



The University of Sydney

An Empirical Study of Off-line Permutation Packet Routing on 2-dimensional meshes based on the Multistage Routing Method

Technical Report Number 477

February, 1994

Antonios Symvonis and Jonathan Tidswell

ISBN 0 86758 904 3

**Basser Department of Computer Science
University of Sydney NSW 2006**

An Emperical Study of Off-Line Permutation Packet Routing on 2-Dimensional Meshes Based on the Multistage Routing Method

Antonios Symvonis

Basser Department of Computer Science
University of Sydney
Sydney, N.S.W. 2006
Australia
symvonis@cs.su.oz.au

Jonathon Tidswell

Hydrographic Sciences Australia P/L
PO Box 85
Cammeray, NSW 2062
Australia
jont@hsa.com.au

February 1, 1994

Abstract

In this paper we present the *multistage off-line method*, a new and rather natural way to model off-line packet routing problems, which reduces the problem of off-line packet routing to that of finding edges disjoint paths on a multistage graph. The multistage off-line method can model any kind of routing pattern on any graph and can incorporate the size of the maximum queue allowed to be created in any processor. The multistage off-line method accommodates all optimal solutions and can be used to obtain lower bounds based on the actual routing problem under consideration. The paths of the packets are computed by a greedy heuristic method. A lot of freedom is left for the user of the method in order to incorporate in the heuristic properties of the underlying interconnection network. Based on the multistage off-line method, we studied the permutation packet routing problem on 2-dimensional meshes. We ran millions of experiments based on random generated data and, for all of our experiments, we were able to compute a solution of length equal to the maximum distance a packet has to travel, and thus, match the actual lower bound for each routing pattern.

Index terms: Mesh, multistage graph, off-line algorithm, packet routing, permutations.

1 Introduction

A crucial component of any large scale parallel machine is the algorithm which is used to route messages (also called packets) between nodes in the underlying network. 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 (queues are created while two packets are competing for the same communication channel). This is the task of the packet routing algorithm.

Obviously, during the course of the execution of a parallel algorithm, several communication patterns are generated. It is fair to assume that these patterns are not known in advance, and thus, all the routing decisions have to be taken during the execution of the algorithm. In other word, routing is performed by an on-line algorithm.

However, this is not always the case. In several algorithms the communication patterns are known in advance. Algorithms which perform matrix operations form such an example. When this situation arises, we can handle the communication part of the algorithm off-line. Solving the routing problem in an off-line fashion has great practical benefits. The router can become part of the compiler. Besides its usual and well known functions, the compiler will also generated code which will route packets of information through specific paths [3]. This results in faster execution time. All the overhead that would be required by the on-line routing algorithm is eliminated. We have to emphasize that this amount of overhead is significant. This is probably the reason why, in practice, despite the existence of optimal routing algorithms in terms of time and space, the parallel computing industry prefers to use very simple and nonoptimal algorithms. (Parallel machines based on the mesh interconnection network serve as an example.)

Another reason for which we are keen to investigate the off-line packet routing problem is because we hope that an off-line solution can help us to design better on-line algorithms. A lot of work has been done on on-line packet routing [8, 10, 11, 12, 14, 15, 16, 17, 18, 20]. However, for all nontrivial networks, the question of whether it is possible to route a permutation in optimal time by using no queues (or constant queues of small size, say 2-5 packets) is still open. One way to attack the problem is to obtain an off-line solution and to learn from its structure. This approach was proved successful in [8].

In this paper, we are going to study the off-line packet routing problem. Formally, in an off-line packet routing problem we are given a graph which represents the underlying interconnection network of a parallel machine and a set of routing requests. Each request consists of a tuple (*origin, destination*) where *origin* and *destination* are vertices in the graph. We are required to compute a path for each request. The computed paths have to be such that, when all requests are routed together, the routing is efficient. In other words, it is fast (near the lower bound) and requires small constant size queueing area in each node. Usually lower bound are obtained based on distance and bisection-like arguments.

The only previous work on off-line packet routing which applies to a large class of interconnection networks is that of Annexstein and Baumslag [1]. They presented a method to solve the permutation off-line packet routing problem on product graphs. Their algorithm produces optimal schedules which are within a constant factor of the worst case lower bound. During the routing queues are never created. For the case of $n \times n$ meshes, their algorithm produces paths of length $3n - 3$. Later on, Krizanc [9] presented an algorithm which, for $n \times n$ meshes, creates

paths of length $2.5n$ and uses queues of size at most 2. When queues of size 4 are allowed, the length of the paths reduces to $2.25n$. Krizanc's algorithm can be considered to be a fine tuning of the algorithm of Annexstein and Baumslag. Using a different method, Kaklamani, Krizanc and Rao [6] produced a solution to a permutation problem on a 2-dimensional mesh (of side length n) of size $2n - 1$ which uses queues of size 4. This latter result is very impressive since $2n - 2$ is the diameter of an $n \times n$ mesh and the queue size is quite small. However, in this paper we will demonstrate that it is reasonable to believe that the above result can be further improved. Our experimental work showed that it was always possible to find a solution in which the routing terminated after so many steps as the maximum distance some packet had to travel (in the permutation studied) and without using queues.

We propose a new method for the off-line packet routing problem. Our method can accommodate any kind of interconnection network and any kind of routing pattern. Furthermore, our method can incorporate a queue of variable length which reflects the buffering capabilities of the processors in the parallel machine. An additional feature is that it tries to compute paths which are of length close to the actual lower bound dictated by the routing pattern under consideration and not a general worst case lower bound.

The rest of the paper is organized as follows: In the next section, we define the terminology used in the paper. In Section 3, we present the *multistage off-line routing method*, our way of modeling off-line packet routing, we discuss its modeling power and we present the general form of a routing algorithm which can be used in conjunction with it. In Section 4, we apply our method to examine off-line permutation routing on 2-dimensional meshes. We present an extended study of the permutation problem based on simulations. For all the simulations we run on randomly generated data, the number of routing steps required by our algorithm is equal to the maximum distance a packet has to travel, and thus, optimal. We also present upper bounds on the length of the solutions obtained by using the multistage method for routing permutations on 2-dimensional meshes. We conclude in Section 5, with a discussion on the potential of our method and on open problems.

2 Preliminaries

A *finite directed graph* $G = (V, E)$ is a structure which consists of a finite set of vertices V and a finite set of edges $E = \{e_1, e_2, \dots, e_{|E|}\}$. Each edge is incident to the elements of an ordered pair of vertices (u, v) . u is the start-vertex of the edge and v is its end-vertex.

Edges with the same start and end-vertices are called *self-loops*. We define the directed *self-loop augmented graph* $G^{SL} = (V, E')$ of $G = (V, E)$ to be the graph with $E' = E \cup \{e^v = (v, v) | v \in V\}$ (one self loop is added for each vertex in G provided that it does not already exist).

A *directed path* is a sequence of edges e_1, e_2, \dots such that the end-vertex of e_{i-1} is the start-vertex of e_i . Edges with the same start-vertex and the same end-vertex are called *parallel*. We say that a directed graph which contains parallel edges is a *directed multigraph*.

The set $Neighbors(v, G)$ is defined to be the set of vertices in G which can be reached from v by crossing just one edge. Formally, $Neighbors(v, G) = \{w | (v, w) \in E \text{ of } G\}$.

The *product graph* $F = G \times H = (V_F, E_F)$ of two graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$ is formed by taking the cross product of G and H . Formally,

$$V_F = \{(v_g, v_h) \mid v_g \in V_G, v_h \in V_H\}$$

and

$$E_F = \{((v_g, v_{h1}), (v_g, v_{h2})) \mid v_g \in V_G, (v_{h1}, v_{h2}) \in E_H\} \\ \cup \{((v_{g1}, v_h), (v_{g2}, v_h)) \mid (v_{g1}, v_{g2}) \in E_G, v_h \in V_H\}.$$

An *off-line* routing problem R is defined by a triple (G, P, k) where $G = (V, E)$ is the directed graph representing the network in which the routing will take place (vertices in V represent processors and edges in E represent unidirectional communication links). The elements in set P represent the m packets to be routed. Formally, $P = \{p_1, p_2, \dots, p_m \mid p_i = (orig_i, dest_i), orig_i, dest_i \in V, 1 \leq i \leq m\}$. Finally, k is the maximum number of packets which are allowed to queue at any processor during the routing.

Note that, in the literature, the maximum queue size allowed was not part of the definition. We decided to include it in the statement of the problem in order to get solutions which are closer to reality and also utilize resources at the maximum.

There is no restriction on the number of packets which originate from, or, are destined for, a certain processor. 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) *packet routing problem*. If $h_1 = 1$ we have a *many-to-one routing problem* (*many* processors send packets to *one* processor), if $h_2 = 1$ 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 *permutation routing problem*.

A solution of length L for the off-line packet routing problem $R = (G, P, k)$ is a set of directed paths, $SOLUTION(R) = \{d_1, d_2, \dots, d_m\}$ where d_i is the directed path corresponding to packet p_i . The paths are taken on graph G^{SL} , the self-loop augmented graph of G , instead of G . We do that in order to make possible to incorporate self loops in the directed paths. A self loop from vertex v in the path of some packet indicates that the packet was queued in processor v at the corresponding routing step. Each directed path contains at most $L + 1$ vertices. For $i = 1 \dots m$ we have that

$$d_i = v_i^0 v_i^1 \dots v_i^l, 0 \leq l \leq L, v_i^0 = orig_i \text{ and } v_i^l = dest_i.$$

In order to have a valid solution for the routing problem, the directed paths must satisfy the following two conditions:

1. At any routing step, each edge which corresponds to a unidirectional communication link appears in at most one directed path, and
2. at any routing step, each self loop appears in at most k directed paths.

Assume an interconnection network, represented by a directed graph G , in which each processor has a queueing area of k packets. Let \mathcal{P} be a class of routing patterns and consider the set $\mathcal{R} = \{(G, P, k) \mid P \in \mathcal{P}\}$ of routing problems. Denote by $LB(G, P, k)$ the number of routing

steps required to solve the routing problem (G, P, k) by any routing algorithm. $LB(G, P, k)$ is a lower bound on the length of any off-line routing solution for (G, P, k) . Because $LB(G, P, k)$ depends on the routing pattern P , we refer to it as the *actual lower bound* (for routing pattern P). Define $LB(G, \mathcal{P}, k) = \max_{P \in \mathcal{P}}(LB(G, P, k))$. We refer to $LB(G, \mathcal{P}, k)$ as the *worst case lower bound* (for the class of routing patterns \mathcal{P}).

When our problem is that of routing permutations on an $a \times b$ mesh, the worst case lower bound is $a + b - 2$ since it takes so many routing steps for a packet initially at the top-left corner of the mesh to reach the bottom-right one. For a specific permutation P , the actual lower bound with respect to P is the maximum of the distances that individual packets have to travel.

3 The Multistage Off-Line Packet Routing Method

In this section we present a new way to model the off-line packet routing problem as a graph theoretic problem. For reasons that will become evident in the rest of the section, we call our method the *multistage off-line packet routing method*.

3.1 The Multistage Graph

Consider any routing problem $R = (G, P, k)$ where $G = (V, E)$ is the directed graph (with no self-loops) which represents the interconnection network in which the routing takes place, P is the set of packets to be routed and, k is the maximum number of packets which can be queued at any processor. Our goal is to achieve routing time near the actual lower bound of the problem. Assume a lower bound of T routing steps for the problem under consideration (for now consider worst case trivial lower bounds).

We construct a multistage directed multigraph $G' = (V', E')$ as follows:

$$V' = \{(v, t) \mid v \in V \text{ and } 0 \leq t \leq T\}$$

and

$$E' = \{((v, t), (w, t + 1)) \mid w \in \text{neighbors}(v, G) \text{ and } 0 \leq t < T\} \cup \{e_v^1, e_v^2, \dots, e_v^k \mid e_v^i = ((v, t), (v, t + 1)), v \in V, 0 \leq t < T, 1 \leq i \leq k\}.$$

The edges in the first term of E' represents the communication that can take place between adjacent vertices of the interconnection network at any time. The edges in the second term of E' represent a queue which resides in any vertex v and can grow up to k packets.

Figure 1 shows the resulting graph when the interconnection network is a chain of length 5 and no queues are allowed during the routing. For permutations, an obvious lower bound of 4 routing steps which is based on a distance argument applies.

Let $tower(G', v)$ be the set of vertices of graph G' (the constructed multistage graph) which correspond to vertex v in G . Formally,

$$tower(G', v) = \{(v, t) \mid v \in V, (v, t) \in V', 0 \leq t \leq T\}.$$

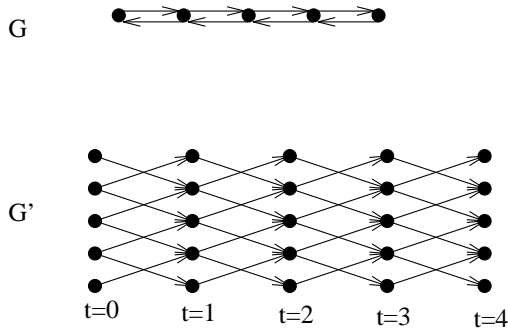


Figure 1: A chain of 5 vertices and its corresponding multistage graph.

Theorem 1 *Assume any routing problem $R = (G, P, k)$ on graph G . R has a solution of length L if and only if for each packet $p_i = (orig_i, dest_i) \in P$ there exist a path from a vertex $(orig_i, t)$ in $tower(G', orig_i)$ to vertex $(dest_i, t')$ in $tower(G', dest_i)$, $t \leq t' \leq L$ and all such paths are mutually edge disjoint.*

Proof First assume a solution $SOLUTION(R) = \{d_1, d_2, \dots, d_m\}$ where d_i is the directed path corresponding to packet $p_i = (orig_i, dest_i)$. We can map all directed paths in $SOLUTION(R)$ to directed paths in the multistage graph G' . For any directed path $d_i = v_i^0 v_i^1 \dots v_i^l$, $0 \leq l \leq L$, $1 \leq i \leq m$, we map edge (v_i^t, v_i^{t+1}) of d_i to edge $((v_i^t, t), (v_i^{t+1}, t+1))$ of G' .

Since in $SOLUTION(R)$, at any time step, each edge which corresponds to transmission of a packet appears in at most one directed path, these edges will never map in the same edge of G' . Also, since in G' there are k edges from vertex $((v, t), (v, t+1))$, we can map the (at most) k identical self loops which appear at any routing step of $SOLUTION(R)$ to different edges of G' .

Similarly,, we can obtain from a set of edge disjoint paths, each of the form $(orig_i, t) \dots (dest_i, t')$, for packet p_i , a solution $SOLUTION(R) = \{d_1, d_2, \dots, d_m\}$ for the routing problem $R = (G, P, k)$ of length L , where L is the maximum stage number out of all stages reached by some path. ■

Theorem 1 states that the problem of obtaining an off-line solution to a routing problem reduces to the problem of finding edge disjoint paths in the directed multigraph G' . This allows us to view the problem from a different view and to treat it as a graph theoretic problem.

Note that the number of stages of the resulting multistage graph G' was defined in terms of a known lower bound. If this lower bound is less than the actual lower bound, we will fail to find a solution. The fact that G' can be extended by the addition of new stages resolves this problem.

3.2 The Modeling Power of the Multistage Off-Line Packet Routing Method

The only known method which solves off-line packet routing problems for a large class of graphs is that of Annexstein and Baumslag [1] for product graphs. They provided an algorithm, based on Hall's matching theorem, to support the following theorem:

Theorem 2 [Annexstein and Baumslag, [1]] *Assume that we know how to route off-line any permutation on graphs G and H in at most $r(G)$ and $r(H)$ routing steps, respectively, without using any queues. Then, we can route off-line any permutation on the product graph $F = G \times H$ in $r(F) = 2 \cdot \min(r(G), r(H)) + \max(r(G), r(H))$ steps (again, without creating any queues).*

It is obvious that the above theorem can be used successfully only for a limited class of routing problems. It is useful only when the routing problem is a permutation and only when the interconnection network is the product of two graphs on which we know how to route permutations efficiently. More importantly, the number of routing steps required for the routing of a given permutation π on F is not expressed as a function of the actual lower bound (for permutation π) but rather as a function of upper bounds on the number of steps required to route any permutation on graphs G and H . Note also that it is not possible to extend the method of Annexstein and Baumslag, at least not in a straight forward way, to use queues and possibly reduce the required number of routing steps.

The definition for a routing problem we use in this paper is quite general. When combined with Theorem 1, it reveals the modeling power of the multistage off-line routing method. The multistage routing method can model routing problems on any graph and of any routing pattern, not just permutations. Queues can be also modeled. This enables the routing algorithm to fully utilize all the available resources of the interconnection network.¹

Theorem 1 reveals another important aspect of the modeling power of the multistage routing method. For a given routing problem, any solution of the problem can be mapped to a set of edge disjoint paths on the produced multistage graph. Thus, the set of paths which corresponds to the optimal solution for the routing problem on hands is also contained in the multistage graph. It is up to us to identify this set of paths. In Section 4, we will provide a heuristic way to find edge disjoint paths for routing permutations on 2-dimensional meshes. That heuristic produced optimal solutions (with respect to the actual lower bound) for all the permutations we studied. For the same problem, i.e., routing permutations on an $a \times b$ mesh, $a \geq b$, we will also provide a path selection algorithm which produces routing solutions of size at most $a + 2b - 3$ steps. The same result can be obtained by the method of Annexstein and Baumslag since an $a \times b$ mesh is the product of two line graphs of size a and b , respectively. However, it is notable that our proof is simpler and shorter.

Another interesting feature of the multistage routing method is that it can be used to derive actual lower bounds. While there are methods (usually based on distance, and bisection-like arguments) which can be used in deriving lower bounds for simple routing patterns such as permutations, they are not very helpful for arbitrary routing patterns. The reader who is familiar with graph theory and the theory of NP-completeness, will realize that the problem of obtaining edge-disjoint paths in a graph reduces to the *multicommodity integral flow* problem. This problem is known to be NP-complete [4]. However, the non-integral version of this multicommodity flow problem is polynomially reducible to linear programming [5] and thus, it can be solved in polynomial time [7]. We can exploit this fact to obtain lower bounds on the required number of routing steps for a given routing problem. Assume that we have constructed a multistage network of L stages. Then, if the non-integral multicommodity flow problem does not have a solution, its integral version will not have one either, and thus, the length of the optimal solution of our routing problem is bounded from below by L . By using a binary-like search, we

¹The definition of the *routing problem* as well as the construction of the multistage graph can be easily modified to accommodate queues of non-uniform size.

can compute the smallest L for which the non-integral multicommodity flow problem accepts a solution. This will be an actual lower bound for the routing problem on hands. To the best of our knowledge, the above method is the only known way to compute actual lower bounds for non-trivial routing problems and it can prove quite useful in evaluating the performance of heuristic routing methods.

3.3 A Class of Routing Algorithms

Up to now, we developed a method for studying the off-line packet routing problem as a graph theoretic routing problem. For each packet $p_i = (orig_i, dest_i)$ we want to find a path in G' from some node in $tower(G', orig_i)$ to some node in $tower(G', dest_i)$ such that all the paths are edge disjoint. As we mentioned in the previous section, the problem of obtaining edge-disjoint paths in a graph reduces to the *multicommodity integral flow* problem. An immediate consequence is that it is unlikely to find an efficient algorithm. However, more research is required in order to take into account the fact that the underlying graph is a multistage network. Also more research is needed in the area where the initial interconnection network is some special graph.

Even though there is not an efficient way to obtain edge-disjoint paths, the multistage off-line method is suitable for the empirical study of the off-line packet routing problem. For most of the routing problems it is the only method available. Even for permutation problems on product graphs, it is worthwhile to spend at least the same amount of time with the method of Annexstein and Baumslag, trying to find solutions which are closer to the actual lower bound (for the permutation on hands).

It is obvious, since G' can be always extended by adding new stages, that a solution to the off-line packet routing problem always exists. However, we are interested in solutions of minimum length. In our study, we will use algorithms which fall within following greedy framework:

Algorithm Find_Edge_Disjoint_Paths

1. Sort all packet according to some ordering criterion.
2. For each packet $p = (orig, dest)$, according to its position in the sorted list, do:
 - Find a path from a node in $tower(G', orig)$ to some node in $tower(G', dest)$
 - Update G' by removing the edges which belong in the found path.

The above framework leaves the designer of the routing algorithm with a lot of freedom. Firstly, an *ordering criterion* has to be selected. The order in which paths are assigned to packets can greatly affect the length of the solution. For some ordering criteria, we will be able to prove upper bounds on the length of the produced solutions. After fixing the order in which the paths will be assigned to the packets, we have to provide a *path selection scheme*. Again, several choices are available. In the case where more than one minimal paths for each *(origin, destination)* pair exist, one path has to be selected. Non-minimal paths might be also considered. In general, the choice of the path selection method will affect the quality of the solution and our ability to claim

bounds about it. The interconnection network under consideration will be of great influence in our selection of the ordering criterion and the path selection scheme. Also, the experience gained from the performance of on-line algorithms can provide significant feedback.

Since the way we choose paths in Algorithm *Find_Edge_Disjoint_Paths* is not fully specified, it is not possible to give a precise time complexity analysis. The analysis will be different for each ordering criterion and each path selection scheme. The space requirements of the method are dictated by two factors: the space required for the multistage graph and the space needed to report the solution. In section 3.1, we presented our method in a static way. A multistage graph was derived from the initial packet routing problem. When the memory space required for the storage of the multistage graph is of great consideration, several approaches can be taken to reduce the space to the minimum. (Equal to the space we need to report the solution.) The multistage graph can be constructed during the execution of the algorithm. If new stages are needed we add them in run time. Also we may choose to add only the part of a stage which is used by some paths. In order to be able to do this we need to maintain a linked list representation of the multistage graph. This will have the effect of slowing down the algorithm by a factor of $O(N)$, N is the number of vertices of the original interconnection network. This is a trade off any algorithm designer has to face.

In Section 4, we will use the multistage off-line routing method to study the permutation packet routing problem on 2-dimensional meshes. For the used ordering criteria and path selection schemes, detailed analysis will be possible.

4 Routing Permutations on Two Dimensional Meshes

In this section, we use the multistage off-line routing method to study permutation routing on 2-dimensional meshes. Given an $a \times b$ mesh, $a \geq b$, in which we have to route a permutation, we form a multistage network of $a + 2b$ stages. We have chosen the number of stages to be equal to $a + 2b$ since the product graph method guarantees to give a solution of that exactly length. We do not allow packets to be queued at any intermediate processor during their routing. However, we assume that each packet can wait for some time at its origin node. This is a reasonable assumption since at least so much space is needed in order to store the packet. The same assumption was made in [14]. The space needed to store the multistage graph is $O((ab)(a + b))$. This is because the maximum degree of any node in a 2-dimensional mesh is 4.

In the rest of this section, when we refer to the multistage routing method, we implicitly assume algorithms which fall in the framework of Algorithm *Find_Edge_Disjoint_Paths*. So, in our study of routing permutations on 2-dimensional meshes, we have to fully specify the ordering criteria and the path selection schemes.

4.1 Packet Ordering Criteria

Several ordering criteria are possible and, as it turns out, they have a significant effect on the quality (i.e., length) of the routing solution. In our study, we experimented with both destination address dependent and independent orderings.

Destination Address Independent Orderings The ordering of the packets does not depend on their destination addresses. The order is inherited from an explicit ordering of the processors in which the packets reside. So, for each processor ordering scheme, we get an implicitly defined ordering for the packets. Commonly used orderings are (see Leighton’s book [13] for detailed definitions):

1. *Row-major ordering.*
2. *Column-major ordering.*
3. *Snake-like* versions of row-major and column-major orderings.
4. *Random ordering.* This ordering is independent of the origin addresses as well. The order is assigned in a random manner. A random permutation of N elements can be easily generated in $O(N)$ time.

Destination Address Dependent Orderings The ordering of the packets depends on their destination addresses. The inherited ordering from the destination addresses can be also defined as in the destination address independent ordering but they are of no special interest. This is because, inverting the roles of origins and destinations (and following the paths backwards) results to the same routing problem. So, we are more interested on orderings which depend on the distance each packet has to travel to reach its destination. If we associate with each packet p the pair (p_H, p_V) where p_H is the distance from the origin-column to the destination-column of p and p_V is the distance from the origin-row to p ’s destination-row, we can define the following ordering criteria:

5. *Longest total distance first.* Packets are sorted in decreasing order with respect to their $p_H + p_V$ values. Ties can be broken in an arbitrary way or by using any of the inherited criteria mentioned above.
6. *Longest lexicographically-horizontal distance first.* Packets are sorted in decreasing order with respect to their p_H values. Ties are resolved as above.
7. *Longest lexicographically-vertical distance first.* Packets are sorted in decreasing order with respect to their p_V values. Ties are resolved as above.
8. *Shortest total distance first.* Packets are sorted in increasing order with respect to their $p_H + p_V$ values. The lexicographic versions of this ordering can be defined as in the *longest-distance* orderings.

Sorting the packet according to any of the above ordering criteria can be done in $O(N)$ time for a problem of N packets. This is achieved by using variants of the *counting sorting method* [2].

Note that, for several problems more complicated orderings, which use information about the relative position between groups of packets and about minimal paths, might be required to achieve optimal solutions. This is the case for routing permutations on trees of N nodes. An $O(N^2)$ time algorithm was developed to produce an ordering which, in turn, was used to achieve an optimal routing solution of length at most $N - 1$ for any permutation [19].

4.2 Path Selection Schemes

After sorting the packets according to one of the ordering criteria mentioned in the previous section, we assign paths to them in the order they appear in the sorted list. When a path is assigned, the edges which belong to the path are deleted from the multistage graph. Then, we proceed with the assignment of a path to the next packet.

The following algorithm is used to assign a path to packet $p_i = (orig_i, dest_i)$ in the multistage graph G'_i . G'_i is the graph obtained from the initial multistage graph G' of our method, after the deletion of the edges which belong to the paths of the first $i - 1$ packets. Algorithm $Path_Selection(P_i, G'_i)$ was obtained after numerous experimentations with several path selection schemes, several of which were quite complicated (based on the reachability between nodes of G'_i) and did not take into consideration the special mesh structure of the interconnection network.

Algorithm Path_Selection(P_i, G'_i)

1. $stage = 0$
2. **While** (a path is not yet assigned to p_i and $stage < a + 2b - distance(orig_i, dest_i)$) **do**
 - If** it is possible to route the path from node $(orig_i, stage)$ horizontally to the correct column and then vertically to the destination, **then** assign that path to p_i
 - else if** it is possible to route the path from node $(orig_i, stage)$ vertically to the correct row and then horizontally to the destination **then** assign that path to p_i
 - else** $stage = stage + 1$
3. **If** a path is not yet assigned to packet p_i **then** signal the failure of the algorithm.

If the algorithm terminates because it failed to route some packet, this means that for the specific routing problem in hands and for the given path selection algorithm, the product graph method performs better. However, as we will see in the next section, after performing millions of experiments, we were not able to identify such a routing problem.

The time spent in the routing of any path, in the worst case, is $O((a + b)^2)$. This is because, we might fail $O(a + b)$ times to assign a path to some packet and, at each try, we have to check $O(a + b)$ edges. Thus, the algorithm terminates after $O((a + b)^2 ab)$ time. For a square mesh of N processors, this translates to $O(N^2)$ time.

If we succeed in routing all paths, we will need $O(ab)$ space (in the word model) to report the solution. This is because each path produced by our routing algorithm can be described just by specifying the stage in which the routing of the packet starts and the initial direction (horizontally or vertically) of the path. Note that, in general, reporting a solution requires $\Omega(abL)$ space, where L is the length of the optimal solution.

4.3 Performance

In this section, we present experimental results of the algorithm based on random permutation routing problems. We ran our algorithm using random input data on square and rectangular meshes. The data were generated with the help of the random number generator function *random()* on a MIPS computer system. *random()* uses a non-linear additive feedback random number generator employing a default table of size 31 long integers to return successive pseudo-random numbers in the range from 0 to $2^{31} - 1$. The period of this random number generator is very large, approximately $16((2^{31}) - 1)$.

When we started our experimental study, we hoped that we will be able to compute for each random permutation a solution of length less than the diameter of the mesh. Surprisingly, when we combined the “longest total distance first” ordering criterion together with Algorithm *Path_Selection()* our method surpassed our expectations. For all the random routing problems, it revealed a solution of length equal to the maximum distance some packet had to travel. This is the best we can expect since the length of that solution matches the distance lower bound for the specific routing pattern. Table 1 contains the number of experiments we run for each square mesh.² The number of experiments decrease as the sidelength of the meshes increases. This is because we spent a fixed amount of time on the study of each individual mesh and the fact that for larger meshes the algorithm requires more time to compute a solution. The numbers of experiments are not in exact proportion with the sidelengths since we performed our experiments in a multiuser system of variable load. For each of the different random problems we produced an optimal solution.

We also performed experiments on rectangular meshes. For each 2^k -node rectangular mesh, $7 \leq k \leq 14$, we studied rectangular $2^m \times 2^{k-m}$ meshes. The number of experiments we performed in each case are given in Tables 2, 3, 4, 5, 6, 7, 8, 9. Again, we were not able to identify a routing pattern for which our method fails to produce an optimal solution.

Obviously, checking all possible permutations is out of question since for an N -node mesh there are $N!$ permutations. For a 4×4 mesh there are $16!$ ($\approx 2.092 * 10^{13}$) permutations. However we were able to check all $9!$ ($= 362880$) possible permutations on a 3×3 mesh. For rectangular meshes with less than 12 nodes, we checked exhaustively all permutations. For all of them, we were able to construct an optimal solution. So we can state the theorem:

Theorem 3 *The multistage off-line method when using the “longest total distance first” ordering criterion together with Algorithm *Path_Selection()* produces optimal solutions for any permutation problem on any rectangular mesh of 12 or less nodes.*

Quite interesting are also the results of our simulations with different ordering criteria (and the path selection scheme described by Algorithm *Path_Selection()*). For the row-major, column-major (and their snake-like variants), random and the “shortest total distance first” (and its lexicographic variants) orderings, we were able to identify permutations for which solutions of length greater than the diameter of the mesh were produced. However, these solutions were never worse than what the product method guarantees. For the “longest lexicographically-horizontal

²All experimental results reported in tables of this paper were obtained using the “longest total distance first” ordering criterion and the path selection scheme of Algorithm *Path_Selection()*.

distance first” and “longest lexicographically-vertical distance first” orderings, we identified permutations for which our method produced solutions of length greater than the actual distance lower bound (for the permutation on hands). However, we were not able to identify any permutation for which our method failed to produce a solution of length smaller or equal to the diameter of the mesh. This is quite interesting since it suggests that the lexicographic orderings might be useful in proving upper bounds on the length of the solutions produced by the multistage method. Experimentation with variants of Algorithm *Path_Selection()* in which all generated paths extend horizontally (vertically) and then vertically (horizontally) revealed similar behavior. However, these variants failed to match the performance of Algorithm *Path_Selection()* in producing solutions of length equal to the actual distance lower bound (for the permutation on hands) when the “longest total distance first” ordering criterion was used. In this case, solutions of length less than or equal to the diameter of the mesh were produced.

4.4 Some Upper Bounds

In this section, we prove some upper bounds on the length of the routing solutions produced when the multistage method is applied for routing permutations on 2-dimensional meshes. For all of our proofs, we will assume a path selection scheme which is a variant of that of Algorithm *Path_Selection()*. More specifically, all produced paths will first extend horizontally and then vertically. For completeness, we give a high level description of this path selection scheme.

Algorithm Horizontal_First_Path_Selection(P_i, G'_i)

1. *stage* = 0
2. **While** (a path is not yet assigned to packet p_i) **do**
 - if** it is possible to route the packet from node (*orig_i*, *stage*) horizontally to the correct column and then vertically to the destination
 - then** assign that path to p_i
 - else** *stage* = *stage* + 1

In a similar fashion, we can define the routing scheme *Vertical_First_Path_Selection()*. In the proofs of our theorems, the following observation is crucial.

Observation 1 *Assume that the path selection scheme *Horizontal_First_Path_Selection()* is used in deriving off-line routing solutions with the multistage method for meshes. Consider packets p and q and further assume that a path has been already assigned to packet q . Then, during the path assignment process, packet q can “delay” p by at most one step.*

The above observation also holds when the path selection schemes *Path_Selection()* and *Vertical_First_Path_Selection()* are used. By saying that “packet q delays packet p ” we mean that at some stage of the path assignment process for packet p , we failed to assign a path to p because the path already assigned to q used an edge of the multistage graph required by p ’s tentative path. The observation follows from the fact that in the produced routing schedules a packet is never delayed after it starts moving.

Theorem 4 *Let π be a permutation to be routed on an $a \times b$ mesh, $a \geq b$. Then, by using any packet ordering criterion and path selection scheme `Horizontal_First_Path_Selection()`, the multistage off-line routing method produces a routing schedule for π of length at most $2(a+b)-4$.*

Proof Consider any packet p . Since p will move first horizontally and then vertically, there are at most $a - 1$ packets which can delay p in its origin-row and at most $b - 1$ packets which can delay p in its destination-column. Taking into account that each of these $a + b - 2$ packets can delay p exactly once, and the fact that p is initially at most $a + b - 2$ steps away from its destination, we conclude that p 's routing schedule is of length at most $2(a + b) - 4$. ■

Theorem 5 *Let π be a permutation to be routed on an $a \times b$ mesh, $a \geq b$. Then, by using the “longest lexicographically-horizontal distance first” packet ordering criterion and path selection scheme `Horizontal_First_Path_Selection()`, the multistage off-line routing method produces a routing schedule for π of length at most $a + 2b - 3$.*

Proof Consider any packet p . Let h and v be the distances which p has to travel in the horizontal and vertical direction, respectively. W.l.o.g., assume that p has to move towards the west. Then, since we are using the “longest lexicographically-horizontal distance first” ordering criterion, the packets which are originally at the h west-most positions in p 's origin-row cannot delay p . So, at most $a - h - 1$ packets initially in p 's origin-row can delay it. During the vertical routing, a delay of at most $b - 1$ steps is possible. Taking into account that p has to travel $h + v$ distance and that $v \leq b - 1$, we conclude that p 's routing schedule is of length at most $(a - h - 1) + (b - 1) + h + (b - 1) = a + 2b - 3$. ■

Note that, the length of the routing schedule guaranteed by Theorem 5 is exactly the same with that promised by the product method when applied to meshes of the same dimensions. However, the proof is much simpler and shorter. Note also, that the upper bounds were obtained by a combination of ordering criteria and a path selection scheme which in practice resulted to non-optimal routing schedules.

5 Conclusions

In this paper we presented the multistage off-line method, a new and rather natural, way to model off-line packet routing problems. Then, the problem of off-line packet routing reduces to the problem of finding edges disjoint paths on a multistage graph. The multistage off-line method can model any kind of routing pattern on any graph. It also accommodates the optimal solutions and can be used to obtain lower bounds which are not based in worst case situations but rather on the actual routing problem under consideration. Based on the multistage method, we studied permutation routing on 2-dimensional meshes. We were able to produce optimal solutions for all (several millions) the random permutations we studied.

Several interesting problems are raised from our method and deserve further investigation. The general problem of finding edge disjoint paths reduces to the integral multicommodity flow which, in turn, is NP-Complete. However, we do not know how the complexity of the problem is affected by the fact that we have to deal with a multistage graph which is constructed in a special way.

Also, the kind of the graph the routing takes place might affect the complexity. Another area which deserves further research is that of the path selection algorithms. We provided a general framework for the algorithm which can accommodate several heuristic methods depending on the underlying interconnection network. However, our study for the mesh was so successful that, for any random permutation problem we studied we produced the optimal algorithm. This makes us investigate the particular algorithm and try to prove that it always produces an optimal solution, or to produce a counter-example. The fact that it turns to be an optimal algorithm for all the small meshes we exhaustively investigated, is very encouraging. Nevertheless, given the fact that in a real parallel system a small portion of the available processors will be allocated to a particular application (a small submesh in a mesh connected computer), having an optimal routing schedule it is something very important. It can easily lead to speed-ups of important magnitudes (especially when the routing pattern presents some locality).

References

- [1] F. Annexstein, M. Baumslag, "A unified approach to Off-Line Permutation Routing on Parallel Networks", Proceedings of 1990 ACM Symposium on Parallel Algorithms and Architectures, SPAA'90, Crete, Greece, July 1990, pp. 398-406..
- [2] T.H. Cormen, C.E. Leiserson, R.L. Rivest, "Introduction to Algorithms", MIT Press/McGraw-Hill, Cambridge, MA, 1990.
- [3] E. Dahl, "Mapping and Compiled Communication on the Connection Machine", Proceedings of the 5th distributed Memory Computing Conference, Charleston, South Carolina, April 1990, pp.756-766.
- [4] M.R Garey, D.S. Johnson, "Computers and Intractability. A Guide to the Theory of NP-Completeness", 1979, W.H Freeman and Company, New York.
- [5] A. Itai, "Two-Commodity Flow", Journal of the Association for Computing Machinery, Vol. 25, No. 4, October 1978, pp. 596-611.
- [6] C. Kaklamanis, D. Krizanc, S. Rao, "Simple Path Selection for Optimal Routing on Processor Arrays", Proceedings of the 4th Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA'92, June 1992, San Diego, pp. 23-30.
- [7] N. Karmarkar, "A New Polynomial-Time Algorithm for Linear Programming", *Combinatorica*, Vol. 4, pp. 373-395, 1984.
- [8] M. Kaufmann, J. Sibeyn, T. Suel, "Derandomizing Algorithms for Routing and Sorting on Meshes", Proceedings of the Fifth Annual ACM-SIAM Symposium on Discrete Algorithms, Arlington, VA, 1994, pp. 669-679.
- [9] D. Krizanc, "A Note on Off-Line Routing on a Mesh-Connected Processor Array", *Parallel Processing Letters*, Vol. 1, No. 1, 1991, pp. 67-70.
- [10] D. Krizanc, S. Rajasekaran, Th. Tsantilas, "Optimal Routing Algorithms for Mesh-Connected Processor Arrays", *VLSI Algorithms and Architectures (AWOC'88)*, J. Reif, editor, *Lecture Notes in Computer Science 319*, 1988, pp. 411-422.
- [11] M. Kunde, "Routing and Sorting on Mesh-Connected Arrays", *VLSI Algorithms and Architectures (AWOC'88)*, J. Reif, editor, *Lecture Notes in Computer Science 319*, 1988, pp. 423-433.

- [12] F.T. Leighton, "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.
- [13] F.T. Leighton, "Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes", Morgan Kaufmann, San Mateo, California, 1992.
- [14] F.T. Leighton, B. Maggs, S. Rao, "Universal Packet Routing Algorithms", Proceedings of the 29th Annual Symposium on the Foundations of Computer Science, 1988, pp. 256-269.
- [15] F.T. Leighton, F. Makedon, I.G. Tollis, "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.
- [16] F. Makedon, A. Symvonis, "An Efficient Heuristic for Permutation Packet Routing on Meshes with Low Buffer Requirements", IEEE Transactions on Parallel and Distributed Systems, Vol. 3, No. 4, March 1993, pp. 270-276.
- [17] S. Rajasekaran, R. Overholt, "Constant Queue Routing on a Mesh", Journal of Parallel and Distributed Computing, Vol. 15, 1992, pp. 160-166.
- [18] A.G. Ranade, "How to Emulate Shared Memory", Proceedings of the 28th IEEE Symposium on Foundation of Computer Science, 1987, pp. 185-194.
- [19] A. Symvonis, "Routing on Trees", TR-471, Basser Dept. of Computer Science, University of Sydney, September 1993. Submitted for publication. An abstract appeared in ACSC-17, Christchurch, New Zealand, January 1994.
- [20] L.G. Valiant, G.J. Brebner, "Universal Schemes for Parallel Communication", Proceedings of the 13th Annual ACM Symposium on the Theory of Computing, May 1981, pp. 263-277.

Mesh	# of Experiments
10×10	2075360
20×20	322190
30×30	103840
40×40	46700
50×50	36380
60×60	35130
70×70	34640
80×80	17420
90×90	15600
100×100	11420
110×110	9000
120×120	3690
130×130	2330
140×140	2290
150×150	2200
160×160	2120
170×170	1680
180×180	1410

Table 1: Number of experiments performed for each mesh. All experiments succeed in producing an optimal solution.

Mesh	# of Experiments
$2^3 \times 2^{11}$	1280
$2^4 \times 2^{10}$	2840
$2^5 \times 2^9$	3540
$2^6 \times 2^8$	3980
$2^7 \times 2^7$	5120

Table 2: Number of experiments performed each 2^{14} -node mesh. All experiments succeed in producing an optimal solution.

Mesh	# of Experiments
$2^2 \times 2^{11}$	900
$2^3 \times 2^{10}$	1180
$2^4 \times 2^9$	2500
$2^5 \times 2^8$	2880
$2^6 \times 2^7$	3360

Table 3: Number of experiments performed each 2^{13} -node mesh. All experiments succeed in producing an optimal solution.

Mesh	# of Experiments
$2^2 \times 2^{10}$	2180
$2^3 \times 2^9$	2700
$2^4 \times 2^8$	3520
$2^5 \times 2^7$	4320
$2^6 \times 2^6$	7760

Table 4: Number of experiments performed each 2^{12} -node mesh. All experiments succeed in producing an optimal solution.

Mesh	# of Experiments
$2^2 \times 2^9$	4060
$2^3 \times 2^8$	4300
$2^4 \times 2^7$	5580
$2^5 \times 2^6$	8220

Table 5: Number of experiments performed each 2^{11} -node mesh. All experiments succeed in producing an optimal solution.

Mesh	# of Experiments
$2^2 \times 2^8$	3040
$2^3 \times 2^7$	3200
$2^4 \times 2^6$	5300
$2^5 \times 2^5$	7000

Table 6: Number of experiments performed each 2^{10} -node mesh. All experiments succeed in producing an optimal solution.

Mesh	# of Experiments
$2^2 \times 2^7$	14320
$2^3 \times 2^6$	11520
$2^4 \times 2^5$	16400

Table 7: Number of experiments performed each 2^9 -node mesh. All experiments succeed in producing an optimal solution.

Mesh	# of Experiments
$2^2 \times 2^6$	44500
$2^3 \times 2^5$	71340
$2^4 \times 2^4$	108320

Table 8: Number of experiments performed each 2^8 -node mesh. All experiments succeed in producing an optimal solution.

Mesh	# of Experiments
$2^2 \times 2^5$	160120
$2^3 \times 2^4$	205860

Table 9: Number of experiments performed each 2^7 -node mesh. All experiments succeed in producing an optimal solution.