JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING **33,** 84–90 (1996)
ARTICLE NO. 0027

Routing with Locality on Meshes with Buses[1]

STEVEN CHEUNG AND FRANCIS C. M. LAU

Department of Computer Science, University of California, Davis, California 95616; and Department of Computer Science, The University of Hong Kong, Hong Kong

Routing with locality is studied for meshes with buses. In this problem, packets' distances are bounded by a value, d, which is less than the diameter of the network. This problem arises naturally when specific known algorithms are implemented on meshes. Solving this problem in ordinary meshes requires at least a routing time of d steps. To do better than this, we propose adding a kind of short bus to ordinary meshes. By using a technique which we call iterative walk-and-ride, we show how the routing time can be reduced by approximately one-third for solving the problem (including the multipacket version) on one- and two-dimensional short-bus meshes. The bounds we develop are tight. © 1996 Academic Press, Inc.

1. INTRODUCTION

Permutation routing on mesh-connected computers has been extensively investigated [9, 10, 12, 13, 14, 16, 17, 22, 25, 29]. In permutation routing, each processor initially has a packet to send, and packets have unique destinations. A good routing algorithm should route packets to their destinations as quickly as possible and should require as few buffers at each processor as possible. Among the best known is the deterministic algorithm by Leighton *et al.* [16] which can solve the problem for an $n \times n$ mesh in $2n - 2$ steps, which is optimal in the worst case since the mesh's diameter is $2n - 2$. Unfortunately, the buffer size required by their algorithm is impractically large. The result has been improved by Rajasekaran and Overholt [22], Gu and Gu [9], Sibeyn *et al.* [25], Kaufmann *et al.* [12], and Kaufmann *et al.* [10]. However, these results are no longer time optimal when the packets only have to travel a distance less than the diameter, say $\sqrt{n}$ or $n/2$. This routing with locality problem (or restricted distance routing problem) has been studied in [6, 11, 13, 14]. Routing with locality problems arises naturally when specific known algorithms are implemented on the mesh [13] and when other types of processor or process networks are embedded onto mesh-connected computers to yield a small dilation. In fact, one of the goals in designing parallel algorithms is to try to make distances between related processes as short as possible [4]. If the maximum distance of all the packets is bounded by d, the authors have presented in [6] a $d + O(d/f(d))$ step, $O(f(d))$ buffer size routing algorithm which is asymptotically optimal if $f(d)$ is chosen to be a large constant. In our study, we assume all the processors operate in synchronous MIMD mode. At any time step, each processor can communicate with all of its grid neighbors and can both send and receive one packet along each mesh link. In addition, processors can also store packets in their own queues. This model (hereafter referred to as the base model) is the same as the ones used in [9, 10, 12–16, 22, 25].

The main disadvantage of the mesh topology is its large diameter, which has direct impact on the communication times of many parallel algorithms. Augmenting arrays of processors with various faster mechanisms has been suggested as a means to speed up communication among the processors. Examples are meshes with multiple buses which have a bus in each column and each row [1, 20], generalized meshes with multiple buses which are composed of smaller meshes with multiple buses [8], meshes with separable row and column buses in which row/column buses can be separated into multiple shorter buses through turning on/off bus switches [24], and reconfigurable meshes in which links can be connected to form buses [2]. These enhanced meshes are capable of solving problems that only require a limited amount of global communication significantly faster. This paper shows how to utilize these buses in some ''high-bandwidth'' routing problems.

In all the related bused mesh models, except reconfigurable meshes, broadcast buses are added to the base model. Each broadcast bus is connected to a set of processors. In each time step, only one processor attached to a bus can send a packet via the bus. In addition, a processor can receive packets from all the buses attached to it in a time step. A number of proposed bused mesh models assume the propagation delay of a bus to be a constant which is independent of the number of processors attached to it. This assumption is thought to be a reasonable one in practical situations [1, 2, 3, 20, 27]. However, Lu *et al.* [18] investigated physical implementations of buses and found that short buses and long buses do have a difference in performance and that the constant-delay assumption is more appropriate with short-bus models. In this paper, we assume the propagation delay of the buses to be one time step which is also assumed in [1, 8, 17, 27]. As we will see,

[1] A preliminary version of this paper appears in *Proceedings of the 6th IEEE Symposium on Parallel and Distributed Processing,* October 1994.

0743-7315/96 $18.00

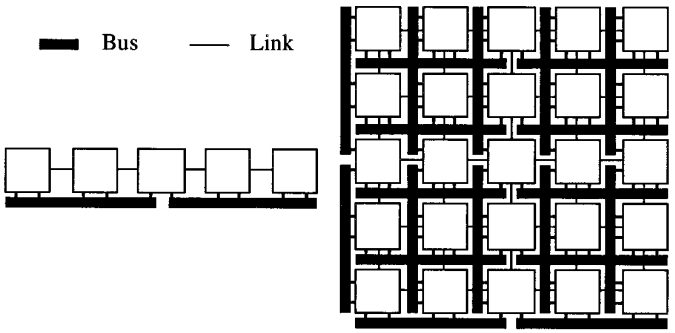

FIG. 1. Examples of short-bus meshes.

our algorithms can be tuned to use short, or constant-length buses.

Upper and lower bounds for unrestricted distance routing on meshes with fixed and reconfigurable buses can be found in [7, 17, 21, 26, 28]. For instance, for meshes with multiple buses, Leung and Shende [17] proved that $2n/3$ is a tight worst-case time bound for permutation routing on an n-processor one-dimensional mesh.

In this paper, we study how and to what extent buses can help in solving the restricted distance routing problem on bused meshes. In the following, we first define our short-bus mesh model, and prove some lower bounds for the problem. Then we present our routing algorithms for permutation routing with locality in one- and two-dimensional short-bus meshes. Finally, we state our results on multipacket routing.

In a two-dimensional short-bus mesh, each row/column has a sequence of equal-length buses and any two adjacent buses are incident on a common processor. These short buses can be active simultaneously. Our short-bus mesh model can be realized by a simple modification of fixed-bus meshes like generalized meshes with multiple buses [8] or reconfigurable-bus meshes like meshes with separable row and column buses [24]. In generalized meshes with multiple buses and meshes with separable buses, adjacent buses do not meet at a common processor. Figure 1 shows some examples of one- and two-dimensional short-bus meshes. Short-bus meshes have several advantages over meshes with multiple row/column buses. First, short buses have shorter propagation delay. Second, Chung [8] and Maeba *et al.* [19] showed that short-bus meshes can solve certain fundamental problems significantly faster than meshes with multiple buses. Third, short-bus meshes are relatively easy to partition into bused submeshes to serve different computational requests.

2. LOWER BOUNDS

When the maximum distance between the source and the destination of any packet is d and $d \leq 2n/3$, we show that permutation routing cannot be solved on meshes with multiple buses in fewer than d steps. The proof, shown below, is a generalization of Leung and Shende's $2n/3$ time steps lower bound proof for the permutation routing problem on meshes with fixed buses [17].

For simplicity, we consider a one-dimensional mesh with a global bus and d is even in the proof, which can be easily generalized to two- and higher dimensional cases. Consider processor i, where $i \in [1, 2, ..., d/2]$, which has a packet to send to processor $i + d$. Moreover, processor $i + d$ also has a packet to send to processor i. Each of those d packets must have a bus ride in order to reach its destination within d steps. However, the bus can only carry one packet during each time step. Thus, any algorithm needs at least d time steps for this problem.

We now prove a lower bound for restricted distance permutation routing on one-dimensional short-bus meshes. If $d \leq n/2$, consider the scenario that all the packets in a processor subarray of length $2d$ send a packet to the processor which has distance d from it and within the same subarray. In the middle of the subarray, we have $2d$ packets crossing a cut of size 3—a bidirectional link and a bus. Hence the $2d/3$ time bound. Note that this lower bound is valid also for two-dimensional short-bus meshes. It can be proved by applying this one-dimensional construction to each row of the meshes. Using the same construction, we have a $2d$-step lower bound for reconfigurable meshes (because buses are unidirectional) and a d-step lower bound for the generalized meshes with multiple buses. The bounds for the latter two types of meshes are due to the gap between two adjacent buses.

3. ROUTING ON ONE-DIMENSIONAL SHORT-BUS MESHES

In this section, we present an asymptotically $(2d/3)$-step algorithm for the restricted distance permutation routing problem. The one-dimensional short-bus mesh is assumed to have buses of length b, where b is an odd integer[2] and $b \ll d$. In odd-numbered steps, each bus segment will be used to route packets from left to right, and vice versa for even-numbered steps. The processors at the ends of buses are called *bus terminals*. When packets are transmitted via buses, they usually travel from bus terminal to bus terminal, except when their destinations are in the middle of a bus segment, in which case the packets will go directly to their destinations instead of to the bus terminal at the other end of the bus. Note that each bus terminal has two adjacent buses attached to it.

ALGORITHM 1 (Iterative Walk-and-Ride Algorithm). Consider an arbitrary processor P.

Case 1. P is not a bus terminal. If P receives a packet[3]
- from a mesh link, P forwards the packet by mesh link;

[2] In case we have an even b, we can use two additional buffers in each bus terminal to simulate a virtual neighboring processor.

[3] For simplicity, at time 0, every packet is treated as having come from a mesh link.

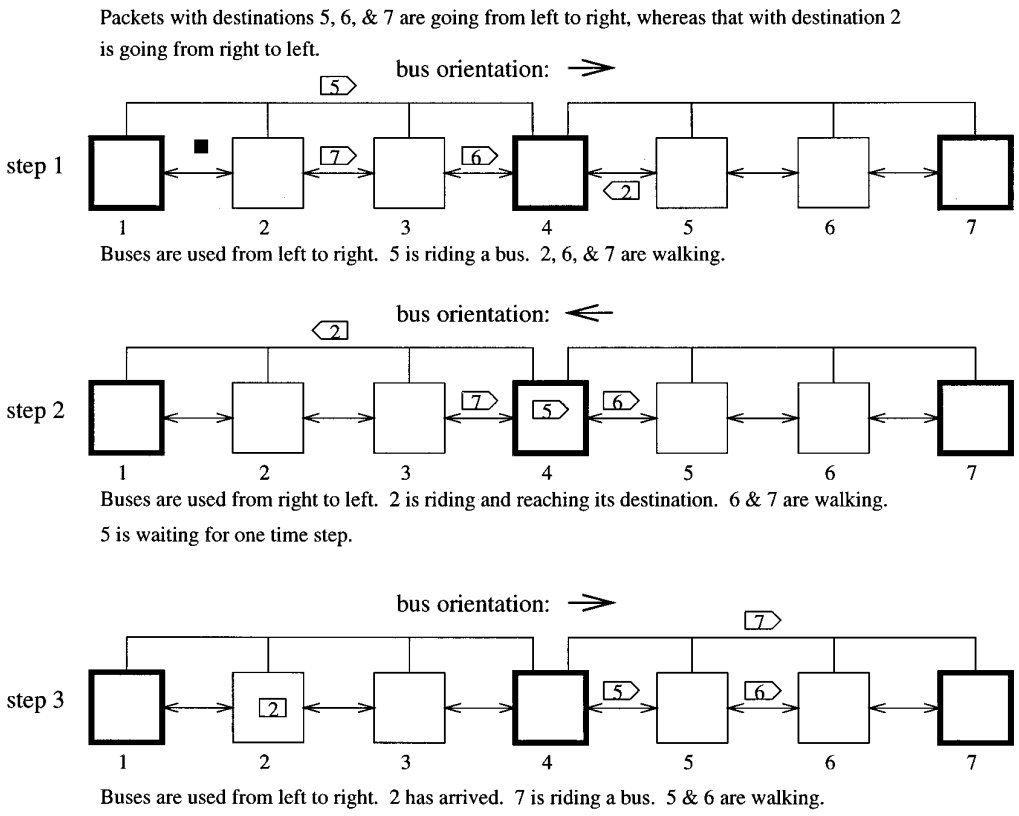

FIG. 2. An example of the walk-and-ride algorithm.

• from a bus (that is, P is its destination), P stores the packet into its local memory.

Case 2. P is a bus terminal. If P receives a packet

• from a mesh link and bus is available in the next time step, *P* routes it by bus to the next bus terminal or to its destination, whichever is nearer;

• from a mesh link but bus is not available in the next time step, *P* routes it by mesh link;

• from a bus, the packet has to wait one time step before it leaves by walking.

The basic idea of Algorithm 1 is to ensure that after a packet has "walked" (i.e., being transmitted through links) say for r steps, it can take a bus to cover say s units of distance. As a result, all packets with source-destination distance d can reach their destinations in approximately $d \times r/(r + s)$ steps. This walking–riding cycle continues until the packet reaches its destination. Hence, no packet will suffer from too long a walk. The lower bound shown above gives us a cue for the ratio r/s. Algorithm 1 makes the packets to wait one step after they have ridden a bus because a bus and a link are available for transporting two packets in the corresponding direction in those time steps. Figure 2 shows a few steps of an example of executing this algorithm. In the example, we have $b = 3$, nodes 1, 4, and 7 are bus terminals, and packets are identified by their destination address. At time step 1 (odd), the buses are oriented for going right, and at time step 2 (even), the other way around. At time step 2, the packet destined for node 2 takes a bus ride directly to its destination, which is not a terminal.

LEMMA 1. *Given a permutation routing problem on a linear array, all packets will have their source-destination distances reduced by D (or they will reach their destinations in case their source-destination distance is* $<D$*) after running Algorithm 1 for* $(D - \lfloor D/3b \rfloor \times b) + \lceil D/3b \rceil \times 2$ *steps or less.*

Proof. Except the ones that had a bus ride in the previous step, every packet advances in each time step. Without loss of generality, consider only those packets that are moving to the right. Because b is odd, we can ensure that, in every $2b$ steps, any packet can reach the left end of a bus in one of these steps and the corresponding bus will serve it in the next step. In other words, each packet travels a distance of $3b$ in $2b + 2$ steps because it only walks $2b$ steps and the cost of a bus ride followed by waiting is 2 time steps. Any packet traveling for a distance of D takes at least $\lfloor D/3b \rfloor$ bus rides, and it walks for at most $(D - \lfloor D/3b \rfloor \times b)$ steps. Moreover, it only spends at most $2 \times \lceil D/3b \rceil$ time steps for riding buses and waiting at bus terminals. ■

THEOREM 1. *For the permutation routing problem on a linear array with the maximum source–destination dis-*

tance d, Algorithm 1 *can be used to solve it in approximately* $2d/3$ *time steps and only a constant number of auxiliary buffers are required per processor.*

4. ROUTING ON TWO-DIMENSIONAL SHORT-BUS MESHES

The algorithm for permutation routing on two-dimensional short-bus meshes has two main ingredients, namely, sorting and greedy algorithm. Sorting is used to redistribute the packets so as to ensure that packets destinated at the same row are routed along many columns. Thus not too many packets will turn at any processor, and hence the number of buffers required for each processor is small. The idea of using sorting to lower buffer complexity was first proposed by Kunde [14]. The greedy algorithm we use is as follows: every packet is first routed along its column (column routing) until it reaches its destination row. Then it is routed along its destination row (row routing) to its destination.

Before proving the two-dimensional case, we first define some notation and prove a result on the one–many routing problem on a linear array. In one–many routing, an arbitrary number of packets can start as a processor, and packets have unique destinations. Let b be the length of each bus and $f(d)$ be a function of d. Let the processors of a linear array be numbered from 0 to n. Define a processor to be of type i, $i \in \{0, 1, 2\}$, if its *ID* mod 3 equals i. A packet is of type i if its destination processor is of type i. Bus terminals have an extra set of labels and they are labeled from 0 to n/b and a type $\in \{0, 1, 2\}$ is assigned to each of them, similarly based on these labels. Informally speaking, we want to divide the packets into types and schedule them according to their types in such a way that packets of different types do not interfere with each other and packets of the same type have similar behavior (e.g., the paths they travel). For simplicity, we assume $3b$ divides d henceforth.[4]

Next, we will prove a result on one–many routing with the restriction that all type-i packets reside in type-i terminals initially. Then we will describe how to perform one–many routing without this restriction.

ALGORITHM 2 (One–Many Routing with Restriction).

1. A packet with source–destination distance $(d - x)$ starts moving after step $\lfloor x/3b \rfloor \times (2b + 2) + \lfloor (x - \lfloor x/3b \rfloor \times 3b)/3 \rfloor \times 2$.

2. The packet performs the following repeatedly until it reaches its destination:

(a) Walking for 2 segments.

(b) Taking a bus ride to the next terminal. Right-moving packets wait one step at each bus terminal just after each bus ride, while left-moving ones do so before each ride.[5]

LEMMA 2. *Given a one–many routing problem on a linear array with maximum source–destination distance d and all type-i packets residing in type-i terminals initially, then Algorithm* 2 *can route all packets to their destinations in* $(1 + 1/b)2d/3$ *time steps.*

Proof. Without loss of generality, consider the packets moving to the right only. Basically, we make packets with a farther distance start earlier. For packets of the same type, their distances differ by a multiple of three. For any two type-i packets with destinations that are three processors apart, the one with a farther destination will be scheduled to leave two steps earlier than the other one. This is because we enforce that type-i packets can leave type-i terminals only at odd-numbered steps, while the right to leave type-i terminals at even-numbered steps is reserved for packets of type $((i - 1) \bmod 3)$, which will be explained later.

Note that for any segment, there are only two different types of packets that might walk through it, while the remaining ones not belonging to those two types take the bus associated with that segment. Moreover, as each type-i packet leaves type-i terminals in odd-numbered steps, while type-$((i - 1) \bmod 3)$ packets leave type-i terminals in even-numbered steps, packets of these two different types will not interfere with each other. This is the case because the segment length b is chosen to be an odd integer. Observe that once a packet starts moving, it will never be delayed by any other packet. Hence it needs at most $(d - x) - (\lfloor (d - x)/3b \rfloor \times b) + \lceil (d - x)/3b \rceil \times 2$ more steps to reach its destination by Lemma 1. Therefore, the time required for a packet with distance $(d - x)$ to reach its destination is $\leq 2d/3 + 2d/3b$. ■

Lemma 2 shows us how to route any one–many problem in which type-i packets start at type-i bus terminals. In the following, we make use of this result to route any one–many problem without this restriction. In short, our strategy is first compute the starting time step of every packet in the latter problem and then apply the routing scheme of the former problem. For a type-i packet p whose source is s, let its *virtual source* be the nearest type-i bus terminal in the opposite direction from its destination.[6] In the following, let packet p be at a distance $(d - x)$ from its destination and be at a distance y from its virtual source.

ALGORITHM 3 (One–Many Routing without Restriction).

[4] As we are interested in the case $b \ll d$, so we can make d a multiple of $3b$ without causing any significant increase in time complexity.

[5] Because a bus can take only one packet at a time, left-moving packets and right-moving packets are scheduled to use the buses in alternate steps. Alternatively, we can have left-moving packets start one step later than the right-moving ones. In this case, all packets wait one time step after each bus ride.

[6] For packets near the ends that do not have a virtual source, we extend the array conceptually to create virtual sources for them.

1. Packet p starts moving after step $\lfloor (3b + x - y)/3b \rfloor \times (2b + 2) + \lfloor ((3b + x - y) - \lfloor (3b + x - y)/3b \rfloor \times 3b)/3 \rfloor \times 2 + \min\{y, 2b\}$.

2. The path taken by packet p is the portion of the path starting from s if p were originated at its virtual source and Algorithm 2 were used.

LEMMA 3. *Given a one–many routing problem on a linear array with maximum source–destination distance d in which packets are arbitrarily distributed among processors. Packets can be scheduled so that all of them can reach their destinations in* $(1 + 1/b)2d/3 + (2b + 2)$ *time steps.*

Proof. This one–many problem is now transformed to one in which type-i packets are originated from type-i terminals with maximum distance $(d + 3b)$ as in Algorithm 2. Specifically, we make use of Lemma 2 by pretending that each packet is originated from its virtual source. If p were originated from its virtual source, v, its distance from its destination would be $((d + 3b) - (3b + x - y))$, or it would start at step $\lfloor (3b + x - y)/3b \rfloor \times (2b + 2) + \lfloor ((3b + x - y) - \lfloor (3b + x - y)/3b \rfloor \times 3b)/3 \rfloor \times 2$ by Lemma 2. Furthermore, it would leave s $\min\{y, 2b\}$ steps later. The starting time step of p is calculated by adding the values of these two expressions. There are two cases concerning the $\min\{y, 2b\}$ term. If the distance between v and s is less than $2b$, the packet would walk from v to s in the entire imaginary journey, thus s will transmit this packet y steps after leaving v. If the distance between v and s is equal to or more than $2b$, s will transmit the packet over the bus $2b$ steps after the starting time at v. The stated time bound follows from Lemma 2. ■

In the following, we present an algorithm, the Sorting + Greedy Algorithm, for solving permutation routing on two-dimensional short-bus meshes.

ALGORITHM 4 (Sorting + Greedy Algorithm).

1. Partition the short-bus mesh into submeshes of size $d_0/f(d_0) \times d_0/f(d_0)$ each, where d_0 is the maximum source–destination distance before sorting. Sort all the packets within each submesh in row-major ordering[7] with respect to the row-major indexing of the processors.[8]

2. Perform the following in parallel with row routing being started $(b + 4)$ steps later than column routing.

(a) Packets not situated in their destination rows participate in column routing according to Algorithm 1;

(b) Others participate in row routing as in Algorithm 3.

THEOREM 2. *The restricted distance permutation problem with maximum distance* d_0 *on a two-dimensional* $n \times n$ *short-bus mesh can be solved by Algorithm* 4, *which has time complexity* $2d(1 + 1/b)/3 + 3b + 6 + O(d_0/f(d_0))$ *and buffer complexity* $O(f(d_0))$ (*per processor*), *where* $d = d_0 + 2d_0/f(d_0)$.

Proof. After sorting is performed in each submesh, a packet may be displaced from its destination for a distance of $d_0/f(d_0)$ vertically and $d_0/f(d_0)$ horizontally. Thus the maximum source–destination distance after sorting, d, is $d_0 + 2d_0/f(d_0)$. Once a packet starts moving, it always advances to its destination by first moving along column edges and then along row edges. It is obvious that column routing and row routing work correctly by their own. To prove the correctness of this algorithm, we only need to show that a packet can reach its destination row by the time it has to start row routing as in Lemma 3. Consider a packet that has horizontal distance $d - d_v$. By Lemma 1 (instead of considering packets moving in the left and right directions, we now consider packets moving in the up and down directions), any packet with vertical distance d_v reaches its destination row after at most $2d_v/3 + \lceil d_v/3b \rceil \times 2 + b$ steps (which bounds $(d_v - \lfloor d_v/3b \rfloor \times b) + \lceil d_v/3b \rceil \times 2$ from above). From the proof of Lemma 3 and the fact that row routing is started $b + 4$ steps later than column routing, we know that this packet starts row routing only after step

$$\begin{aligned}\lfloor (3b + d_v - y)/3b \rfloor \times (2b + 2) + \lfloor ((3b + d_v - y) \\ - \lfloor (3b + d_v - y)/3b \rfloor \times 3b)/3 \rfloor \times 2 \\ + \min\{y, 2b\} + (b + 4) \geq 2d_v/3 \\ + 2d_v/3b + b + 2.\end{aligned}$$

Thus, we can guarantee that any packet can finish the column routing on time.

The total time of this algorithm consists of two components: Step 1 requires $O(d_0/f(d_0))$ time steps [23]; performing column and row routing in parallel requires $2d(1 + 1/b)/3 + 3b + 6$ steps—this is because $2d(1 + 1/b)/3 + (2b + 2)$ steps are needed to complete the row routing phase and row routing starts $(b + 4)$ steps after column routing starts. Regarding buffer size complexity, each processor requires only $O(f(d_0))$ buffers as there are only $O(f(d_0))$ packets turning at any processor. We omit the proof of the buffer complexity which can be found in [6]. ■

COROLLARY 1. *Restricted distance permutation routing with maximum distance d on an* $n \times n$ *short-bus mesh can be solved in approximately* $2d/3$ *steps with a constant number of buffers per processor, if* $b = o(d)$.

[7] Let (r, c) denote the destination address of a packet, where r and c are the row and column address, respectively, of the destination processor; then the row-major ordering of two packets with destination addresses (r_1, c_1) and (r_2, c_2) is defined as $(r_1, c_1) \leq (r_2, c_2)$ iff $r_1 \leq r_2$.

[8] The processors are indexed by a one–one mapping onto $\{1, ..., n^2\}$ and the sorting problem with respect to this mapping is to move the ith smallest packet to the ith indexed processor. In row-major indexing of the processors, processor (r_1, c_1) has a smaller index than that of processor (r_2, c_2) when either $r_1 < r_2$, or $r_1 = r_2$ and $c_1 < c_2$.

5. k–k ROUTING

A more general version of the permutation routing is called k–k routing, in which each processor has exactly k packets to send and exactly k packets to receive at the end. If the maximum source–destination distance of all the packets is d, our lower bound for permutation routing can be adapted to yield a $2kd/3$ bound for k–k routing on a short-bus mesh with side length n, where $d \le n/2$. Because of space limitations, we only state our results for k–k routing on one- and two-dimensional short-bus meshes below. Readers are referred to [5] for the details.

THEOREM 3. *Given a k–k routing problem on a linear array, all packets can have their source–destination distance reduced by D (or they will reach their destinations in the cases where their source–destination distances are $<D$) in $\le (D - \lfloor D/3b \rfloor \times b)k + \lceil D/3b \rceil \times 2$ steps if k is odd, or $\le (D - \lfloor D/3b \rfloor \times b)k + \lceil D/3b \rceil \times 2 + (\lceil D/b \rceil - \lfloor D/3b \rfloor)$ steps if k is even.*

COROLLARY 2. *On a linear array with the maximum source–destination distance d, if $b = o(d)$, k–k routing can be solved in approximately $2kd/3$ time steps and only a constant number of auxiliary buffers are required per processor.*

The main issue in extending our permutation routing result to k–k routing in the two-dimensional case is that there could be more than one packet sharing the same destination, which makes row routing a bit more complicated. As packets may start their row routing phase by riding buses in the middle of bus segments, we need to schedule the buses in such a way that no two packets try to ride the same bus at the same time. We achieve this by letting each packet know how many other packets share the same destination with it before it starts its row routing phase. Therefore, we divide the whole algorithm into c phases, which is a parameter to be tuned.

THEOREM 4. *The restricted distance k–k routing problem with maximum source–destination distance d_0 on a two-dimensional $n \times n$ short-bus mesh can be solved in $(c/c-1)((1 + 1/b)2kd/3 + (2b + 2)k) + O(kd_0/f(d_0)) + O(cbkf(d_0))$ steps with buffer size $O(kf(d_0))$ per processor, where $d = d_0 + 2f(d_0)$.*

COROLLARY 3. *On an $n \times n$ mesh with the maximum source–destination distanced, if $b = o(d)$, k–k routing can be solved in approximately $2kd/3$ time steps with buffer size $O(k)$ per processor.*

6. CONCLUDING REMARKS

We have presented an iterative walk-and-ride technique for solving routing with locality problems on short-bus meshes. With respect to ordinary meshes, one third of the running time is saved after the short buses are employed. A short-bus mesh can be derived from a fixed-bus mesh such as the generalized mesh with multiple buses [8] or from a reconfigurable bus mesh such as the mesh with separable buses [24]. The results developed in this paper are also directly applicable to short-bus rectangular meshes and tori.

ACKNOWLEDGMENTS

We thank Charles Martel for his comments on an earlier version of this paper. We are also grateful to the referees for their suggestions and comments.

REFERENCES

1. A. Bar-Noy and D. Peleg, Square meshes are not always optimal. *IEEE Trans. Comput.* **40**(2) (Feb. 1991), 196–203.
2. Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, The power of reconfiguration. *J. Parallel Distrib. Comput.* **13**(2) (Oct. 1991), 139–153.
3. S. H. Bokhari, Finding maximum on an array processor with a global bus. *IEEE Trans. Comput.* **C-33**(2) (Feb. 1984), 133–139.
4. M. C. Chen, A design methodology for synthesizing parallel algorithms and architectures. *J. Parallel Distrib. Comput.* **3**(4) (Dec. 1986), 461–491.
5. S. Cheung, Packet routing on mesh-connected computers. M. Phil. thesis, Department of Computer Science, The University of Hong Kong, Oct. 1992.
6. S. Cheung and F. C. M. Lau, Mesh permutation routing with locality. *Inform. Process. Lett.* **43**(2) (Aug. 1992), 101–105.
7. S. Cheung and F. C. M. Lau, A lower bound for permutation routing on two-dimensional bused meshes. *Inform. Process. Lett.* **45**(5), (Apr. 1993), 225–228.
8. K. L. Chung, Prefix computations on a generalized mesh-connected computer with multiple buses. *IEEE Trans. Parallel Distrib. Systems* **6**(2), (Feb. 1995), 196–199.
9. Q. P. Gu and J. Gu, Two packet routing algorithms on a mesh-connected computer. *IEEE Trans. Parallel Distrib. Systems* **6**(4), (Apr. 1995), 436–440.
10. M. Kaufmann, U. Meyer, and J. F. Sibeyn, Towards practical permutation routing on meshes. *Proc. 6th IEEE Symposium on Parallel and Distributed Processing.* Oct. 1994, pp. 664–671.
11. M. Kaufmann and J. F. Sibeyn, Optimal multi-packet routing on the torus. *Proc. 3rd Scandinavian Workshop on Algorithm Theory, Lecture Notes in Computer Science 621,* Springer-Verlag, Berlin/New York, July 1992, pp. 118–129.
12. M. Kaufmann, J. F. Sibeyn, and T. Suel, Derandomizing algorithms for routing and sorting on meshes. *Proc. 5th Symposium on Discrete Algorithms.* ACM-SIAM, Jan. 1994, pp. 669–679.
13. D. Krizanc, S. Rajasekaran, and T. Tsantilas, Optimal routing algorithms for mesh-connected processor arrays. *Proc. 3rd Aegean Workshop on Computing: VLSI Algorithms and Architectures, Lecture Notes in Computer Science 319.* Springer-Verlag, Berlin/New York, June 1988, pp. 411–422.
14. M. Kunde, Routing and sorting on mesh-connected arrays. *Proc. 3rd Aegean Workshop on Computing: VLSI Algorithms and Architectures, Lecture Notes in Computer Science 319.* Springer-Verlag, Berlin/New York, June 1988, pp. 423–433.
15. M. Kunde and T. Tensi, (k–k) routing on multidimensional mesh-connected arrays. *J. Parallel Distrib. Comput.* **11**(2) (Feb. 1991), pp. 146–155.
16. T. Leighton, F. Makedon, and I. G. Tollis, A $2n - 2$ step algorithm for routing in an $n \times n$ array with constant size queues. *Proc. 1989 ACM Symposium on Parallel Algorithms and Architectures.* pp. 328–335.

17. J. Y. T. Leung and S. M. Shende, On multi-dimensional packet routing for meshes with buses. *J. Parallel Distrib. Comput.* **20**(2) (Feb. 1994), pp. 187–197.
18. Y. W. Lu, J. B. Burr, and A. M. Peterson, Permutation on the mesh with rconfigurable bus: Algorithms and practical considerations. *Proc. 7th International Parallel Processing Symposium.* Apr. 1993, pp. 298–308.
19. T. Maeba, S. Tatsumi, and M. Sugaya, Algorithms for finding maximum and selecting median on a processor array with separable global buses. *Electron. Comm. Japan Part III* **73**(6), (June 1990), pp. 39–48.
20. V. K. Prasanna Kumar and C. S. Raghavendra, Array processor with multiple broadcasting. *J. Parallel Distrib. Comput.* **4**(2), (Apr. 1987), 173–190.
21. S. Rajasekaran, Mesh connected computers with fixed and reconfigurable buses: Packet routing, sorting, and selection. *Proc. 1st Annual European Symposium on Algorithms, Lecture Notes in Computer Science 726.* Springer-Verlag, Berlin/New York, Oct. 1993, pp. 309–320.
22. S. Rajasekaran and R. Overholt, Constant queue routing on a mesh. *J. Parallel Distrib. Comput.* **15**(2) (June 1992), 160–166.
23. C. P. Schnorr and A. Shamir, An optimal sorting algorithm for mesh connected computers. *Proc. 18th Annual ACM Symposium on Theory of Computing.* 1986, pp. 255–263.
24. M. J. Serrano and B. Parhami, Optimal architectures and algorithms for mesh-connected parallel computers with separable row/column buses. *IEEE Trans. Parallel Distrib. Systems* **4**(10) (Oct. 1993), 1073–1080.
25. J. F. Sibeyn, B. S. Chlebus, and M. Kaufmann, Shorter queues for permutation routing on meshes. *Proc. 19th International Symposium on Mathematical Foundations of Computer Science.* Springer-Verlag, Berlin/New York, Aug. 1994, pp. 597–607.
26. J. F. Sibeyn, M. Kaufmann, and R. Ramen, Randomized routing on meshes with buses. *Proc. 1st Annual European Symposium on Algorithms, Lecture Notes in Computer Science 726.* Springer-Verlag, Berlin/New York, Oct. 1993, pp. 333–344.
27. Q. F. Stout, Mesh-connected computers with broadcasting. *IEEE Trans. Comput.* **C-32**(9) (Sept. 1983), 826–830.
28. T. Suel, Permutation routing and sorting on meshes with row and column buses. *Parallel Process. Lett.* **5**(1) (March 1995), 63–80.
29. L. G. Valiant and G. J. Brebner, Universal schemes for parallel communication. *Proc. 13th Annual ACM Symposium on Theory of Computing.* 1981, pp. 263–277.

Received April 19, 1994; revised April 3, 1995; accepted October 2, 1995

STEVEN CHEUNG received the B.S. and M.Phil. degrees in computer science from the University of Hong Kong in 1989 and 1992, respectively. He is currently a Ph.D. student in the Department of Computer Science at the University of California, Davis. His research interests include parallel algorithms and computer security.

FRANCIS C. M. LAU received the B.Sc. degree from Acadia University, Canada in 1979 and MMath and Ph.D. degrees from the University of Waterloo, Canada in 1980 and 1986, respectively. He joined the Department of Computer Science at the University of Hong Kong in 1987, where he is now a senior lecturer. His research interests are in parallel and distributed computing, object-oriented programming, and operating systems.