Multipacket Hot-Potato Routing on Processor Arrays

Christos Kaklamanis[*]
DIMACS Center
Rutgers University
Piscataway, NJ 08855

Danny Krizanc[*]
School of Computer Science
Carleton University
Ottawa, Ontario K1S 5B6

(Preliminary Version)

Abstract

In this paper, we consider the problems of multipacket batch and balanced routing on d-dimensional (constant $d \geq 2$) torus and mesh-connected processor arrays. We present new "hot-potato" routing algorithms which achieve the best known average-case and worst-case time bounds for both problems on all such networks. In particular, our results include the following:

1. Algorithms that route almost all batch routing problems where each node is the source of at most $\lfloor d/2 \rfloor - 1$ packets in $dn/2 + O(\log^2 n)$ time steps on the d-dimensional n^d-node torus and in $dn + O(\log^2 n)$ time steps on the d-dimensional n^d-node mesh.
2. Randomized algorithms that route any routing problem where each node is the source and destination of at most $\lfloor d/2 \rfloor - 1$ packets in $dn + O(\log^2 n)$ time steps on the d-dimensional n^d-node torus and in $2dn + O(\log^2 n)$ time steps on the d-dimensional n^d-node mesh, with high probability.

[*]This research was partially supported by the NEC Research Institute.

1 Introduction

A number of researchers [1, 4, 3, 2, 5, 6, 7, 8, 10, 11, 14, 12] have suggested algorithms for routing packets in a network with the property that on each step, each node in the network sends all of the packets it received on the previous step along one of its outgoing edges (with at most one packet leaving per edge). Such schemes are generally referred to as *hot-potato* or *deflection* routing schemes since the packets are always moving (i.e., they are treated as "hot-potatoes") and if two packets conflict for the use of an edge one of them may be "deflected" to a node further from its destination. They have the distinct advantage over traditional store-and-forward packet routing algorithms, that use buffers to store packets between time steps, of requiring only one buffer per incoming edge. If the choice of which edge to send the incoming packets out is a simple one, then the resulting algorithm is easy to implement with minimal time between routing steps. Variants of hot-potato routing are used in the Connection Machine [9], the HEP multiprocessor [16] and in high-speed communications networks [12].

While the apparent advantages of hot-potato algorithms have been borne out by numerous simulation studies [1, 6, 7, 11, 12, 15], exact analysis of their behavior (i.e., without any independence assumptions) has proven to be difficult. Hajek [8] showed that for a natural algorithm on the N-node hypercube, k packets with worst-case destinations are delivered in $2k + \log N$ time steps. An algorithm for the hypercube suggested by Borodin and Hopcroft [4] was shown by Prager [15] to terminate in $O(\log N)$ steps on a special class of permutations. Feige and Raghavan [5] were the first to present an exact analysis of the average-case and worst-case behavior of hot-potato algorithms for two- and three-dimensional tori. Feige and Raghavan also showed that a straightforward algorithm for the hypercube terminates in $O(\log N)$ steps on average. Newman and Schuster [14] analyse the worst-case behavior of deterministic hot-potato algorithms for d-dimensional meshes ($d \geq 2$) and for the hypercube.

The model we use is the same as that in [5]. A network is modelled as a directed graph where the nodes are processors and the unidirectional edges are communication links between processors. The routing is performed in discrete, synchronous time steps. We only consider *deflection networks* where the out-degree of each node is greater or equal to its in-degree and the digraph is strongly connected. During each step, a processor receives

zero or one packet along each incoming edge and must send all the packets it received out along outgoing edges with at most one packet leaving per outgoing edge. Note that no buffers are required to hold the packets between time steps (except possibly at the source or destination nodes).

In the *k-batch* routing problem all processors generate k packets simultaneously with arbitrary destinations. We assume that k is less than or equal to the outdegree of the nodes. The $k-k$ routing problem is a special case of the k-batch routing problem where each node is the destination of k packets. The time required to route a given routing problem is the maximum over all packets of the number of time steps required to deliver each packet.

In this paper, we present new hot-potato routing algorithms for the d-dimensional n^d-node torus and for the d-dimensional n^d-node mesh, for any constant d. We give algorithms that route almost all $(\lfloor d/2 \rfloor - 1)$-batch routing problems in $dn/2 + O(\log^2 n)$ time steps on the d-dimensional mesh and in $dn + O(\log^2 n)$ time steps on the d-dimensional mesh. (By "for almost all" we mean all but a $1/n^c$ fraction for some $c \geq 1$.) Note that these algorithms are optimal in the sense that for almost all batch routing problems there is a packet that is, initially, $dn/2 - o(n)$ distance from its destination on the torus and $dn - o(n)$ distance from its destination on the mesh. Our bounds are the best known for all $d \geq 6$ and can be easily adjusted to match the bounds known for $d < 6$. Kaklamanis, Krizanc and Rao [10] gave algorithms that route almost all 1-batch routing problems in $dn/2 + O(\log^2 n)$ time steps on the d-dimensional torus and in $dn + O(\log^2 n)$ time steps on the d-dimensional mesh, for any constant d. Recently, Meyer auf der Heide and Westermann [13] presented an algorithm that routes almost all $d/88$-batch routing problems in $dn + O(d^3 \log n)$ steps, for any $d = O(n^\epsilon)$, with $0 < \epsilon < 1/2$. The results of Ben-Dor, Halevi and Schuster [3] imply a simple greedy algorithm can perform a d-batch routing problem in $O(n^d)$ steps on a d-dimensional mesh in the worst-case, but no bounds on the average-case are given.

Combining the above average-case algorithms with well-known randomized routing techniques [17] we give randomized algorithms that route every $(\lfloor d/2 \rfloor - 1) - (\lfloor d/2 \rfloor - 1)$ routing problem on a d-dimensional torus in $dn + O(\log^2 n)$ time steps and on a d-dimensional mesh in $2dn + O(\log^2 n)$ time steps, with high probability. (By "with high probability" we mean with probability $1 - 1/n^c$ for some $c \geq 1$.) Again, these are the best known bounds for all $d \geq 6$ and the only bounds we are aware of for worst-case multipacket routing on multidimensional tori and meshes. Kaklamanis, Krizanc

and Rao [10] give algorithms which solve any $1-1$ routing problem on a d-dimensional torus in $dn + O(\log^2 n)$ time steps and on a d-dimensional mesh in $2dn + O(\log^2 n)$ time steps, with high probability.

All of our algorithms have the same general structure: Packets are divided into groups which are routed in parallel in disjoint subtori along paths that closely approximate their natural greedy paths. Small cycles (or "snakes") that pass through all of the nodes in small regions of the subtori are used to facilitate moving from one dimension to another. Informally, the snakes are being used as a storage mechanism at the turns of a packet's greedy path.

It is possible for a deflection routing algorithm on certain inputs to route a set of packets in such a manner that they mutually deflect each other in an infinite loop from which they never recover. This situation is referred to as *livelock*. All of the above algorithms are constructed so as to guarantee they are free from livelock. The proof of this assertion is straightforward and is not presented in this paper.

In the next section we present in detail our algorithm for the average-case multipacket batch routing problem on the d-dimensional torus. This is followed by a discussion of how it can be extended to perform average-case multipacket batch routing on the d-dimensional mesh, and worst-case $(\lfloor d/2 \rfloor - 1) - (\lfloor d/2 \rfloor - 1)$ routing on the d-dimensional torus and mesh.

2 Multipacket Routing on the d-Dimensional Torus

In this section we prove our main result:

Theorem 1 *For any constant $d \geq 4$, there exists a hot-potato routing algorithm which routes almost all $(\lfloor d/2 \rfloor - 1)$-batch routing problems on the d-dimensional torus in $dn/2 + O(\log^2 n)$ steps.*

The proof of the theorem follows easily from the following lemma which shows how a set of packets can "fix" one of their dimensions using only edges in two dimensions of the torus. A packet is said to be *oscillating* in a dimension x if it is moving back and forth along the same edge in the x dimension on successive time steps.

Lemma 1 *Consider a two-dimensional subtorus of a d-dimensional torus consisting of nodes in the xy plane. Assume $(1-\epsilon)n$ packets are oscillating on edges in each row of the x dimension of the torus with the exception every $(c \log n)$th row (c satisfying the probabilistic conditions below). Assume further that each packet is within $c \log n$ distance of its original x-coordinate in a given batch routing problem. Then, there exists a hot-potato routing algorithm using only edges in the x and y dimensions of the torus which for almost all batch routing problems routes each packet to its final destination in the x dimension and within $c \log n$ of its starting position in the y dimension with the exception of packets destined for every $(c \log n)$th column. At the end of the routing the packets are oscillating in the y dimension. For a randomly chosen batch routing problem, the algorithm performs this task in $n/2 + O(\log^2 n)$ steps, with high probability.*

Proof: We first divide the $n \times n$ torus into $n/c \log n$ vertical strips that consist of $c \log n$ columns each. Each strip is subdivided into blocks that consist of $c \log n$ rows each, i.e., of size $c \log n \times c \log n$.

In each block, for the purposes of the routing, we construct two edge disjoint directed cycles that visit every node in the block. In particular the first cycle, referred to as the xy "snake" below, goes through the nodes in column-major snake-like order "starting" at the top-left node in the block and then moving alternately down along odd columns and up along even columns until it reaches the top right node in the block (assuming $c \log n$ is even) and then returning along the top row to the top left node of the block. The second cycle, referred to as the yx snake below, goes through the nodes in row-major snake-like order starting at the top-left node in the block and then moving alternately across odd rows and back along even rows until it reaches the bottom left node in the block and then returning along the leftmost column to the top left node of the block. The "backbone" of each snake (i.e., the top row of xy snake and the leftmost column of the yx snake) correspond to the unused rows and columns mentioned in the lemma. Note also that the backbone in one block overlaps with the last row or column of the snake corresponding to its neighboring block.

We use the following algorithm to perform the above task:

1. At a fixed time, all packets stop oscillating and instead continue to move in the direction dictated by their last oscillation. Note that this may

not be in the shortest path direction towards the packet's destination in the x dimension.

2. Packets not travelling in their shortest path direction attempt to switch directions on each step by either finding a packet moving in the opposite direction that is also going in the "wrong" direction or by finding an empty position in the flow of packets moving in its desired direction.

3. Once a packet reaches the block of processors containing its destination column in the x dimension, it attempts to turn in the y dimension in the opposite direction of the flow of the xy snake in the block. It attempts this turn at every column of the block. If it fails to turn at all columns in the block it becomes a low priority packet and is routed along arbitrary free edges so as to not interfere with the movement of the other packets.

4. Once a packet has entered the xy snake it moves around the snake in the "wrong" direction until it finds an empty position in the flow of packets moving in the snake direction at which point it reverses direction.

5. After $n/2 + 2(c \log n)^2$ steps from the beginning of the algorithm, all packets in xy snakes switch to moving in the yx snakes covering their same block.

6. As a packet travelling in the yx snake passes its final x dimension column it attempts to turn in the y dimension in either the positive or negative direction (with the exception of packets destined for the backbone of the snake). If it is successful it oscillates in the y dimension along the edge it turned into. If it fails to turn it continues around the snake until the next position that intersects its final x dimension column and attempts to turn there. It continues in this manner until it successfully turns in its destination column and is oscillating in the y dimension.

The lemma is easily obtained from the following claims concerning the algorithm above.

Claim 1 *For a random batch routing problem, all packets successfully execute step 2 (i.e., start moving in the correct direction) within $c \log n$ steps, with high probability.*

Proof: For a random batch routing problem, for each packet, the probability that its final destination column is in the postive x direction is $1/2$. It is easy to show, using Chernoff bounds, that with high probability in any segment of $c \log n$ at most $(1-\epsilon)(c \log n)$ packets have destination columns in the positive x direction. Therefore any packet with a destination column in the positive x direction that begins travelling in the opposite direction will find an empty position in the flow of packets travelling in the positive direction within $c \log n$ steps, with high probability. □

Claim 2 *With high probability, for any block and any time window of length $(c \log n)^2$, there are at most $O(1)$ packets whose destinations are inside that block and that arrive during that time window.*

Proof: For a packet to enter a block during the time window $[t, t + (c \log n)^2]$ the distance between its origin and the block must be between $t + (c \log n)^2$ and $t - c \log n$ (by claim 1). Thus for a particular block and for a particular time window there are $O(\log^3 n)$ packets that could enter the block and each has its destination in the block with probability $c \log n / n$. From this it is easy to show that at most $O(1)$ packets arrive at the block during a particular time window, with high probability. Since there are $O(n^2)$ blocks and $O(n)$ time windows the claim holds, with high probability. □

Claim 3 *With high probability, all packets that have successfully executed step 3, will successfully execute step 4 within $(c \log n)^2$ steps.*

Proof: From Chernoff bounds we know that the number of packets with destination column in a given block and origin row coinciding with the block is $(1-\epsilon)(c \log n)^2 + o(\log^2 n)$, with high probability. I.e., at any time there are at most this many packets travelling in the correct direction in the snake. Since the snake is $(c \log n)^2$ nodes long, a packet travelling in the opposite direction must find an empty position in the flow of packets travelling in the correct direction. □

Claim 4 *With high probability, for any block and any time window of size $(c \log n)^2$, any packet whose destination is inside the block and arrives at that block during that time window successfully completes step 3 within $O(1)$ steps of arrival.*

Proof: The proof is by induction on successive time windows for some particular block. For the basis step consider a block and the packets that are destined for that block and arrive in the first $(c \log n)^2$ time steps. With high probability there are at most $O(1)$ such packets according to claim 2. Also there are at most another $O(1)$ such packets that arrive in the next time window. Now consider one of the packets arriving in the first time window. After it arrives at the destination block it starts trying to turn in the y dimension and thus enter the "wrong" direction of the snake. At each attempt it fails if it encounters some other packet that is also moving in the "wrong" direction of the snake. Such a packet must have of course arrived at the destination block before the moment of the encounter. Furthermore, since the attempts are made at successive columns of the snake, any consecutive failures must be due to distinct packets. Therefore, the packet can fail at most $O(1)$ times due to packets that arrive during the first $(c \log n)^2$ time steps, and if these failures occur very close to time $(c \log n)^2$, at most another $O(1)$ times due to packets in the second time window. Thus it will successfully complete step 3 within $O(1)$ steps of its arrival at the block.

Now let us assume that the claim is true for the first m time windows of size $(c \log n)^2$. Then consider some packet that arrives at the destination block during the $(m+1)$th time window of size $(c \log n)^2$. Because of the inductive hypothesis and claim 3 all other packets destined for the block that arrived during the first $m-2$ time windows of size $(c \log n)^2$ are already moving in the correct direction of the snake. Thus the packet can only conflict with other such packets that arrive during the $(m-1)$th, the mth, the $(m+1)$th (i.e. the current), or the $(m+2)$th window of size $(c \log n)^2$. According to claim 2 with high probability there are at most $O(1)$ such packets and each one of them can be encountered only once. Therefore, every packet destined for the block that arrives during the $(m+1)$th time window of size $(c \log n)^2$ successfully completes step 3 within $O(1)$ time steps with high probability. □

Claim 5 *For a random batch routing problem, all packets successfully complete step 6 within* $(c \log n)^2$ *steps, with high probability.*

Proof: From the above claims, by $n/2 + 2(c \log n)^2$ steps after the beginning of the algorithm, all packets are travelling in the correct direction in a xy snake in the block containing their destination column, with high probability. At this point, all the packets begin moving in a yx snake in the block.

The yx snake crosses each column $c \log n$ times (with the exception of the first and last column of the snake) giving each packet $2c \log n$ oppurtunities to turn in the y direction and begin their oscillation. By Chernoff bounds, there are at most $c \log n$ packets with a destination in a particular column and therefore each packet must find a free edge to turn into in one of its attempts, with high probability. This completes the proof of the claim and of the lemma. $\Box$

Sketch of Proof of Theorem 1: Randomly distribute the $\lfloor d/2 \rfloor - 1$ packets at each node to one of $\lfloor d/2 \rfloor$ different groups. The algorithm consists of d phases. In each phase, the algorithm of lemma 1 is applied to fix a single dimension of the packets in each of the groups using edge disjoint tori in parallel. During the first phase (numbered 0), packets in group i use the torus consisting of dimensions $2i$ and $2i+1$ to fix their dimension $2i$, $i = 0, 1, \ldots, \lfloor d/2 \rfloor - 1$. During the kth phase, they use the torus consisting of dimensions $(2i+k) \bmod d$ and $(2i-1+k) \bmod d$ to fix their dimension $(2i+k) \bmod d$. During the last phase, they use the torus consisting of dimensions $(2i-1) \bmod d$ and $2i \bmod d$ to fix their dimension $(2i-1) \bmod d$. At this point they are in a snake that passes through their final destination and the routing is completed.

Note that for the lemma to apply we must be certain that (1) $(1-\epsilon)n$ packets are to be routed in each row during each application of the algorithm and (2) none of the packets to be routed have origins in a snake backbone row or a destination in a snake backbone column. For a randomly chosen batch routing problem, using Chernoff bounds one can show the number of packets being routed in any row during any application of the lemma is at most $(1-2/d)n + o(n) = (1-\epsilon)n$, for an appropriately chosen ϵ. To insure that no packets are routed in backbone rows or columns, any packet that would use such a column or row according to its random group assignment and the schedule above is routed along a shifted path reached using initial and final redistribution phases of $O(\log^2 n)$ steps each. Since the number of such packets is small, this can be done in such a way that only $o(n)$ packets are added to any group.

By repeatedly applying the lemma, we get a hot-potato algorithm which routes almost all $(\lfloor d/2 \rfloor - 1)$-batch routing problems in $dn/2 + O(\log^2 n)$ steps.

As described above, the algorithm is not livelock-free. To make it livelock-free, route any packets that are unsuccessful at any stage of the algorithm

along arbibtary free dimensions until time $dn/2 + k \log^2 n$, for an appropriately chosen k, at which time these packets can be routed using one large snake encompassing all nodes of the torus. □

An algorithm similar to the one above can be used to show:

Theorem 2 *For any constant $d \geq 4$, there exists a hot-potato routing algorithm which routes almost all $(\lfloor d/2 \rfloor - 1)$-batch routing problems on the d-dimensional mesh in $dn + O(\log^2 n)$ steps.*

Proof: Omitted. □

Using the standard two phase routing technique [17] the theorems above can be used to show:

Theorem 3 *For any constant $d \geq 4$, there exists a hot-potato routing algorithm which routes every $(\lfloor d/2 \rfloor - 1)$-$(\lfloor d/2 \rfloor - 1)$ routing problem on the d-dimensional torus in $dn + O(\log^2 n)$ steps, with high probability.*

Proof: Omitted. □

Theorem 4 *For any constant $d \geq 4$, there exists a hot-potato routing algorithm which routes every $(\lfloor d/2 \rfloor - 1)$-$(\lfloor d/2 \rfloor - 1)$ routing problem on the d-dimensional torus in $dn + O(\log^2 n)$ steps, with high probability.*

Proof: Omitted. □

References

[1] Acampora, A. and Shah, S., "Multihop lightwave networks: a comparison of store and forward and hot-potato routing", IEEE INFOCOM, 1991, 10–19.

[2] Bar-Noy, I., Raghavan, P., Schieber, B. and Tamaki, H., "Fast Deflection Routing for Packets and Worms", Symp. on Principles of Distributed Computing, 1993, 225-234.

[3] Ben-Dor, A., Halevi, S. and Schuster, A., "Potential Function Analysis of Greedy Hot-Potato Routing", Symp. on Principles of Distributed Computing, 1994, 225-234.

[4] Borodin, A. and Hopcroft, J., "Routing, merging, and sorting on parallel models of computation", Journal of Computer and System Sciences, 30, 1985, 130–145.

[5] Feige, U. and Raghavan, P., "Exact analysis of hot-potato routing" , Symposium on the Foundations of Computer Science, 1992, 553–562.

[6] Greenberg, A. and Goodman, J., "Sharp approximate models of deflection routing in mesh networks", IEEE Transactions on Computers, to appear.

[7] Greenberg, A. and Hajek, B., "Deflection routing in hypercube networks", IEEE Transactions on Computers, to appear.

[8] Hajek, B., "Bounds on evacuation time for deflection routing", Distributed Computing, 5, 1991, 1–6.

[9] Hillis, D., "The Connection Machine", MIT Press, Cambridge, Massachusetts, 1985.

[10] Kaklamanis, C., Krizanc, D. and Rao, S., " Hot-Potato Routing on Processor Arrays", Symp. on Parallel Algorithms and Architectures, 1993, 273-282.

[11] Lawrie, D. and Padua, D., "Analysis of message switching with shuffle-exchanges in multiprocessors", Interconnection Networks, IEEE Computer Society Press, 1984.

[12] Maxemchuk, N., "Comparison of deflection and store and forward techniques in the Manahattan street and shuffle-exchange networks", IEEE INFOCOM, 1989, 800–809.

[13] Meyer auf der Heide, F. and Westermann, M., "Hot-Potato Routing on Multi-Dimensional Tori", Workshop on Graph-Theoretic Concepts in Computer Science (WG 95), 1995.

[14] Newman, I. and Schuster, A., "Hot-potato algorithms for permutation routing", Technion Technical Report, CS-LPCR 9201, 1992.

[15] Prager, R., "An algorithm for routing in hypercube networks", University of Toronto Technical Report, 1986.

[16] Smith, B., "Architecture and applications of the HEP multiprocessor computer system", Real Time Signal Processing, 1981, 241–248.

[17] Valiant, L., "A Scheme for fast parallel communication", SIAM Journal of Computing, 11, 1982, 350–361.