

M. Tech. (Computer Science) Dissertation Series

**Development of Heuristics for Permutation Routing
in a General Network and Algorithms
for some Specific Networks**

**a dissertation submitted in the partial fulfilment of the
requirement for the M. Tech. (Computer Science)
degree of the Indian Statistical Institute**

By

Niloy Kumar Dana

under the supervision of

Dr. B. P. Sinha

Professor & Head

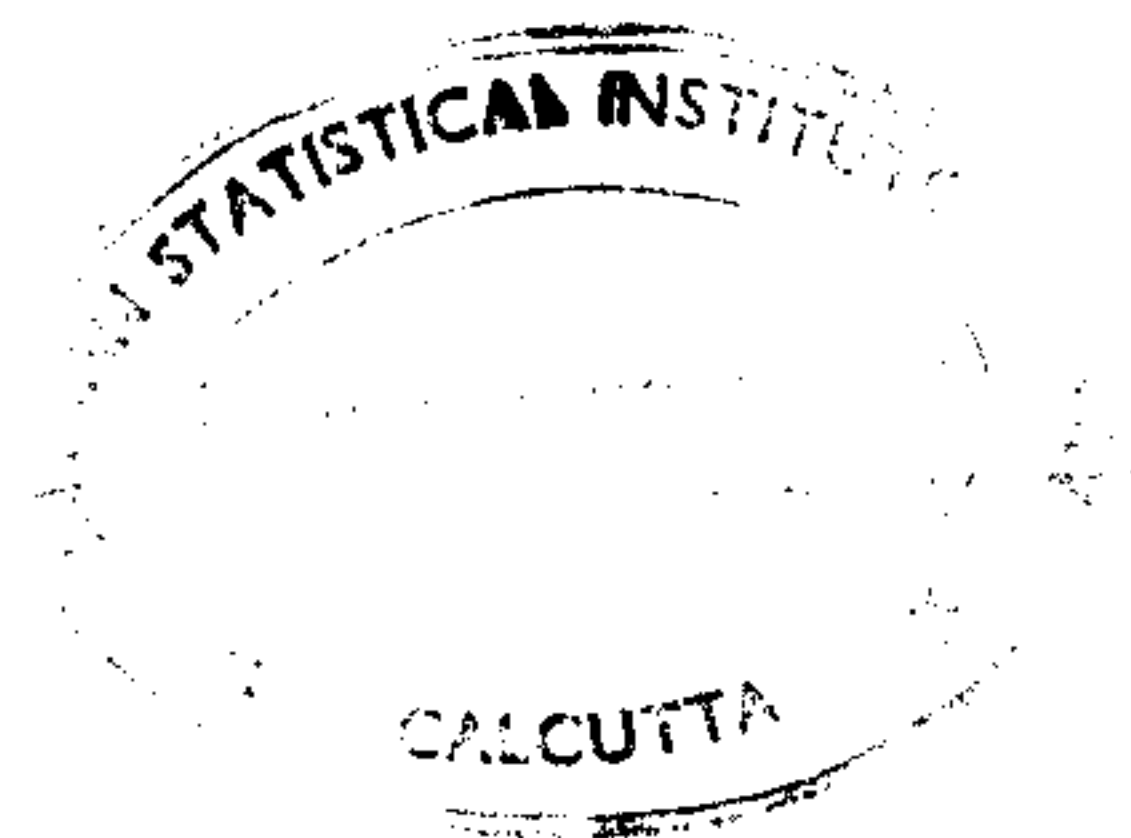
Advance Computing & Microelectronics Unit

INDIAN STATISTICAL INSTITUTE

203, Barrackpore Trunk Road

Calcutta - 700 035

1998



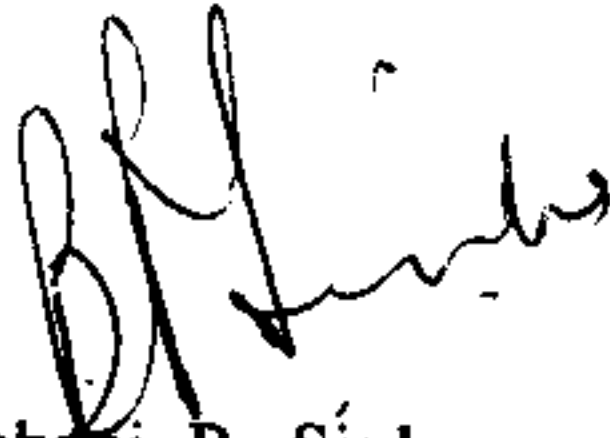
Indian Statistical Institute

203, Barrackpore Trunk Road

Calcutta - 700 035

Certificate Of Approval

This is to certify that the dissertation work entitled "Development of Heuristics for Permutation Routing in a General Network and Algorithms for some Specific Networks" submitted by Niloy Kumar Dana, in partial fulfilment of the requirements for M. Tech in *Computer Science* degree of the *Indian Statistical Institute* is an acceptable work for the award of the degree.



Prof. Bhabani P. Sinha
Professor & Head
Advance Computing & Microelectronics Unit

ACKNOWLEDGEMENTS

I would like to acknowledge my guide, Dr. B. P. Sinha for providing me invaluable support throughout this project. During the course of this project he has always given me helpful suggestions whenever I was in trouble.

I would like to thank all members of ACM unit , for helping me directly or indirectly throughout the work.

Finally I would like to thank all my teachers and batchmates for their support throughout this project.

Niloy Kumar Dana.

Abstract

Presented here is a heuristic for routing permutations on a general network. Discussed also are some deterministic algorithms on some special kind of networks. Linear array, Two-dimensional array, and hypercube are included in the special kind of networks. Both off-line and on-line routing problems are considered here. For a two dimensional ($N \times N$) array every permutation can be routed in $6N + o(N)$ steps with a queue size of only one.

Contents

Sl. No	Topic	Page No
1.	Introduction	3
2.	Packet Routing Algorithm	5
3.	Heuristic for general network	6
4.	Algorithms for special networks	9
5.	Intersection Graph	10
6.	Node coloring formulation	11
7.	Off-line Routing	12
8.	On-line Routing	18
9.	Conclusions	24
10.	References	25

1 Introduction

The exchange of information between Processing Units is one of the most important problems on parallel computers in which the PUs communicate through an interconnection network. The basic communication step is that of transferring packets. These are portions of information generated and received by PUs. Usually a PU is connected to a small number of other

PUs and the packet sent by it need to travel through a sequence of intermediate PUs to their final destination. The handling of all the packets moving through the network is called *packet routing*.

A *packet routing problem* consists of a set of M packets, each with a desired destination address P_h . Initially the packets are stored individually among the N nodes of the network, and the desired destinations are all different. The problem is to route the packet to the desired destinations using local control in as few steps as possible.

The efficiency of a protocol for solving a packet routing problem is measured by two parameters : 1) The time required to complete a set of packet sending requests, and 2) the size of an auxiliary memory in each PU to store such incoming packets which have to stay for a while before moving on.

In a permutation routing every PU is source and destination of precisely one packet. So here every PU will have some packet to a destination PU. This sources and destinations will have a one-to-one correspondence.

In case of partial permutation routing every PU may not send a packet but here also there will be a one-to-one correspondence.

Circuit switch fixed routing is an increasingly popular communication model wherein there is between every source-destination pair a single path that is system-determined by a fixed routing rule.

Packet Routing can be broadly classified into off-line and on-line routing.

If the path followed by each packet in each of the permutations can be pre-computed and stored at the originating processor then this type of routing is called *off-line routing*.

The algorithms for which the local routing decisions are made without pre-computation and without knowledge of global routing problem are called *on-line routing* algorithm. Obviously this will be harder.

2 Packet Routing Algorithms

When routing a permutation f [where every node h needs to send a single message to node to $f(h)$], path conflicts often occur and cause communication overhead, namely link conflict, node conflict, and path length. It can be shown that the impact of node conflict and path length is negligible in circuit-switched fixed routing systems. While the impact of node contention is more dominant. So we assume throughout that the communication overhead is due to link contention only.

To minimise the communication overhead when routing a permutation , the permutation has to be scheduled. *Scheduling* a permutation consists of partitioning the set of nodes into m subsets E_1, E_2, \dots, E_m , for some m , such that for every $h = 1, 2, \dots, m$, the paths originating from the nodes in E_h do not conflict over links, i.e. they can be established simultaneously and their corresponding messages can be delivered in parallel. E_h represents the set of source nodes that can send data to their destinations at time h , $h = 1, 2, \dots, m$. *Optimal scheduling* is the process of finding a partition of minimum size m , and the partition is called an optimal schedule.

3 Heuristic for a general network :

Input :

- 1) A weighted graph where the weights denote the cost .
- 2) A permutation of all the vertices which are actually the destination nodes.

Data-structures used:

```

struct set {
    int first;
    int second;
    struct busy_time *tptr;
    struct set *next;
};

struct busy_time {
    int enter_time;
    int exit_time;
    struct busy_time *next;
}

```


2. Sort these shortest times in non-increasing order. Let S_1, S_2, \dots, S_n be the sorted order of the set T_1, T_2, \dots, T_n and the corresponding set U of ordered sets are B_1, B_2, \dots, B_n .

Initialise $A \leftarrow B_1$.

3. Till all the sets are finished.

Perform $V = U - A$

Take set $B \leftarrow$ left-most of V .

If (any element is common between two sets A & B)

If (if the busy_times of these two sets in the common link is intersecting)

find the entering node and exiting node of the link in which they are intersecting. In the set B try to get another path through these two nodes without taking the link in which they are intersecting..

If (there exists such a path)

change the route-links of set B and add these links to the links of set A to get a combined set.

Else

$B \leftarrow$ next set of V .

Break.

Else

add the links corresponding to set B to those of set A to get a combined set.

Else

add the links corresponding to set B to those of set A to get a combined set.

Set $A \leftarrow$ combined set $\{ A, B \}$.

Set $B \leftarrow$ right set of B in the sorted list.

4 Routing algorithm for Special networks.

In linear arrays there is only one path between any pair of nodes. Therefore, there can only be one fixed routing rule. In the case of rings, we will consider the *clockwise rule* that selects the clockwise path. This forces the ring to be a directed graph. The *counter-clockwise rule* can be treated similarly. In meshes the row-column rule will be considered. In this rule, the source-destination path goes row-wise to the correct column and the column-wise to the destination.

In the k -cube, call the bits that differ in two nodes x and y of binary levels $x_{k-1} \dots x_1 x_0$ and $y_{k-1} \dots y_1 y_0$ the *changeable bits*. It is evident that a shortest path from x to y can be generated from x one node at a time by complementing every changeable bit exactly once in some order. We will define two fixed routing rules: the *e-cube rule* selects the path corresponding to complementing the changeable bits from right to left. The *e⁻¹-cube rule* selects the path corresponding to complementing the changeable bits from left to right.

5 Intersection graphs

Let A_1, A_2, \dots, A_n be n sets. The intersection graph modelled by these sets is the graph $G = (V, E)$ where $V = \{ 1, 2, \dots, n \}$ and $E = \{ (h, j) \mid A_h \cap A_j \neq \emptyset \}$. If the sets are intervals on real axis, the intersection graph is an *interval graph*. If the sets are arcs of a circle, the graph is called *circular arc graph*.

6 Node Coloring Formulation

Let $G = (V, E)$ be a network with a fixed-routing rule s , and f a permutation of V to be scheduled on G . Our node coloring formulation aims at constructing a graph $\Gamma_f(G)$ such that every schedule of f in G corresponds to a node coloring of $\Gamma_f(G)$ and vice versa. In particular the chromatic number of $\Gamma_f(G)$ is equal to the minimum number of passes needed to route f in G , i.e. the size of the optimal schedule of f .

Let $\Gamma_f(G)$ be the intersection graph modelled by the sets $A_i = \{ e \mid e \text{ is an edge in the path } h \rightarrow f(h) \}$, for all $h \in V$, i.e. $\Gamma_f(G) = (V, E)$ where $V \rightarrow V$ and $(h, j) \in E$ if and only if s -determined paths $h \rightarrow f(h)$ and $j \rightarrow f(j)$ overlap (i.e. , conflict over at least one link in G).

Theorem 6.1. (a) A partition E_1, E_2, \dots, E_m of V is a schedule of f in G if the m -coloring of

$\Gamma_f(G)$ derived by coloring the nodes E_j with color j for $j = 1, 2, \dots, m$ is a correct node coloring.

(b) The chromatic number of $\Gamma_f(G)$ is equal to the size of the optimal schedule of f .

Proof . (a) Assume that E_1, E_2, \dots, E_m is a schedule of f in G . Then by definition, for every $j = 1, 2, \dots, m$, the paths $(h \rightarrow f(h))_{h \in E_j}$ are mutually non-overlapping (i.e. non-conflicting), and, therefore the corresponding nodes $h \in E_j$ are mutually nonadjacent in $\Gamma_f(G)$. Thus all the nodes $h \in E_j$ can be colored with the same color, say color j . Conversely if for every j the nodes in E_j can be colored with the same color in $\Gamma_f(G)$. Then their corresponding paths are non-conflicting in G , and therefore, $(E_j)_{1 \leq j \leq m}$ is a schedule of f in G .

(b) This will immediately follow from part (a).

7 Off-line Routing

7.1 Permutation scheduling on linear array

There can be two possibilities : The first is when the edges are half-duplex , i.e. data can flow over a link in either direction but not simultaneously. The second case is when the links are full-duplex, i.e. data can over a link in both directions simultaneously. Let us consider the half-duplex case.

It can be observed that if G is a linear array L_N the graph $\Gamma_f(L_N)$ is an interval graph. This is so because two paths $h \rightarrow f(h)$ and $j \rightarrow f(j)$ conflict over links in L_N if and only if the real intervals $[h, f(h)]$ and $[j, f(j)]$ are overlapping. Note that an interval $[a, b]$ is the set $\{ x \mid x \text{ is a real number and } a < x < b \}$. Note also that if $h > f(h)$, the interval $[h, f(h)]$ is simply taken to denote $[f(h), h]$.

Algorithm :

begin

1. Sort the end points of the intervals.
2. Let Q be a list of available colors ,and m , an integer variable representing the maximum number of colors needed so far. Initialise Q to \emptyset and m to 0.
3. Scan the sorted endpoints from left to right. When an endpoint x is reached do

if (x is a left end of an interval I) **then**

if (Q is not empty) **then**

delete a color c from Q and color I with c ;

else

increment m and color I with color m ;

else /* x is the right end of I */
 insert the color of I into the list Q ;

end

7.2 Permutation scheduling on hypercubes

In this section , the permutations that are routable in one pass (i.e. $m = 1$) under one of the rules e^{-1} -cube and e-cube , will be characterised assuming that the links are full-duplex. Using this characterisation , it will be shown that the very useful Ω -realisable (resp. Ω^{-1} -realisable permutations are routable in one pass on the hypercube under the e^{-1} -cube (resp. , e-cube) rule. Afterward, the Benes routing algorithm will be used to schedule arbitrary permutations in 2 passes on the hypercube.

Throughout this section , the hypercube under consideration is a k -cube of $N = 2^k$ nodes and the permutations are permutations of $\{ 0, 1, \dots, N - 1 \}$. we will also follow the notation that every node x has the binary label $x_{k-1} \dots x_1 x_0$.

Lemma 7.1. Let $s = s_{k-1} \dots s_1 s_0$ and $d = d_{k-1} \dots d_1 d_0$ be two nodes in the k -cube under the e^{-1} -cube rule. The path $s \rightarrow d$ goes through an edge (x, y) in the r th dimension if and only if $x = d_{k-1} \dots d_{r+1} s_r \dots s_0, y = d_{k-1} \dots d_r s_{r-1} \dots s_0$ and $d_r = \text{com}(s_r)$ where $\text{com}(s_r)$ is the complement of s_r .

Proof : Refer [2].

Lemma 7.2. Two paths $s \rightarrow d$ and $s_1 \rightarrow d_1$ conflict over a link in the k -cube under the e^{-1} rule if and only if there exists an integer r , $0 \leq r \leq k-1$, such that $d_{k-1} \dots d_r s_{r-1} \dots s_0 = d_1_{k-1} \dots d_1_r s_1_{r-1} \dots s_1_0$ and $d_r = d_1_r = \text{com}(s_r) = \text{com}(s_1_r)$. Where com denotes complement.

Proof. Refer [2].

Lemma 7.3. Two paths $s \rightarrow d$ and $s' \rightarrow d'$ conflict over a link in Ω of 2^k inputs if and only if there exists an integer r , $0 \leq r \leq k-1$, $d_{k-1} \dots d_r s_{r-1} \dots s_0 = d'_{k-1} \dots d'_r s'_{r-1} \dots s'_0$.

Proof. Refer [2].

So based on the above lemmas we have the following theorem.

Theorem 7.1. a) Every permutation realised by the Ω network without conflict is routable in one pass on the k -cube under the e^{-1} rule.

b) Every permutation realised by the Ω^{-1} network without conflict is routable in one pass on the k -cube under the e rule.

Since many interesting and frequently used permutations are realised by Ω and Ω^{-1} , it follows that many interesting permutations are routable in one pass on the fixed routing hyper cubes which uses the e -cube rule.

Now we know that every Benes network is identical to the network derived from concatenating an Ω^{-1} network with an Ω network.

So to yield a 2-pass scheduler :

1. Use Lee's algorithm which for any given permutation f , finds the switch setting of $\Omega^{-1} \Omega$ to realise f .
2. Let g be the permutation realised by the settings of the Ω^{-1} part, and let h be the permutation realised by the switch settings of the Ω part. Clearly $f = g \circ h$.
3. Route g in one pass on the hypercube under the e-cube rule and then route h in one pass on the hypercube under the e^{-1} cube rule.

7.3 Permutation scheduling on a two-dimensional array

Off-line algorithm can solve any permutation routing problem in $3N - 1$ steps on a $N \times N$ array using queue of size 1.

Steps :

- 1) The packets are permuted within each column so that at most one packet in each row is destined for each column.
- 2) Each packet is routed within its row to the correct column.
- 3) Each packet is routed within its column to its correct row (i.e. destination).

So in the off-line part we need to figure out how to permute the packets within each column during step 1 so that there will be at most one packet in each row destined for any column.

Steps of permuting the packets within each column

Let us consider the more general problem of routing packets on an $r \times s$ array.

Steps:

- 1) Construct a bipartite *routing graph* $G = (U, V, E)$, for an arbitrary routing problem f , containing $2s$ nodes $U = \{u_1, u_2, \dots, u_s\}$ and $V = \{v_1, v_2, \dots, v_s\}$ and rs edges $E = \{e_1, e_2, \dots, e_{rs}\}$. The *routing graph* contains one edge for

each packet , and the k th edge connects u_{ik} to v_{jk} , where ik is the starting column of the k th packet and jk is the destination column of the k th packet ($1 \leq k \leq rs$).

- 2) Label each edge of G with an integer in the interval $[1, r]$ so that any two edges incident to the same node will have different labels. This labelling is possible since G is a r -regular bipartite graph. (r -regular means every node of G is incident to r edges).
- 3) Each packet p is sent to row l_p , where l_p denotes the label for the edge corresponding to packet p .

Using bipartite matching it can be shown that every permutation can be routed on $p \times q$ meshes in q passes under the row-column routing rule.

Theorem 7.3.

Let $G = (U, V, E)$ be a bipartite graph such that for every subset A of U , we have $|\Gamma(A)| \geq |A|$, where $\Gamma(A)$ is a subset of nodes in V that are adjacent to nodes in A . Then G has a perfect matching , that is, a matching of size $= \min(|U|, |V|)$.

Assume that the node in row x_1 and column x_2 is labelled x_1x_2 , for all $x_1 = 0, 1, \dots, p-1$ and $x_2 = 0, 1, \dots, q-1$. We are to select, in every pass , p paths $x_1x_2 \rightarrow f(x_1x_2)$ such that their sources belong to distinct rows and their destinations belong to distinct columns so that paths do not conflict.

This will be accomplished using perfect matching in a bipartite graph $G = (V_1, V_2, E)$ to be defined next. V_1 is the set of columns and V_2 is the set of columns. For every pair of nodes (x_1, y_2) in $V_1 \times V_2$, (x_1, y_2) is an edge of label y_1x_2 if $f(x_1x_2) = y_1y_2$. Thus G is a bipartite multigraph.

Lemma 7.4. Let $G = (V_1, V_2, E)$ be as just defined. For every subset A of V_1 , we have $|\Gamma(A)| \geq |A|$.

Proof. Refer [2].

Using the previous lemma and previous theorem it can be concluded that

$G = (V_1, V_2, E)$ has a perfect matching M of size equal to $\min(|V_1|, |V_2|) = \min(p, q) = p$ (say).

Every edge of (x_1, y_2) of some label y_1x_2 in M corresponds to the path $x_1x_2 \rightarrow f(x_1x_2)$ that is $x_1x_2 \rightarrow y_1y_2$ consisting of row path $x_1x_2 \rightarrow x_1y_2$ in row x_1 , followed by the column path $x_1y_2 \rightarrow y_1y_2$ in column y_2 .

The p paths corresponding to the p edges of matching M are mutually non - conflicting because every two such paths have their sources in two distinct rows and their destinations in two distinct columns due to the fact that M is a matching. Thus all these paths can be established simultaneously in single pass.

By deleting M from G , we obtain a new bipartite graph that can be similarly shown to have a perfect matching. By repeating this process q times we obtain q perfect matchings yielding a q pass schedule for f under the row-column routing rule. We thus have a q pass scheduling algorithm for meshes.

8 Online Routing:

Permutation Routing on a linear array

Here we assume that the links are full-duplex.

For linear array we use this scheme. At each step, each packet that still needs to move right or leftward does so. The algorithm terminates when all packets have reached there destination.

The first thing to notice about greedy algorithm for linear arrays is that it is well defined (at least in case of permutation routing). In particular, two packets will never be contending for use of the same edge (in the same direction) at the same time. Hence, whenever a packet is supposed to move according to the algorithm, it is able to do so. In this case no two packets which are travelling in the same direction will ever reside in the same processor at the same time. More over in this greedy algorithm on a linear array each packet reaches its destination in d steps, where d is the distance that the packet needs to travel. Hence the algorithm always terminates in at most $N - 1$ steps.

Permutation routing on a two-dimensional array

When we apply greedy algorithm with this type of network there arises a problem. Here two or more packets might be contending for the same edge (in the same direction) at the same time.

One of the strategies for arbitrating between packets that are contending for the same edge is that the packet that need to go farthest goes first. For preventing the build-up of large queues we could preset a maximum threshold q , and simply not advance a packet forward into a processor with a queue that is at or near the threshold.

Here we allow the queues to grow arbitrarily and edge contention is resolved by giving priority to the packet that needs to go farthest in that direction (farthest-first).

Steps :

- 1) each packet is routed to its correct column,
- 2) Then it is routed to its destination along the column.

Here every packet reaches its correct column in first $N - 1$ steps. This is because there are never any contention for row edges. Each row acts like a linear array and packets needing to move rightward or leftward do so in a stepwise fashion. Though it is possible for packets to pile up at a node still it does not affect.

So after $N - 1$ steps every packet is in correct column. However several packets within a column might be piled up in large queues.

Now we take the help of the following lemma to prove the rest.

Lemma 8.1. Consider an N -node linear array in which each node contains an arbitrary number of packets, but for which there is at most one packet destined for each node. If edge contention is resolved by giving priority to the packet that needs to go farthest, then the greedy algorithm routes all the packets in $N - 1$ steps.

Proof : Refer [1] .

So any permutation can be routed on a two dimensional ($N \times N$) mesh in $2N - 2$ steps but only if we allow queues of packets to build up at some processors. In the worst case, some queues can grow to contain as many as $\uparrow(N)$ packets.

As we can see routing in a two-dimensional mesh can be done in diameter time but only at the cost of very high queue size.

It will be shown here that queue size can be reduced if we sacrifice a little bit of the speed.

Here is given a $6N + \alpha(N)$ step algorithm using queues of size 1.

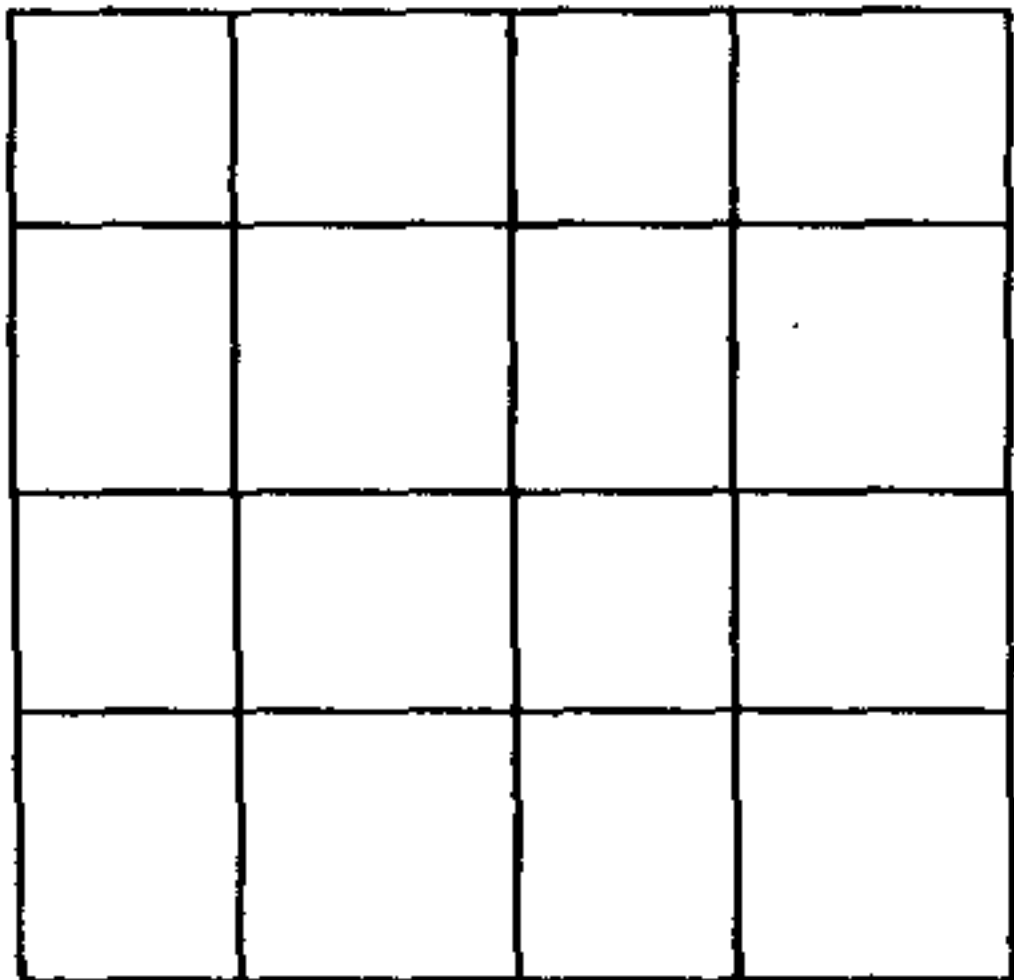
8.1 Steps .

1. The packets are sorted into column-major order according to the column destination of each packet. (Ties can be broken by row destination).

Here we are using the procedure to sort a $\sqrt{N} \times \sqrt{N}$ mesh into mesh in column-snakelike order.

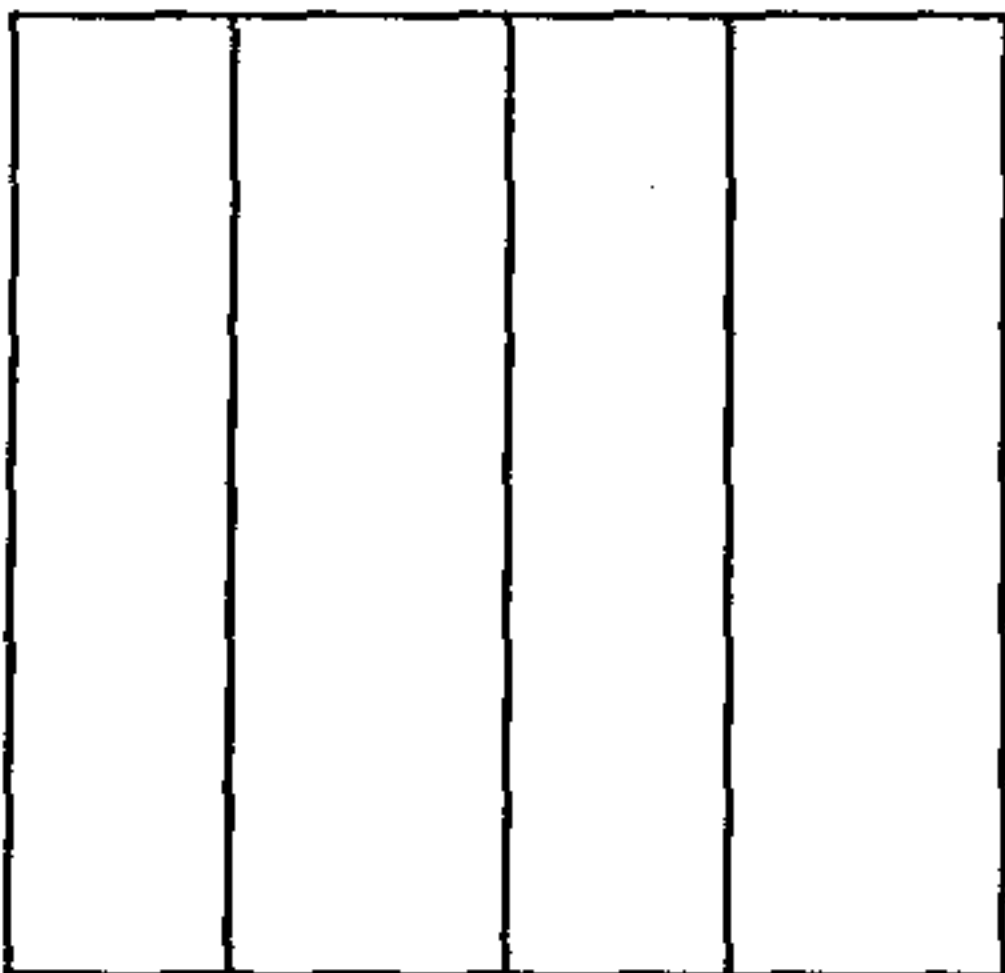
Steps of the procedure column-snakelike order.

1. Divide the mesh into $N^{1/4}$ **blocks** of size $N^{3/8} \times N^{3/8}$ and simultaneously sort each block in snakelike order.
2. Perform an $N^{1/8}$ -way unshuffle of the columns, In particular permute the columns so that the $N^{3/8}$ columns in each block are distributed evenly among the $N^{1/8}$ vertical slices.(A vertical slice is simply a column of blocks. Similarly a horizontal slice is a row of blocks).
3. Sort each block into snakelike order.
4. Sort each column in linear order.
5. Collectively sort blocks 1 and 2, blocks 3 and 4, etc. , of each vertical slice into snake-like order.
6. Collectively sort blocks 2 and 3, blocks 4 and 5, etc. , of each vertical slice into snake-like order.
7. Sort each row in linear order according to the direction of the overall N-cell snake.
8. Perform $2N^{3/8}$ steps of odd-even transposition sort on the overall N-cell snake.

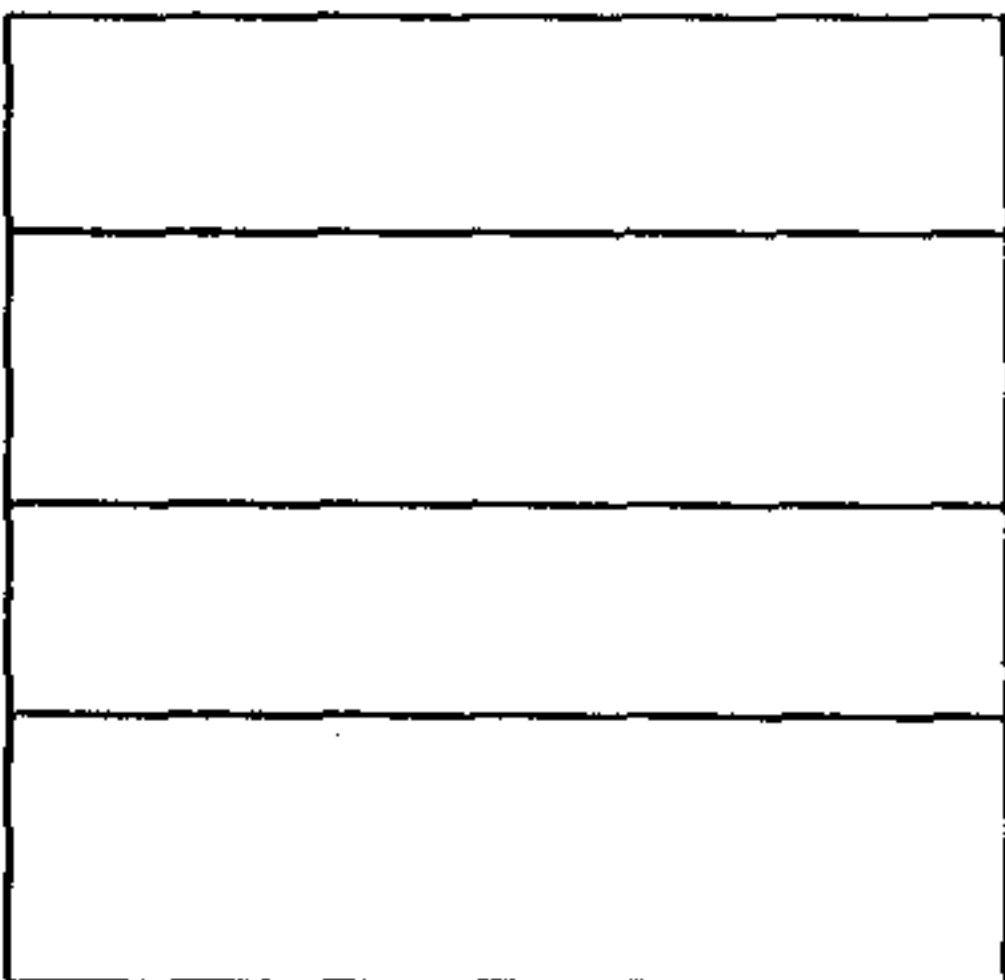


$N^{3/8}$

Fig. 2



Vertical slices



Horizontal slices

Lemma 8.2. A $\sqrt{N} \times \sqrt{N}$ mesh can be sorted in $\sqrt{N} (\log N + 1)$ steps using *shear sort*.

Proof: Refer [1].

So the steps 1, 3, 5, and 6 of the above procedure can be accomplished using *shear sort* in at most $O(N^{3/8} \log N)$ steps.

Steps 4 and 7 can be accomplished using odd-even transposition sort in $2\sqrt{N}$ steps.

Step 2 can be accomplished in a variety of ways using no more than $\sqrt{N} + O(N^{3/8})$ steps.

Step 8 also uses odd-even transposition sort, but only for $2N^{3/8}$ steps.

Hence the total running time is $3\sqrt{N} + o(\sqrt{N})$ steps.

And it takes $\sqrt{N} - 1$ more steps to produce column-major order from snake-like column major order.

If there are precisely N packets, then routing is completed at the end of this step only.

2. Each packet is routed to its correct column.
3. Each packet is routed to its correct destination.

In this algorithm there is never any contention for edges. This is because the sorting phase rearrange the packet so that at most one packet in each row is destined for each column. Hence when row routing is completed at the end of step 2, there will be at most one packet in each node at the start of step 3. So there won't be any contention during the column routing in step 3.

The algorithm can be generalised so that it runs in a $(2 + 4/q)N + o(N/q)$ steps with queues of size $2q - 1$.

Steps :

1. The grid is partitioned into q^2 blocks of size $\sqrt{N}/q \times \sqrt{N}/q$, and we sort the packet into column major order as done in the previous algorithm within each block according to column destination of each packet.
2. Each packet is routed to its destination using the basic greedy algorithm.

Analysis:

Step 1 takes $4\sqrt{N}/q + o(\sqrt{N}/q)$ steps.

Step 2 takes $2\sqrt{N} - 2$ steps. Hence this routing takes a total of $(2 + 4/q)\sqrt{N} + o(\sqrt{N}/q)$ steps.

9 Conclusions

Here we have considered the off-line and online permutation routing for mesh, hypercube. It can be extended to multimesh. Multimesh can be considered as a 4-dimensional mesh and the idea of two-dimensional mesh can be applied here to develop an algorithm for it.

10 References

- [1] F. T. Leighton. Introduction to Parallel algorithms and architectures: arrays . trees . hypercubes.
- [2] Abdou Youssef. Off-line permutation routing on circuit-switched fixed-routing networks. *Networks* , an international journal. Vol-23, pp 441-448.