packets with common source and destination blocks are approximately evenly distributed among the r sets.

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-balanced 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 all $\epsilon > 0$.

Corollary 2.1.1 For any $\delta = o(n)$, there exists a deterministic algorithm that routes every δ-balanced ϵ-permutation in time $\epsilon n + o(n)$ with a queue size of 2. For general ϵ-permutations, a running time of $2n/\sqrt{\epsilon} + o(n)$ can be achieved.

2.3 Routing on Multi-Dimensional Meshes with Buses

In the following, we apply the techniques 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 $(2 - 1/r)n + o(n)$ 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.

Our algorithm is based on a well-known scheme for off-line routing on r-dimensional meshes described by Annexstein and Baumslag [2]. 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 thus takes at most n steps on the standard mesh. Hence, the entire routing is completed after $(2r-1)n$ steps. This bound can be matched on meshes with buses, even if only buses are used to route the packets. (On the mesh with fixed buses, this running time can easily be reduced by using the $2n/3$ time algorithm of Leung and Shende [33] to perform the linear array routing.)

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. 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. While the routing is similar in this respect to the scheme studied in the previous subsection, it is also important to realize the differences between the two schemes. In particular, we are not aware of any interpretation of the r-dimensional scheme as an instance of the Open Shop Scheduling Problem. On the other hand, it does not appear to be possible to generalize the two-dimensional scheme to higher dimensions.

The details of the computations necessary to obtain the intermediate locations of the packets can be derived from the description in Section 1.7.5 of [31]. For our purposes, it is enough to understand that the running time of these computations is polynomial in n^r, the number of packets in the network. More precisely, to obtain the intermediate locations of the packets after phase i, $1 \leq i \leq r-1$, it suffices to compute a sequence of n^i perfect matchings of size n^{r-i}. In order to convert this 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.

of more than 200 to obtain a running time of $1.2n$.) On the mesh with reconfigurable buses, our algorithm improves upon the best previously known randomized algorithm of Rajasekaran and McKendall [43]. 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.

Our algorithm can also be easily adapted to the *Polymorphic Torus* network described in [35]. (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 $n/2$.

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 [7] 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 gives an $n + o(n)$ algorithm for permutation routing on the *Mesh of Trees* [31].

(Note: Very recently, Kunde [27] and Kaufmann, Sibeyn, and Suel [25] have described optimal deterministic algorithms for k–k sorting on the standard mesh based on ideas similar to Leighton's Columnsort [29]. Using the same approach, it is possible to obtain algorithms for 1–1 routing and sorting on the Mesh of Trees that nearly match the lower bound of $n/2$ due to bisection width. This approach can also be used to design algorithms for the PARBUS and the Polymorphic Torus that match the running times of our algorithms.)

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. Moreover, our algorithm has a queue size of 2, while the only optimal deterministic algorithms previously known for the standard mesh [32, 44] have a queue size of at least 112. (Optimal deterministic algorithms with substantially smaller queue sizes have very recently been described in [8, 25].) 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 [2], as well as the $3n + o(n)$ sorting algorithm of Schnorr and Shamir [45], achieve 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 difficult to design any algorithm with a queue size of 1 that uses the buses to transmit packets.

An even greater speed-up over the standard mesh can be achieved if we consider certain restricted classes of permutations. Consider the problem of routing a partial permutation with only a very small number of packets (say, at most ϵn^2 packets). In the case of the standard mesh without buses, this problem would still require 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. It turns out that the above algorithm can be used to solve this problem, provided that the sources and destinations of the packets are approximately evenly distributed over the mesh.

Algorithm ROUTE:

(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 sorting and 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 using the buses.

(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) Perform 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 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 also 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 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.

We can decrease the queue size to 3 by assuming that the elements in the diagonal of B_i do not receive any of the packets entering the block across the row edges. To get a queue size of 2, 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 then rearrange the packets in each block and compute another schedule to deliver the remaining packets; the details of this construction are omitted. This establishes the following result.

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 [34] in both running time and queue size. (As an example, the algorithm in [34] requires a queue size 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 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 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 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 is 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 increases 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 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 $O\left((n^{1-\alpha})^{5/2}\right)$. 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 $O\left((n^{1-\alpha})^{9/2}\right) = o(n)$. In order to implement this computation on a bused mesh, 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 clearly be done in time $o(n)$, since only a small amount of information has to be transmitted. 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. All in all, we obtain the following algorithm:

for all i, since every column is 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 $U = \{P_0, \dots, P_{n-1}\}$, $V = \{R_0, \dots, R_{n-1}\}$, 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 Equations (1) and (2) 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 $O(n^{5/2})$ using the algorithm of Hopcroft and Karp [19]. Thus, the entire schedule can be computed in time $O(n^{7/2})$. Of course, this makes the algorithm inappropriate for use as an on-line algorithm. In the next subsection, we show how this off-line algorithm can be converted into an on-line algorithm that runs in time $n + o(n)$.

2.2 Permutation Routing on Two-Dimensional Meshes with Buses

In order to get a running time of $n + o(n)$, we 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 then takes 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. This is described more formally in the following.

We partition the bused mesh into blocks B_i, $0 \le i < n^{2-2\alpha}$, 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, $0 \le i < n^{1-\alpha}$, has exclusive ownership of its n^α column buses, while each resource R_j, $0 \le j < n^{1-\alpha}$, consists of n^α row buses. At most one process is allowed to access a single resource at any point in the algorithm. 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 this 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

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 new and improved algorithms for permutation routing and sorting 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 [33, 34]. 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 the two-dimensional mesh with buses. In Subsection 2.3 we apply our technique to obtain improved deterministic algorithms for multi-dimensional meshes with buses. In Subsection 2.4, we give another algorithm for the two-dimensional case. Finally, in Subsection 2.5 we describe our algorithms for 1–1 sorting.

2.1 Off-line Routing

In the off-line routing scheme of Leung and Shende [33, 34], 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 make use of 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 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 initially located 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 needs to access 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* [12].

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

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

holds for all j, since every row is the destination of exactly n packets. Correspondingly, we also have

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

time is actually slightly better than this bound.)

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 the mesh with fixed and reconfigurable buses, respectively, due to the bisection width of the network. For the mesh with fixed buses, Rajasekaran [42] and Sibeyn, Kaufmann, and Raman [46] describe randomized algorithms that match this 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 algorithms for the standard mesh given in [24]. Very recent work by Kunde [27] and Kaufmann, Sibeyn, and Suel [25] implies that this bound can also be matched deterministically. If we drop the assumption of bidirectional communication in subbuses of length 1, then about kn steps are necessary and sufficient.

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 two fairly simple deterministic algorithms for permutation routing on the $n \times n$ mesh with buses that achieve a running time of $n + o(n)$ and a queue size of 2. We also give an algorithm for r-dimensional meshes, $r \geq 3$, with a running time of $(2 - 1/r)n + o(n)$ and a queue size of 2. We then show how to obtain algorithms for 1–1 sorting whose running times match those for permutation routing, within a lower order additive term. An interesting feature of all our algorithms is that they can be implemented on a variety of different classes of networks, within the same bounds on time and queue size. For the mesh with fixed buses, our algorithms offer a significant improvement over the best previously known deterministic algorithms [33, 34] with respect to both running time and queue size. For the mesh with reconfigurable buses, our algorithms also improve over the best known randomized algorithms [42]. Our 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 is of independent interest, and that it 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, this situation is completely different in the case of dynamic routing. We describe an algorithm for the mesh with fixed buses that assures that in any time interval of length T, every packet is routed in time $O(\lg T + n^{1/3})$, with high probability. For the mesh with reconfigurable buses, we propose a very simple routing scheme that routes every packet in $O(\lg T + \lg n)$ steps. For the mesh with fixed CRCW buses, a delivery time of $O(\lg T + \lg n \lg \lg n)$ can be achieved. In contrast, the expected time for the completion of a routing request on the standard mesh is easily seen to be $\Theta(n)$.

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

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 [34], they obtain an improved algorithm for the two-dimensional case, running in time $(1+\epsilon)n+o(n)$ with a queue size of $O(1/\epsilon)$.

Rajasekaran and McKendall [43] and Rajasekaran [42] describe randomized algorithms for routing and sorting on a mesh in which the mesh edges have been replaced by a reconfigurable bus system. This model, hereinafter 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. This means that in this case 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 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 show how to obtain randomized algorithms for sorting with the same bounds on running time and queue size.

Comparing the two different models described above, we observe that in the case of the mesh with fixed buses, we cannot remove the standard mesh edges without losing the capability of efficiently performing local communication among groups of adjacent processors. On the other hand, on the mesh with reconfigurable buses the standard mesh edges are not really necessary, since they can be simulated efficiently by partitioning the buses appropriately. In this context, the assumption of bidirectional communication in subbuses of length 1 assures that any step of the standard mesh can be simulated by a single step of the mesh with reconfigurable buses. While this assumption may be technologically feasible, it can also be perceived as somewhat unsatisfactory from a theoretical point of view, since it adversely affects the simplicity of the model. We point out here that both the algorithms for two-dimensional networks in [43] and those presented in this paper do not make use of this assumption. Note that in this case, there is a lower bound of n steps for permutation routing due to the smaller bisection width.

Very recently, and independent of this work, Sibeyn, Kaufmann, and Raman [46] 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 values of r, their lower bounds are approximately $\frac{r-1}{r}n$. The lower bound for the two-dimensional case was also independently discovered by Cheung and Lau [7]. 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 a fairly simple algorithm with a running time of $0.87n$, and a more complicated algorithm with a running time of $0.78n$. They also give an algorithm for d-dimensional networks that achieves a running time of $(2-1/r)n+o(n)$, for all r. (For small values of r, the running 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. We also use the term CRCW to refer to buses with broadcast capability and some form of write conflict resolution. Note that there is a close relationship between a shared memory cell in a CREW/CRCW PRAM and a global bus of the same type [37].

Additional features that have been studied include buses with non-unit delay [35, 38], and buses that allow pipelining of messages under certain conditions [15].

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 will also run on more powerful models, such as the *Polymorphic Torus* [35] or the PARBUS [51], which can be reconfigured into a mesh with row and column buses. Unless explicitely stated otherwise, we assume the buses to be CREW. However, we 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 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 [30, 31] and the hypercube [47], we are not aware of any previous analysis of 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 proposed by Leung and Shende [33]. They assume a model of computation, hereinafter 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 running in $2n/3$ steps with small constant queue size. They also show a matching lower bound of $2n/3$ for this problem; this lower bound can be extended to networks of higher dimensions. For the two-dimensional mesh, Leung and

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 [31]. 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, still require at least $n - 1$ communication steps.

To remedy this situation, it was proposed by several authors [4, 22, 48] 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 [1, 4, 10, 38], Prefix Sums [3, 6, 10, 28, 41, 48], Selection [5, 16, 41, 48], as well as various algorithmic problems in image processing and graph theory [20, 21, 38, 40, 49]. Additional literature can be found in [35] and the above references.

Due to the low communication requirements of most of the above problems, significant speed-ups over the standard mesh can be achieved. The exact time complexities of the proposed algorithms 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. On the mesh without buses, at least $n - 1$ steps are needed. 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 [1, 4, 38, 48]. Most of the work on local buses has focused on the mesh with row and column buses [10, 41, 49], although other architectures have been proposed [37, 49].

(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*. In a system with reconfigurable buses, the possible partitions of the buses depend on the layout of the bus system. As an example, consider an $n \times n$ mesh with reconfigurable row and column buses laid out in the obvious way. Then each of the n row buses (column buses) can only be partitioned into subbuses connecting groups of consecutive processors of the respective row (column).

(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

Routing and Sorting on Meshes with Row and Column Buses

Torsten Suel[*]

Department of Computer Sciences
University of Texas at Austin
torsten@cs.utexas.edu

Abstract

We study the complexity of permutation routing, sorting, and dynamic routing 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 give improved deterministic algorithms for permutation routing and sorting. Among our results, we obtain two fairly simple algorithms for permutation routing on two-dimensional meshes with buses that achieve 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, within a lower order additive term. An interesting feature of our algorithms is that they can be efficiently implemented on a variety of different models of meshes with buses. The algorithms are obtained through a new technique that allows us to convert certain off-line routing schemes into deterministic on-line algorithms.

In the second part of the paper, we study the performance of meshes with buses on dynamic routing problems. We analyze fast routing schemes under several different assumptions about the properties of the bus system. We observe that the time for the completion of a dynamic routing request can be significantly reduced by adding buses to the network.

[*]Supported by the Texas Advanced Research Program under Grant Nos. #003658–480 and #003658–461, and by a Schlumberger Graduate Fellowship. A preliminary version of this paper appeared in [50].