Experimental Evaluation of Hot–Potato Routing Algorithms on 2–Dimensional Processor Arrays*

Constantinos Bartzis[1**], Ioannis Caragiannis[2], Christos Kaklamanis[2], and Ioannis Vergados[2]

[1] Department of Computer Science and Engineering
University of California, Santa Barbara, USA
[2] Computer Technology Institute and
Dept. of Computer Engineering and Informatics,
University of Patras, 26500 Rio, Greece.

Abstract. In this paper we consider the problem of routing packets in two–dimensional torus–connected processor arrays. We consider four algorithms which are either greedy in the sense that packets try to move towards their destination by adaptively using a shortest path, or have the property that the path traversed by any packet approximates the path traversed by the greedy routing algorithm in the store–and–forward model. In our experiments, we consider the static case of the routing problem where we study permutation and random destination input instances as well as the dynamic case of the problem under the stochastic model for the continuous generation of packets.

1 Introduction

We consider a form of packet routing known as *hot–potato* routing. The network is modeled as a directed graph where the nodes are the processors and the unidirectional edges are communication links between processors. Each processor has an injection buffer and a delivery buffer. When a new packet is generated, it is stored in the injection buffer of its source processor; when a packet reaches its destination processor, it is stored in the delivery buffer. The routing is performed in discrete, synchronous time steps. 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. No buffers are required to hold the packets between the time steps. Any packet that arrives at a node other than its destination must immediately be forwarded to another node. The topology we consider in this paper is that of the 2–dimensional torus–connected processor array.

* This work was partially funded by the European Union under IST FET Project ALCOM–FT and RTN Project ARACNE. An extended version of the paper can be found at http://students.ceid.upatras.gr/~caragian/bckv00.ps

** Part of this work was performed while the author was at the Department of Computer Engineering and Informatics, University of Patras, Greece.

A. Bode et al. (Eds.): Euro-Par 2000, LNCS 1900, pp. 877–881, 2000.

In static (or batch) routing problems, all processors generate a single packet simoultaneously. The running time of a routing problem is the number of time steps required to deliver all packets to their destinations.

In dynamic routing problems, each node continously generates packets with an injection rate λ. New packets are stored in the injection buffer and wait to be served. When a processor receives less than four packets along its incoming edges, it considers a packet from its injection buffer. When a packet starts moving, it is never buffered at any node until it reaches its destination, where it is stored in the delivery buffer and absorbed.

The first hot-potato algorithm was proposed by Baran [1]. Borodin and Hopcroft [4], Prager [13] and Hajek [8] presented algorithms for hypercubes. Hot–potato routing algorithms for 2–dimensional meshes and tori were proposed by Bar–Noy et al. [2], Ben–Aroya et al. [3], Newman and Schuster [12], Kaufman et al. [10], Feige and Raghavan [7], and Kaklamanis et al. [9]. All of them deal with batch routing problems. The only study of the dynamic case we are aware of is that of Broder and Upfal [5].

An important class of hot–potato routing algorithms is that of greedy algorithms. In these algorithms, each node forwards each packet closer to its destination whenever possible. Although greedy algorithms have been observed to work well in practice (for static routing problems), the known theoretical results for their performance are far from being tight (see Busch et al. [6]).

Especially for meshes and tori, a class of hot–potato routing algorithms that has received much attention is that of algorithms that make packets follow paths that approximate their natural greedy path (i.e., the path utilized by the greedy routing algorithm in the store–and–forward model [11]). Such algorithms were proposed and analyzed in [9].

2 Short Description of the Algorithms

In this section, we briefly describe four algorithms on the 2–dimensional torus network, namely the greedy algorithm, algorithm A1, algorithm KKR, and algorithm A2. The greedy algorithm is a variation of the folklore algorithm mentioned in the bibliography. Algorithm KKR was proposed in [9]. Algorithm A1 is a simple algorithm that "approximates the greedy path" while algorithm A2 is a variation of algorithm KKR. To our knowledge, algorithms A1 and A2 have not been studied in previous work.

The greedy algorithm. The greedy algorithm which was implemented tries to move packets towards their destination by adaptively using shortest paths, also trying to minimize the difference between the horizontal and the vertical distance of each packet from its destination.

The decisions the algorithm makes are local, since they depend on the destination of the incoming packets and the order in which these packets are examined. The implementation of the algorithm obeys the one–pass property [8].

A simple algorithm that approximates the greedy path. From the point of view of the motion of the packets, packets routed by algorithm A1, start

moving into their rows and continuously turn right so that they move around their destination row, until they reach it (see Figure 1).

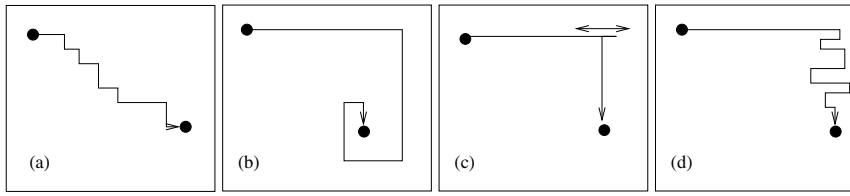

Fig. 1. Typical movement of packets performed by (a) the greedy algorithm, (b) algorithm A1, (c) algorithm KKR, and (d) algorithm A2.

The algorithm KKR [9]. Each packet p starts moving along its row, following the shortest path towards its destination column. When it reaches its destination column, p attempts to enter the column and moves towards its destination. If it fails, it moves "back and forth" until it succesfully turns into the right column.

A variation of the algorithm KKR. Algorithm A2 is based on the following idea: during the time steps that a packet p is moving "back and forth" along its row trying to turn into the correct column, it could also try to turn at some other node trying to decrease the vertical distance of the packet from its destination.

This can be seen as a movement of a packet in its row until it reaches its destination column, and then, greedy movement to the destination processor. Thus, algorithm A2 maintains both properties: it is greedy and also approximates the greedy path.

The typical shape of the paths traversed by the packets during the execution of the algorithms is depicted in Figure 1.

3 Experimentation

The four algorithms where implemented in C, and the results were conducted by simulation experiments on a Pentium III/500Mhz running Solaris 7.

In the static model, packets are initially stored at the injection buffer and start moving according to the routing algorithm. Initially, each processor has one packet stored in its injection buffer. We consider routing problems with *random destinations* (i.e., each packet is assigned as destination a processor, selected among all the processors of the network, uniformly at random) and *random permutations* (i.e., the routing problem is selected uniformly at random among all possible permutations).

In our experiments, the parameter of interest was the running time of the algorithms. Statistics on the running time of the algorithms on routing problems with random destinations are depicted in Table 1. The results for random permutations are close to those for random destinations.

	200×200			500×500		
	Average	Max.	Std dev.	Average	Max.	Std dev.
Greedy	202.69	207	1.509	505.85	512	2.032
A1	201.39	205	1.214	501.16	506	1.412
KKR	205.19	214	2.881	505.13	511	2.553
A2	204.98	217	3.387	505.2	515	2.785

Table 1. Statistics from the execution of the four algorithms on 100 routing problems with random destinations in tori 200×200 and 500×500. The average and maximum observed running time, as well as the standard deviation of the running time is shown.

The performance of all algorithms is very close to the optimal. We believe that all the four algorithms route (almost all) batch routing problems in time $n + O(\log n)$ on the $n \times n$ torus. Such a strong theoretical result has only been proved for algorithm KKR in [9]. Algorithm A1 has slightly better performance than the other three algorithms. Surprisingly, algorithm A2 does not improve the running time of algorithm KKR.

In the dynamic model, packets with random destinations are continuously generated at each processor with a rate λ (i.e., at each time step, a processor generates a packet with probability λ, independently from the other processors) and stored in the injection buffer. Injection buffers have been implemented as FIFO (first–in–first–out) queues; so the network together with the injection buffers is considered as a queueing system. Once a packet leaves the injection buffer of its origin processor, it starts moving according to the routing algorithm until it reaches its destination, where it is stored in the delivery buffer and absorbed (leaves the system).

In our experiments under the dynamic model, parameters of interest were the delay of packets, the number of packets in the system (i.e., the number of packets in injection buffers and packets being routed) and the network throughput, i.e., the maximum injection rate for which the system is stable.

A theoretical maximum value for the maximum injection rate on the $n \times n$ torus is given by $\lambda_{max} = 8/n$ [11]. We alternatively express the injection rate (and the network throughput) as a percentage of its theoretical maximum value. Although we never observe stable behavior of the system for injection rates very close to the theoretical maximum, we observed network throughput up to $72,5\%$.

We performed experiments on the 200×200 torus for injection rates smaller than 50% (see Table 2). In this case, for the four algorithms we study, the network is stable. We observe that, even for small injection rate, packets experience significant delays when routed with the greedy algorithm, while the average number of the packets in the network is large. Especially the average (total) size of injection buffers when the greedy algorithm is used is more than twice the average size of injection buffers when any of the other three algorithms is used.

In our experiments on the 200×200 torus, the network throughput observed was about $0,0276$ (69%) for the greedy algorithm, 0.0254 (63.5%) for algorithm A1 0.0265 (66.25%) for algorithm KKR, and 0.029 (72.5%) for algorithm A2.

	Average delay			Average size of injection buffers		
	$\lambda = 0.005$	$\lambda = 0.01$	$\lambda = 0.02$	$\lambda = 0.005$	$\lambda = 0.01$	$\lambda = 0.02$
Greedy	0.76	2.547	10.932	189	1,026	9,088
A1	0.336	0.943	4.664	156	386	3,835
KKR	0.475	1.304	6.456	78	543	5,383
A2	0.444	0.88	4.444	46	486	3,720

Table 2. Average delay of packets and average total size of injection buffers. The average on the delay was taken among all packets that reached their destinations within $2,500$ steps of execution. The average size of injection buffers was computed for the steps of execution 1000–2500.

References

1. P. Baran. On Distributed Communication Networks. *IEEE Transactions on Communications*, pp. 1–9, 1964.
2. A. Bar–Noy, P. Raghavan, B. Shieber, and H. Tamaki. Fast Deflection Routing for Packets and Worms. In *Proc. of the 12th Annual ACM Symposium on Principles of Distributed Computing*, pp. 75–86, 1993.
3. I. Ben–Aroya, T. Eilam, and Schuster. Greedy Hot–Potato Routing on the Two–Dimensional Mesh. *Distributed Computing*, 9(1):3–19, 1995.
4. A. Borodin and J. Hopcroft. Routing, Merging, and Sorting on Parallel Models of Computation. *Journal of Computer and System Sciences*, 30:130–145, 1985.
5. A. Broder and E. Upfal. Dynamic Deflection Routing on Arrays. In *Proc. of the 28th Annual ACM Symposium on the Theory of Computing*, pp. 348–358, 1996.
6. C. Busch, M. Herlihy, and R. Wattenhofer. Randomized Greedy Hot–Potato Routing. In *Proc. of the 11th Annual ACM/SIAM Symposium on Discrete Algorithms (SODA '00)*, pp. 458–466, 2000.
7. U. Feige and P. Raghavan. Exact Analysis of Hot–Potato Routing. In *Proc. of the 33rd Annual IEEE Symposium on Foundations of Computer Science*, pp. 553–562, 1992.
8. B. Hajek. Bounds on Evacuation Time for Deflection Routing. *Distributed Computing*, 5:1–6, 1991.
9. C. Kaklamanis, D. Krizanc, and S. Rao. Hot–Potato Routing on Processor Arrays. In *Proc. of the 5th Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 273–282, 1993.
10. M. Kaufmann, H. Lauer, and H. Schroder. Fast Deterministic Hot–Potato Routing on Meshes. In *Proc. of the 5th International Symposium on Algorithms and Computation*, LNCS 834, Springer–Verlag, pp. 333-341, 1994.
11. F.T. Leighton. Average Case Analysis of Greedy Routing Algorithm on Arrays. In *Proc. of the 2nd Annual ACM Symposium on Parallel Algorithms and Architectures*, pp. 2–10, 1990.
12. I. Newman and A. Schuster. Hot–Potato Algorithms for Permutation Routing. *IEEE Transactions on Parallel and Distributed Systems*, 6(11): 1168–1176, 1995.
13. R. Prager. An Algorithm for Routing in Hypercube Networks. Master's thesis, University of Toronto, 1986.