On the Hot-Potato Permutation Routing Algorithm of Borodin, Rabani and Schieber

Richard Thomas Plunkett *Antonis Symvonis*
Basser Department of Computer Science
University of Sydney
Sydney, N.S.W. 2006
Australia
{*richard,symvonis*}*@cs.su.oz.au*

Abstract

Borodin, Rabani and Schieber [3] presented an $O(n\sqrt{n})$-step algorithm for hot-potato routing of permutations on $n \times n$ meshes. They conjectured that their algorithm completes the routing of a permutation in $O(n)$ steps. In this paper, we present worst-case partial permutations which force their algorithm to use $\Omega(n\sqrt{n})$ routing steps.

Keywords Packet routing, hot-potato routing, permutation routing, routing algorithm.

1 Introduction

Message routing has been abstracted in several ways. In *packet routing* it is assumed that a message can be transmitted between two adjacent processors (vertices of the undirected graph) in a single step as a *packet*. We concentrate on the routing model known as *deflection* (or *hot-potato*) routing in which packets continuously move between processors from the time they are injected into the network until the time they are consumed at their destination.

The advantage of deflection routing is obvious. No queueing area is required at the processors. However, the fact that packets always move implies that at any step each processor must transmit the packets it received during the previous step (unless they were destined for it). As a result, several packets might be derouted away from their destination. This makes the analysis extremely difficult.

The work of Feige and Raghavan [4] which provided analysis for deflection routing algorithms for the torus and the hypercube, renewed the interest in deflection routing. As a result, several papers appeared with deflection routing as their main theme. Kaklamanis, Krizanc and Rao [5] considered batch and permutation routing on torii and meshes. Bar-Noy et al [1] gave a nearly optimal deterministic algorithm for routing permutations on the $n \times n$ mesh and torus. Based on sorting algorithms, Newman and Schuster [7] derived asymptotically optimal algorithms for routing permutations on the mesh and the torus networks and a near optimal algorithm for the hypercube. Kaufmann et al [6] fine tuned their results for the case of meshes and managed to reduce the constant hidden in the asymptotic analysis.

Ben-Aroya and Schuster [2] considered the general situation where a mesh is loaded with k packets that have to be routed to their destinations in a hot-potato manner. Through clever analysis of their greedy algorithm they showed that routing will terminate within $dist + 2(k-1)$ steps where $dist$ is the initial maximum distance a packet has to travel. Independently, Borodin et al [3] formalized the notion of the *deflection sequence*, a nice way to charge each deflection of an individual packet to distinct packets travelling on the network. By using their method, they show that routing k packets in a hot-potato manner can be completed within $dist + 2(k-1)$ steps for trees, butterflies and multidimensional meshes.

For permutation routing on two-dimensional meshes, the algorithm of Borodin et al [3] completes the routing in $O(n\sqrt{n})$ steps. It was conjectured that the actual performance of the algorithm is $O(n)$. In this paper, we disprove the conjecture of Borodin et al [3] regarding the performance of their algorithm for permutation routing on two-dimensional meshes by demonstrating worst case input permutations that require $\Omega(n\sqrt{n})$ for their routing. The paper is organised as follows: In the next section, we describe the algorithm of Bar-Noy et al [1] together with a worst case input permutation which causes the algorithm to take $\Omega(n\sqrt{n})$ routing steps[1]. In Section 3 we describe the algorithm of Borodin et al [3] while, in Section 4,

[1]There is nothing new about this worst case routing pattern (it can be considered to be an answer to Exercise 7.4 in [8]). However, it demonstrates the basic idea used in the construction presented in Section 4.

Proceedings of CATS'96 (Computing: The Australasian Theory Symposium), Melbourne, Australia, January 29–January 30 1996.

we present routing patterns which causes the algorithm to perform poorly. We conclude in Section 5.

2 Preliminaries

2.1 The Routing Model

Throughout the paper we consider hot-potato permutation routing on an $n \times n$ mesh of processors. The processor at the intersection of the i-th row and the j-th column is represented by the tuple (i, j), $1 \leq i, j \leq n$. Processor (i, j) is connected with processors $(i-1, j)$, $(i+1, j)$, $(i, j-1)$ and $(i, j+1)$, provided they exist, through bidirectional communication links.

If at most h_1 packets originate from any processor, and, at most h_2 packets are destined for any processor, then we say that we have an (h_1, h_2) *routing problem.* When $h_1 = 1$ and $h_2 > 1$ we have a *many-to-one routing problem* (*many* processors send packets to *one* processor), when $h_1 > 1$ and $h_2 = 1$ we have a *one-to-many routing problem* (*one* processor sends packets to *many* processors), and when $h_1 = h_2 = 1$ we have the *(partial) permutation routing problem.*

When the routing is performed in a hot-potato manner, the processors operate in synchronous mode and they execute a routing algorithm which iterates over the following 3 steps:

1. *Receive the packets that want to enter the processor (if any).* These packets were transmitted during the previous step from neighboring processors.

2. *Assign to each of the packets received during the previous step a distinct communication link leading to one of the processor's neighbores.* The way we assign the communication links to the packets is determined by the routing algorithm.

3. *Transmit the packets through the communication links assigned to them.*

While it is possible to perform *dynamic routing* (where packets can be generated when routing is in progress), we concentrate on *static routing*, i.e., the situation where all packets are available at the beginning of the routing. The fact that we examine only permutations implies that queues of packets are never created. Actually, this is true for any (h_1, h_2) routing pattern in which each processor has initially at most as many packets as the number of its neighbor processors.

2.2 The Algorithm of Bar-Noy, Schieber, Raghavan and Tamaki

We describe the algorithm of Bar-Noy et al [1] for two reasons. Firstly, the algorithm of Borodin, Rabani and Schieber [3] is a refinement of it and thus, its description becomes easier. Secondly, the worst case permutation for the algorithm of Bar-Noy et al demonstrates the underlying idea in the construction of the routing pattern we use to disprove the conjecture of Borodin at al.

According to the algorithm of Bar-Noy et al [1], each packet moves in its origin row towards the east and when it reaches the east-most column of the mesh it reverses direction and moves towards the west, reversing again its direction of movement when it reaches the west-most column, and so on. During each westbound pass, each packet attempts to enter its destination column, preferably towards the correct direction. Packets moving in their destination column have priority over packets that attempt to turn into the column. Once a packet succeeds to turn into its destination column, it moves from end to end (in a similar way with its row movement) and is guaranteed to reach its destination in less than $2n$ steps.

Bar-Noy et al [1] proved that their algorithm terminates after $O(n\sqrt{n})$ routing steps. They achieved that by exploiting the fact that it takes exactly 2 packets moving in their destination column to prevent a third packet from turning into that column. Moreover, a packet cannot be prevented from entering its destination column by another packet (which collaborates with a third one) more than once.

We now describe a partial permutation that causes the above algorithm to use $\Omega(n\sqrt{n})$ routing steps. This worst case permutation is illustrated in Figure 1. The packets of interest are initially located in a sub-mesh of dimensions $s \times s$ and are all destined for the same column C and for rows which are not spanned by the sub-mesh. We refer to the segment of column C which consists of the processors at the rows spanned by the sub-mesh as segment D. Since we route a permutation, we have to ensure that a sufficient number of destination processors exists in column C. This implies that $s^2 \leq n - s$.

During the routing, all packets move as a block, henceforth referred to as block B, which looses at most $4s$ of its packets during each westbound pass over column C. To see that, observe that when the last column of block B crosses C during a westbound pass, at most $2s$ packets are in segment D. All of these packets have succeeded in turning into their destination column, i.e., column C. Also notice that during the s steps that it takes block B to cross column C, at most $2s$ packets (2 per step) can exit segment D.

Since during each westbound pass of block B over column C, it looses at most $4s$ packets, some packets in the block execute at least $s^2/4s = s/4$ passes. Each *complete pass*, i.e., the trip that a

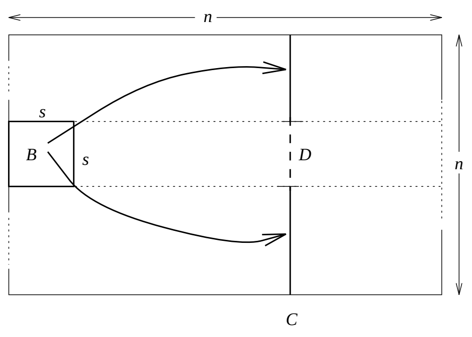

Figure 1: A routing patterns that causes the algorithm of Bar-Noy, Schieber, Raghavan and Tamaki to use $\Omega(n^{3/2})$ routing steps.

packet makes in order to return to its original position after bouncing at both the east and the west boundaries of the array, takes $2n$ steps and thus the algorithm terminates after $\frac{ns}{2}$ routing steps. By choosing the maximum value that satisfies $s^2 \leq n-s$, we get that $\Omega(n\sqrt{n})$ routing steps are necessary. Note that, if we are interested in the constant factor hidden in the $\Omega()$ notation, we should have block B located at the top or the bottom rows of the mesh. In this way, at most 1 packet can exit segment D in every step.

3 The Algorithm of Borodin, Rabani and Schieber

The Algorithm of Borodin, Rabani and Schieber [3] is a modification of the algorithm of Bar-Noy et al [1]. The modified algorithm allows a packet heading in the wrong direction (either in its origin row or destination column) to reverse direction immediately, provided that this change of direction is not in conflict with the movement of another packet that is headed towards its destination column (if it is moving along a row) or its destination node (if it is moving along its destination column). The modified algorithm also allows for packets to enter their destination column during their eastbound passes as well as their westbound passes.

It is useful to examine how the worst case permutation that forces the algorithm of Bar-Noy et al to perform poorly is routed when the modified algorithm is used. When block B reaches column C, it looses its first 2 columns. This is because at most $2s$ packets can at any time be located in segment D, executing their column movement. The remaining of the block rolls over itself and changes direction by starting moving towards the west. When it reaches the last column of the block, it reverses direction again and follows it. From now on, each packet appears to be part of a closed chain of packets which moves in *cycles*, each cycle "touching" the destination column C. During the routing, the shape of the block changes. However, at every routing step at most 2 packets manage to exit segment D. The fact that the chains of packets that cycle around in any row intersecting segment D always have some packet in a processor of segment D (executing its row movement) guarantees that at every step at least 1 packet manages to exit segment D or at least 1 packet of the block starts its column routing in segment D. Since the number of packets initially in block B is at most $n-s$, we conclude that the modified algorithm routes the permutation that forced the original algorithm to take $\Omega(n\sqrt{n})$ steps in only $O(n)$ steps.

There is one aspect of the modified algorithm that is not fully specified. According to the algorithm [3, pg. 6], *"packets heading in the wrong direction will reverse direction whenever possible."* This statement accepts two different interpretations when it is concerned with a packet which is located at its destination column but failed to turn into it. This packet, no matter in which direction it moves during the next step, moves away of its destination column. We can resolve this ambiguity by considering 2 options:

Option 1 *Bounce at the destination column.* The packet changes direction of movement if it fails to enter its destination column (Figure 2).

Option 2 *Overshoot the destination column.* The packet maintains the momentum of its movement by continuing moving in the same direction (Figure 3).

We interpret that the intension of Borodin, Rabani and Schieber was to consistently employ one of these two ways in their routing algorithm. It turns out that if the second option is used, the construction of the worst case permutation is more complicated. In the next section, we provide worst case permutations for both options. We refer to the version of the algorithm that uses the first option as the *bounce algorithm* and to the one that uses the second option as the *overshoot algorithm.*

4 Permutations that Require $\Omega(n\sqrt{n})$ Routing Steps

4.1 A Worst Case Permutation for the *Bounce* Algorithm

The routing pattern that forces the *bounce* algorithm to use $\Omega(n\sqrt{n})$ routing steps is illustrated in Figure 4(a). The packets of interest are grouped into k blocks $B_1, B_2, \cdots, B_k$. Initially, each block has length l_1 and height h_1. Let D_i denote the segment at the intersection of column C_i and the rows spanned by the blocks. The packets of block B_i are all destined to the north or south of segment D_i in column C_i, $1 \leq i \leq k$. This implies that $l_1 h_1 \leq n - h_1$.

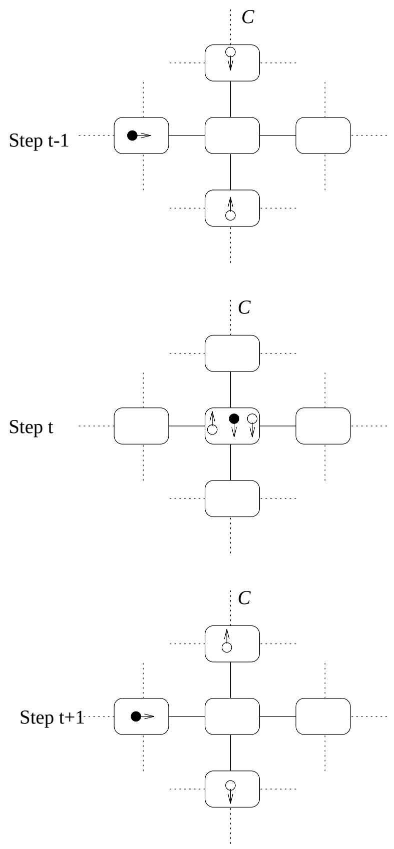

Figure 2: Demonstration of the routing that takes place if the *bounce* option is used in the algorithm of Borodin et al.

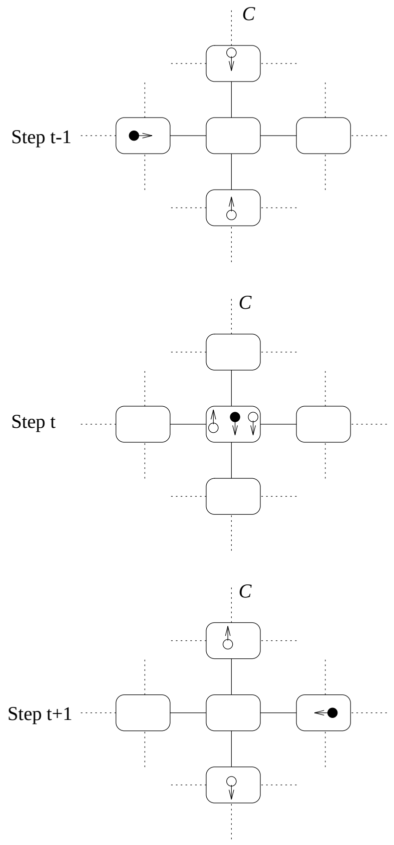

Figure 3: Demonstration of the routing that takes place if the *overshoot* option is used in the algorithm of Borodin et al.

In the following discussion we assume that the columns of the block are numbered. The first column of a block is the rightmost one, the second column is the one to the left the first one, and so on. What is important is that this numbering remains the same when the block rolls over itself and changes direction, i.e., the numbering is based on the initial position of the block.

The destinations of the packets in block B_i, $1 \leq i \leq k$ are illustrated in Figure 4(b). An arrow pointing upwards (downwards) indicates packet(s) that are destined north (south) of segment D_i. All packets in the first column and the top row of block B_i are destined south of D_i. All but the topmost packet of the second column and all but the rightmost packet of the bottom row of the block are destined north of D_i. The destinations of the remaining packets of block B_i will be specified later.

Let's observe the movement of block B_1 when it reaches column C_1. All the packets in the first column of B_1 will start their column movement. During the next step, all packets in the second column also start their column movement. At the end of the second step, each processor in segment D_1 but the ones at its north and south ends contains 2 packets, one moving towards the north and the other towards the south. During the next step, the

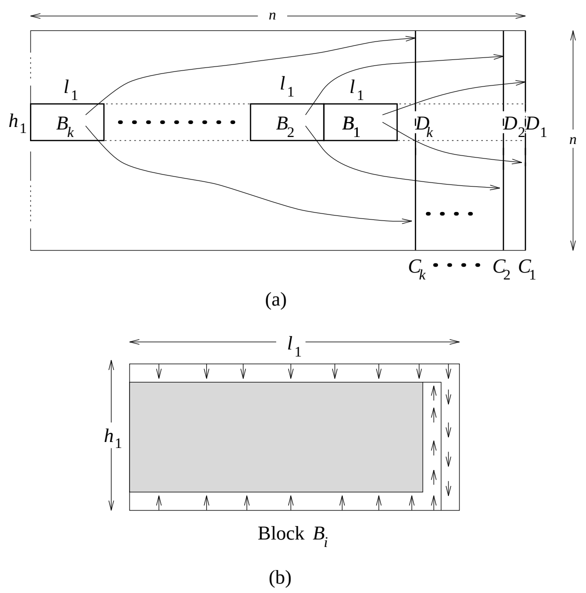

Figure 4: A worst case permutation for the *bounce* algorithm.

packets at the top and bottom row of the third column start their column movement while the remaining packets of the third column (i.e., those that belong into the grey area of Figure 4(b)) bounce and start moving towards the west. The same scenario continues till the last column of the block reaches C_1. The packets (in the grey area) of the second last column bounced during the previous step and are now at the processors of segment D_2.

These packets of the last 2 columns of B_1 start changing direction of movement and after at most $4(h_1 - 2)$ steps have all started their column movement. To see that, observe that all the packets that had already started their column movement in D_1, exit the part of segment D_1 spanned by the gray area within $h_1 - 2$ steps. Consider an arbitrary packet out of the remaining $2(h_1 - 2)$ packets that want to turn at column C_1 and exit segment D_1. Because that packet can only meet with packets that initially are at even distance from it, only $(h_1 - 3)$ can delay it. By employing the analysis of Borodin, Rabani and Schieber we know that the packet will reach its destination after $2m + dist$ steps, where m is the number of packets that can delay it and $dist$ is the distance to its destination. In our case, $m = h_1 - 3$ and assuming that the packets destination is the first processor outside D_1 (north or south of it), $dist \leq h_1 - 2$. We conclude that all packets of interest exit segment D_1 after at most $4(h_1 - 2)$ steps.

The remaining packets of block B_1 move towards the west. They form a smaller block of length $l_2 = l_1 - 4$ (since the original block lost its first 2 and its last 2 columns) and height $h_2 = h_1 - 2$ (since the original block lost its top and bottom rows). The new smaller block can change direction of movement only when its first column meets the last column of block B_k.

The behaviour of the remaining blocks is similar. The only exception is block B_k which looses only its first 2 columns. As a result, the movement of the blocks/packets resembles that of a *belt* that cycles around while at the same time it gets shorter and thinner in a very regular way. Note that in order to allow the blocks to roll over themselves without creating "empty" columns that separate them, the length of each block must be even.

Now it is easy to specify the destinations of the remaining packets (those in the grey area) of block $B_i,\ 1 \leq i \leq k-1$. For simplicity assume that the packets in the last column of B_i are destined to the north of D_i while those in the second last column are destined to the south of D_i. The destination of the remaining packets in the $(l_1 - 4) \times (h_1 - 2)$ sub-block are set according to the pattern described in Figure 4(b). For B_k, we also set the destination of the packets in the grey area according to Figure 4(b). However, the last 2 columns of its grey area are not treated in a special way since these columns are present in the new smaller block after each pass.

Let l_j be the length and h_j be the height of block $B_i,\ 1 \leq i \leq k-1$, after $j-1$ passes over the block's destination column. Assuming that the packet destination were set as above, we have:

$$l_j = l_{j-1} - 4 \text{ and } h_j = h_{j-1} - 2.$$

The length of block B_k decreases by 2 columns instead of 4.

Let a *complete pass* start when the first column of block B_1 reaches column C_1 (and finish just before the start of the next pass). The j-th pass is completed after at least kl_j routing steps while during the last pass the height of each block is equal to 2 (or the length of each block but B_k is equal to 4). By setting $l_1 = \sqrt{n},\ k = \sqrt{n} - 1$ (since in Figure 4(a) the blocks do not initially overlap with the destination columns) and $h_1 = \sqrt{n}/4$ (in order to guarantee that the packets in the last two columns of block B_{k-1} have started their column movement before the packets of block B_1 arrive at column C_{k-1}), we get that after $\sqrt{n}/8$ passes the packets in all blocks have started their column movement.

This implies that the row routing finishes after

$$\begin{array}{rcl} t & \geq & \sum_{j=1}^{\sqrt{n}/8} kl_j \\ & = & k(l_1 + l_2 + \cdots + l_{\sqrt{n}/8}) \\ & = & (\sqrt{n} - 1)(\sqrt{n} + \sqrt{n} - 4 + \cdots + \sqrt{n}/2) \\ & = & \Omega(n\sqrt{n}) \end{array}$$

routing steps. Notice that the above choices for l_1 and h_1 satisfy $l_1 h_1 \leq n - h_1$ as required. A final comment regarding the position of the blocks. Even though the number of packets in each block that are destined north of its corresponding D segment is not equal to the number of packets destined south of it, the number of packets in each block $(n/4)$ is too small to cause any problem. For any initial placement of the blocks spanning the middle rows of the mesh, it is easy to assign destinations to the packets.

4.2 A Worst Case Permutation for the *Overshoot* Algorithm

Firstly observe that when the *overshoot* algorithm is used to route the permutation described in the previous section, the belt of blocks appears to loose its original shape after a few passes. This shape deterioration is not that dramatic (the blocks of packets still look like a belt but the boundaries of the blocks are not so clear since adjacent blocks now overlap and the block shape is not a rectangle anymore) but is enough to make unclear how to use the same pattern to prove a lower bound on the

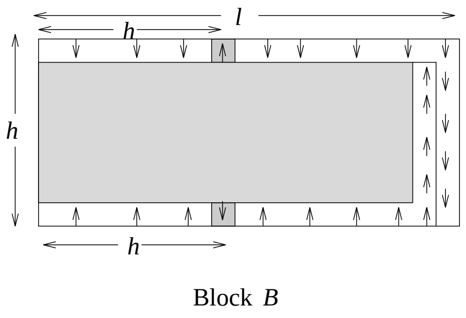

Figure 5: The block structure used in the worst case permutation for the *overshoot* algorithm.

behaviour of the *overshoot* algorithm, although it is unlikely that the *overshoot* algorithm routes this pattern in $o(n\sqrt{n})$ routing steps). So, we describe a different, more complicated pattern that forces the *overshoot* algorithm to use $\Omega(n\sqrt{n})$ routing steps.

The basic structure of the new pattern consists again from a sequence of blocks, where the packets in each block are destined for the same destination column. To enable the reader to understand the construction, we first present the structure of a block and its routing behaviour in two different situations. The block structure is described in Figure 5. All packets of the block are destined for the same column C, north or south of the D segment (the processors at the intersection of C and the rows spanned by the block). The length of the block is l and its height is h, where $l > h + 1$. By comparing the block structure of Figure 5 with that of Figure 4(b), we observe that they only differ in the destination of 2 *special* packets, i.e., the h-th last packet of the top row is destined north of D and the h-th last packet of the bottom row is destined south of D.

To understand the reason for this modification, observe the routing that takes place when the pattern of Figure 6 is routed by the *overshoot* algorithm. When the first column of block B reaches column C, the packets in it start their column movement. At the next step, the packets of the second column do so. When the third column arrives at C, no packet of the grey area succeeds to turn into the column. These packets overshoot column C and arrive at column $C + 1$ where they change direction of movement. All the remaining columns of the block behave in a similar manner and thus the grey area of the block rolls over itself and moves towards the west where it will roll over again when it meets with the last column of the block of packets that are destined east of column C. However, notice that the new smallest block is not a rectangle anymore. More specifically, it has lost the top and the bottom packets of its last column. This is because when these packets return to column C from column $C + 1$, they succeed in starting their column routing. The top packet of the last column of the grey area enters the column and moves towards the south while the bottom packet of the last column moves towards the north. This small disturbance in the shape of the "rectangle" has a significant effect when a block that follows B tries to roll itself behind B. The gaps that were created in B's rectangle are filled with packets from the block that follows B.

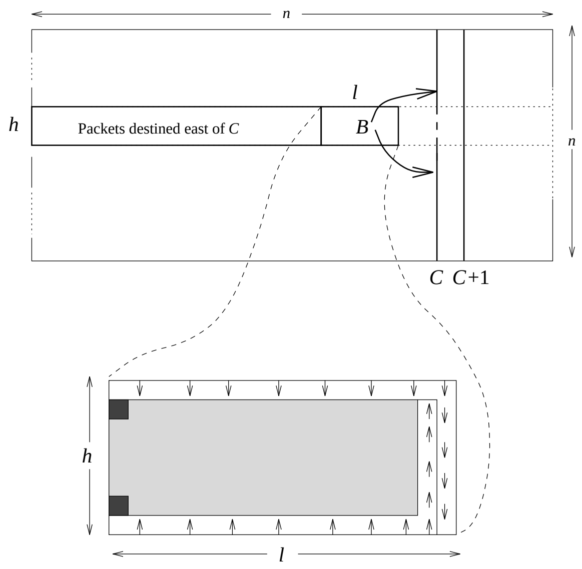

Figure 6: A routing pattern in which the block structure used for the *bounce* algorithm is routed by the *overshoot* algorithm.

The block structure of Figure 4(b) was modified to that of Figure 5 in order to maintain the rectangular shape of the block when it rolls over itself after overshooting its destination column. When the 2 special packets that appear to be at odds with the other packets in their row enter the destination column, they move in the wrong direction. These 2 packets traverse the entire segment D moving in the wrong direction. They change direction of movement when they reach the processors at the endpoints of segment D and move in the correct direction again. But at the next step, we have at the second top (bottom) row of segment D two packets that have started their column movement and the top (bottom) packet of the last column of the grey area. The top and the bottom packets of the grey area will fail to start their column movement and thus, the shape of the new block remains a rectangle. Note how crucial in the construction is the fact that the 2 special packets are the h-th last packets in their corresponding rows. The described construction places an additional restriction to the height h of the block. h must be even. If h was odd, the 2 special packets would have met in the middle of segment D, and they would have both changed direction of movement. The fact that after one pass over its destination column the new block has 2 rows less than the original block, guarantees

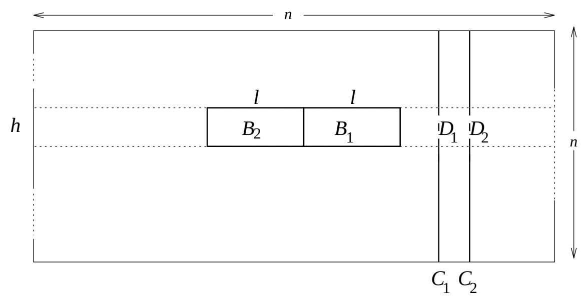

Figure 7: A routing pattern which consists of 2 blocks (as described in Figure 5) and is to be routed by the *overshoot* algorithm.

that we are able to repeat the construction for the packets in the grey area (which form the new block).

Consider now the configuration described in Figure 7. Each block is configured as in Figure 5. The packets of block B_1 and B_2 are destined for C_1 and C_2, respectively, all south or north of their corresponding D segment. Let the blocks move eastwards and consider the time where the first column of B_1 arrives at C_1. After $l - 2$ steps following that time, block B_1 has lost 2 columns and 2 rows, is facing towards the west, and its last column is at C_1 ready to move towards the east. At that step, the first column of B_2 is to the left of C_1. As a result, This means that after 4 steps, block B_1 starts rolling over itself behind B_2. It rolls over at a column that is exactly $l - 1$ positions to the west of column C_2 (without counting C_2). Now the 2 blocks which have formed a belt start making their passes over the 2 columns. With every pass the belt gets shorter (and thinner) since its span reduces by 4 columns (the two columns at the front of the block and the four columns its back). We also assume that $l > 6h$. This allows enough time for the packets that execute their column movement in C_1 (C_2) to exit the D_1 (D_2) segment before block B_1 (B_2) arrives for its new pass.

Now we are ready to describe the worst case permutation for the *overshoot* algorithm which is shown in Figure 8. Blocks $B_1, \cdots, B_{\lceil \frac{k}{2} \rceil}$ have length L_1 while blocks $B_{\lceil \frac{k}{2} \rceil + 1}, \cdots, B_k$ have length l_1, where $L_1 > 2l_1$ and $k = l_1 + 1$. All blocks have height h_1 and, as previously, the packets of block B_i are destined for C_i. Each block is configured as in Figure 5.

When block B_1 reaches C_1, it looses 2 columns and 2 rows, rolls over itself and moves towards the west where it will roll over itself again behind B_k. The movement of blocks $B_2, \cdots, B_{k-2}$ is similar. Let's examine the behaviour of blocks B_{k-1} and B_k. Consider the first time where the first column of B_{k-1} is at C_{k-1}. Since the configuration consisting only of blocks B_{k-1} and B_k is identical to that of Figure 7, after $l_1 + 2$ steps the 2 blocks have

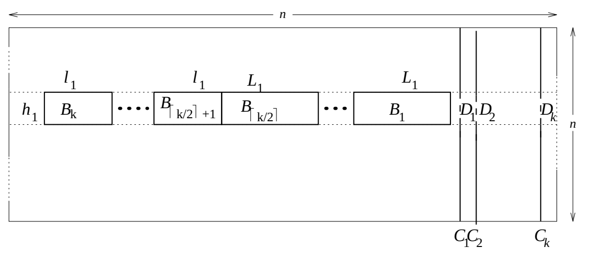

Figure 8: A worst case permutation for the *overshoot* algorithm.

formed a belt that spans up to column C_{k-l_1+1} which is C_2 since $k = l_1 + 1$. At that time, block B_1 arrives at C_1 but it will not interfere with the belt of B_{k-1} and B_k since it never goes east of C_2. So, for the next L_1 steps that columns of B_1 arrive at C_1, the belt of blocks B_{k-1} and B_k makes a full pass over columns C_{k-1} and C_k. (This was the reason for $L_1 > 2l_1$.) After its first pass, it spans up to column C_4. Now, B_2 starts arriving in C_2 but, again, it doesn't interfere with the belt of B_{k-1} and B_k since it never goes east of C_3. This scenario continues and by the step that block $B_{\lceil \frac{k}{2} \rceil}$ arrives at $C_{\lceil \frac{k}{2} \rceil}$ the belt of B_{k-1} and B_k has been consumed (i.e., all the packets in these 2 blocks have started their column movement).

Consider now how the initial belt have shrunk. After 1 full pass, the belt (which consists of all blocks) lost 2 whole blocks (i.e., B_{k-1} and B_k) and each of the remaining blocks lost 2 columns and 2 rows. Otherwise, the configuration after one pass appears to be identical to the one we started of. We will not go into the tedious calculations of the exact values for l_1, k, L_1 and h_1. Analysis similar to that of Section 4.1 shows that when l_1, k, L_1 and h_1 are $\Theta(\sqrt{n})$, the constructed permutation requires $\Omega(n\sqrt{n})$ steps for its routing by the *overshoot* algorithm. Of course, the selected values for l_1, k, L_1 and h_1 have to satisfy all the relevant restrictions mentioned in this section.

5 Conclusion

Even though there have been some encouraging results regarding hot-potato routing of permutations on two-dimensional meshes, a simple greedy-like algorithm with provably good performance, say $O(n)$, is not yet known. There are several simple and greedy-like algorithms developed over the years but the efforts to match their good experimental behaviour with an exact analysis have failed. In this paper which is a progress report of our work on packet routing, we showed that the algorithm of Borodin, Rabani and Schieber [3], which appears to belong in the class of simple and greedy-like algorithms, actually needs $\Omega(n\sqrt{n})$ steps for the routing of some permutations. Currently we are trying to establish nontrivial upper and lower bounds for the permutation routing algorithm of Ben-Aroya and Schuster [2].

References

[1] A. Bar-Noy, B. Schieber, P. Raghavan and H. Tamaki. Fast deflection routing for packets and worms. In *Proceeding of the 12th Annual ACM Symposium on Principles of Distributed Computing (PODC 93), Ithaca, NY*, pages 75–86, August 1993.

[2] I. Ben-Aroya and A. Schuster. Greedy hot-potato routing on the two-dimensional mesh. In Jan van Leeuwen (editor), *Proceedings of the 2nd European Symposium on Algorithms ESA '94 (Utrecht, September 1994)*, LNCS 855, pages 365–376. Springer-Verlag, 1994.

[3] A. Borodin, Y. Rabani and B. Schieber. Deterministic many-to-many hot potato routing. Technical Report RC 20107 (6/19/95), IBM Research Division, T.J. Watson Research Center, Yorktown Heights, NY 10598, June 1995.

[4] U. Feige and P. Raghavan. Exact analysis of hot-potato routing. In *Proceedings of the 33rd Annual Symposium on Foundations of Computer Science (Pittsburgh, Pennsylvania, October 24–27, 1992)*, pages 553–562, Los Alamitos-Washington-Brussels-Tokyo, 1992. IEEE Computer Society Press.

[5] Christos Kaklamanis, Danny Krizanc and Satish Rao. Hot-potato routing on processor arrays. In *Proceedings of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'93 (Velen, Germany, June 30 – July 2, 1993)*, pages 273–282, New York, 1993. ACM SIGACT, ACM SIGARCH, ACM Press.

[6] M. Kaufmann, H. Lauer and H. Schroder. Fast deterministic hot-potato routing on processor arrays. In D.Z. Du and X.S. Zhang (editors), *Proceedings of the 5th International Symposium on Algorithms and Computation ISAAC '94 (Beijing, P.R. China, August 1994)*, LNCS 834, pages 333–341. Springer-Verlag, 1994.

[7] I. Newman and A. Schuster. Hot-potato algorithms for permutation routing, 1993. Unpublished manuscript.

[8] M. Tompa. Lecture notes on message routing in parallel machines. Technical Report 94-06-05, Department of Computer Science and Engineering, University of Washington, June 1994.