Math. Systems Theory **29**, 311–330 (1996)

**Mathematical Systems Theory**



# Flit-Serial Packet Routing on Meshes and Tori

F. Makedon[1] and A. Symvonis[2]

[1] Department of Computer Science, Dartmouth College, Hanover, NH 03755, USA
makedon@dartmouth.edu

[2] Basser Department of Computer Science, University of Sydney, Sydney, N.S.W. 2006, Australia
symvonis@cs.su.oz.au

**Abstract.** In this paper we consider the *flit-serial packet-routing problem*, where each packet consists of a sequence of $k$ *flits* and is, thus, called a *snake*. Based on the properties of the snake during the routing, we give a formal definition of three different packet-routing models, namely, the *store-and-forward* model, the *cut-through* model, and the *cut-through with partial cuts* model. Surprisingly, all previous work has focused on the store-and-forward model. We also introduce the *restricted cut-through* model, which is unrealistic, but is proved to be a very powerful tool in the effort to bound the time required by a routing problem. We study the cut-through with partial cuts model which is most commonly used in practice. We present the first algorithms, deterministic and probabilistic, based on this model for the permutation routing problem on a chain, on a square mesh, and on a square torus.

## 1. Introduction

The development of efficient packet-routing algorithms is a very important task in the construction of parallel computers. In a parallel architecture different computing elements must be able to communicate with each other fast. This means that each processor must be able to route packets of information fast. Furthermore, for the routing scheme to be efficient, the size of the queues that will be created during the routing must be bounded.

In this paper we have chosen to study the packet-routing problem on the mesh architecture because the two-dimensional layout of most implementation technologies suggests a two-dimensional grid topology which makes it easy to interconnect the pro-

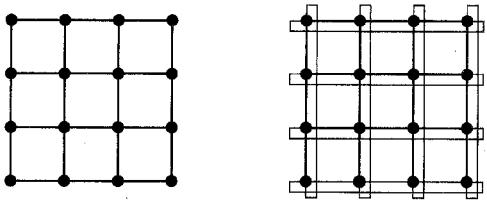


**Fig. 1.** A 4 × 4 mesh (a) and a 4 × 4 torus (b).

cessors. Since the distance between neighbouring processing elements is constant on the grid, the time needed for communication between any pair of connected elements is also constant. Furthermore, although the grid has a large diameter $(2n - 2)$, its topology is well matched with many problems.

An $n \times n$ *mesh of processors* is defined to be a graph $G = (V, E)$, where $V = \{(i, j) | 1 \leq i, j \leq n\}$ and an edge $e = ((i, j), (k, l))$ belongs to $E$ if $|k - i| + |l - j| = 1$. Processor $(i, j)$ will be also denoted by $P_{i,j}$. The $n \times n$ mesh is illustrated in Figure 1(a). At any one step, each processor can communicate with all of its neighbours by the use of bidirectional links (channels). We define the *distance* between two processors $P = (i, j)$ and $Q = (k, l)$ as $distance(P, Q) = |k - i| + |l - j|$. If $distance(P, Q) = 1$, then $P$ and $Q$ are *neighbours*. If the mesh is supported with wrap-around connections, then the resulting network is called *torus* [3] (Figure 1(b)). For the torus, we have that $distance\_torus(P, Q) = \min(n - |k - i|, |k - i|) + \min(n - |l - j|, |l - j|)$. The *width of the channel* is defined to be the number of bits that can be transmitted in one step from a processor to one of its neighbours. Processors have a local storage area where they can queue incoming packets. The processors are assumed to work in a synchronous MIMD model.

In a *permutation* problem, each processor has one packet to transmit to another processor. At the end, each processor receives exactly one packet (1–1 routing). The problem here is to route all packets to their destinations fast and without the use of large additional queueing area (buffers) in each processor.

Usually, the size of a packet is larger than the number of bits the channel can accommodate in one step. This forces us to break each packet into smaller pieces, called *flits*. So, we can think of a packet as an ordered collection of $k$ flits, where $k$ depends on the width of the channel and the size of the entire packet.

The packet-routing problem can be studied by two different approaches. In the first one we consider the $k$ flits as $k$ distinct packets, which we try to route independently (*independent split approach*). There are several problems with this approach. First, each processor will receive $k$ flits which it must be able to combine in order to retrieve the initial packet. Since the order in which the flits are received is not fixed, we must associate with each flit a *flit number* that will indicate the position of the flit in the initial packet. Secondly, since each flit is routed independently from the others, a destination address must be attached to it. So, if the constants of the problem (packet-size and width of channel) require to break the initial packet into $k$ flits , then $(\log k + 2 \log n)$ bits are to be stored in each flit for bookkeeping information, on top of actual message information.

However, another problem arises if the width of the channel is less than $(\log k + 2 \log n)$ bits. In this case this scheme cannot work, since there is no space left within a flit to host any part of the initial message.

The second approach (*flit-serial routing*) [1], [4], [9], [18], [19] is to consider the $k$ flits as a *snake*. All flits will follow the first one, called the *head*, which knows the destination address. Moreover, the snake cannot be broken, i.e., consecutive flits in the initial packet will always be at the same or at adjacent processors. The last flit of the snake is called the *tail*. Note that in the flit-serial routing approach no additional space is needed for information used to combine flits when they arrive at their destination. The order in which the flits arrive at their destination specifies their position in the initial packet.

More formally, at time $t$, a $k$ flit snake $s$ can be represented by the $(k+1)$-tuple $[s_1, s_2, .., s_i, .., s_k, t]$, where $s_i$ is a processor and $s_i$, $s_{i+1}$ are identical or adjacent processors, $i = 1, \ldots, k-1$. We define the *length of snake s at time t* to be $length(s,t) = \sum_{i=1}^{k-1} distance(s_i, s_{i+1}) + 1$, that is, the number of processors over which the snake is distributed. We say that a snake $s$ is travelling in *full extension* at time $t$, if $length(s,t) = k$. For a snake $s$ at time $t$, we define the function $collisions(s,t)$ to be $collisions(s,t) =$ (number of processors that hold more than one flit of snake $s$ at time $t$). The processors that have more than one flit of the same snake at a given time are called *collision points*. If a snake is travelling in full extension, then $collisions(s,t) = 0$, i.e., there are no processors that hold two or more flits of that snake. Using the previously defined functions *length*( ) and *collisions*( ), we can now distinguish the following three types of flit-serial routing models:

- *Store-and-forward model.* A snake $s$ that is routed using the store-and-forward model at any time $t$ has $length(s,t) \leq 2$ (Figure 2(a)).
- *Cut-through model.* A snake $s$ that is routed using the cut-through routing model has, at any time $t$, $collisions(s,t) \leq 2$. Moreover, the collision points are at the processors that hold the head or the tail of the snake (Figure 2(b)).
- *Cut-through with partial cuts model.* In the cut-through with partial cuts routing model, for any snake $s$ at time $t$, we have $0 \leq collisions(s,t) \leq k/2$, i.e., we may have as many as $k/2$ collision points in snake $s$. Observe that in the cut-through routing model, whenever a collision occurs, the whole snake must reach the processor at the collision point before the snake continues its routing. This does not hold for cut-through with partial cuts routing (Figure 2(c)).

We should point out here that a standard terminology about the different routing models does not exist in the literature. For example, the cut-through with partial cuts model was first described in [9] by Kermani and Kleinrock. However, it is referred to as *wormhole* in [18] by Ngai. A different definition for wormhole routing is given by Dally and Seitz in [4]. The most frequently used definitions are those found in [4] and [9].

Because of its restricted nature, the store-and-forward model is probably the simplest to analyse of the three flit-serial models defined above. In fact, all of the previous work on the permutation routing problem is based on the store-and-forward model. For example, the authors in [21], [11], [16], [17], [20], and [24] consider the case where all flits of a packet are transmitted in one step. These results can be immediately interpreted to the store-and-forward model if we multiply the final time complexity by the number of flits

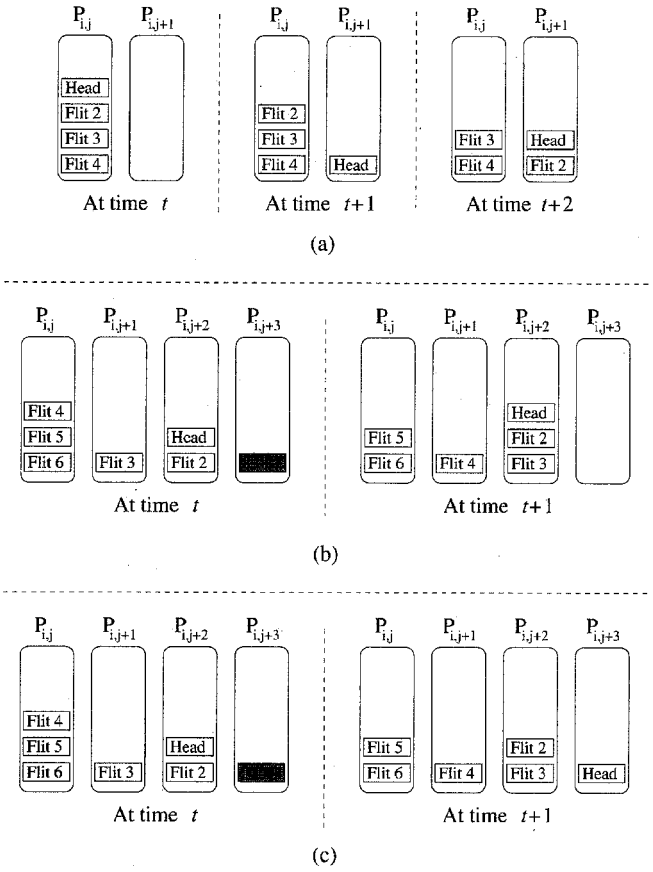


**Fig. 2.** Three flit-serial routing models: (a) The store-and-forward model, (b) the cut-through model, and (c) the cut-through with partial cuts model.

in a packet. There have been deterministic [11], [16], [20] and probabilistic [21], [24] approaches based on this model. The trivial greedy algorithm routes the packets to the correct column and then to the correct row in $2n-2$ steps. The size of the queues, however, can be as bad as $2n/3$ [15]. An average case analysis of several greedy algorithms on arrays under a variety of assumptions can be found in [14]. In that paper, Leighton shows that certain greedy algorithms perform surprisingly well on the average. The nontrivial solutions given to this problem are based on parallel sorting algorithms [22], [23]. Kunde [11] was the first to use parallel sorting to obtain a deterministic algorithm that completes the routing in $2n + O(n/f(n))$ steps and with queues of size $O(f(n))$. Later, Leighton *et al.* [16] derived a deterministic algorithm that completes the routing in $2n-2$ steps using constant-size queues of about 1000 packets. The queue size was reduced to about 150 packets[1] by Rajasekaran and Overholt [20]. Probabilistic algorithms have been derived

[1] Originally a queue size of 58 packets was claimed, but an error in there analysis was reported in [2].

as well. The first one, derived by Valiant and Brebner [24], completed the routing in $3n + o(n)$ steps using $O(\log n)$-size queues. Later, Krizanc *et al.* [10], [21] derived an algorithm that routes the packets in $2n + O(\log n)$ steps using constant-size queues. All probabilistic algorithms route the packets correctly with high probability. Finally, a heuristic based on the *odd–even transposition* method which requires small queues of constant size was designed by the authors [17]. Experimental results on randomly generated data show that the number of steps required to complete the routing is almost equal to the maximum distance a packet has to travel.

Kunde [11] considered the problem of permutation routing by splitting a packet into $k$ flits. He derived that the routing can be completed after $kn + o(n)$ routing steps. However, he used the $k$-independent split approach, where each of the $k$ flits of a packet is routed independently from the others and an overhead of information is required.

As was indicated earlier, all previous work has focussed on the store-and-forward model or on routing all of the $k$ flits independently. Here we study the cut-through with partial cuts model which is the one most commonly used in practice. We present the first algorithms based on this model for the permutation routing problem on a chain, on a square mesh, and on a square torus.

The remainder of the paper studies the packet[2]-routing problem using the cut-through with partial cuts routing model. The paper is divided into sections as follows. In Section 2 we study the permutation routing on a chain of processors, as well as two more general routing problems on a chain, namely, the *many-to-one* and the *one-to-many* routing problems. In Section 3 we study the routing problem on the mesh. We describe an algorithm that is based on different colourings of the packets. We give a deterministic colouring that results in an algorithm that completes the routing after $3nk/2 + o(n)$ routing steps, using queues of size $3f(n)/2$ packets. A random colouring of the packets results in an algorithm that completes the routing in $nk + o(n)$ routing steps with high probability. We also show that if we break the packet into two snakes instead of one, we can reduce the number of routing steps for the deterministic algorithm. In Section 4 we examine the permutation routing problem on the torus. We conclude in Section 5.

## 2. Routing on a Chain of Processors

In this section we describe a simple algorithm based on the cut-through with partial cuts routing model for the permutation routing problem on a chain of $n$ processors. We give an algorithm that completes the routing after $n(k + 1)/2 + k$ steps, for any permutation routing problem. We also state and prove, using the same model, some lemmata concerning routing on a chain of processors, where the initial and the final distributions of the packets on the chain do not form a permutation problem. These lemmata are used later, when we examine the routing in the two-dimensional mesh of processors.

[2] In the rest of the paper the terms "snake" and "packet" are used interchangeably.

### 2.1. *Permutation Routing on a Chain*

Each processor in the chain has a FIFO queue associated with each one of its two channels (bidirectional edges) that connects it with its left and right neighbours. Such a queue can hold up to $k$ flits. Initially, each processor places the packet it has to transmit in the queue associated with the channel the packet has to cross in order to approach its destination.

**Algorithm** *Route_on_a_Chain*

At step 0: Each processor transmits the head of the snake towards its destination.
At step $i$, $i > 0$: Each processor:

1. Appends the flits it received at step $i-1$ to the end of the appropriate queues.
2. Transmits the flits which are at the front of the queues.

Observe that with this algorithm each processor always transmits if it has at least one flit in its queue. This has the effect that no packet can over pass another one during its trip to the destination.

In order to prove that routing can be completed in $n(k+1)/2 + k$ steps, we define a "weaker" routing model which we call the *restricted cut-through* model.

**Definition.** We say that routing is done using the *restricted cut-through* model, if, for every snake $s$, a time instance $t = lk$ exists such that:

1. $length(s, t') \leq 2$, for any $t' \leq t$.
2. $length(s, t+i) = i+1$, for any $0 \leq i \leq k-1$.
3. $length(s, t') = k$, for any $t' \geq t+k-1$.
4. $length(s, t''+i) = k-i$, for any $0 \leq i \leq k$, where $t''$ is the step when the head of snake $s$ reaches its destination.

In other words, a packet gains full extension at time $t$ and maintains that full extension until it reaches its destination. Before time $t$, the snake is routed (as condition 1 imposes) according to the store-and-forward model. Also, transmission of a packet can start only at time instances which are multiples of the number of flits in a packet. A collision occurs at time $t$ at processor $P$, if flits that belong to two different packets are in the FIFO queue of processor $P$ at time $t$.

**Lemma 1.** *When the routing on a chain of n processors is performed using the restricted cut-through model, no collision can occur after step $nk/2$.*

*Proof.* We assume that each processor is equipped with an oracle that helps it to decide when is the appropriate time for a packet to gain full extension according to the rules of the restricted cut-through model. Without loss of generality, consider packets that go from left to right. Let them form the sequence $A = (P_m, P_{m-1}, \ldots, P_1)$, $m \leq n$. If the length of sequence $A$ is less than or equal to $n/2$, the lemma is true since, every $k$ steps, at least one packet gains full extension. Now, we examine the case when the length of sequence $A$ is greater than $n/2$. Assume that a collision occurs after $(n/2+j)k$

steps, $j \geq 0$. Say that packet $P$ collided with packet $Q$. This happened because $P$ did not gain full extension by time $(n/2 + j)k$, $j \geq 0$. In turn, $Q$ did so in order to avoid a collision with packet $R$ which did not gain full extension by time $(n/2 + j)k - k$, $j \geq 0$, and so on. Thus, in sequence $A$ we can identify the packet that had the collision and also $(n/2 + j)$ packets that caused the collision (since, at least full extension is obtained every $k$ steps). We now concentrate on those packets. They form the subsequence $B = (Q_{(n/2)+j+1}, Q_{(n/2)+j}, Q_{(n/2)+j-1}, \ldots, Q_1)$, where $Q_{(n/2)+j+1}$ is the packet that had the collision. Exactly one of these packets is gaining full extension every $k$ steps (starting from $Q_1, Q_2, \ldots$). Let $I_\lambda$ be the initial position (processor) of packet $Q_\lambda$, of sequence $B$. Observe that the destination of packet $Q_\lambda$, initially at $I_\lambda$, is to the right of position $(I_{\lambda-1}+\lambda)$. ($Q_\lambda$ did not gain full extension in order to avoid a collision with $Q_{\lambda-1}$. If the destination of packet $Q_\lambda$ was to the left of position $(I_{\lambda-1}+\lambda)$ a collision would not be possible after step $\lambda k$.) Thus, all packets are going to the right of position $n/2 + j$. However, there are only $n/2 - j$ such positions to accommodate $n/2 + j + 1$ packets. Since our initial problem was a permutation problem, we run into a contradiction. So, our assumption that a collision can occur after step $nk/2$ was false. This completes the proof. □

**Lemma 2.** *If the routing on a chain of n processors is performed by Algorithm* Route_on_a_Chain *using the cut-through with partial cuts routing model, no collision can occur after step* $nk/2$.

*Proof.* At any step of the routing, all flits of a packet in the restricted cut-through model are further from their destination, compared with the flits of the same packet in the cut-through with partial cuts model. From Lemma 1, we know that $nk/2$ steps are enough for the restricted cut-through model. Thus, $nk/2$ steps are also enough for the cut-through with partial cuts model. □

**Theorem 1.** *Using the cut-through with partial cuts routing model, the permutation routing problem on a chain of n processors, where each packet consists of k flits, can be completed after* $n(k + 1)/2 + k$ *steps.*

*Proof.* Use Algorithm *Route_on_a_Chain*. It guarantees that the snakes remain connected during the routing. From Lemma 2, we know that no collision can occur after step $nk/2$. The maximum distance a packet has still to travel is $n/2$. This corresponds to the case where the initial position of the packet is at one end of the chain, and the destination at the other. After $nk/2$ steps, the whole packet has already travelled at least half of the distance. Now it can gain full extension. So, it needs $n/2 + k$ more steps for its tail to reach its destination. Thus, the total number of steps required is $nk/2 + n/2 + k = n(k + 1)/2 + k$. □

### 2.2. *General Routing on a Chain*

In this section we define and examine two more general routing problems in a chain of $n$ processors. Later, we use these results in the algorithm for the permutation routing problem on a square mesh.

**Problem 1** (Many-to-One Routing). Initially, each processor has at most one packet to route. At the end, a processor can receive more than one packet.

**Problem 2** (One-to-Many Routing). Initially, a processor might have more than one packet to route. At the end, each processor receives at most one packet.

**Theorem 2.** *Using the cut-through with partial cuts routing model, the many-to-one routing problem on a chain of n processors can be solved in* $(k-1)m+n$ *steps, where m is the total number of packets on the chain, and k is the number of flits in a packet.*

*Proof.* Route the packets using Algorithm *Route_on_a_Chain*. To show that $(k-1)m+n$ steps are enough to complete the routing, we use again the restricted cut-through model. We want to prove that, using this model, routing will be completed after $(k-1)m+n$ steps. Then we can conclude that $(k-1)m+n$ steps are enough to route all packets, even when we are using the cut-through with partial cuts model. Without loss of generality, consider packets that are going from left to right. In the worst case, all $m$ packets belong to that category. The use of the restricted cut-through model guarantees that in every $k$ steps, at least one packet gains full extension. So, after $km$ steps, all packets have gained full extension. Consider now the tail of the leftmost packet. It can be at most $n-m$ steps away from its destination. Thus, we need $mk+(n-m)=(k-1)m+n$ steps to complete the routing. □

**Theorem 3.** *Using the cut-through with partial cuts routing model, the one-to-many routing problem on a chain of n processors, can be solved in* $(k-1)m+n$ *steps, where m is the total number of packets on the chain, and k is the number of flits in a packet.*

*Proof.* The routing algorithm is a modification of Algorithm *Route_on_a_Chain*. However, we now have a queue, since, initially, a processor might have more than one packet to send. At any instant, when a processor has just completed the transmission of the last flit of a packet, it starts the transmission of the packet that has to go farthest. In order to prove that $(k-1)m+n$ steps are enough to complete the routing we use again the restricted cut-through model. Without loss of generality, consider only the packets that have to move to the right. Consider such a packet before it gains full extension, and at time instances that are multiples of $k$. It is either delayed by another packet that has to go further, or it is advanced to the next processor (using the store-and-forward model). We assume that, until the time it gains full extension it is delayed $\mu$ times. This implies that by time $mk$ it advances $m-\mu$ times. If, by $m-\mu$ advances, it reaches its destination we are done. Otherwise, by time $mk$ its tail will be at most $n-\mu$ steps away from its destination. To see that, observe that since it was delayed by $\mu$ packets, it is not destined for the rightmost $\mu$ processors. Also it has advanced $m-\mu$ positions closer to its destination. By combining these two facts, we get that its tail still has to travel at most $(n-\mu)-(m-\mu)=n-m$ positions to the right. This implies that the routing will be completed after $mk+(n-m)=(k-1)m+n$ steps. □

Before we proceed, we have to emphasize that the restricted cut-through model that was used in the proofs of the first three theorems is not used by our algorithms. This model is an unrealistic one because it is provided with the ability to guess whenever or not a collision will occur in the future routing of any packet. However, it turns to be a very useful tool for the proof of these theorems.

## 3. Routing on an $n \times n$ Mesh of Processors

In this section we describe an algorithm for the permutation routing problem on an $n \times n$ mesh of processors. The algorithm is based on a colouring of the packets with two different colours, *black* and *white*. From now on, when we talk about a *black* (*white*) *packet* (*or snake*), we mean a packet (or snake) that is coloured black (white). We present two simple deterministic colouring strategies which guarantee that the routing can be completed after $3nk/2 + O(kn/f(n))$ steps. Using this algorithm, at most $3f(n)/2$ packets are queued at any one processor at any one time. A third algorithm which relies on random colourings completes the routing with high probability in $nk+o(n)$ steps using queues of the same size. The technique used to bound the queue size at a processor was introduced by Kunde in [11] and has been used by several other researchers [15], [20]. Typical functions that we can use as $f(n)$ are the $\log n$ and the $\sqrt{n}$ functions. Provided that an efficient colouring exists, and that it can be computed quickly, our algorithm can route all packets in $nk + O(kn/f(n))$ steps. Finally, we give a deterministic algorithm that achieves this time by breaking each snake into two equal parts and routing them separately (as opposed to Kunde's $k$-independent splitting).

### 3.1. *The Algorithm*

Our routing algorithm needs to sort the packets located in square submeshes as one of its steps. Two different orderings are used, namely, the *row-wise row-major* and *column-wise column-major* orderings. So, before we proceed with the description of the routing algorithm, we define these orderings. For the purposes of this paper, we assume that the packets to be sorted are $n^2$ pairs of integers $(i, j)$, $0 \leq i, j \leq n-1$. The term *row-wise* means that after sorting, the $(in + j + 1)$th smallest element is located at processor $P_{i+1,j+1}$, $0 \leq i, j \leq n-1$.[3] The term *column-wise* means that after sorting, the $(in + j + 1)$th smallest element is located at processor $P_{j+1,i+1}$, $0 \leq i, j \leq n-1$. The terms *row-major* and *column-major* refers to the ordering relation between packets. Assuming that the packets to be sorted are distinct, in *row-major* ordering $(i, j) < (k, l)$ if and only if $i < k$ or $(i = k$ and $j < l)$ while, in *column-major* ordering $(i, j) < (k, l)$ if and only if $j < l$ or $(j = l$ and $i < k)$. In the case where the pairs to be sorted are not distinct, ties are broken arbitrarily. Figure 3(a) shows the packets in a $4 \times 4$ mesh after being sorted (according to their destinations) in *row-wise row-major* order. Figure 3(b) shows the *column-wise column-major* ordering. Note that in this example more than one packets are destined for a single processor. In the case that all destinations are distinct, the results look identical (Figure 3(c)).

[3] Recall that rows/columns of the mesh are numbered from 1 to $n$.

| | | | |
|---|---|---|---|
| (1,1) | (1,1) | (1,1) | (1,1) |
| (1,1) | (1,1) | (1,1) | (1,1) |
| (2,4) | (2,4) | (2,4) | (2,4) |
| (3,3) | (3,3) | (3,3) | (3,3) |

(a)

| | | | |
|---|---|---|---|
| (1,1) | (1,1) | (3,3) | (2,4) |
| (1,1) | (1,1) | (3,3) | (2,4) |
| (1,1) | (1,1) | (3,3) | (2,4) |
| (1,1) | (1,1) | (3,3) | (2,4) |

(b)

| | | | |
|---|---|---|---|
| (1,1) | (1,2) | (1,3) | (1,4) |
| (2,1) | (2,2) | (2,3) | (2,4) |
| (3,1) | (3,2) | (3,3) | (3,4) |
| (4,1) | (4,2) | (4,3) | (4,4) |

(c)

**Fig. 3.** Row-wise row-major (a) and (c) and column-wise column-major (b) and (c) orderings.

The algorithm for the permutation routing problem on an $n \times n$ mesh using the cut-through with partial cuts model is based on a colouring strategy and is as follows:

**Algorithm** *Route_on_a_Square_Mesh*

Step 1. Using a colouring strategy, colour each packet black or white.
Step 2. 1. Divide the $n \times n$ mesh into $n/f(n) \times n/f(n)$ submeshes.
2. For all submeshes, in parallel do:
(a) Sort all black packets in row-wise row-major order.
(b) Sort all white packets in column-wise column-major order.
(Note that, after the sorting, a processor may contain a black and a white packet.)
Step 3. In parallel do:
1. Route all black packets vertically, until they reach their row destination.
2. Route all white packets horizontally, until they reach their column destination.
Step 4. When the routing of step 3 is completed, in parallel do:
1. Route all black packets horizontally, until they reach their destination.
2. Route all white packets vertically, until they reach their destination.

**Definition** A colouring is said to be an $(a_1, a_2, b_1, b_2)$-*colouring* if and only if:

1. Initially, at most $a_1$ white packets are in any row.
2. Initially, at most $a_2$ black packets are in any column.
3. At most $b_1$ white packets are destined for any column.
4. At most $b_2$ black packets are destined for any row.

The following lemma shows that the sorting phase of Algorithm *Route_on_a_Square_Mesh* does not destroy the initial colouring, but slightly modifies it.

**Lemma 3.** *Assume an arbitrary* $(u, v, w, x)$*-colouring of the packets. Then the division of the mesh into* $n/f(n) \times n/f(n)$ *submeshes and the individual sorting of each submesh results in a* $(u + f(n), v + f(n), w, x)$*-colouring, provided that* $u + f(n) \leq n$ *and* $v + f(n) \leq n$.

*Proof.* Obviously, $w$ and $x$ will remain the same. This is because sorting is not changing the colour of any packet. It just redistributes the packets within a particular submesh. So, we just have to prove that $u$ and $v$ will become $u + f(n)$ and $v + f(n)$, respectively. We show this only for $u$. The proof for $v$ is similar. The mesh can be considered as a collection of $f(n)$ disjoint horizontal strips, each one consisting of $n/f(n)$ rows. Each strip consists of $f(n)$ submeshes. Without loss of generality, consider the first strip. Since initially we had a $(u, v, w, x)$-colouring, in the first strip there are at most $un/f(n)$ white packets.

Let $R_{ij}$ denote the number of white packets in the $i$th row of the $j$th submesh before the sorting, and $R'_{ij}$ after. The $j$th submesh contains

$$W_j = \sum_{i=1}^{n/f(n)} R_{ij}$$

white packets. After the sorting, we will have that $R'_{ij} \leq \lceil W_j/(n/f(n)) \rceil = \lceil f(n)W_j/n \rceil$. Thus, in the $i$th row, we will have at most

$$\begin{aligned}\sum_{j=1}^{f(n)} \left\lceil \frac{f(n)W_j}{n} \right\rceil &\leq \sum_{j=1}^{f(n)} \left( \frac{f(n)W_j}{n} \right) + f(n) \\ &= \frac{f(n)}{n} \sum_{j=1}^{f(n)} W_j + f(n) \\ &\leq \frac{f(n)}{n} \frac{un}{f(n)} + f(n) = u + f(n)\end{aligned}$$

white packets. Similarly, we can prove that the number of black packets in any column will be at most $v + f(n)$. So, after the sorting, we will have a $(u + f(n), v + f(n), w, x)$-colouring. □

**Corollary 1.** *Assume an arbitrary $(u, v, w, x)$-colouring. If, in each submesh, the number of white and the number of black packets are multiples of $n/f(n)$, then, after the sorting, we still have a $(u, v, w, x)$-colouring.*

*Proof.* We have that $W_j$ is a multiple of $n/f(n)$. This implies that $\lceil f(n)W_j/n \rceil = f(n)W_j/n$. We use this relation in the inequalities at the proof of Lemma 3, to prove the corollary. □

**Lemma 4.** *Assume an initial $(u, v, w, x)$-colouring. Then, if Algorithm* Route_on_a_Square_Mesh *is used, at most $(1 + \max(w, x)/n) f(n)$ packets might be queued at any processor, waiting for the same channel.*

*Proof.* We use sorting in order to bound the queue size, as introduced by Kunde in [11], where according to our definitions, all packets have the same colour. We prove the lemma only for packets that can wait for a vertical channel, i.e., white packets. Consider

any processor $P = (r, c)$. We compute the number of white packets that might be queued at processor $P$. We concentrate on the strip that contains processor $P$. Let $a_i$ denote the number of white packets after the sorting in the $i$th submesh of the strip, that are in the $r$th row and have destination in column $c$. We wish to compute

$$\sum_{i=1}^{f(n)} a_i.$$

In each submesh there are at least $(a_i - 1)n/f(n) + 1$ white packets. So,

$$\sum_{i=1}^{f(n)} \left\lfloor \frac{(a_i - 1)n}{f(n)} + 1 \right\rfloor \leq w.$$

By simple calculations, we obtain

$$\sum_{i=1}^{f(n)} a_i \leq \left(1 + \frac{w}{n}\right) f(n).$$

A similar result can be obtained for black packets. Thus, no queue can have more than $(1 + \max(w, x)/n) f(n)$ packets at any time. □

**Theorem 4.** *Using an initial $(u, v, w, x)$-colouring, Algorithm* Route_on_a_Square_Mesh *completes the routing after* $(k-1)[\min(\max(u+f(n), v+f(n)), n)+\max(w, x)]+2n + O(kn/f(n)) + t_{\text{colour}}$ *steps, using queues of size at most* $(1 + \max(w, x)/n) f(n)$ *packets each.*

*Proof.* Step 1 requires $t_{\text{colour}}$ routing steps. $t_{\text{colour}}$ depends on how complex the computations are which are needed to determine the colouring. Step 2 of the algorithm can be performed in $O(kn/f(n))$ routing steps, using any one of the algorithms for sorting in a square mesh given in [22] and [23]. Step 3 describes two many-to-one problems, one for black and one for white packets. By Theorem 2, we know that we can solve these problems in $(k - 1)[\min(\max(u + f(n), v + f(n)), n)] + n$ steps. Finally, step 4 describes two one-to-many routing problems. From Theorem 3, we know that we can solve the one-to-many problems in $(k - 1)[\max(w, x)] + n$ steps. By summing the time required for each step of Algorithm *Route_on_a_Square_Mesh*, we obtain that $(k-1)[\min(\max(u+f(n), v+f(n)), n)+\max(w, x)]+2n+O(kn/f(n))+t_{\text{colour}}$ steps are needed. The argument about the queue size being at most $(1 + \max(w, x)/n) f(n)$ packets was proved in Lemma 4. □

**Corollary 2.** *Assume an arbitrary colouring. If in each submesh the number of white and the number of black packets are multiples of $n/f(n)$, then Algorithm* Route_on_a_Square_Mesh *completes the routing after* $(k - 1)[\max(u, v) + \max(w, x)] + 2n + O(kn/f(n)) + t_{\text{colour}}$ *steps, using queues of size at most* $(1+\max(w, x)/n) f(n)$ *packets each.*

*Proof.* By using Corollary 1 in the proof of Theorem 4. □

### 3.2. *Three Possible Colourings*

In the following we describe three possible colourings of the packets. The first two colourings are derived in a deterministic way. These colourings can be considered to be symmetric and result in the same routing time. The algorithm that uses the first colouring needs queues of size $2f(n)$ packets. We show that use of the second colouring can reduce the size of the queues to $3f(n)/2$ packets. The third colouring is a random one. All three colourings can be obtained in $O(1)$ time.

*Colouring A.* Colour the processors in such a way that the mesh looks like a chessboard. A packet is coloured white (black), if it is initially at a white (black) processor. So, initially, there are exactly $\lceil n/2 \rceil$ white packets in any row, and exactly $\lceil n/2 \rceil$ black packets in any column. Observe that it is possible that $n$ white (black) packets have the same column (row) destination (Figure 4). Thus, colouring $A$ is a $(\lceil n/2 \rceil, \lceil n/2 \rceil, n, n)$-colouring. Also observe that, for even $n$, the number of white (black) packets in each submesh is a multiple of $n/f(n)$. Thus, after the sorting, we still have a $(n/2, n/2, n, n)$-colouring (Corollary 1). For odd $n$, after the sorting, only one row in each submesh has an extra packet compared with the others. We can arrange that at the $i$th submesh the extra packet will be at the $i$th row. So, after the sorting, we will have an $(\lceil n/2 \rceil, \lceil n/2 \rceil, n, n)$-colouring. Thus, Corollary 2 is correct for even values of $n$, and gives an underestimate by $k$ routing steps for odd values of $n$. So, at most $3nk/2 + n/2 + (k-1)f(n) + O(kn/f(n)) + O(1) + k = 3nk/2 + n/2 + o(n)$ steps are needed to complete the routing. The queues have size at most $2f(n)$ packets. A more careful analysis, one that takes into consideration that we never start the routing with any two adjacent black (white) packets, shows that we can save $n/2$ routing steps from the time required to solve

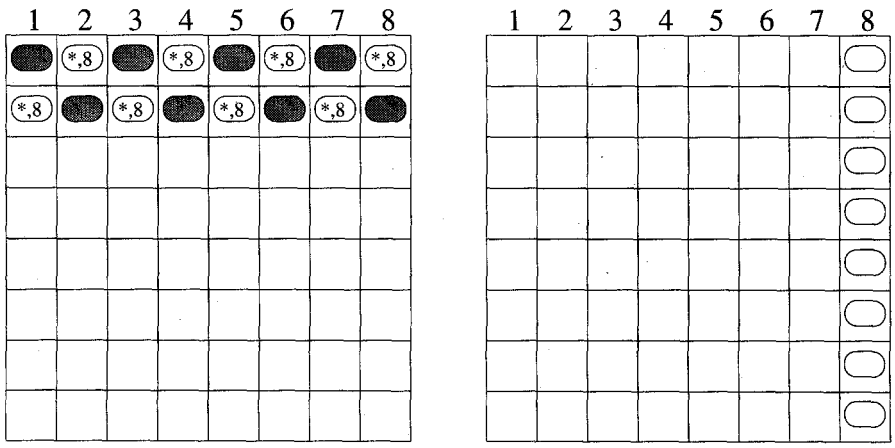


**Fig. 4.** Colouring $A$ is an $(n/2, n/2, n, n)$ colouring. (a) Before routing and (b) after routing.

the many-to-one problem of step 3 in our algorithm. To see that, consider the worst-case scenario for the many-to-one problem, namely when all $n/2$ packets are destined for the rightmost processor. Exactly $nk/2$ flits want to enter that processor, and they start doing so immediately. Also observe that, at every step, one flit reaches its destination. So, the many-to-one routing is completed after $nk/2$ steps. This implies that the total number of steps needed to complete the routing is $3nk/2 + o(n)$.

*Colouring B.* Colour the processors as before (chessboard). Now, colour a packet white (black), if it is destined for a white (black) processor. In this colouring exactly $n/2$ white (black) packets are destined for any column (row), but it is now possible that $n$ white (black) packets are initially in the same row (column) (Figure 5). Thus, colouring $B$ is an $(n, n, n/2, n/2)$-colouring. By Corollary 2, we have that routing can be completed after $3nk/2 + n/2 + o(n)$ routing steps, using queues of size $3f(n)/2$ packets. Again, a more careful analysis, one that takes into consideration that no two white (black) packets are destined for any two adjacent processors, shows that the one-to-many problem of step 4 in our algorithm can be solved in $nk/2$ steps. Thus, again, $n/2$ steps can be saved, and $3nk/2 + o(n)$ routing steps are enough to complete the routing.

*Colouring C.* Each processor decides the colour (black or white) of the packet it has by flipping a fair coin. The probability that a given packet is black (white) is 1/2. We show that:

**Lemma 5.** *If we colour all packets randomly with* $\Pr\{P = White\} = \Pr\{P = Black\} = 1/2$ *for every packet* $P$, *then the resulting colouring will be an* $(n/2 + a, n/2 + a, n/2 + a, n/2 + a)$*-colouring with high probability, where* $a = \sqrt{3n \ln 2n}$.

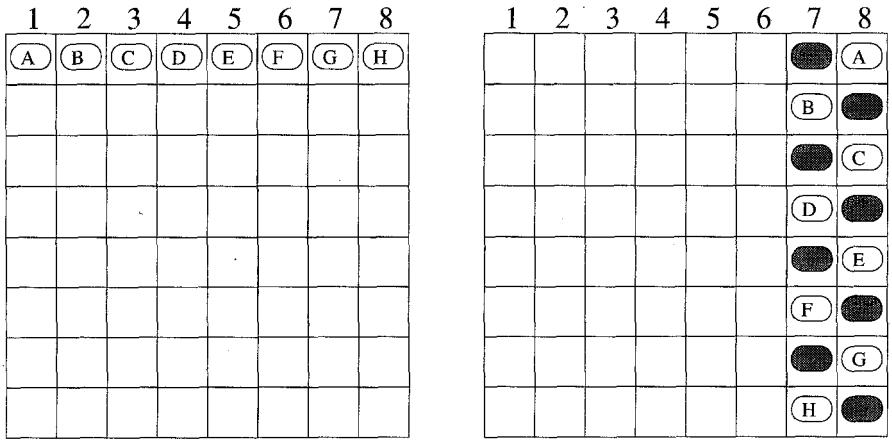


**Fig. 5.** Colouring $B$ is an $(n, n, n/2, n/2)$ colouring. (a) Before routing and (b) after routing.

*Proof.* Consider a sequence of $n$ Bernoulli trials where in the $i$th trial, for $i = 1, 2, \ldots, n$, success occurs with probability $p_i = p$ and failure occurs with probability $q_i = 1 - p$. If $X$ is a random variable that describes the total number of successes in this sequence of $n$ trials, then for $0 < a < np$ we have [5]

$$\Pr\{X \geq np + a\} \leq e^{-a^2/3np}.$$

In our case, success is considered to be the event of colouring a packet black and failure the event of colouring a packet white. For every packet $P_i$, $i = 1, 2, \ldots, n$, we have that $\Pr\{P_i = Black\} = p = 1/2$.

Thus, we have that

$$\Pr\left\{X \geq \frac{n}{2} + a\right\} \leq e^{-2a^2/3n}.$$

For $a = \sqrt{3n \ln 2n}$ we get:

$$\begin{aligned}
\Pr\left\{X \geq \frac{n}{2} + \sqrt{3n \ln 2n}\right\} &\leq \exp\left(\frac{-2(\sqrt{3n \ln 2n})^2}{3n}\right) \\
&= \exp\left(\frac{-2 \cdot 3n \ln 2n}{3n}\right) \\
&= \exp(-2 \ln 2n) \\
&= (2n)^{-2} \\
&= \frac{1}{4n^2}.
\end{aligned}$$

This means that, for any row, the probability of having more than $n/2 + a$ black packets is less than $1/4n^2$. In a similar way, we can prove that for any row the probability of having less than $n/2 - a$ black packets is less than $1/4n^2$. Thus, for any row, the probability to fail to have $X$ black packets, where $n/2 - a \leq X \leq n/2 + a$, is less than $1/2n^2$, for $a = \sqrt{3n \ln 2n}$. Thus, the probability to fail to have at every row $j$, $1 \leq j \leq n$, $X_j$ black packets, where $n/2 - a \leq X_j \leq n/2 + a$, is less than $n(1/2n^2) = 1/2n$, for $a = \sqrt{3n \ln 2n}$. Similarly, we can prove that the probability to fail to have at every column $j$, $1 \leq j \leq n$, $X_j$ white packets, where $n/2 - a \leq X_j \leq n/2 + a$, is less than $1/2n$, for $a = \sqrt{3n \ln 2n}$. Given that the above facts hold for both the origins of the packets as well as their destinations, we conclude that with probability $1 - 1/n$ the resulting colouring will be an $(n/2 + a, n/2 + a, n/2 + a, n/2 + a)$-colouring. □

From Theorem 4, we then conclude that the routing is completed with high probability in $nk + o(n)$ routing steps. Thus, we can now state the following theorem:

**Theorem 5.** *Using the cut-through with partial cuts routing model, a permutation routing problem on an $n \times n$ mesh of processors can be solved deterministically after $3nk/2 + o(n)$ routing steps, using queues of size at most $3f(n)/2$ packets. Furthermore, the problem can be solved in $nk + o(n)$ routing steps with high probability if we colour the packets randomly.*

### 3.3. *Breaking Each Packet into Two Snakes Can Help*

If we insist that our algorithm has to be deterministic we can reduce the routing time by breaking each packet into two snakes. The main characteristic of the first two colourings of the previous section is that they do not result in an algorithm that has "balanced" performance. In other words, colouring $A$ ($B$) achieves optimal routing time at step 3 (step 4) of the routing algorithm, and performs badly at step 4 (step 3). The ideal colouring for our algorithm would be an $(n/2, n/2, n/2, n/2)$-colouring. This colouring results in an algorithm that completes routing after $nk + o(n)$ steps.

We can obtain a balanced colouring by breaking each packet into two snakes, each one consisting of $k/2$ flits. For every packet, we colour one of its two snakes white and the other black. Now, we have an $(n, n, n, n)$-colouring which is balanced. (Observe that there are $2n^2$ snakes in the mesh). Using Corollary 2, since the number of black and white packets in each submesh is a multiple of $n/f(n)$, we conclude that routing can be completed after $(k/2-1)(n+n)+2n+o(n) = kn+o(n)$ steps, instead of $3nk/2+o(n)$ steps without snake-splitting. Thus, we have:

**Theorem 6.** *Using the cut-through with partial cuts model, if we break each packet into two snakes and route them separately, a permutation routing problem in an $n \times n$ mesh of processors can be solved after $nk + o(n)$ routing steps, using queues of size $f(n)$ packets.*

## 4. Routing on the Torus

In this section we show that the permutation routing problem on the torus can be solved deterministically after $3nk/4 + o(n)$ steps and with high probability after $nk/2 + o(n)$ steps. We also show that the breaking of each packet into four snakes can reduce the number of steps of the deterministic algorithm to $nk/2+o(n)$. Since the basic components of our algorithm are routines that do the routing on rings, we analyse two special forms of such routing.

### 4.1. *Routing on a Ring*

**Lemma 6.** *A many-to-one flit-serial routing problem on a ring of $n$ processors that has the property that packets are initially distributed on nonadjacent processors, can be solved in $kn/4$ routing steps, where each packet consist of $k$ flits.*

*Proof.* Consider the simple algorithm where each packet is routed using the shortest path to its destination and no over passing of packets that are routed in the same direction is allowed. The maximum distance that a packet has to travel is $n/2$. Since at the beginning no two packets are adjacent, in the worst case, at most $nk/4$ flits want to enter (and possibly exit from) any processor. Furthermore, they will start doing this immediately, and at every step one flit will enter any processor. This implies that after $nk/4$ steps the routing will be completed. □

**Lemma 7.** *Any one-to-many flit-serial routing problem on a ring of $n$ processors can be solved in $kn/2$ routing steps, where each packet consists of $k$ flits.*

*Proof.* We can achieve this routing time by using the store-and-forward model and the following method. Each processor computes, for each of the packets it holds, the direction of the shortest path to its destination. Consider any packet that has to travel a distance $\mu$ along the shortest path. Then the routing of the packet starts at time $(n/2 - \mu - 1)k$. By observing that no packets can compete for the same channel, we conclude that each packet will reach its destination at step $nk/2$. (Remember that the store-and-forward model is used.) □

### 4.2. *Extending Algorithm* Route_on_a_Square_Mesh *to Work on Tori*

**Theorem 7.** *Using the cut-through with partial cuts model, a permutation routing problem on an $n \times n$ torus can be solved deterministically after $3nk/4 + o(n)$ routing steps, using queues of size $2f(n)$ packets. Furthermore, the problem can be solved in $nk/2 + o(n)$ routing steps with high probability if we colour the packets randomly.*

*Proof.* For the deterministic part of the theorem we use colouring $A$. For the probabilistic part we colour the packets randomly. We then modify steps 3 and 4 of Algorithm *Route_on_a_Square_Mesh* so that packets are routed using the methods described in Lemmata 6 and 7. So, step 3 and step 4 of the deterministic algorithm require $nk/4$ and $nk/2$ routing steps, respectively. Thus, in total, we need $3nk/4 + o(n)$ routing steps. Again, the size of queues is at most $2f(n)$ packets, since colouring $A$ is an $(n/2, n/2, n, n)$-colouring. The proof for the probabilistic part of the theorem uses the result of Lemma 5. □

### 4.3. *Breaking Each Packet into Four Snakes Can Help*

We show that if we break each packet into four snakes of equal size, we can reduce the routing time of the deterministic algorithm from $3nk/4 + o(n)$ to $nk/2 + o(n)$ steps. We colour each snake of a packet using four different colours, say, black, white, red, and green. Let the black and green snakes of a given packet $i$ together form a new snake $BG_i$. Similarly, let the white and red snakes of a given packet $i$ form the snake $WR_i$. In the sorting stage of the routing algorithm (step 2), we sort the WR snakes of each submesh in column-wise column-major order, and the BG snakes of each submesh in row-wise row-major order. In step 3 we route the snakes as follows:

- *White snakes*: Horizontally, with direction to the east, until they reach their column destination.
- *Red snakes*: Horizontally, with direction to the west, until they reach their column destination.
- *Black snakes*: Vertically, with direction to the south, until they reach their row destination.
- *Green snakes*: Vertically, with direction to the north, until they reach their row destination.

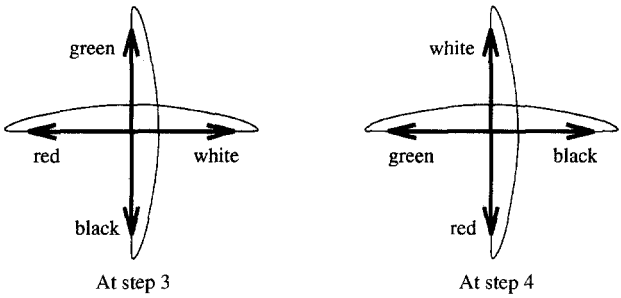


**Fig. 6.** The directions of the snakes.

In step 4 we route the snakes as follows:

- *White snakes*: Vertically, with direction to the north, until they reach their final destination.
- *Red snakes*: Vertically, with direction to the south, until they reach their final destination.
- *Black snakes*: Horizontally, with direction to the east, until they reach their final destination.
- *Green snakes*: Horizontally, with direction to the west, until they reach their final destination.

The moves that the snakes with different colours make are illustrated in Figure 6. Observe that at steps 3 and 4, one direction is dedicated to all snakes with the same colour. Step 3 will take $nk/4$ steps, since $n$ snakes of length $k/4$ each want to move to a given direction. For the same reason, step 4 will take at most $nk/4$ steps also. The queue size of any processor at each of the four channels is at most $f(n)/2$ packets. So, we have:

**Theorem 8.** *Using the cut-through with partial cuts model, if we break each packet into four snakes and route them separately, the permutation routing problem on an $n \times n$ torus, can be solved after $nk/2 + o(n)$ routing steps, using queues of size at most $f(n)/2$ packets.*

## 5. Conclusions

In this paper we studied the flit-serial routing problem on two-dimensional meshes and tori under the cut-through with partial cuts routing model. We presented the first deterministic and probabilistic algorithms for permutation routing. Since the submission of the original manuscript, several papers related to packet routing on meshes have appeared. A selective sample of the most important ones (in our opinion) includes [2], [6]–[8], and [12].

## Acknowledgement

We would like to thank Tom Leighton for suggesting these problems to us and for many helpful and useful discussions.

## References

[1] B. Aiello, F. T. Leighton, B. Maggs, M. Newman, Fast Algorithms for Bit-Serial Routing on a Hypercube, *Mathematical Systems Theory*, Vol. 24, pp. 253–271, 1991.

[2] B. Chlebus, M. Kaufmann, J. Sibeyn, Deterministic Permutation Routing on Meshes, *Proceedings of the Fifth Symposium on Parallel and Distributed Processing*, December 1993, pp. 814–821.

[3] W. J. Dally, C. L. Seitz, The Torus Routing Chip, *Distributed Computing*, Vol. 1, pp. 187–196, 1986.

[4] W. J. Dally, C. L. Seitz, Deadlock-Free Message Routing in Multiprocessors Interconnection Networks, *IEEE Transactions on Computers*, Vol. 36, No. 5, pp. 547–553, 1987.

[5] T. Hagerup, C. Rüb, A Guided Tour of Chernoff Bounds, *Information Processing Letters*, Vol. 33, pp. 305–308, 1990.

[6] C. Kaklamanis, D. Krizanc, S. Rao, Simple Path Selection for Optimal Routing on Processor Arrays, *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA* '92, June 1992, San Diego, CA, pp. 23–30.

[7] M. Kaufmann, S. Rajasekaran, J. Sibeyn, Matching the Bisection Bound for Routing and Sorting on the Mesh, *Proceedings of the Fourth Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA* '92, June 1992, San Diego, CA, pp. 31–40.

[8] M. Kaufmann, J. Sibeyn, T. Suel, Derandomizing Algorithms for Routing and Sorting on Meshes, *Proceedings of the Fifth Annual ACM–SIAM Symposium on Discrete Algorithms*, Arlington, VA, 1994, pp. 669–679.

[9] P. Kermani, L. Kleinrock, Virtual Cut-Through: A New Computer Communication Switching Technique, *Computer Networks*, Vol. 3, No. 4, pp. 267–286, 1979.

[10] D. Krizanc, S. Rajasekaran, Th. Tsantilas, Optimal Routing Algorithms for Mesh-Connected Processor Arrays, in *VLSI Algorithms and Architectures (AWOC* '88), J. Reif, editor, Lecture Notes in Computer Science, Vol. 319, pp. 411–422, Springer-Verlag, Berlin, 1988.

[11] M. Kunde, Routing and Sorting on Mesh-Connected Arrays, in *VLSI Algorithms and Architectures (AWOC* '88), J. Reif, editor, Lecture Notes in Computer Science, Vol. 319, pp. 423–433, Springer-Verlag, Berlin, 1988.

[12] M. Kunde, Balanced Routing: Towards the Distance Bound on Grid, *Proceedings of the Third Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA* '91, July 21–24, 1991, pp. 260–271.

[13] M. Kunde, T. Tensi, ($k$–$k$) Routing on Multidimensional Mesh-Connected Arrays, *Journal of Parallel and Distributed Computing*, Vol. 11, No. 2, 1991, pp. 146–155.

[14] F. T. Leighton, Average Case Analysis of Greedy Routing Algorithms on Arrays, *Proceedings of the Second Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA* '90, Crete, July 2–6, 1990, pp. 2–10.

[15] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures*: *Arrays*, *Trees*, *Hypercubes*, Morgan Kaufmann, Los Altos, CA, 1991.

[16] F. T. Leighton, F. Makedon, I. G. Tollis, A $2n-2$ Algorithm for Routing in an $n \times n$ Array with Constant Size Queues, *Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA* '89, Santa Fe, NM, June 18–21, 1989, pp. 328–335.

[17] F. Makedon, A. Symvonis, An Efficient Heuristic for Permutation Packet Routing on Meshes with Low Buffer Requirements, *IEEE Transactions on Parallel and Distributed Systems*, Vol. 4, No. 3, 1993, pp. 270–276.

[18] J. Y. Ngai, A Framework for Adaptive Routing in Multicomputer Networks, Ph.D Thesis, Technical Report CS:TR:89-09, Caltech Computer Science Department.

[19] J. Y. Ngai, C. L. Seitz, A Framework for Adaptive Routing in Multicomputer Networks, *Proceedings of the ACM Symposium on Parallel Algorithms and Architectures, SPAA* '89, June 1989, pp. 1–9.

[20] S. Rajasekaran, R. Overholt, Constant Queue Routing on a Mesh, *Journal of Parallel and Distributed Computing*, Vol. 15, No. 2, 1992, pp. 160–166.

[21] S. Rajasekaran, T. Tsantilas, Optimal Routing Algorithms for Mesh-Connected Processor Arrays, *Algorithmica*, Vol. 8, 1992, pp. 21–38.

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

[23] C. D. Thompson, H. T. Kung, Sorting on a Mesh-Connected Parallel Computer, *Communications of the Association for Computing Machinery*, Vol. 20, 1977, pp. 263–270.

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

*Received February* 12, 1991, *and in revised form April* 29, 1993, *and in final form September* 28, 1994.