Routing with Locality in Partitioned-Bus Meshes

Steven Cheung

Department of Computer Science
University of California
Davis, CA 95616, USA

Francis C.M. Lau

Department of Computer Science
The University of Hong Kong
Hong Kong

Abstract

We show that adding partitioned-buses (as opposed to long buses that span an entire row or column) to ordinary meshes can reduce the routing time by approximately one-third for permutation routing with locality. A matching time lower bound is also proved. The result can be generalized to multi-packet routing.

1 Introduction

Permutation routing on mesh-connected computers has been extensively investigated [7] [8] [10] [11] [19] [21]. 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 algorithm by Leighton, Makedon, and Tollis [10] which can solve the problem for an $n \times n$ mesh in $2n-2$ steps using a constant-sized buffer. Because the mesh's diameter is $2n-2$, their algorithm is optimal in the worst case. However, it is no longer time optimal when the packets only have to travel a short distance, say $\log n$ or $\sqrt{n}$. This routing with locality problem (or restricted distance routing problem) has been studied in [4] [7] [6] [8]. Routing with locality problems arise naturally when specific known algorithms are implemented on the mesh [7] and when other types of processor or process networks are embedded onto mesh-connected computers to yield a small dilation [4]. If the maximum distance of all the packets is bounded by d, the authors have presented in [4] a $d + O(d/f(d))$ step, $O(f(d))$ buffer size routing algorithm which is asymptotically optimal if $f(d)$, a function of d, is chosen to be a large constant (because an obvious lower bound is d steps). In our mesh model, all the processors are assumed to run 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 [7] [8] [9] [10].

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 fixed buses which have a bus in each column and each row [1] [13], 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 [16], and reconfigurable meshes in which mesh links can be connected together to form buses [2]. 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 [18] [3] [13] [1] [2]. However, Lu, Burr, and Peterson [12] investigated physical implementations of buses and found that meshes with short buses outperform those with long buses for permutation routing because short buses have shorter propagation delay. We assume that the propagation delay of the buses is one time step which is also assumed in [11] [1] [18]. As we will see, our algorithms can be tuned to use short, or constant-length buses.

In the bused mesh models we consider in this paper, 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. Upper and lower bounds for unrestricted distance routing on meshes with fixed and reconfigurable buses can be found in [11] [5] [14] [17] [20]. For instance, for meshes with fixed buses, Leung and Shende [11] proved that $2n/3$ is the tight worst case

0-8186-6427-4/94 $04.00 © 1994 IEEE

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 restricted distance permutation routing problem on bused meshes. In the following, we first define partitioned-bus mesh, and prove lower bounds for the routing with locality problem on bused meshes. Then we present our routing algorithms for local permutation routing in one- and two-dimensional partitioned-bus meshes.

Our partitioned-bus mesh is a variant of the mesh with fixed buses. In a two-dimensional partitioned-bus mesh, each row/column bus can be partitioned into several short buses and any two adjacent short buses are incident on a common processor. In addition, these short buses can be active simultaneously. Our model is similar to mesh with separable row and column buses studied in [16], except that adjacent short buses do not meet at a common processor in that model. Note that any algorithm designed for meshes with separable buses can be used on our partitioned-bus meshes without time loss. To enable bus partitioning, there are two locally controllable bus switches for the column buses and the row buses to which it attaches within each processor. Turning on the row bus switch connects the left and right row buses, and turning on the column bus switch connects the upper and lower column buses. In addition, each of these switches can be set independently before the entire computation starts. Figure 1 shows some examples of one- and two-dimensional partitioned-bus meshes, in which each row/column bus is partitioned into two shorter buses. The partitioned-bus mesh has a higher

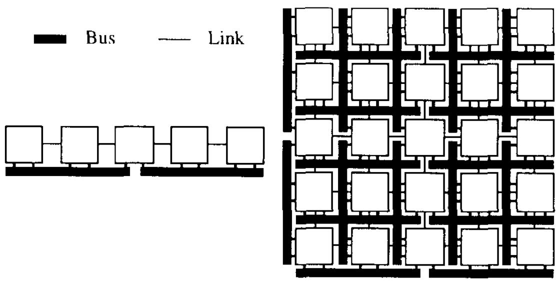

Figure 1: Examples of Partitioned-Bus Meshes

implementation cost than the mesh with fixed buses because of the switch overhead. However, the increase in cost may be justified by the gain in performance. Furthermore, the short-bus model is a candidate most suitable for being partitioned into bused submeshes to serve different computational requests. Partitioned-buses instead of short fixed buses are used because the best value of the bus length depends on the maximum source-destination distance. We will briefly discuss the possibility of using meshes with short fixed buses at the end of this paper.

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 fixed buses in less than d steps. The proof, shown below, is a generalization of Leung's and Shende's $2n/3$ time steps lower bound proof for the permutation routing problem on meshes with fixed buses [11].

For simplicity, we consider one-dimensional mesh with fixed buses and d is even in the proof, which can be easily generalized to two- and higher-dimensional cases. Consider processor with index i, where $i \in [1, 2, \ldots, d/2]$, which has a packet to send to processor $i + d$ and vice versa. Any 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 every time step. Thus, any algorithm needs at least d time steps for this problem. The reason why this architecture is not suitable for restricted distance routing problems is that only one packet can ride the bus during any time step, and because of the packets' restricted distances, they cannot really afford to wait too long for the bus. This observation led us to consider another variant—the partitioned-bus mesh model. This model looks more promising in handling restricted distance packets because more than one packet can be riding a bus in a column or row simultaneously.

We now prove a lower bound for restricted distance permutation routing on one-dimensional partitioned-bus meshes. If $d \leq n/2$, consider the scenario that all the packets in a processor sub-array of length $2d$ send a packet to the processor which has distance d from it and within the same sub-array. In the middle of the sub-array, 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 partitioned-bus meshes. It can be proved by applying this one-dimensional construction to each row of the meshes. Moreover, we get a d steps lower bound if the middle link of this construction is mapped to the gap between two adjacent short buses in meshes with separable buses. In other words, buses in meshes with separable buses cannot help in this local routing problem.

3 Routing on One-Dimensional Partitioned-Bus Meshes

In this section, we present an asymptotically $(2d/3)$-step algorithm for the restricted distance permutation routing problem. The one-dimensional partitioned-bus mesh is assumed to be partitioned into segments of length b each (that is, each segment has b links in it), where b is an odd integer 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*.[1] 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*[2]

- *from a mesh link, P forwards the packet by mesh link;*
- *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 for 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) r distance units, it can take a bus to cover s units. 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 for 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. One may be tempted to give as many bus rides as possible to packets that have a long distance to travel instead of forcing them to walk. However, as it is shown later, with our particular design, the algorithm is asymptotically time optimal in the worst case. This design also makes the algorithm simpler to operate and analyze.

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. By choosing the segment length b to be 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 destination d, Algorithm 1 can be used to solve it in asymptotically $2d/3$ time steps and only a constant number of auxiliary buffers are required per processor.*[3]

Proof: Substitute d for D in Lemma 1 and choose b to be $o(d)$ and an odd integer. Note that only two auxiliary buffers are needed for each processor to hold the last packet arriving at it by bus. □

[1] In our model, we can always choose b to be an odd integer by setting the switches appropriately. In case we really have to choose an even b, we can use two additional buffers in each bus terminal to simulate a virtual neighboring processor.

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

[3] We assume that in each processor there are two buffers associated with each bidirectional link and one buffer for each bus attached to the processor. It is possible that all these buffers are filled at the start of the algorithm. Auxiliary buffers refer to any additional buffers that are needed other than these buffers.

4 Routing on Two-Dimensional Partitioned-Bus Meshes

The algorithm for permutation routing on two-dimensional partitioned-bus meshes has two essential 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 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 many-one routing problem on a linear array. In many-one routing, an arbitrary number of packets can start at 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. As before, we partition the array into segments of length b and assign one bus to each segment. 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 many-one routing with the restriction that all type i packets reside in type i terminals initially. Then we will describe how to perform many-one routing without this restriction.

Algorithm 2 *(Many-One 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 for 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 many-one 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 to start earlier. Packets are divided into batches according to their distances. Those with distances $(d-3b+1)$ to d are of the first batch and they start moving during the first $(2b+2)$ time steps. Then packets with distances $(d-6b+1)$ to $(d-3b)$ start in the second $(2b+2)$-step interval, and so on.

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. It is because we enforce that type i packets can leave type i terminals only at odd-numbered steps, while the right of leaving 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, time required for a packet with distance $(d-x)$ to reach its destination

[4] As we are interested in the case $b \lll d$, 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 for one time step after each bus ride.

= starting time + traveling time
$\leq 2x/3 + \lfloor x/3b \rfloor \times 2 + (d-x) - (\lfloor (d-x)/3b \rfloor \times b) + \lceil (d-x)/3b \rceil \times 2$
$\leq 2x/3 + \lfloor x/3b \rfloor \times 2 + (d-x) - (d/3 - \lfloor x/3b \rfloor \times b) + 2d/3b - \lfloor x/3b \rfloor \times 2$
$\leq 2d/3 + 2d/3b$
□

Lemma 2 shows us how to route any many-one problem in which type i packets start at type i bus terminals. In the following, we make use of this result to route any many-one 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 *(Many-One Routing without Restriction)*

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 many-one 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 many-one 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 were 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 the 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, Sorting + Greedy Algorithm, for solving permutation routing on two-dimensional partitioned-bus meshes.

Algorithm 4 *(Sorting+Greedy Algorithm)*

1. *Partition the partitioned-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 $n \times n$ partitioned-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

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

[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 $\{1, \cdots, n^2\}$ and the sorting problem with respect to this mapping is to move the i^{th} smallest packet to the i^{th} 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 $c_1 < c_2$.

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

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

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 [15]; performing column and row routing in parallel requires $2d(1 + 1/b)/3 + 3b + 6$ steps. It 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 [4]. □

5 Concluding Remarks

Routing with locality problems on partitioned-bus meshes have been studied. In particular, we have proposed an algorithm for the restricted distance permutation routing problem with maximum distance d_0 on an $n \times n$ partitioned-bus mesh which needs $2d(1 + 1/b)/3 + O(d_0/f(d_0)) + 3b + 6$ time steps and $O(f(d_0))$ buffers, where $d = d_0 + 2f(d_0)$. It is asymptotically time and buffer optimal if $f(d_0)$ is chosen as a large constant and $b = o(d_0)$. One third of the running time is saved after the partitioned-bus is employed. After setting b, our algorithms do not require any further reconfiguration of the buses. Thus, one may want to replace partitioned-buses by cheaper short fixed buses. By fixing b as a large constant, we can still achieve close to optimal results for large d. Our lower bound and upper bound results can be generalized to local multi-packet (or k-k) routing algorithms giving an asymptotically tight $2kd/3$ bound in one- and two-dimensional partitioned-bus meshes. In the k-k problem, each processor initially has at most k packets to send, and no more than k packets share the same destination. Finally, the results developed in this paper can also be used in partitioned-bus rectangular meshes and tori.

References

[1] A. Bar-Noy and D. Peleg, "Square Meshes are not always Optimal". *IEEE Transactions on Computers*, Vol. 40, No. 2, February 1991, pp. 196-203.

[2] Y. Ben-Asher, D. Peleg, R. Ramaswami, and A. Schuster, "The Power of Reconfiguration". *Journal of Parallel and Distributed Computing*, Vol. 13, 1991, pp. 139-153.

[3] S.H. Bokhari, "Finding Maximum on an Array Processor with a Global Bus". *IEEE Transactions on Computers*, Vol. C-33, No. 2, February 1984, pp. 133-139.

[4] S. Cheung, and F.C.M. Lau, "Mesh Permutation Routing with Locality". *Information Processing Letters*, Vol. 43, 1992, pp. 101-105.

[5] S. Cheung, and F.C.M. Lau, "A Lower Bound for Permutation Routing on Two-Dimensional Bused Meshes". *Information Processing Letters*, Vol. 45, 1993, pp. 225-228.

[6] M. Kaufmann, and J.F. Sibeyn, "Optimal Multi-Packet Routing on the Torus". *Proceedings of the 3^{rd} Scandinavian Workshop on Algorithm Theory. Lecture Notes in Computer Science 621*, Springer-Verlag, July 1992, pp. 118-129.

[7] D. Krizanc, S. Rajasekaran, and T. Tsantilas, "Optimal Routing Algorithms for Mesh-Connected Processor Arrays". *Proceedings of the 3^{rd} Aegean Workshop on Computing: VLSI Algorithms and Architectures. Lecture Notes in Computer Science 319*, Springer-Verlag, June 1988, pp. 411-422.

[8] M. Kunde, "Routing and Sorting on Mesh-Connected Arrays". *Proceedings of the 3^{rd} Aegean Workshop on Computing: VLSI Algorithms and*

Architectures. Lecture Notes in Computer Science 319, Springer-Verlag, June 1988, pp. 423-433.

[9] M. Kunde, and T. Tensi, "(k-k) Routing on Multidimensional Mesh-Connected Arrays". *Journal of Parallel and Distributed Computing*, Vol.11, 1991, pp.146-155.

[10] T. Leighton, F. Makedon, and I.G. Tollis, "A 2n − 2 Step Algorithm for Routing in an n × n Array with Constant Size Queues". *Proceedings of the 1989 ACM Symposium on Parallel Algorithms and Architectures*, pp. 328-335.

[11] J.Y.T. Leung and S.M. Shende, "On Multidimensional Packet Routing for Meshes with Buses". *Journal of Parallel and Distributed Computing*, Vol. 20, No. 2, February 1994, pp. 187-197.

[12] Y.W. Lu, J.B. Burr, and A.M. Peterson, "Permutation on the Mesh with Reconfigurable Bus: Algorithms and Practical Considerations". *Proceedings of the 7th International Parallel Processing Symposium*, April 1993, pp. 298-308.

[13] V.K. Prasanna Kumar and C.S. Raghavendra, "Array Processor with Multiple Broadcasting". *Journal of Parallel and Distributed Computing*, Vol. 4, 1987, pp. 173-190.

[14] S. Rajasekaran, "Mesh Connected Computers with Fixed and Reconfigurable Buses: Packet Routing, Sorting, and Selection". *Proceedings of the 1st Annual European Symposium on Algorithms. Lecture Notes in Computer Science 726*, Springer-Verlag, October 1993, pp. 309-320.

[15] C.P. Schnorr, and A. Shamir, "An Optimal Sorting Algorithm for Mesh Connected Computers". *Proceedings of the 18th Annual ACM Symposium on Theory of Computing*, 1986, pp. 255-263.

[16] M.J. Serrano, and B. Parhami, "Optimal Architectures and Algorithms for Mesh-Connected Parallel Computers with Separable Row/Column Buses". *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 10, October 1993, pp. 1073-1080.

[17] J.F. Sibeyn, M. Kaufmann, and R. Ramen, "Randomized Routing on Meshes with Buses". *Proceedings of the 1st Annual European Symposium on Algorithms. Lecture Notes in Computer Science 726*, Springer-Verlag, October 1993, pp. 333-344.

[18] Q.F. Stout, "Mesh-Connected Computers with Broadcasting". *IEEE Transactions on Computers*, Vol. C-32, No. 9, September 1983, pp. 826-830.

[19] T. Suel, "Optimal Deterministic Routing and Sorting on Mesh-Connected Arrays Processors". Technical Report TR-93-18, Department of Computer Sciences, University of Texas at Austin, September 1993.

[20] T. Suel, "Routing and Sorting on Meshes with Row and Column Buses". Technical Report TR-94-09, Department of Computer Sciences, University of Texas at Austin, April 1994.

[21] L.G. Valiant, and G.J. Brebner, "Universal Schemes for Parallel Communication". *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, 1981, pp. 263-277.