A Note on Off-line Permutation Routing on a Mesh-Connected Processor Array

Danny Krizanc
Department of Computer Science
University of Rochester
Rochester, NY 14627

Abstract

We show how to off-line route any permutation of a $n \times n$ mesh-connected processor array in $2.5n - 3$ steps with at most 6 packets per processor per time step and in $2.25n + 3$ steps with at most 8 packets per processor per time step.
Keywords: Parallel architectures, parallel algorithms, mesh-connected array, off-line permutation routing.

1 Introduction

The mesh-connected array is the basis for a number of proposed and implemented parallel computers due to its simplicity and the regularity of its interconnection pattern which makes it ideal for VLSI implementation. The performance of an algorithm on a packet-switched mesh depends on the mesh's ability to realize the communication patterns between processors arising from the algorithm. Often it is the case that these communication patterns are permutations of the processors which are known at the time of the algorithm's design. This leads naturally to the problem of off-line routing of permutations on the mesh, i.e., given a fixed permutation of the processors, each processor being the origin and destination of one packet, find paths connecting the origin-destination pairs and a schedule of when packets cross the edges of their path, which minimizes the time required to realize the

permutation. At the same time we are interested in minimizing the number of packets a single processor must store between steps of the routing.

Our model of the $n \times n$ mesh is sometimes referred to as the MIMD model. It consists of n^2 processors with the interconnections between them defined by the two dimensional grid without wraparound edges. Edges are assumed to be bidirectional and a processor can send (and receive) a single packet along each of its edges in single time step.

There is an obvious lower bound of $2n-2$ steps for our problem since in the worst case a permutation may have a packet originating in the top left corner processor and destined for the bottom right corner processor. Lieghton, Makedon and Tollis [3] gave an algorithm for on-line permutation routing (and therefore for off-line as well) which achieves this bound. However, in their algorithm the processors must store a large number of packets between communication steps. (The number is constant but estimated to be in excess of 500.) As a corollary to one of their theorems, Annexstein and Blaumslag [1] show how to off-line route in $3n - 3$ steps with at most 3 packets per processor. In this note we extend this result by showing that by allowing only a slight increase in the amount of storage required a great improvement in the time can be made. In particular, by allowing at most 6 packets per processor per step any permutation can be off-line routed in $2.5n - 3$ steps and by allowing 8 packets per processor per step any permutation can be off-line routed in $2.25n + 3$ steps.

2 Off-line Routing

Our results require the following lemmas which are of interest in their own right. We assume that $n = 4k$, for some k, whenever it is convient in the discussion below.

Lemma 1 *On an n processor linear array, any permutation can be routed (on- or off-line) in $n - 1$ steps with at most 3 packets per processor per time step.*

Proof. The straightforward greedy algorithm achieves this bound. Note that at most 2 packets enter a processor during a step and they both leave on the next step unless the given processor is their destination. Since we are

routing a permutation, at most 1 packet has a processor as its destination so that at most 3 packets ever reside at a processor. □

Lemma 2 *On an n processor linear array, 2 permutations can be simultaneously routed (on- or off-line) in $n+1$ steps with at most 4 packets per processor per time step.*

Proof. We use the well known reduction of routing to sorting by destination and show how to sort 2 sequences in $n+1$ steps. It is easy to show that odd-even transposition sort sorts a sequence on a linear array in n steps (see [2]). During odd steps of the sort, packets are compared and exchanged along odd edges of the array, and during the even steps along even edges. To perform 2 sorts simultaneously, start a second sort after the first step of the first sort using the odd edges while the first sort uses the even edges and vice versa. The second sort completes after $n+1$ steps. On any step a processor receives at most 2 packets and at most 2 packets are destined for a processor, so that at most 4 packets reside at a processor between steps. □

Lemma 3 *On an $n \times n$ mesh, 2 permutations can be simultaneously routed off-line in $3n-3$ steps with at most 6 packets per processor per time step.*

Proof. Recall how Annexstein and Blaumslag [1] perform off-line permutation routing on the mesh. They show that any permutation can be realized by first performing a permutation of each of the rows, followed by a permutation of the columns and finishing with another permutation of the rows. (The three permutations are calculated off-line.) We observe that a second permutation may be simultaneously routed by first performing a permutation of each of the columns (while the first permutation is being routed in the rows) followed by permutations in the rows and then again in the columns. Using lemma 1 the above can be performed in $3n-3$ steps. On any step a processor receives at most 4 packets and at most 2 packets are destined for a processor, so that at most 6 packets reside at a processor between steps. □

Lemma 4 *On an $n \times n$ mesh, 4 permutations can be simultaneously routed off-line in $3n+3$ steps with at most 8 packets per processor per time step.*

Proof. Analagous to lemma 3, we send 2 permutations first by row, then by column and finally by row and send the other 2 permutations first by column,

then by row and finally by column. Using lemma 2, the routing is completed in $3n + 3$ steps. On any step a processor recieves at most 4 packets and is the destination of at most 4 packets, so that at most 8 packets reside at a processor between steps. □

We are now ready to show our main results.

Theorem 1 *On an $n \times n$ mesh, any permutation can be off-line routed in $2.5n - 3$ steps with at most 6 packets per processor per time step.*

Proof. Divide the mesh into 4 $n/2 \times n/2$ submeshes in the natural fashion (i.e., into quadrants). Imagine the submeshes as forming a 2×2 mesh. Our algorithm consists of 3 phases. In the first phase, packets move either 0 or $n/2$ steps in their row so as to be in their correct column of the 2×2 mesh. After this phase each processor contains 0, 1, or 2 packets. During the second phase we route within the submeshes. If a packet's destination is within the submesh it goes there. If its destination is within the other submesh in the column of submeshes, it goes to the image of its final destination if the two submeshes were overlaid. Any processor is now the intermediate destination of 0, 1 or 2 packets. It is easy to see that the $2 - to - 2$ mapping thus defined can be completed to form 2 permutations which may be routed off-line using lemma 3. During the last phase, packets travel either 0 or $n/2$ steps to their final destination. The storage requirements of the algorithm are the same as those of the algorithm for lemma 3. □

Theorem 2 *On an $n \times n$ mesh, any permutation can be off-line routed in $2.25n + 3$ steps with at most 8 packets per processor per time step.*

Proof. The proof is analogous to theorem 1. Divide the mesh into 16 $n/4 \times n/4$ submeshes with the submeshes forming a 4×4 mesh. First we correct the column submesh by moving 0, $n/4$, $n/2$ or $3n/4$ steps. The second phase consists of a $4 - to - 4$ mapping which is realized using lemma 4. During the last phase, packets must travel 0, $n/4$, $n/2$ or $3n/4$ steps to their final destination. By sending those that must go furthest first, this can be achieved in $3n/4$ steps. The storage requirements are the same as those of the algorithm in lemma 4. □

3 Discussion

It should be noted that the above algorithms are very easy to implement requiring only $O(\log n)$ bit headers and the ability to check a flag, subtract 1, and test if a value equals 0. The question of wether every permutation can be routed (on- or off-line) in $2n - 2$ steps using a small amount of storage (say less than 10 packets) remains open.

References

[1] F. Annexstein and M. Blaumslag, *A Unified Approach to Off-Line Permuation Routing on Parallel Networks,* Symp. on Parallel Algorithms and Architectures, 1990, pp. 398-406.

[2] T. Leighton, C. E. Leiserson, B. Maggs, S. Plotkin and J. Wein, *Lecture Notes for Theory of Parallel and VLSI Computation,* MIT/LCS/Research Seminar Series 1, 1988.

[3] T. Leighton, F. Makedon and I. Tollis, *A* $2n - 2$ *Step Algorithm for Routing in an* $n \times n$ *Array with Constant Size Queues,* Symp. on Parallel Algorithms and Architectures, 1989, pp. 328-335.