JOURNAL OF PARALLEL AND DISTRIBUTED COMPUTING 22, 37–43 (1994)

Optimal Algorithms for Multipacket Routing Problems on Rings

FILLIA MAKEDON* AND ANTONIOS SYMVONIS†

*Department of Mathematics and Computer Science, Dartmouth College, Hanover, New Hampshire 03755; and
†Basser Department of Computer Science, University of Sydney, New South Wales 2006, Australia

We study multipacket routing problems on rings of processors. We prove a new lower bound of $2n/3$ routing steps for the case that k, the number of packets per processor, is at most 2. We also give an algorithm that tightens this lower bound. For the case where $k > 2$, the lower bound is $kn/4$. The trivial algorithm needs in the worst case $k\lfloor n/2 \rfloor$ steps to terminate. An algorithm that completes the routing in $kn/4 + 2.5n$ routing steps is given. © 1994 Academic Press, Inc.

1. INTRODUCTION

A great deal of work has been devoted to the study of the packet routing problem [1–16]. This is because the packet routing problem is closely related to parallel computation. Through the routing of messages (packets) we are able to emulate shared memory [14]. More generally, for a parallel computer to be computationally effective, it must be able to route messages from their origin processors to their destination processors quickly and with small, preferably constant size queues. These queues are created while two or more packets are waiting to cross the same communication channel.

In this paper, we concentrate on permutation routing problems on a ring of processors. We prove a new lower bound for the case that the number of packets per processor is at most 2, and we give an algorithm that matches the lower bound. For the case that k is greater than 2, we present an algorithm that routes the packets in near optimal time.

Surprisingly, not too much attention has been given to the packet routing problem on rings. According to our knowledge, it is the first time that an in-depth investigation has been attempted. A majority of the previous works on packet routing problems on multidimensional meshes focus on the two-dimensional case. Probabilistic [2, 10, 13, 15, 16] as well as deterministic [3, 4, 5, 6, 8, 9, 10, 12, 15] algorithms have been proposed. All of these algorithms try to minimize the number of routing steps required to complete the routing and the size of the extra memory used for queueing purposes. The only work on r-dimensional meshes, $r > 2$, is by Kunde [4, 5].

A *ring of n processors* is defined to be graph $G = (V, E)$ where $V = \{i \mid i = 0, 1, 2, \ldots, n-1\}$ and $E = \{(i, i+1 \bmod n) \mid i = 0, 1, \ldots, n-1\}$. Set V represents processors while set E represents the communication links between them. During a single routing step, each processor can transmit a packet to each of its two neighbors and at the same time it can receive a packet from each of them. The processors are assumed to work in the MIMD model.

We define the *distance along the shortest path* between processors $P = i$ and $Q = j$, denoted $D_s(P, Q)$, to be the minimum number of links (edges) that a packet has to traverse starting from processor P and destined for processor Q. Obviously, $D_s(P, Q) = D_s(Q, P)$. The notation $[xy]_{\text{dir}}$ will be used to denote the distance from processor x to processor y in direction dir. dir can be clockwise (cw) or counterclockwise (ccw). Observe that $[xy]_{\text{cw}} = n - [yx]_{\text{cw}} = n - [xy]_{\text{ccw}}$. For clarity, in the rest of the paper processor i will be denoted by P_i. We also assume that the ring is laid down in such a way that the processor numbers increase in the clockwise direction.

In a *permutation routing problem* each processor has one packet to transmit to some other processor. At the end, each processor receives exactly one packet. In the *multipacket permutation problem* each processor has k packets, all of which are destined for the same processor. At the end, each processor receives exactly k packets. In the rest of the paper, we will refer to the multipacket routing problem as the *k-packet routing problem* to make explicit the number of packets per processor. Formally, a multipacket permutation problem R on a ring can be defined as a triple $\langle n, k, F \rangle$, where n is the number of processors on the ring, k is the number of packets per processor, and F is a function $F: \{0, 1, \ldots, n-1\} \rightarrow \{0, 1, \ldots, n-1\}$ such that F defines a permutation. The multipacket permutation problem arises when a single packet in the permutation routing problem consists of k flits. Some work has already been done on square meshes for this case: Makedon and Symvonis [10] and Symvonis [15] treated the k flits as an unbreakable "snake," while Kunde and Tensi [6] routed the flits of a packet independently. More recently, Rajasekaran and Raghavachari [13] presented randomized algorithms using both approaches and Kunde [5] derived a new deterministic algorithm for the case where the packets are routed independently. Up to now no work has been done on rings. Note that the above definition of the multipacket routing prob-

0743-7315/94 $6.00

lem is different from the definition of $k - k$ routing in [6] and [13]. In $k - k$ routing, the restriction that all packets originating from the same processor have the same destination is relaxed.

We obtain lower bounds on the number of steps required to solve a routing problem using two different arguments. The first one is a lower bound based on the maximum distance a packet has to travel (*distance bound*). The second one is based on the *bisection bound* of the network used. Then, the lower bound is max(*distance bound, bisection bound*). For the case of r-dimensional meshes of side-length n, the distance bound is $r(n - 1)$ and the bisection bound is $nk/2$, where k is the number of packets each processor holds. Thus, the lower bound on the number of steps required to solve the multipacket permutation routing problem on the r-dimensional mesh is $\max(r(n - 1), nk/2)$. Similarly, for the r-dimensional torus (an r-dimensional mesh with wraparound connections), the lower bound is $\max(r(n - 1)/2, nk/4)$. For r-dimensional meshes and tori, the distance bound dominates when $k \leq 2r$. For rings or processors, this corresponds to $k \leq 2$.

It should be pointed out that the domination of the distance bound for the case where $k \leq 2r$ is based on worst case scenarios. There are problems for which $k > 2r$ ($k > 2$ for rings) that still can be solved in less time than that indicated by the distance lower bound. For rings, one such trivial example is when all processors have packets destined for the processor immediately after them in the clockwise direction. Obviously, this routing problem can be solved in k steps.

The remainder of the paper is organized into sections as follows. In Section 2, we concentrate on 2-packet permutation routing problems on rings of n processors. We present a lower bound of $2n/3$ steps and we give an algorithm that matches the bound. In Section 3, we investigate the k-packet permutation routing problem, $k > 2$, on a ring of n processors. We present an algorithm that routes any problem in at most $kn/4 + 2.5n$ routing steps. We conclude in Section 4.

2. THE 2-PACKET PERMUTATION PROBLEM ON RINGS

2.1. A New Lower Bound

Let us assume that we have a ring of n processors, n is a multiple of 3, and we wish to route a multipacket permutation on it. Each processor has two packets that will be routed independently. Consider the following situation: Initially, P_i contains 2 packets destined for $P_{(i+n/3) \bmod n}$, $0 \leq i < n$. Hence, in the counterclockwise direction, each packet has to travel distance $2n/3$. If some packet decides to move in the counterclockwise direction, then at least $2n/3$ steps are required, since the distance between origin and destination is exactly $2n/3$ in that direction. So, if we want to achieve a better routing time, we have to send all packets in the clockwise direction. In this case, each of a total of $2n$ packets will travel for $n/3$ steps. Then, the total movement (number of wire crossings) is $2n^2/3$. Since all packets are moving in the same direction, only n communication links are used. Hence, this movement required at least $2n/3$ routing steps to be accomplished.

THEOREM 1. *There is a 2-packet permutation routing problem on a ring of* n *processors that requires* 2n/3 *routing steps for its solution.*

2.2. An Algorithm That Tightens the Lower Bound

An algorithm that tightens the lower bound given in Theorem 1 is the following:

Algorithm *Route_1*

At step 1: Each processor determines the minimum distance its packets have to travel, say s where $0 \leq s \leq n/2$. It also determines the direction in which the packets have to travel so that their distance to the destination is s. Let this direction be denoted by K (cw or ccw).

- If $s \leq \lceil n/3 \rceil$, then both packets are sent in direction K.
- If $\lceil n/3 \rceil < s \leq n/2$, then the processor sends one packet in the cw direction and one in the ccw direction.

At step t, t > 1: Each processor transmits a packet toward its destination, if it has one.
A processor never changes the direction of a packet.
The packet that has to go further in a given direction has higher priority.

LEMMA 1. *If Algorithm Route_1 is used for the routing of a 2-packet permutation routing problem then, at any time t, any processor has at most 2 packets that want to move in the same direction.*

Proof. The lemma is obvious, since (i) each processor transmits a packet toward a given direction, if it has one, (ii) at any step, it can receive at most one packet that must be sent in that direction, and (iii) the initial load of each processor is at most 2 packets that want to travel in the same direction. ■

DEFINITION. A *cut* c_i is defined to be the edge that connects $P_{(i-1) \bmod n}$ and P_i.

LEMMA 2. *If Algorithm Route_1 is used for the routing of a 2-packet permutation routing problem, then there are at most* $2\lceil n/3 \rceil$ *packets that want to cross any cut in the same direction.*

Proof. Without loss of generality, we consider cut $c_{2\lceil n/3 \rceil}$ and we examine packets moving only in the clockwise direction. The proof for the counterclockwise movement is symmetric. Let $P_0, \ldots, P_{\lceil n/3 \rceil - 1}$ constitute segment A of the ring, and $P_{\lceil n/3 \rceil}, \ldots, P_{2\lceil n/3 \rceil - 1}$ constitute segment B

of the ring (Fig. 1). Initially each processor in segment A has at most 1 packet that wants to move in the clockwise direction and also wants to cross the cut. Assume that the number of packets in segment A that want to cross the cut is m, $0 \le m \le \lceil n/3 \rceil$. Then these packets must be destined for segment C, where $P_{2\lceil n/3 \rceil}, \ldots, P_{n-1}$ constitute segment C. Since we are examining permutation routing, for any packet in segment A that wants to cross the cut in the clockwise direction, there must exist a unique destination in segment C. This means that there exist m positions in segment C, none of which can be the destination of a packet initially in segment B. Thus, in segment B, there must exist at most $\lceil n/3 \rceil - m$ processors that have packets destined for segment C. In the worst case, all of these processors will send 2 packets toward the cut. The remaining m processors in segment B will send all together at most m packets toward the cut. So, the total number of packets which are initially in segment B and want to cross the cut is at most $2(\lceil n/3 \rceil - m) + m$. This implies that the total number of packets that want to cross the cut in the clockwise direction is at most $(2\lceil n/3 \rceil - m) + m = 2\lceil n/3 \rceil$. ■

THEOREM 2. *Using Algorithm Route_1, a 2-packet permutation routing problem on a ring of n processors can be solved in* $2\lceil n/3 \rceil$ *steps.*

Proof. We will show that all packets that want to cross any cut in a given direction will do so within $2\lceil n/3 \rceil$ steps. Without loss of generality, let us consider cut $P_{2\lceil n/3 \rceil}$. We simulate the routing process as follows: Assume that we have two tapes, tape A and B. Tape A has $2\lceil n/3 \rceil$ cells and can move to the right. Tape B has $\lceil n/3 \rceil$ cells, is not allowed to move, and is placed, initially, on top of the $\lceil n/3 \rceil$ rightmost cells of tape A. Each cell of a tape can hold at most one *pebble*. A pebble represents a packet that wants to cross the cut in the clockwise direction. The fact that only tape A is able to move represents the fact that only 1 packet can be transmitted in one step in a given direction by any processor. If a pebble is placed on a cell of tape B, then a pebble must also exist on the underlying cell of tape A. We now observe that, initially, we have the following situation: If μ pebbles are on tape B, then at least μ cells in the $\lceil n/3 \rceil$ leftmost positions of tape A are empty. We associate the ith pebble of tape B, $0 \le i < \lceil n/3 \rceil - 1$, where we count from right to left, with the ith empty cell (*hole*) of tape A, where we count from left to right (Fig. 2). This is a 1–1 relation. A hole can be thought to correspond to (1) a nonexisting packet, (2) a packet that is not moving in the clockwise direction, or (3) a packet that is moving in the clockwise direction but does not need to cross the cut. Now we start moving tape A to the right. In the worst case, each pebble in tape B will drop into its corresponding hole in tape A. Some further explanation is needed in the case where a hole represents a packet that is moving in the clockwise direction but does not want to cross the cut. When a hole of this kind gets under a pebble of tape B, it means that in the same processor there are two packets that want to use the same communication link. The packet that is represented by the pebble has to go farther, and thus, since in our algorithm a packet that has to go farther has higher priority, there will be an exchange between the pebble of tape B and the hole of Tape A.

Since the whole tape A will cross the cut after exactly $2\lceil n/3 \rceil$ steps, all pebbles cross the cut in at most $2\lceil n/3 \rceil$ steps. ■

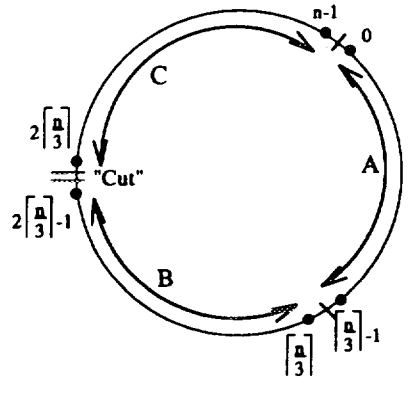

FIG. 1. The ring is divided into three sections that are used in the proof of Lemma 2.

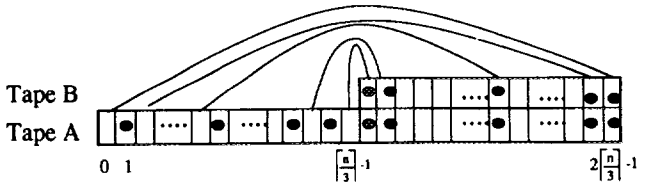

FIG. 2. The simulation of the routing by two tapes as used in the proof of Theorem 2.

3. THE k-PACKET PERMUTATION PROBLEM ON RINGS, $k > 2$

In this section, we consider k-packet permutation routing problems, $k > 2$, on rings of n processors. In this case, the lower bound based on the bisection of the ring is $kn/4$. The trivial greedy algorithm that routes each packet along the shortest path to its destination takes, in the worst case, at least $k\lfloor n/2 \rfloor$ steps. In what follows, we describe an algorithm that completes the routing in the worst case after $kn/4 + 2.5n$ routing steps.

Before we proceed with the description of the algorithm, we give definitions for certain variables we use: Let S_i denote the distance that the packets initially located at P_i have to travel along the shortest path to their destination. S_i can be written as $S_i = \lambda_i n$, $0 \le \lambda_i \le 1/2$, where λ_i is a coefficient used in our algorithm. Our algorithm routes a fraction of the packets which are initially at P_i along the shortest path to their destination, and

routes the remaining packets along the longest path. The number of packets that are routed along the shortest path and are originating from P_i is denoted by σ_i.

Algorithm *Route_2*

At step 1: P_i, $0 \le i < n$, determines the number of packets σ_i it will send along the shortest path using the following rule: $\sigma_i = k - \lfloor \lambda_i k \rfloor$

It then adds σ_i out of the k packets to a queue associated with the link on the shortest path, and the remaining packets to the queue associated with the other link.

The packets at the front of the queues are transmitted.

at step t, $t > 1$: Each processor transmits a packet toward its destination, if it has one.

A processor never changes the direction of a packet.

The packets are transmitted using a FIFO policy.

Our efforts to analyze Algorithm *Route_2* in a way similar to the analysis of Algorithm *Route_1* were not successful. In particular, we were not able to introduce in a proof of that kind the fact that the routing problem is a permutation. So, we proceed with a totally different approach. Again, we consider the number of packets that cross any cut in any direction. But now we prove that the given routing problem is "easier" to be solved than a special kind of routing problem for which we can make statements regarding its complexity. An upper bound obtained for this special problem will clearly be an upper bound on the initial (and "easier") routing problem. This is the first time that this method for proving upper bounds on routing algorithms is applied. A first simple proof of that type was introduced by Makedon and Symvonis [10] and Symvonis [15].

A *half-ring* of a ring consisting of n processors, n being even, is any section of the ring consisting of exactly $n/2$ consecutive processors. Note that a half-ring is well defined only on rings that consist of an even number of processors. It is undefined when there is an odd number of processors on the ring. Two half-rings of the same ring of processors are said to be *disjoint* if they share no common processors.

A multipacket routing problem on a ring of n processors, n being even, is said to be *symmetric with respect to cut* c_i if all the packets at the half-ring composed by P_i, ..., $P_{(i+n/2-1) \bmod n}$ are destined for the half-ring composed by $P_{(i+n/2) \bmod n}$, ..., $P_{(i-1) \bmod n}$ and vice versa. If a multipacket routing problem is not symmetric with respect to cut c_i then it is called *asymmetric* (with respect to cut c_i). Again, a symmetric (asymmetric) multipacket routing problem with respect to cut c_i is well defined only for rings consisting of an even number of processors.

LEMMA 3. *Assume a ring of n processors, n being even, any cut c_i, any direction, and an asymmetric k-packet permutation routing problem, $k > 2$, with respect to cut c_i that is to be routed. Also assume that Algorithm Route_2 is used for the routing. Then, there exists a symmetric $(k+1)$-packet permutation problem with respect to cut c_i such that, if it is routed using Algorithm Route_2 and the extra packet is sent toward the cut in the direction under consideration, it will send at least as many packets to cross the cut in the given direction as the initial asymmetric routing problem. (Informally we can say that the new routing problem is "harder.")*

Proof. Without loss of generality, we concentrate on the number of packets that cross the cut in the clockwise direction. A diameter that passes through the cut divides the ring into two disjoint half-rings. Half-ring A consists of P_i, ..., $P_{(i+n/2-1) \bmod n}$ and half-ring B consists of $P_{(i+n/2) \bmod n}$, ..., $P_{(i-1) \bmod n}$. We show how to transform any asymmetric k-packet routing problem, $k > 2$, (with respect to cut c_i) to a symmetric $(k+1)$-packet routing problem (with respect to cut c_i). Furthermore, the number of packets that cross the cut in the new routing problem is greater than or equal to the number of packets that cross the cut if the original problem is routed. (Both problems are routed by Algorithm *Route_2*.)

First observe that if there is a processor in half-ring A that has packets destined for another processor in half-ring A, then there exists a processor in half-ring B that has packets destined for a processor in half-ring B. The above observation follows from the pigeonhole principle. Our transformation consists of picking two processors, one in each half-ring, that have packets destined for the half-ring to which they belong. Then, the destinations are switched. By performing the above transformation for at most $n/2$ times, we will get a symmetric multipacket routing problem (with respect to cut c_i). Now it remains to prove that if we load each processor at the new problem with one additional packet which is to be routed toward the cut, the new problem sends at least as many packets to cross the cut as the initial one (and thus, it is "harder"). We distinguish the four cases that are described in Fig. 3.

Case 1. The packets at processors b and d are destined for processors a and c, respectively. After the switch of the destinations, the packets at processor b are destined for processor c and the packets at processor d are destined for processor a. This is the most interesting case. Before the switch, packets from both origins want to cross the cut while, after the switch, only packets initially located at processor b want to do so. Before the switch, processor b sends at most $(k/n)[ab]_{cw}$ packets toward the cut. Processor d sends at most $(k/n)[cd]_{cw}$ packets toward the cut. Thus, before the switch, at most $(k/n)([ab]_{cw} + [cd]_{cw})$ packets cross the cut in the clockwise direction. Consider now the number of packets (originating from processor b) that want to cross the cut after the switch. We have to distinguish two cases. In the

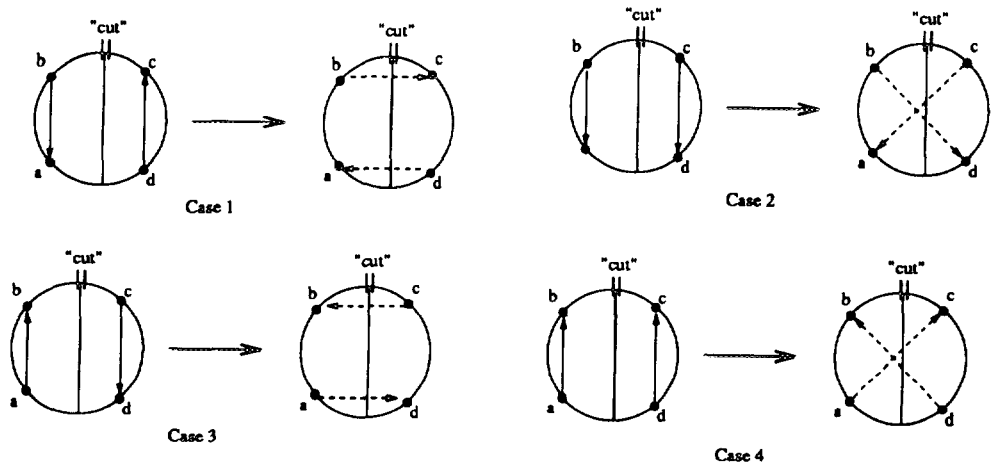

FIG. 3. The four cases considered in the proof of Lemma 3.

first case the packets that cross the cut are routed along a shortest path. Then at least

$$k - \left\lfloor \frac{[bc]_{cw}}{n} k \right\rfloor \geq k - \frac{[bc]_{cw}}{n} k - 1 = \frac{k}{n}(n - [bc]_{cw}) - 1 = \frac{k}{n}[cb]_{cw} - 1$$

packets will cross the cut. In the case where the packets are routed along the longest path, the number of packets that cross the cut is at least $(k/n)[cb]_{cw}$. So, in any case, after the switch at least $(k/n)[cb]_{cw} - 1$ packets will cross the cut in the clockwise direction. But $[cb]_{cw} \geq [ab]_{cw} + [cd]_{cw}$. Thus, if we load processor b with one extra packet and we force it to cross the cut in the clockwise direction, the new problem sends at least as many packets to cross the cut as the original one.

Case 2. The transformation is described in Fig. 3. Observe that before the switch no packet wants to cross the cut in the clockwise direction. After the switch, a portion of the packets located at processor a might cross the cut in the clockwise direction. So, the new problem sends at least as many packets to cross the cut as the original one.

Case 3 and 4. The transformations are described in Fig. 3. They are handled in a way similar to Case 2.

In the worst case, when we convert an asymmetric k-packet routing problem with respect to some cut c_i into a symmetric one with respect to the same cut, we might always have to perform the transformation described in Case 1. This observation proves the lemma. ▩

LEMMA 4. *Assume a ring of n processors, n being even, any cut c_i, and any direction. Also assume that a symmetric k-packet permutation routing problem, $k > 2$, with respect to cut c_i is to be routed. If Algorithm Route_2 is used for the routing, then the number of packets that cross the cut in any direction is at most equal to the number of packets that cross the cut if a new symmetric $(k + 2)$-packet routing problem with respect to cut c_i is routed, where, in the new problem, the packets at P_j are destined for $P_{(j+n/2) \bmod n}$, $0 \leq j < n$, and the extra 2 packets are routed toward the cut in the given direction.*

Proof. Without loss of generality we consider the packets that cross the cut in the clockwise direction. Only packets that originate in the half-ring defined by $P_{(i+n/2) \bmod n}, \ldots, P_{(i-1) \bmod n}$ will cross the cut in that direction. We can convert the initial symmetric problem to the one defined in the lemma by executing Algorithm *Symmetric_Conversion(i)*.

Algorithm *Symmetric_Conversion(cut)*
Begin
  For $j = 0$ **to** $n - 1$ **do**
    Convert(cut, j)
End

Convert(cut, j)
Begin
- Locate the processor that has packets destined for $P_{(cut+j) \bmod n}$. Let it be processor a.
- Let the destination of the packets at $P_{(cut+j+n/2) \bmod n}$ be processor b.
- Switch the destinations of the packets such that:
  - The packets at $P_{(cut+j+n/2) \bmod n}$ are destined for $P_{(cut+j) \bmod n}$.
  - The packets at processor a are destined for processor b.

End

The transformation that occurs in a single execution of procedure *Convert(cut, j)* for $cut = i$, is illustrated in Fig. 4.

We concentrate on the half-ring immediately before the cut in the clockwise direction. Consider iteration j, $0 \leq$

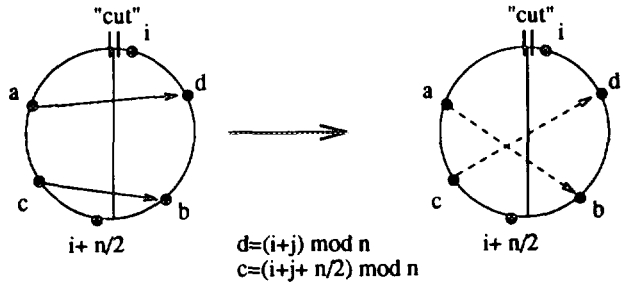

FIG. 4. The transformation that occurs in a single iteration *Convert-(cut, j)*.

$j < n/2$. Denote $P_{(cut+j) \bmod n}$ by c and $P_{(cut+j) \bmod n}$ by d. Observe that before the switch packets from processors a and c want to cross the cut in the clockwise direction. The packets originally at processor a will travel along their shortest path. So, there are at most

$$k - \left\lfloor \frac{[ad]_{cw}}{n} k \right\rfloor \leq k - \frac{[ad]_{cw}}{n} k + 1$$

$$= \frac{k}{n} (n - [ad]_{cw}) + 1 = \frac{k}{n} [da]_{cw} + 1$$

packets originating from processor a that will cross the cut. The packets originally at processor c will travel along their longest path. So, there are at most

$$\left\lfloor \frac{[bc]_{cw}}{n} k \right\rfloor \leq \frac{[bc]_{cw}}{n} k$$

packets originating from processor c that will cross the cut. Thus, before the switch at most $(k/n)([da]_{cw} + [bc]_{cw}) + 1$ packets will cross the cut in the clockwise direction.

After the switch, again, packets from both processors a and c will cross the cut. For the packets initially at processor a we have to consider two cases, namely, when they travel along the shortest or the longest path. In the case where they travel along the shortest path, at least

$$k - \left\lfloor \frac{[ab]_{cw}}{n} k \right\rfloor \geq k - \frac{[ab]_{cw}}{n} k$$

$$= \frac{k}{n} (n - [ab]_{cw}) = \frac{k}{n} [ba]_{cw}$$

packets will cross the cut. In the case where they travel along the longest path, at least

$$\left\lfloor \frac{[ba]_{cw}}{n} k \right\rfloor \geq \frac{[ba]_{cw}}{n} k - 1$$

packets will do so. Thus, in any case, at least $(k/n)[ba]_{cw} - 1$ packets originating from processor a will cross the cut. Let us now turn our attention to the packets originating from processor c. They have to travel exactly $n/2$ steps. We treat them as though they were routed along the shortest path. Then, at least

$$k - \left\lfloor \frac{[cd]_{cw}}{n} k \right\rfloor \geq k - \frac{[cd]_{cw}}{n} k$$

$$= \frac{k}{n} (n - [cd]_{cw}) = \frac{k}{n} [dc]_{cw}$$

packets originating from processor c will cross the cut. Thus, after the switch, a total of at least $(k/n)([ba]_{cw} + [dc]_{cw}) - 1$ packets will cross the cut. Recall that the number of packets that wanted to cross the cut before the switch was $(k/n)([da]_{cw} + [bc]_{cw}) + 1$. But $([ba]_{cw} + [dc]_{cw}) = ([bc]_{cw} + [ca]_{cw} + [db]_{cw} + [bc]_{cw}) = ([da]_{cw} + [bc]_{cw})$. Thus, if we load processor c with 2 extra packets and we route both of them toward the cut the new routing problem sends at least the same number of packet to cross the cut. ■

Up to now, we have concentrated on the case where n, the number of processors on the ring, is even. In what follows, we describe how to augment a routing problem on a ring that consists of an odd number of processors to a routing problem on a ring that has one extra processor. This augmentation will never occur in practice. It is used only as a tool in proving statements regarding the number of packets that cross a cut in some direction if the number of processors in the initial problem is odd. For this reason, the augmentation will be defined with respect to a given cut in the initial ring and a given direction. Informally, the augmented routing problem of the routing problem R with respect to cut c_i and direction dir, denoted by $R^R_{i,\text{dir}}$, is obtained as follows: A new processor is placed on the ring such that it becomes the $\lceil n/2 \rceil$th processor after cut c_i in direction dir. The new processor is loaded with k packets that are destined for itself. Then the processors on the ring are renamed and the initial routing problem is updated so that the destinations reflect the new names.

THEOREM 3. *Using Algorithm Route_2, a k-packet permutation problem $R = \langle n, k, F \rangle$, $k > 2$, on a ring of processors can be solved in*

a. $kn/4 + 1.5n$ routing steps	*if n and k are even*
b. $kn/4 + 2n$ routing steps	*if n is even and k is odd*
c. $kn/4 + 2n$ routing steps	*if n is odd and k is even*
d. $kn/4 + 2.5n$ routing steps	*if n and k are odd.*

Proof. First we prove that if Algorithm *Route_2* is used for the routing then the number of packets that cross any cut in any direction is at most equal to the number of routing steps stated in the theorem.

Without loss of generality consider cut c_i and the packets that cross it in the clockwise direction. First we prove cases *a* and *b*, i.e., when n is even. Lemmata 3 and 4

imply that a routing problem that is symmetric with respect to cut c_i and has initial load of $k + 3$ packets per processor, all destined for the processor located after $n/2$ positions in the clockwise direction, will send at least an equal number of packets to cross the cut c_i in the clockwise direction with any other multipacket permutation routing problem with initial load k packets per processor. (The 3 extra packets will be routed in the clockwise direction toward the cut.) For that "harder" problem, in the case where k is even, Algorithm *Route_2* will send exactly $kn/4$ packets to cross the cut. In the case where k is odd, at most $n/2$ extra packets will cross the cut. This is caused by the $\lfloor\ \rfloor$ operator that is involved in the computation of σ_i at step 1 of the algorithm. Now, by adding the extra flow created by the 3 extra packets that are routed toward the cut (exactly $1.5n$ extra packets will cross it) we get the quantities stated in cases *a* and *b* of the lemma.

In the case where n is odd, we consider the augmented problem $R^R_{i,\mathrm{cw}}$. We make the following observation: If the augmented problem is routed using Algorithm *Route_2*, then the number of packets that cross cut c_i in the clockwise direction might be reduced by at most $n/2$. This is because of the renaming of the processors. The renaming causes the distances between processors to change. So, the number of packets that a processor sends toward the cut might change also. However, this number changes by at most 1 packet, and furthermore it is reduced only at the processors in the half-ring immediately before the cut in the direction under consideration. Working in the same lines as before, in order to overcome the above problem and guarantee that the augmented problem is at least as hard as the original, we load each processor in that half-ring with one extra packet that is routed toward the cut, i.e., the processors in the half-ring before the cut in the augmented problem have $k + 1$ packets. Then cases *c* and *d* are proven by Lemmata 3 and 4 and by arguing as in cases *a* and *b* for k even and odd, respectively.

Now, if at every step of the algorithm one packet wants to cross the cut, the theorem is obviously true. In the case where there is a period during the execution of the routing algorithm that no packet crosses the cut, say for μ steps, then there are at least μ processors that do not have any packets that want to cross the cut. Thus, the claimed number of packets that cross the cut was overestimated by at least μ. So, the theorem holds. ■

4. CONCLUSIONS

In this paper, we have examined multipacket permutation routing problems on rings. We presented a new lower bound for the case of 2-packet routing problems and we gave an algorithm that tightens the lower bound. For the case of k-packet routing problems, $k > 2$, we presented an algorithm that solves any problem within $kn/4 + 2.5n$ routing steps. Thus, we have succeeded in presenting an algorithm that approximates within an additive factor the number of routing steps required in the worst case. An interesting problem is to design an algorithm that matches the lower bound.

Received December 13, 1991; revised September 29, 1992; accepted February 23, 1993

REFERENCES

1. Aielo, B., Leighton, F. T., Maggs, B., and Newman, M. Fast algorithms for bit-serial routing on a hypercube. *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '90.* July 2–6, 1990, Crete, Greece.
2. Krizanc, D., Rajasekaran, S., and Tsantilas, Th. Optimal routing algorithms for mesh-connected processor arrays. In Reif, J. (Ed.). *VLSI Algorithms and Architectures (AWOC'88).* Lecture Notes in Computer Science Vol. 319. Springer-Verlag, 1988, pp. 411–422.
3. Kunde, M. Routing and sorting on mesh-connected arrays. *VLSI Algorithms and Architectures (AWOC'88).* Lecture Notes in Computer Science Vol. 319. Springer-Verlag, 1988, pp. 423–433.
4. Kunde, M. Balanced routing: Towards the distance bound on grids. *Proceedings of the 3rd Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'91.* 1991.
5. Kunde, M. Concentrated regular data streams on grids: Sorting and routing near to the bisection bound. *Proceedings of the 32nd IEEE Symposium on Foundation of Computer Science.* 1991.
6. Kunder, M., and Tensi, T. Multi-packet routing on mesh connected arrays. *Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89.* June 1989, pp. 336–343.
7. Leighton, F. T. Average case analysis of greedy routing algorithms on arrays. *Proceedings of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'90.* July 2–6, 1990, Crete, Greece.
8. Leighton, F. T., Makedon, F., and Tollis, I. G. 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.* June 1989, pp. 328–335.
9. Makedon, F., and Symvonis, A. An efficient heuristic for permutation packet routing on meshes with low buffer requirements. *IEEE Trans. Parallel Distrib. Syst.* **4,** 3 (March 1993) 270–276.
10. Makedon, F., and Symvonis, A. On bit-serial packet routing for the mesh and the torus. *Proceedings of the 3rd Symposium on the Frontiers of Massively Parallel Computation.* October 8–10, 1990, pp. 294–302.
11. Ngai, J. Y., and Seitz, C. L. A framework for adaptive routing in multicomputer networks. *Proceedings of ACM Symposium on Parallel Algorithms and Architectures, SPAA'89.* June 1989, pp. 1–9.
12. Rajasekaran, S., and Overholt, R. Constant queue routing on a mesh, *J. Parallel Distrib. Comput.* **15,** 2 (June 1992) 160–166.
13. Rajasekaran, S., and Raghavachari, M. Optimal randomized algorithms for multipacket and wormhole routing on the mesh. *Proceedings of the 3rd IEEE Symposium on Parallel and Distributed Processing.* Dallas, 1991.
14. Ranade, A. G. How to Emulate Shared Memory. *Proceedings of the 28th IEEE Symposium on Foundation of Computer Science.* 1987, pp. 185–194.
15. Symvonis, A., Packet routing problems on mesh connected machines and high resolution layouts. Ph.D. thesis, University of Texas at Dallas, 1991.
16. Valiant, L. G., and Brebner, G. J. Universal schemes for parallel communication. *Proceedings of the 13th Annual ACM Symposium on the Theory of Computing.* May 1981, pp. 263–277.