A New Approach to Off-Line Packet Routing Case Study: 2-Dimensional Meshes

Antonios Symvonis **Jonathon Tidswell**

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

Abstract

In this paper we present the *multistage off-line routing model*, a new and rather natural, way to model off-line packet routing problems. With this model, the problem of off-line packet routing reduces to the problem of finding edge disjoint paths on a multistage graph. The multistage off-line routing model can model any kind of routing pattern on any graph. Furthermore, it incorporates the size of the maximum queue that can be created in any processor. The multistage off-line routing model accommodates all optimal solutions and can be used to obtain lower bounds that are based not in worst case situations but rather on the actual routing problem under consideration. The paths of the packets are computed by a greedy heuristic method that is based on the distance the packets have to travel. A lot of freedom is left for the user of the model in order to incorporate in the heuristic properties of the underlying interconnection network. We used the multistage off-line model to study the off-line 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 had to travel, and thus, match the actual lower bound for each routing pattern.

1 Introduction

A crucial component of any large scale parallel machine is the algorithm that 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 made during the execution of the algorithm. In other words, 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 that perform matrix operations serve as 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 generate code that will route packets of information through specific paths [?]. This results in faster execution time. All of the overhead that would be required by the on-line routing al-

amount of overhead is significant. This is probably the reason that, 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 [?, ?, ?, ?, ?, ?, ?, ?, ?]. 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.

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 that 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 is by Annexstein and Baumslag [?]. They presented a method to solve the permutation off-line packet routing problem on product graphs. Their algorithm produces optimal schedules that 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 that are of length $3n-3$. Later on, Krizanc and Narayanan [?] presented an algorithm that, for $n \times n$ meshes, creates paths of length $2.25n$ and uses queues of size at most 14. Their algorithm can be considered to be a fine tuning of the algorithm of Annexstein and Baumslag.

We propose a new method for the off-line packet interconnection network. Not just product graphs. It can also treat any type of routing pattern. Not just permutations. Furthermore, our model can incorporate a queue of variable length that reflects the buffering capabilities of the processors in the parallel machine. An additional feature is that it tries to compute paths that 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 Section 2, we give definitions for terms we use in the paper. In Section 3, we present the *multistage off-line routing model*, our model for off-line packet routing. In Section 4, we demonstrate the modeling power of the multistage off-line routing. We explore its features and we discuss its advantages. In Section 5, we present the general form of a routing algorithm that can accommodate the multistage off-line routing model. In Section 6, we apply our model 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 ran on randomly generated data, the number of routing steps that is required by our algorithm is equal to the maximum distance a packet has to travel, and thus, optimal. We conclude in Section 7, with a discussion on the potential of our model and we give some future directions for this research.

2 Definitions

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 $E = \{e_1, e_2, \ldots, e_{|E|}\}$. 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 $e_1, e_2, \ldots$ 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

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

An *off-line* packet routing problem R is defined by a triple (G, P, k) where $G = (V, E)$ is the directed graph that represents 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 that are to be routed. Formally, $P = \{p_1, p_2, \ldots, p_m|\ p_i = (orig_i, dest_i),\ orig_i, dest_i \in V,\ 1 \leq i \leq m\}$. Finally, k is the maximum number of packets that are allowed to be queued at any processor during the routing.

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

There is no restriction on the number of packets that originate from, or, are destined for, a certain processor. Also, several packets may have identical origin and destination processors. 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*, if $h_2 = 1$ we have a *one-to-many routing problem*, 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 $SOLUTION(R) = \{d_1, d_2, \ldots, d_m\}$ 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 it possible to incorporate self loops in the directed paths. A self loop from vertex v in the path of some packet indicates that that packet was queued in processor v at the corresponding routing step. Each directed path contains at most $L + 1$ vertices. For $i = 1 \ldots m$ we have that

$$d_i = v_i^0 v_i^1 \ldots v_i^l,\ 0 \leq l \leq L$$

where, $v_i^0 = orig_i$ and $v_i^l = dest_i$.

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

an unidirected communication link appears in at most one directed path.

2. At any routing step, each self loop appears in at most k directed paths.

3 A New Model for Off-Line Packet Routing

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 model the *multistage off-line routing model.*

Consider any routing problem $R = (G, P, k)$ where, $G = (V, E)$ is the directed graph (with no self-loops) that 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 that can be queued at any processor. Our goal is to achieve routing time near the actual lower bound of the problem. Assume a trivial upper bound of T routing steps for the problem under consideration.

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

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

and

$$E' = \{((v, t), (w, t+1))|\ w \in neighbors(v, G)$$

$$\text{and } 0 \leq t < T\} \cup$$

$$\{e_v^1, e_v^2, \ldots, e_v^k|\ e_v^i = ((v, t), (v, t+1)),$$

$$v \in V,\ 0 \leq t < T,\ 1 \leq i \leq k\}.$$

The edges in the first term of E' represent 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 that resides in any vertex v and can grow up to k packets.

Figure ?? 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 that is based on a distance argument applies.

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

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

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

Theorem 1 *Let $R = (G, P, k)$ be a routing problem 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 $SOLUTION(R) = \{d_1, d_2, \ldots, d_m\}$ 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 $d_i = v_i^0 v_i^1 \ldots v_i^l$, $0 \leq l \leq L$, $1 \leq i \leq m$, 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 that 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 that appear at any routing step of $SOLUTION(R)$ to different edges of G'.

In a similar way, we can obtain from a set of edge disjoint paths a solution $SOLUTION(R) = \{d_1, d_2, \ldots, d_m\}$ for the routing problem $R = (G, P, k)$ of length L, where L is the maximum stage number out of all stages that are entered by some path. Each disjoint path is of the form $(orig_i, t) \ldots (dest_i, t')$, for packet p_i. ∎

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 approach the off-line packet routing problem from a different point of view.

4 The Modeling Power of the Multistage Off-Line Routing

In this section we are going to demonstrate the power of the multistage off-line routing model . The only known method for off-line packet routing is the method of Annexstein and Baumslag [?] that solves the problem for product graphs. It provides us with solutions that are within a constant factor from the optimal for permutation problems. We will refer to this method as the *standard routing method.* Since the standard routing method is the only known method, we will compare the multistage off-line routing with it.

We are aware that the standard method was designed to solve a special form of the problem on a special type of graphs. It is not our intention to underemphasize its importance by the comparison we attempt to do. In our opinion, the standard method was a successful and very important approach to the off-line packet routing problem. We perform our comparison only to demonstrate the power of our model.

The model of the multistage off-line routing has several advantages over the standard method. It models any routing patterns on any interconnection network. It allows for different queue sizes. It can be used to find solutions near to the actual lower bound. We will discuss each of the above in detail.

The standard off-line method for packet routing can treat only routing problems that are permutations and moreover are routed in a network that is a product graph. If $G = H \times F$ is the product graph, $r(H)$ and $r(F)$ are the numbers of routing steps required to route a permutation on H and F respectively, and if $r(H) < r(F)$, then the standard method routes the permutation on G by routing one permutation on H, then a permutation on F and, finally, a permutation on H again. Thus, it produces a solution of length $2r(H) + r(F)$. H or F can be, in turn, product graphs and thus, the method is applied recursively.

From the above, two limitations of the standard

we have no way to model a routing problem defined on a general graph. Moreover, if G is a product graph, we can use the standard method only if we know how to solve the routing problem on both H and F. Thus, H and F must be relatively simple graphs. The second limitation of the standard method concerns the routing patterns. If the routing problem is not a permutation but an (h_1, h_2) routing problem the standard method will fail to solve it. One possible approach that someone might take is to split the routing problem into a sequence of permutation problems and then solve them one after the other by using the standard method. However, the length of the solution may be much more than the optimal. The multistage off-line routing model does not suffer from any of the above two limitations. It can successfully model any routing problem on any interconnection network.

One important factor that any packet routing scheme tries to minimize is the queue size. If a routing problem is to be routed on an interconnection network in which every processor can use an area of size of up to k packets for queueing purposes, it is a design limitation that the queues are never overloaded. Moreover, it is desirable that maximum utilization of the queues occurs, especially, if this utilization results in reduction of the routing time. The multistage off-line routing model has a built-in mechanism to incorporate the available queuesize. On the contrary, the standard method assumes no queues and cannot be modified, at least in a straight forward way, to incorporate them, while, the multistage off-line routing model can model the case where queueing of packets in any processor is prohibited.

When the standard method routes a permutation on a graph $G = H \times F$, it will produce a solution the length of which depends on the worst case optimal solutions for the permutation problem on graphs H and F. For example, for an $n \times n$ mesh, it produces a solution of length $3n-3$ since, the $n \times n$ mesh can be considered as the product of two n-node line graphs and, any permutation on an n-node line graph can be solved in $n-1$ steps. So, in the worst case, the standard method gives a solution that is $n-1$ steps away from the optimal (for the $n \times n$ mesh there exist a trivial worst case lower bound of $2n-2$ routing steps obtained by a distance argument). However, worst case scenarios do not appear in all routing problems. Several problems require the processors to exchange information with other processors in a relatively small (compared with the diameter of the network) distance. In the extreme case that each processor has to transmit a packet destined for itself, the standard method will produce a solution of length that depends not on the actual routing problem but in worst case situations ($3n-3$ for the $n \times n$ mesh) instead of the trivial solution of length 0.

Figure 2: An example of a permutation problem that has a solution of length much less than the worst case lower bound and greater than the bound obtained by any distance argument.

On the other hand, the multistage off-line routing model can accommodate any optimal solution for the routing problem under consideration. It is interesting here to examine how a lower bound is defined for any specific routing problem. When we try to establish worst case lower bounds we usually use arguments based on the diameter and the bisection of the network under consideration. We will try to obtain a lower bound for a specific permutation problem on the interconnection network of Figure ??. This interconnection network is a connected graph that consists of two cliques of size 1000, two cliques of size 10, and one extra node through which the 4 cliques are connected. Obviously, any two nodes of that graph are within distance 4 of each other. This provides us with a trivial worst case lower bound of 4 steps for any routing algorithm. A better worst case lower bound of 1000 steps is obtained if we consider the case where all packets in the one k_{1000} are destined for the other. So, based on the above arguments, we claim that the worst case lower bound for the graph under consideration is 1000 and we say that any algorithm that routes all packets in 1000 steps is optimal. But what if the routing problem is such that all packets in the one k_{10} are destined for the other, and the packets in the k_{1000}'s are destined for

can be solved with exactly 12 routing steps (it is easy to construct such a solution) and a solution of length 1000 is not satisfactory.

From the above discussion, it is obvious that the lower bound for the actual routing problem might be smaller than the worst case lower bound, and that the actual lower bound may be formed by a combination of distance bounds, wire utilization limitations, and of other factors that we are not aware of. The multistage off-line model will accommodate the optimal solution of the actual problem. It is up to the algorithms which will try to find the edge disjoint paths to locate it. On the other hand, the standard method will always fail to find the optimal solution.

5 Greedy Routing Algorithms

Up to know, we developed a model 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. The reader that 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. An immediate consequence is that it is unlikely to find an efficient algorithm for our problem. 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 model is suitable for the empirical study of the off-line packet routing problem. For most of the routing problems it is the only model available. Even for permutation problems on product graphs, it is worthwhile to spend at least the same time with the standard method trying to find better solutions. This becomes evident from the discussion in the previous section regarding actual and worst case lower bounds.

It is obvious, since G' can always be extended by adding new stages, that a solution to the off-line packet routing problem always exists. However, we are interlowing greedy algorithm to obtain such solutions.

Algorithm Find_Edge_Disjoint_Paths

1. Sort all packets in decreasing order according to the distance they have to travel.
2. For each packet $p = (orig, dest)$, according to its position in the sorted list, do
   - find a path from a node in $tower(G', orig)$ that is as close as possible to an earlier stage, to some node in $tower(G', dest)$
   - remove the edges that belong in the path from G'.

The algorithm is greedy in the sense that the packets that have to travel a greater distance to reach their destination are routed first. It is obvious that step 2 of the algorithm leaves a lot of freedom in the way we route the paths. Several methods can be tried. The interconnection network under consideration will be of great influence. Also, the experience gained from the performance of on-line algorithms can provide significant feedback.

Algorithm *Find_Edge_Disjoint_Paths* is actually a heuristic. Since the way we choose the paths is not specified by step 2, it is not possible to give a precise time complexity analysis. The analysis will be different for each *"path selection"* algorithm.

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, we presented our model 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 that is used by some paths. In order to be able to do this we need

stage 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 that any algorithm designer has to face.

In the next section, we will use the multistage off-line routing model to study the permutation packet routing problem on 2-dimensional meshes. The path selection algorithm that will be used in step 2 will be completely defined. As the simulations show, the improvements on the length of the solutions are significant. As a matter of fact, we were not able to identify a routing pattern for which our method does not reveal an optimal solution.

6 Case Study: 2-Dimensional Meshes

In this section, we use the multistage off-line routing model to study permutation routing on 2- dimensional meshes. Given an $a \times b$ mesh, $a \leq b$, in which we have to route a permutation, we form a multistage network of $2a + b$ stages. We have chosen the number of stages to be equal to $2a+b$ since the standard method guarantees to give a solution of that exactly length. In order to make a valid comparison with the results guarantteed by the standard method, 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 [?].

The space needed to store the multistage graph is $O((ab)(a+b))$. This is because the maximum degree of any mode in a 2-dimensional mesh is 4.

6.1 The Path Selection Algorithm

Algorithm *Find_Edge_Disjoint_Paths* presented in Section ??, forms the skeleton of the algorithm we use. After sorting the packets in decreasing order of the distance they have to travel, we assign paths to them in the order they appear in the sorted list. When a path is assigned, the edges that belong to the path are deleted from the multistage graph. Then, we proceed with the assignment of a path to the next packet.

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 model, after the deletion of the edges that belong to the paths of the first $i-1$ packets.

Algorithm Path_Selection(P_i, G'_i)

1. $stage = 0$
2. **While** (a path is not yet assigned to p_i and $stage < 2a + b - distance(orig_i, dest_i)$) **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 at hand and for the given path selection algorithm, the standard method does better. However, as we will see in the next subsection, 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.

If we succeed in routing all paths, we will need $O(ab)$ space 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.

6.2 Performance

In this section, we present experimental results of the algorithm based on random permutation routing prob-

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 would be able to compute for each random permutation a solution of length less than the diameter of the mesh. Surprisingly, our algorithm surpassed our expectations. For all the random input 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 ?? contains the number of experiments we ran for each square mesh. 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.

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 every permutation our algorithm found the optimal solution, and thus, we can state the theorem:

Theorem 2 *The multistage off-line model together with Algorithm Path_Selection() produce optimal solutions for any permutation problem on a* 3×3 *mesh.*

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 mumber of experiments we performed in each case are given in Tables ??, ??, ??, ??, ??, ??, ??, ??. Again, we were not able to identify a routing pattern for which our algorithm fails to produce an optimal solution.

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.

For rectangular meshes with less than 12 nodes, we checked exhaustively all permutations. For all of them, we were able to construct the optimal solution. So we can state the theorem:

Theorem 3 *The multistage off-line model together with Algorithm Path_Selection() produce optimal solutions for any permutation problem on any rectangular mesh of 12 or less nodes.*

7 Conclusions

In this paper we presented the multistage off-line model, 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 nodel can model any kind of routing pattern on any graph. It also accommodates the optimal solutions and can be use to obtain lower bounds that are not based in worst case situations but rather on the actual routing problem under consideration. Several interesting problems are raised from our model and deserve further investigation. The general problem of finding edge disjoint paths reduces to the multicommodity flow

$2^3 \times 2^{11}$	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.

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 that is constructed in a special way. Also, the kind of the graph the routing takes place might affect the complexity. Another area that deserves further research is that of the path selection algorithms. We provided a general framework for the algorithm that 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).

$2^2 \times 2^{10}$	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.

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] E. Dahl, "Mapping and Compiled Communication on the Connection Machine", Proceedings of the 5^{th} distributed Memory Computing Conference, Charleston, South Carolina, April 1990, pp.756-766.

[3] D. Krizanc, L. Narayanan, "Off-Line Routing with small queues on a Mesh-Connected Processor Array", Proceedings of the 3^{rd} Symposium on Parallel and Distributed Processing, Dallas, Texas, December 1991.

[4] 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.

[5] 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.

[6] F.T. Leighton, B. Maggs, S. Rao, " Universal Packet Routing Algorithms", Proccedings of the 29^{th} Annual Symposium on the Foundations of Computer Science, 1988, pp. 256-269.

[7] F.T. Leighton, "Average Case Analysis of Greedy Routing Algorithms on Arrays", Proceedings of the 2^{nd} Annual ACM Symposium on Parallel Algorithms and Architectures, SPAA '90, July 2-6, 1990, Crete, Greece.

[8] 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.

[9] F. Makedon, A. Simvonis, "An Efficient Heuristic for Permutation Packet Routing on Meshes with Transactions on Parallel and Distributed Systems.

[10] S. Rajasekaran, R. Overholt, "Constant Queue Routing on a Mesh", to appear in the Journal of Parallel and Distributed Computing.

[11] A.G. Ranade, "How to Emulate Shared Memory", Proceedings of the 28^{th} IEEE Symposium on Foundation of Computer Science, 1987, pp. 185-194.

[12] L.G. Valiant, G.J. Brebner, "Universal Schemes for Parallel Communication", Proceedings of the 13^{th} Annual ACM Symposium on the Theory of Computing, May 1981, pp. 263-277.