C 23797
24.09.97.

38

PERMUTATION ROUTING ON MESH AND MULTIMESH

Manoj Kumar Baruah

INDIAN STATISTICAL INSTITUTE LIBRARY
C 23797
24 SEP 1997
CALCUTTA

Prof.Bhabani Prasad Sinha

INDIAN STATISTICAL INSTITUTE LIBRARY
24 SEP 1997
CALCUTTA

July 1997

M. Tech. (Computer Science) Dissertation Series

PERMUTATION ROUTING ON MESH AND MULTI-MESH

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

By

Monoj Kumar Baruah

under the supervision of

Prof. Bhabani Prasad Sinha
Advanced Computing and Microelectonics Unit

INDIAN STATISTICAL INSTITUTE
203,Barrackpore Trunk Road
Calcutta-700035

July , 1997

STATISTICAL INSTITUTE LIBRARY
C 23797
24 SEP 1997
CALCUTTA

24 SEP 1997

Indian Statistical Institute

203 , B. T. Road
Calcutta - 700035

Certificate of Approval

This is to certify that the thesis titled , PERMUTATION ROUTING ON MESH AND MULTI-MESH submitted by Monoj Kumar Baruah , towards partial fulfillment of the requirement of the degree of M.Tech . in Computer Science at the Indian Statistical Institute , Calcutta , embodies the work done under my supervision .

Prof. Bhabani P. Sinha
Advanced Computing and Microelectonics Unit
Indian Statistical Institute
Calcutta - 700035.

Acknowledgments

I express my deep gratitude to Prof. Bhabani Prasad Sinha for his guidance, advice and constant encouragement throughout the period this dissertation was in progress .
My sincere thanks goes to everyone in Advanced Computing and Microelectronics Unit for their advice and help whenever I wanted them most .
I thank Rekha Menon , Debashis Sarkar , Manas R. Jagadev , Arijit Laha and Piyush R. Kumar who have helped me a lot in preparing this report .
Lastly , I take the opportunity to thank all my classmates who have made my two years stay here a nice experience .

Monoj Kumar Baruah

Calcutta
July , 1995

Abstract

Fast communication between the processors in a parallel machine is very critical to efficient performance of parallel algorithms. The time-space trade-off involved in solution of the permutation routing problem on mesh has been studied . Few algorithms for permutation routing on Multi-Mesh has been proposed. The first among these requires large buffers to be present in each processor . Another algorithm which requires no buffers has been proposed. Yet another algorithm which uses buffers of constant size has been proposed . An off-line algorithm which precomputes the paths to be taken by each packet off-line so as to route them without requirement of buffers in the processors has been proposed .

Contents

Chapter 1

Introduction

In quest of finding better ways for performing tasks , man has invented newer and newer devices — wheel , gear , vacuum tube , transistor and modern computer in that order , to name a few . Need for fast and correct computation has given birth of one of the greatest inventions of this century — computer , which has been the subject of lot of research work in the last few decades . This has resulted in rapid evolution of computer in terms of speed . At this point of time , the speed of single-processor computer is on the edge of limit imposed by physics of building materials . Designing computers with more than one processor are the recent area of research . The development of VLSI technology has expedited the research work in this field . This fast speed of computation also proves insufficient in coping up with even faster spreading application fields like image processing , computer vision , intelligence processing , pattern recognition etc .

Designing of multi-processor computer is a highly challenging task because even though the overall speed is highly increased , because of communication overheads between the individual processors it may not be possible to achieve the expected performance . Based on how the processors in a multi-processor computer communicates among themselves , they can be categorized into two classes : *tightly coupled* , where the processors share a common lock and/or memory and *loosely coupled* , where each processor runs independently with a local clock and local memory . The interprocess communication in such computer systems plays a vital role in the overall performance .

Two ways of effecting interprocessor communication is through the use of shared memory and an interconnection network . Thus , the interconnection network has become a critical component in such system .

Interconnection networks can be classified into two categories : static and dynamic . In a static network , links between two processors are fixed once for all and cannot be reconfigured for direct connection to other processors . Ring, star, mesh , hypercube etc. are examples of static networks . In dynamic networks the links can be reconfigured by setting the active switch elements of the network . Examples of such dynamic topologies are baseline, reverse baseline, banyan , omega, Benes , shuffle-exchange , etc.

A static network topology should have the following desirable properties :

(i) Low number of links : This is to keep the cost of the network small .

(ii) Low degree : The maximum degree of a node should be small . This requirement helps to keep the number of I/O ports of a processor to a small value .

(iii) Regularity : The network graph should preferably be regular , i.e. every node should have same number of connections or uniform degree .

(iv) High bisection width : The bisection width is the minimum number of edges along a cut

which divides the network into two equal halves . Thus the bisection width provides a good indication of the maximum communication bandwidth along the bisection of the network .
(v) Low diameter : To minimize the message-passing over-head , the diameter of the network graph should be as small as possible .
(vi) High degree of fault-tolerance : The network should remain operational even in the presence of node / link failures . A common measure of fault-tolerance is the node (edge) connectivity of the graph , which is the minimum number of node (edge) faults that makes the network disconnected . Another useful measure is the f-fault-diameter which is the maximum diameter of the network in the presence of f node / link faults .
(vii) Ease of mapping of algorithms : The versatility of the network for efficient implementation of different kinds of parallel algorithm is another desirable feature .
(viii) Incremental Extensibility : Ideally , the network should be extensible by one node at a time , or at least by a small group of nodes .

Many of the above requirements are mutually conflicting with each other . For example , although low degree and low number of links are desirable they effect the diameter and fault-tolerance properties of the network graph . Design of a network topology optimizing all the above features is almost intractable . One common approach is to optimize some of the desirable features of the network , keeping other parameters within the acceptable limits .

Among various topologies available in literature , mesh , torus , ring , hypercube are popular . Recently , a new architecture , Multi-Mesh(MM) is proposed in [4,5] which is a simple variant of mesh . A Multi-Mesh has n^2 blocks each containing n^2 processors connected as simple $n \times n$ mesh . The interconnections among the blocks are made in such a way that each node has a uniform degree of 4 . It has a diameter of 2n . Various algorithms like sorting , matrix multiplication run faster on this architecture than on a 2D mesh .

Fast communication between processors in a netwiork is very critical to efficient implementation of many algorithms . In many applications , it is required that the prcessors communicate the results of computations among themselves . Various routing algorithms — point-to-point communication ,broadcast [4,5], and complete exchange [16] have been already implemented for MM network .

In this dissertation , algorithms for another important communication problem — permutation routing — have been developed on the Multi-Mesh network . First, a greedy algorithm has been presented which requires buffer of large size to be present in each processor. Next , an algorithm which run in O(n) time in a network of size n^4 without any extra buffer at any processor for queueing . This algorithm has close resemblance with that of sorting . Another algorithm which requires buffers of constant size to be present in each processor has been presented .The algorithms similar in nature with this algorithm runs in better time than the corresponding total sorting based algorithms in two-dimensional and multi-dimensional mesh . An off-line algorithm in which the routes to be taken by the packets are computed before the actual routing is done is also presented . This algorithm routes the packets without requirement of buffers in the processors .

The organization of this dissertation is as follows :

In Chapter 2 , the Multi-Mesh network is described along with some important topological properties . In Chapter 3 , an overview of the various type of existing algorithms for permutation routing has been given. In Chapter 4 , algorithms for permutation routing on Multi-Mesh network are presented .

Chapter 2

Multi-Mesh Topology

2.1 Introduction

Mesh is one of the simplest and most popular network topology. In a two-dimensional mesh with n^2 processors can be identified by two co-ordinates x and y with respect to some chosen origin. Let the processor at the position (x,y) be denoted by P(x,y) where $0 \leq x,y \leq n-1$. The processor P(x,y) is connected to $P(x \pm 1, y \pm 1)$, if they exist. The diameter of this simple mesh is 2(n-1). Here, $(n-2)^2$ number of processors have degree 4, 4(n-2) number of them have 3 and four corner processors have degree 2. The interconnection scheme in ILLIAC IV type processors is a bit complicated with additional wrap-around and end-around connections, bringing down the diameter to n - 1. The degree of each processor in this case is 4. Since the diameter of the of the mesh is $Q(N^{1/2})$, where N is the total number of processor, a lower bound on the time to solve non-trivial problems that involve manipulation of data residing in processors farthest apart in a mesh of size $N(= n^2)$ is $W(N^{1/2})$. In search of an architecture capable of providing faster solutions to such problems, yet retaining the attractive features of a mesh, researchers have studied related interconnections like pyramid [11,12], mesh-of-trees [13], meshes with broadcast buses [14,15] etc.

In this chapter, we discuss about Multi-Mesh (MM) network topology [4]. The MM network is different from multi-dimensional mesh discussed in [7,8]. An MM network having n^4 processors is built around n^2 meshes of size $n \times n$ each. The degree of each processor in MM network is 4. The diameter of an MM network with n^4 processor is 2n. A Hamiltonian cycle exists in the network. Algorithms for point-to-point communication, single node broadcast, multicast and wormhole routing for complete exchange have been developed for this network. Point-to-point communication needs 2n communication steps while one-to-all broadcast and multicast for n^4 processors can be effected in 2n + 8 steps and $n^4 + n^3 + n^2 + n - 1$ steps of data communication respectively. The diameter under single node failure is 2n + 6.

It is shown that an $n^2 \times n^2$ mesh can be emulated on MM in O(1) time. Thus, any algorithm that runs in O(f(n)) time in an $n^2 \times n^2$ mesh, can always be solved in less than or equal to O (f(n)) time. However, the result gives merely an upper bound on the running time of an algorithm on MM network. In practice, many real life problems can be solved on MM network more efficiently than on the corresponding mesh with same number of processors. Specifically, the problems whose time complexity are governed by the diameter of the network, the MM network offers a distinct

2.2 The Multi-Mesh (MM) Network

The Multi-Mesh network is an extension over the simple mesh. In an $n \times n$ mesh, the processors are arranged in n rows and n columns. In MM network, such a mesh is used as the building block where n^2 number of such meshes are arranged in the form of an $n \times n$ matrix.

processors on the four outer boundaries each of which has 3 neighbors within the block . These processors are referred to as *boundary processors* . Also , in each block there are 4 corner processors which have one neighbor within the block . These processors are referred to as *corner processors* . The rest $(n-1)^2$ processors in every block , each having 4 neighbors are referred to as the internal processors . Different blocks are interconnected by inserting suitable links among the boundary processors so that each processors will uniformly have four links in the final topology .

A processor inside a given block can be uniquely identified using two co-ordinates . As the blocks in turn are organized as a matrix , each block can also be uniquely identified using two co-ordinates α and β as $B(\alpha,\beta)$. Thus , each of the n^4 processors can be uniquely identified using a 4-tuple of the co-ordinate values . The first two co-ordinates are used to describe the in which the processor lies and the rest two co-ordinates are used to specify the position of the processor inside that specified block . For example , $P(\alpha, \beta, x, y)$ is the processor lying at the x-th row and the y-th column of the block $B(\alpha, \beta)$. Each of the se co-ordinates may assume a value between 0 and $n-1$.

A special symbol * is used for any one of these 4 co-ordinates to denote the set of all processors with all possible values of the respective co-ordinates . For example , $P(*, *, 0, 0)$ signifies the set of the top left corner of all the n^2 blocks .

If the processors $P(\alpha, \beta_1, x_1, y_1)$ are connected to $P(\alpha, \beta_2, x_2, y_2)$ for all values of α, $0 \leq \alpha \leq n-1$, these set of links are denoted by interconnection between the sets $P(*, \beta_1, x_1, y_1)$ and $P(*, \beta_2, x_2, y_2)$. In a similar manner , a set of blocks can be represented by putting a * in one or both of the two co-ordinates of $B(\alpha, \beta)$.

Interblock connections among the boundary processors are given by the following rules:

1. $\forall \beta, 0 \leq \beta \leq n-1$, $P(\alpha, \beta, 0, y)$ are connected to $P(y, \beta, n-1, \alpha)$, where $0 \leq y, \alpha \leq n-1$,
2. $\forall \alpha, 0 \leq \alpha \leq n-1$, $P(\alpha, \beta, x, 0)$ are connected to $P(\alpha, x, \beta, n-1)$, where $0 \leq x, \beta \leq n-1$.

All these links are two-way connections . Hence, in the MM network all processors have a uniform degree of 4 . These interblock connections among the boundary processors are referred to as the *interblock links* . Rule 1 interconnects two blocks in the vertical direction , and so the corresponding links are called vertical interblock links . Similarly , rule , defines the horizontal interblock links .

It is notable that the processors $P(\alpha, *, 0, \alpha)$ are connected to $P(\alpha, *, n-1, \alpha)$ which lie in the same block by means of interblock links . Similarly , the processors $P(*, \beta, \beta, 0)$ are connected to $P(*, \beta, \beta, n-1)$ again in the same block by interblock links . Thus , four of the boundary processors in each block are connected to some boundary processors in the same block . It should also be noted that one of these two connections is in the horizontal direction and while the other is in the vertical direction .

An example of an MM network for $n = 3$, is given in Fig. 2.1 , where interblock connections in the 0th row of blocks and the 0th column of blocks are only shown . *The interblock connections in the rest of the rows of blocks are same as those in the 0th row and the interblock connections in the rest of the column of blocks are same as those in the 0th column .*

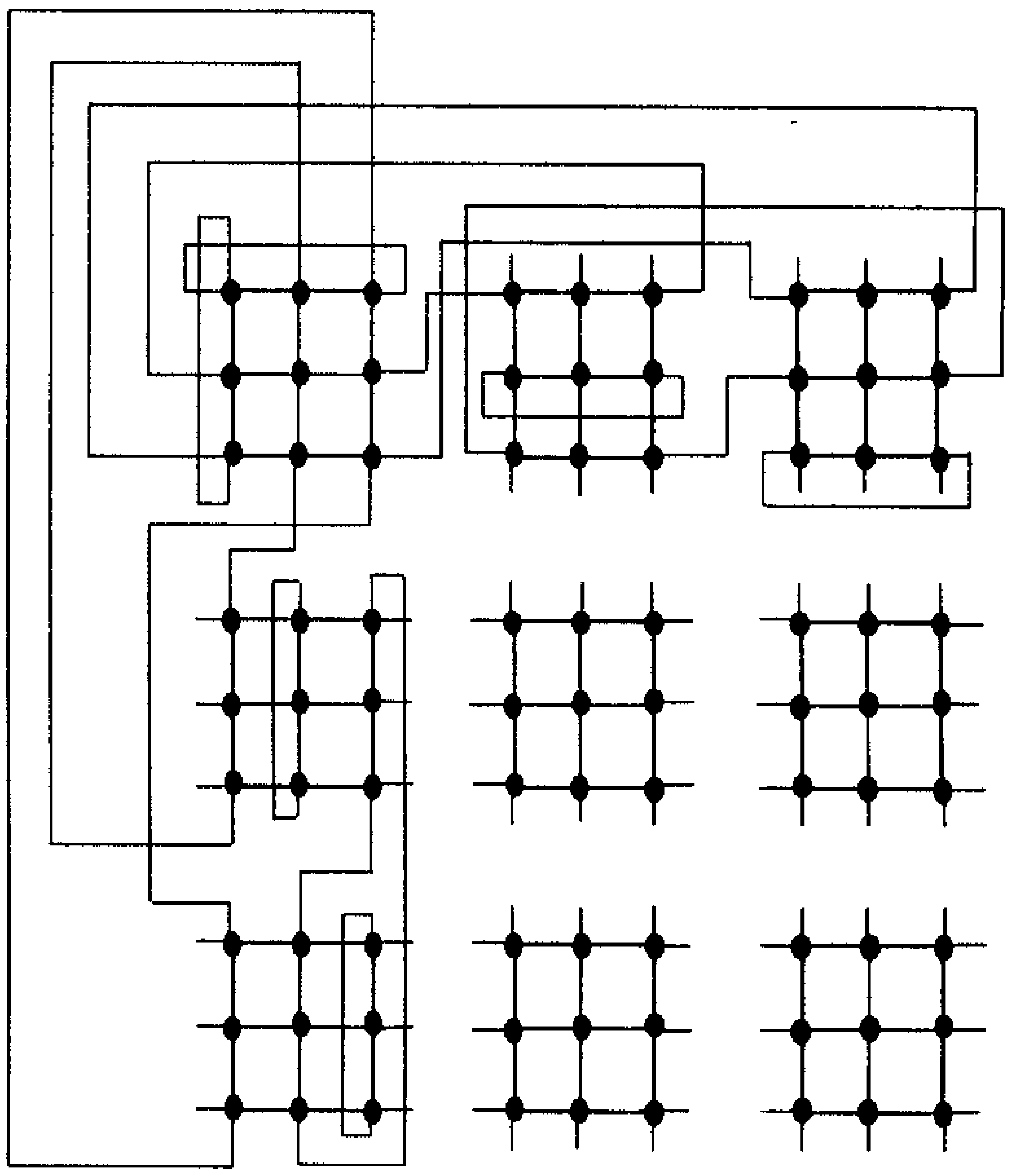


Fig. 2.1. An Example of a Multi-Mesh Network for n = 3
(All interblock links are not shown)

2.3 Topological Properties

Some important properties of the MM network are discussed in this section .

2.3.1 Cycle Structures

The interblock links induce some cycles in the MM network .

Cycles of length n

Every wrap-around link in a block induce a cycle of length n . There are two such cycles in each block , one in horizontal direction and the other in the vertical direction . There are $2n^2$ such cycles of length n in the whole network . The vertical cycles in the ith column of blocks is induced by wrap-around connection in the ith column of processors . Similarly , the horizontal cycles in the ith row of blocks is induced by the wrap-around connections in the ith row of processors .

Cycles of length 2n

Due to horizontal interblock connections of the MM network , there is a cycle of length 2n between the k-th row of the block B(i,j) and the j-th row of the block B(i,k) for $j \neq k$, $0 \leq i \leq n-1$. Thus , there are n(n-1)/2 such horizontal cycles of length 2n in a particular row of blocks . Similarly , due to the vertical interblock connections , there is a cycle of length 2n between the k-th column of block B(i , j) and the i-th column of block (k , j) for $k \neq i$, $0 \leq j \leq n-1$. There are n(n-1)/2 such vertical cycles in a particular column of blocks . In total there are $n^2(n-1)$ cycles of length 2n in the MM network .

Presence of these cycles leads to the following results :

Result 1 : For a given α , if the data elements in $B(\alpha, *)$ are shifted through n positions along the horizontal cycles , then the i-th row elements of $B(\alpha, j)$ will move to j-th row of $B(\alpha, i)$.

Result 2 : For a given β , if the data elements in $B(*, \beta)$ are shifted through n positions along the vertical cycles , then the i-th column of $B(j , \beta)$ will move to the j-th column of $B(i, \beta)$.

An example of data movements along the horizontal cycles in a single row of processors is shown in Fig.2.2 for n=3 .

2.3.2 Diameter of MM network

There exists more than one path between any two processors in the network . However , there exists a path of length less than or equal to 2n between any two processors in the network .

Let the source processor be numbered as and the destination processor be numbered as $P(\alpha_2 , \beta_2 , x_2 , y_2)$,while all co-ordinates may assume values between 0 and n - 1 .

Let the block (α_1, β_1), in which the source processor lies, be referred to as *source block* and block (α_2, β_2) be referred to as the *destination block*. Each of the blocks represented by $B(\alpha_1, *)$ and $B(*, \beta_1)$ is directly connected to the source the source block by one interblock link. Similarly, each of the blocks represented by $(\alpha_2, *)$ and $B(*, \beta_2)$ is directly connected to the destination block by one interblock link. Thus, if $\alpha_1 \neq \alpha_2$ and $\beta_1 \neq \beta_2$, the path from the source to the destination must have to pass through at least one intermediate block. For example, either $B(\alpha_2, \beta_1)$ or $B(\alpha_1, \beta_2)$ can be the intermediate block. The situation is clearly depicted in Fig. 2.3.

Fig. 2.3 shows that there exist 16 different paths between the source and the destination block, each using only one intermediate block. One of these path is of length less than or equal to 2n. To find this, the processors $P(\alpha_1, \beta_1, 0, \alpha_2)$ and $P(\alpha_1, \beta_1, n-1, \alpha_2)$ by an imaginary vertical line and processors $P(\alpha_1, \beta_1, \beta_2, 0)$ and $P(\alpha_1, \beta_1, \beta_2, n-1)$ by an imaginary horizontal line in the source block. These two lines divide the block into four quadrants SQ1, SQ2, SQ3, SQ4 in Fig. 2.3. These four boundary processors are referred to as *source block exits*. These are chosen as exit points as they are connected to either of the intermediate blocks $B(\alpha_2, \beta_1)$ or $B(\alpha_1, \beta_2)$ via one interblock link. Similarly, imaginary lines are drawn in the destination block also, as shown in Fig 2.3. Those lines divide the destination block into four quadrants DQ1, DQ2, DQ3, DQ4. The four boundary processors through which these imaginary lines are drawn are referred to as destination block entries. Each of these destination block entries are connected to one of the intermediate block by one interblock link. From fig 2.3, it is apparent that there are four processor in an intermediate block, out of which two are connected to the source block and are termed as intermediate block entries, while the other two are connected to the destination block and are termed as intermediate block exits. Also, if the intermediate block entries for a certain path are in horizontal boundaries of the intermediate block, then the corresponding intermediate block exits are in vertical boundaries and vice versa.

The source processor $P(\alpha_1, \beta_1, x_1, y_1)$ is contained in one of the quadrants SQ1, SQ2, SQ3, SQ4 in the source block. It is to be noted that each of these quadrants contain exactly two exit points which are two diagonally opposite vertices of the quadrants. Similarly, the destination processor is also contained in one of the four quadrants DQ1, DQ2, DQ3, DQ4 of the destination block. The quadrants in the destination block also have exactly two entry points. The quadrant in which the source processor lies is referred to as *source quadrant* and the quadrant in which the destination processor lie is referred to as *destination quadrant*. It can be shown that one of the two paths that uses only those entry and exit points which are vertices of the source quadrant and the destination quadrant is of length less than or equal to 2n. For example, referring to Fig. 2.3 if the source lies in SQ1 and the destination lies in DQ1, then one of the two paths :

i) $P(\alpha_1, \beta_1, x_1, y_1) \rightarrow P(\alpha_1, \beta_1, \beta_2, 0) \rightarrow P(\alpha_1, \beta_2, \beta_1, n-1) \rightarrow P(\alpha_1, \beta_2, n-1, \alpha_2) \rightarrow$ $P(\alpha_2, \beta_2, 0, \alpha_1) \rightarrow P(\alpha_2, \beta_2, x_2, y_2)$

ii) $P(\alpha_1, \beta_1, x_1, y_1) \rightarrow P(\alpha_1, \beta_1, 0, \alpha_2) \rightarrow P(\alpha_2, \beta_1, n-1, \alpha_1) \rightarrow P(\alpha_2, \beta_1, \beta_2, n-1) \rightarrow$ $P(\alpha_2, \beta_2, \beta_1, 0) \rightarrow P(\alpha_2, \beta_2, x_2, y_2)$

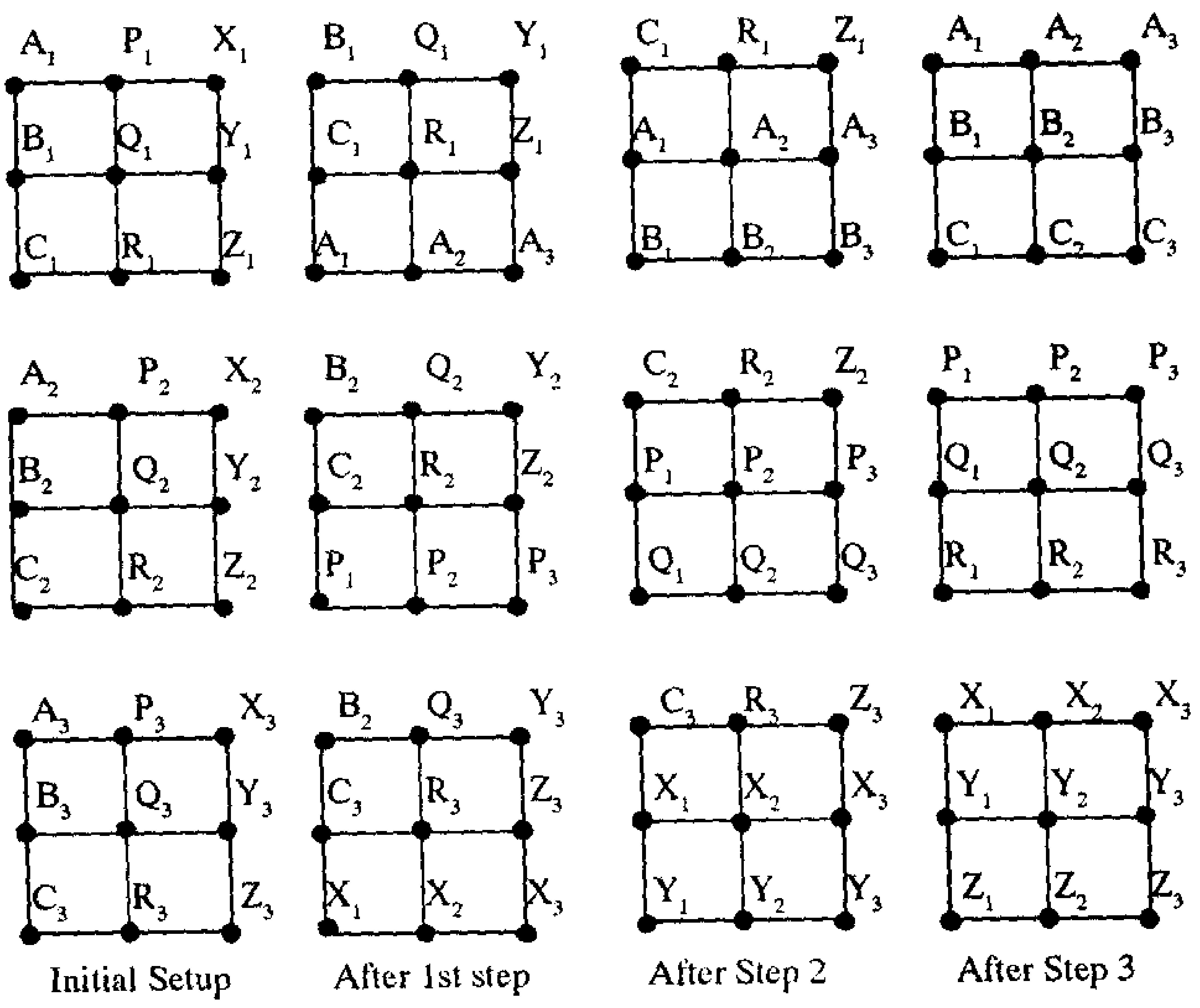


Fig. 2.2. Three Steps of Data Movements in the First Stage of T Operation

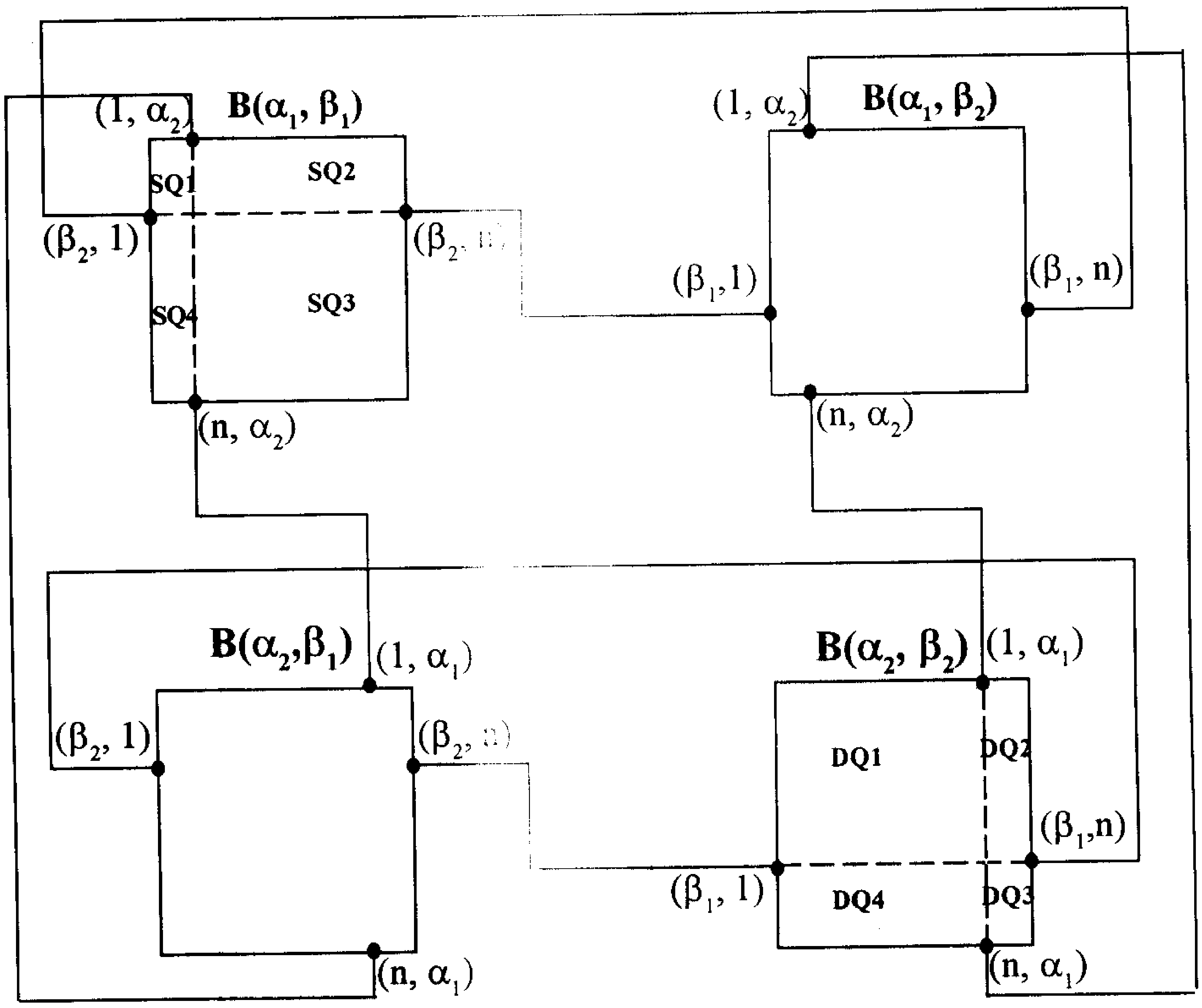


Fig 2.3 Possible Paths Between the Source and the Destination

Chapter 3

Permutation Routing on Mesh

3.1 Introduction

In a network of processors , each processor initially contains one packet destined for some processor in the network . Let v_i denote the packet originating in processor i ; we denote its destination by d(i) . Every processor is destination of one packet . The *permutation routing problem* is to transport v_i to d(i) . In some cases , not all processor has packet to send and hence some processors are not destination of any packet . This kind of permutations are referred to as *partial permutations* .
The algorithms for permutation routing problem can be classified as *oblivious* and *non-oblivious* algorithms . The property of oblivious algorithms is that : the route followed by v_i depends on d(i) alone , and not on d(j) for any $j \neq i$.

Another way of classifying the algorithms is as *on-line* and *off-line* algorithms . In off-line algorithms routes to be taken by the packets are determined off-line and some intermediate destination addresses are appended to each packet so that the packets can be routed to their corresponding destinations without getting queued in intermediate processors .

The permutation routing problem in an $n \times n$ mesh is solved by a variety of algorithms . The algorithms that runs with best time are greedy algorithms where the packets are routed through the shortest path . But these algorithms need extra space to be present in each of the processors . The non-oblivious algorithms which permutes the packets using efficient sorting algorithms needs no extra buffer to be present in the processors . But their running time is higher . In between these two extremes are the algorithms which use sorting algorithm to sort sub-meshes . These algorithms need smaller size buffers to be present in the processors . In fact , the buffer size can be increased to decrease the running time and vice versa .

To reduce running time as well as buffer size , efforts have been made by researchers . In fact , existence of a simple , deterministic , oblivious algorithm which solves the permutation routing problem in optimal time with small buffer size is unknown . Different randomized , oblivious algorithms[9,10] has been proposed which routes the packet in near optimal time using small size queues with high probability . Recently , an efficient heuristic algorithm has been proposed[3] which routes the packets in optimal time in most of the cases . This heuristic is non-oblivious in nature . A recursive algorithm has also been proposed in [2] which routes the packet in optimal time .

In case the permutation to be routed is known in advanced , the routes of each packet can be found out such that the routing needs no buffer to be present in the processors .

In this chapter , we make a study of various types of existing algorithms for permutation routing on mesh .

3.2 Greedy Algorithms

Among the algorithms for solving permutation routing problem, the greedy algorithms run fastest . The idea behind greedy algorithms is to route the packet along a shortest path from its source to the destination . One example of this class of algorithms is an algorithm which routes the packet to its column destination and then to its destination in that column. But the limitation of this class of algorithms is that they require large buffer size . This problem arises because two or more packets might be contending for use of the same edge (in the same direction) at the same time . In that case one of these contending packet will have to be forwarded through that edge and the other(s) will have to be buffered at that node . For example, in the above mentioned algorithm , this kind of a situation can occur when two packet one moving from left to right and another from right to left reaches their column destination in the same step and in the next step both of them has to travel in the same direction (say, upward) . In that case , one of them is selected for forwarding through the upward link and the other is queued in that node . There are many strategies for arbitrating between packets that are contending for the same edge (e.g. , the packet that has to go further goes first) and for preventing the buildup of large queues (e.g., presetting a maximum threshold q , and not advancing a packet forward into a processor with a queue that is at or near the threshold). It can be shown that this algorithm completes routing in 2n - 2 steps in worst-case which is optimal considering the fact that the diameter of mesh is 2n - 2 .

3.3 A Deterministic Algorithm with Queue Size 1

This is a sorting based algorithm which routes the packets in three phases . In Phase 1 , packets are sorted into column-major order according to the column destination of each packet . Ties are broken by row destination . Phase 1 takes 5n + o(n) time in SIMD machine (4n + o(n) time for sorting into column-major snake-like ordering using shear-sort algorithm [17] and n steps to rearrange it to column-major order) and 4n + o(n) time in MIMD machine since in MIMD model shear-sort algorithm takes 3n + o(n) time to sort into column-major ordering . This phase can be completed without requirement of any extra buffer in the processors . If the permutation is complete , then the routing is completed after this phase itself.

INDIAN STATISTICAL INSTITUTE LIBRARY
C 23797
24 SEP 1997

Otherwise , in Phase 2 , packets are routed to their correct column destination . This takes n - 1 steps and can be completed without use of buffer space , since after Phase 1 , in each row there are at most one packet destined for each column . In phase 3 , the packets are routed to their destination . This also takes n - 1 steps to complete .

This algorithm takes 7n - 2 + o(n) steps in SIMD machine and 6n - 2 + o(n) in MIMD machine . This algorithm can handle partial permutations also .

3.4 A modified Algorithm

A better performing algorithm can be constructed where dummy packets are introduced in case of a partial permutation . In this algorithm a tag is added to each packet before running the shear-sort algorithm according to this tag so that after the end of sorting the packets reaches its correct destination . The tag is assigned to each packet in such a way that if the value of the tag is i , then the destination address contains i-th smallest element in the snake-like row-major indexing scheme .

Let the rows and columns be numbered from 0 to n - 1 . If (r,c) is the destination of a packet , then a tag is assigned to it with value r * n + c , if r is even and r * n + (n - c) if r is odd . After running shear-sort algorithm according to this tag with respect to snake-like row-major ordering , the packet reaches their corresponding destination .

This algorithm takes 4n + o(n) steps in SIMD machine and 3n + o(n) steps in MIMD machine.

3.5 Deterministic Algorithm with Constant Queue Size

The running time can be decreased by allowing the processors to have buffer of constant size . Let the processors have buffer of size q . This algorithm [7] consists of three phases . In Phase 1 , the mesh is partitioned into q^2 number of blocks of size $n / q \times n / q$ and packets are sorted in row-major snake-like order inside these blocks according to the row destinations of the packets . Using shear sort algorithm this step can be completed in 4n / q + o(n/q) steps . In Phase 2 , each packet is routed to its row destination . In Phase 3 , packets are routed to the destination . The last two steps take 2n -2 steps . Hence , the entire algorithm takes (2 + 4/q)n + o(n/q) steps . The buffer requirement in each processor is 2q - 1 .

3.6 Randomized Algorithms

The randomized algorithms attempt divide a worst-case routing problem into two average-case routing problem and solve it with small queue size with high probability . A randomized algorithm presented in [9,10] completes permutation routing in three phases . In Phase 1 , each column is partitioned into n / log n intervals and each packet is routed to a randomly selected destination within its interval . In Phase 2 , each packet is routed within its current row to its correct column . In Phase 3, every packet is routed within its correct column to its correct destination . Contention for use of links are handled using farthest-first protocol . The final column routing phase does not start until every packet has completed its row-routing . It has been shown that the resulting algorithm runs in 2n + o(n) steps and uses queue of size at most O(log n) with high probability .

3.7 A Heuristic Algorithm

A heuristic algorithm has been proposed in [3] for permutation routing in an n ′ n mesh . This algorithm is built upon odd-even transposition [18] method . The odd-even transposition method works as follows : At odd time instances , the processors that are at odd positions in a row of processors compares their contents with the content of their left neighbors and a decision is made on whether an exchange of their contents will occur . At even time instances , comparison and a possible exchange takes place between the processors at even position of the row of processors and their left neighbors . This algorithm at each step tries to reduce the summation of the distances that all packets has to travel . Let D(t) and D(t +1) , respectively , be the total distance that all packets have to travel at time instances t and t + 1 . Let at time t , processors P_i and P_{i+1} contain packet p_i and p_{i+1} and let the distances these packet have to travel be $d_i(t)$ and $d_{i+1}(t)$, respectively . During odd-even transposition , the decision on exchanging packet between two neighboring processors at time instance t is taken based on the following two criteria :

1. $D(t+1) \leq D(t)$

or 2. $\max(d_i(t), d_{i+1}(t)) \leq \max(d_i(t+1), d_{i+1}(t+1))$

where $d_i(t+1)$, $d_{i+1}(t+1)$ are the distances the packets have to travel after the exchange .

The algorithm is as follows :

Perform odd-even transposition on the row of the mesh such that the above mentioned criteria are satisfied (packets are moving only horizontally). If a packet reaches the processor that lies at its column destination , and , no other packet in that processor and no other packet at that processor want to move vertically in the same direction , it starts vertical movement . Otherwise , the packet that has to move further in the column , moves vertically , while , the other packet participates in the odd-even transposition that is taking place on the row .

Simulation results has shown that the number of steps required to complete the routing is almost equal to the maximum distance a packet has to travel .

3 .8 Off-line Algorithms

In some applications , the routing is known in advanced . In these cases , the route to be taken is determined off-line so that the packets can reach their destination without waiting in the intermediate processors . Off-line algorithms [1] are particularly advantageous when the same packet routing problem is encountered over and over again .

To ff-lin alg ritom f r p rmutati n r uting n an $n \times n$ m so w rks in tor **poas** . In Poas 1, to pack ts ar p rmut d witoin aco c lumn s toat at m st n pack t in **aco** r w is d stin d f r aco c lumn . In Poas 2 , aco pack t witoin its r w t to c rr ct c **lumn** . In Poas 3 , aco pack t witoin its c lumn t its c rr ct r w (i. . t its d stinati n) .

To oard part f tois alg ritom is figuring ut o w t p rmut to pack ts witoin aco c lumn during Poas 1 s toat to r will b at m st n pack t in aco r w d stin d f r any c lumn . Tois part f to alg ritom is d n ff-lin . Onc tois is d n , aco poas is **asily** impl m nt d in n - 1 st ps . Tous , to n-lin part f tois alg ritom runs in 3n - 3 wito qu u siz f 1 by using to gr dy alg ritom t r ut to pack ts witoin aco r w r c lumn .

To ff-lin part f to alg ritom is as f ll ws : Giv n an arbitrary p rmutati n r uting pr bl m , first a bipartit r uting grapo G= (U , V , E) is c nstruct d wito $U=\{u_1 , u_2 ,....,u_n\}$ and $V= \{v_1 , v_2 ,....,v_n\}$ and n^2 dg s $E=\{ _0, \ _1 ,...., _{n^2-1}\}$ suco toat kto dg $_k$ c nn cts u_{i_k} t v_{j_k} ,wo r i_k is to starting c lumn f to kto pack t and j_k is to d stinati n c lumn f kto pack t ($0 \leq k \leq n^2- 1$) .

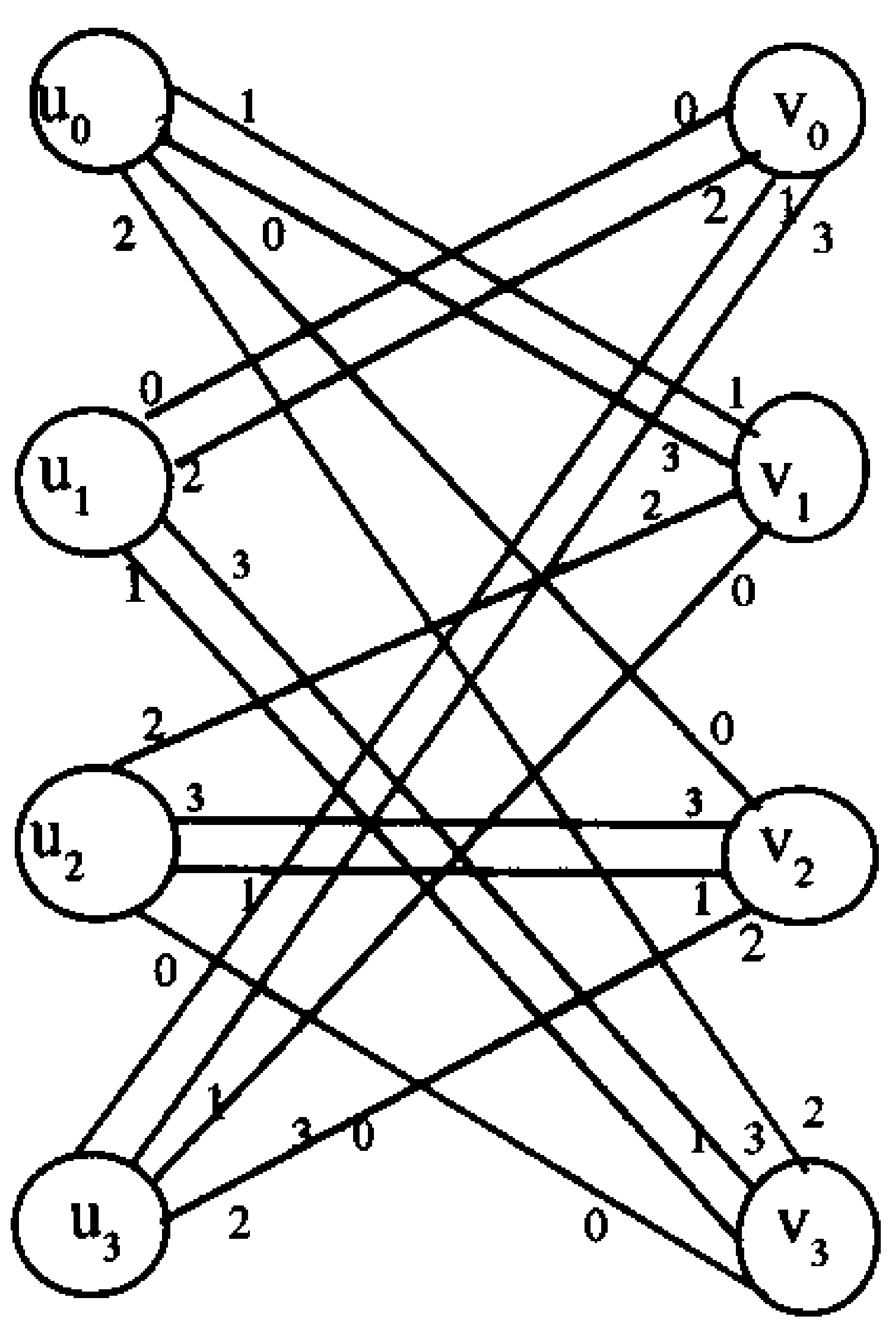
Sinc G in a n-r gular bipartit grapo (i. . v ry n d f G is incid nt t n dg s , and n d s f U ar nly adjac nt t n d s in V and vic v rsa), it is p ssibl t lab l aco dg f G wito an int g r in to int rval [1, r] s toat any tw dg s incid nt t to sam n d will oav diff r nt lab ls.

Onc lab ls f to dg s f G is d t rmin d , it is asy t sp cify o w to pack ts so uld b r arrang d witoin aco c lumn . In particular , aco pack t *p* is s nd t r w l_p , wo r l_p d n t s to lab l f r to dg c rr sp nding t pack t *p* .

An xampl in Fig. 3. 1 , so ws o w to ff-lin part f to alg ritom w rks . Fig 3.1(a) so ws a p rmutati n t b r ut d in a 4×4 m so . To numb rs insid grids impli s to d stinati n f to pack t starting in toat pr c ss r . Fig 3.1 (b) so ws to c rr sp nding bipartit r uting grapo and to lab ls assign d t to dg s aft r c mputati n is d n .To lab ls ar in to rang [0,4]. Fig 3.1(c) so ws to situati n aft r to pack ts ar p rmut d witoin aco c lumn acc rding t to lab ls assign d t to c rr sp nding dg .

	Col 0	Col 1	Col 2	Col 3
Row 0	3,1	0,3	3,2	2,2
Row 1	3,3	2,3	0,1	1,0
Row 2	0,2	0,0	1,3	2,1
Row 3	1,1	2,0	1,2	3,0

(a)



(b)

	Col 0	Col 1	Col 2	Col 3
Row 0	0,2	0,0	1,3	2,1
Row 1	3,1	2,3	3,2	1,0
Row 2	3,3	2,0	0,1	2,2
Row 3	1,1	0,3	1,2	3,0

(c)

Fig 3.1 :An example :(a) a permutation routing problem (b) corresponding bipartite routing graph and labels assigned on the edges (c) after permuting packets within the columns according to the labels

Chapter 4

Permutation Routing on Multi-Mesh

4.1 Introduction

We have observed in the last chapter that a space-time trade-off is involved in the solution of the permutation routing problem . In one extreme are the algorithms which solves the problem with best possible time , but requires large buffer to be present in each processor (Greedy Algorithms) and at the other extreme are the algorithm which solves the problem by applying sorting on the entire mesh without any use of buffer . In between these two extreme are the algorithms which solves the problem by applying sorting on sub-meshes and uses buffer of constant size .

In this chapter , we first give a greedy algorithm which requires buffers of large size to be present in each processor . Next , we present an algorithm which is based on sorting of packet on the entire MM network which requires no buffer space in the processors . Then , we present an algorithm which uses constant buffer size . Similar algorithms for permutation routing on two-dimensional and multidimensional mesh runs in better time than the correspo\nding algorithms which uses sorting on the entire mesh .

Finally , we present an off-line algorithm .

4.2 A Greedy Algorithm

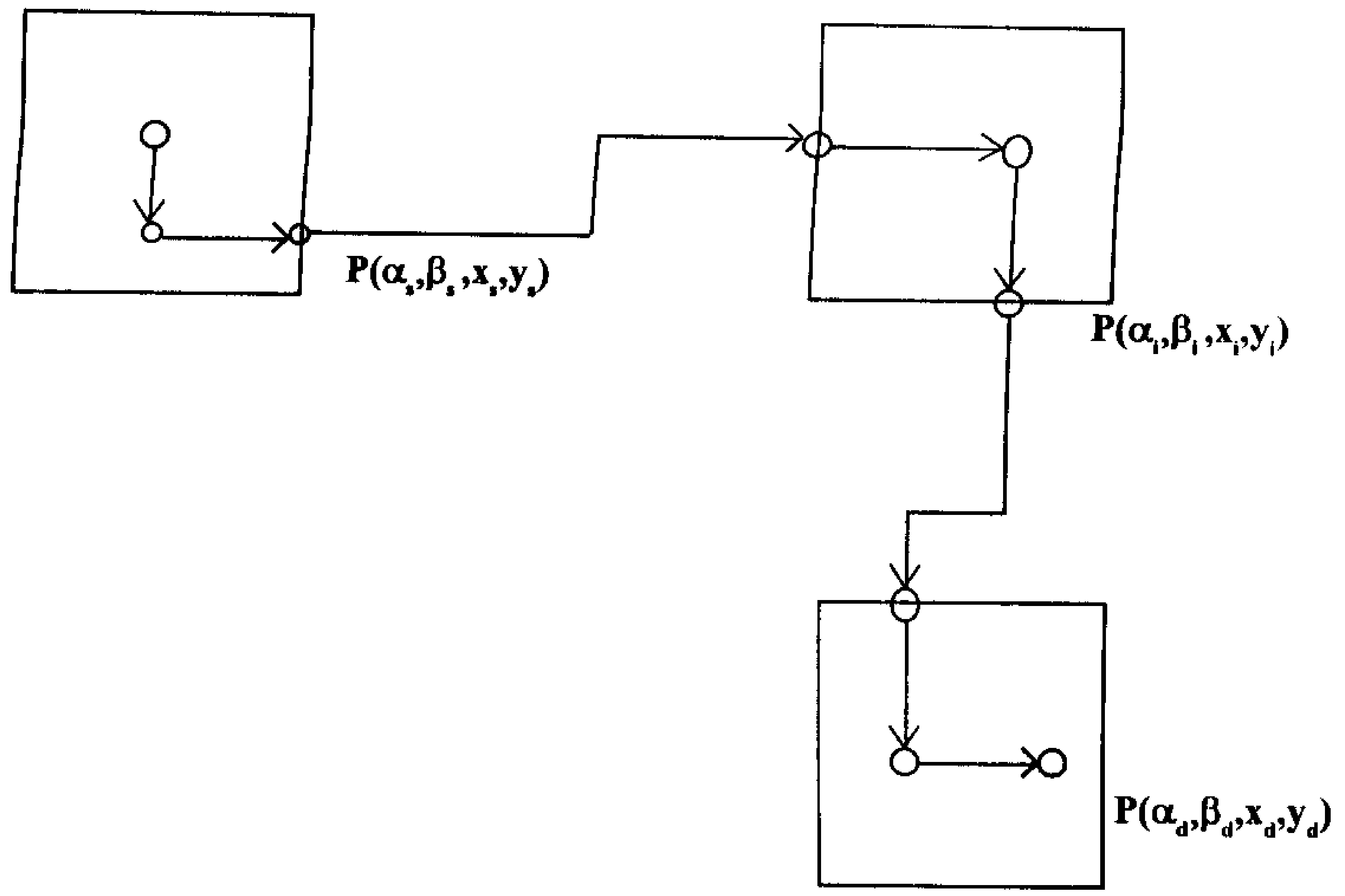
The idea behind this algorithm is to find for each packet the shortest path between its source and its destination and then route the packet through that path . The algorithm works as follows :

As described in the Chapter 2 and shown in Fig. 2.3 , for each packet we can divide both the source block and destination block into four quadrants . And depending on the quadrant in which the source and the destination lie one of the two paths that pass through entry or exit points of the source and the destination quadrant is chosen as the path to be taken by that particular packet . This computation is done in each source processor . The source processors then appends three fields with the data packet . These appended fields are :

Field (1): containing the address of the source block exit processor $(\alpha_s , \beta_s , x_s , y_s)$ if source block and destination block are not the same , otherwise this field is empty .
Field (2): containing the address of the intermediate block exit processor $(\alpha_i , \beta_i , x_i , y_i)$ if source block and the destination block does not lie on the same column of blocks ,otherwise this field is empty .
Field (3): containing the address of the destination processor $(\alpha_d , \beta_d , x_d , y_d)$.
The algorithm is as follows :

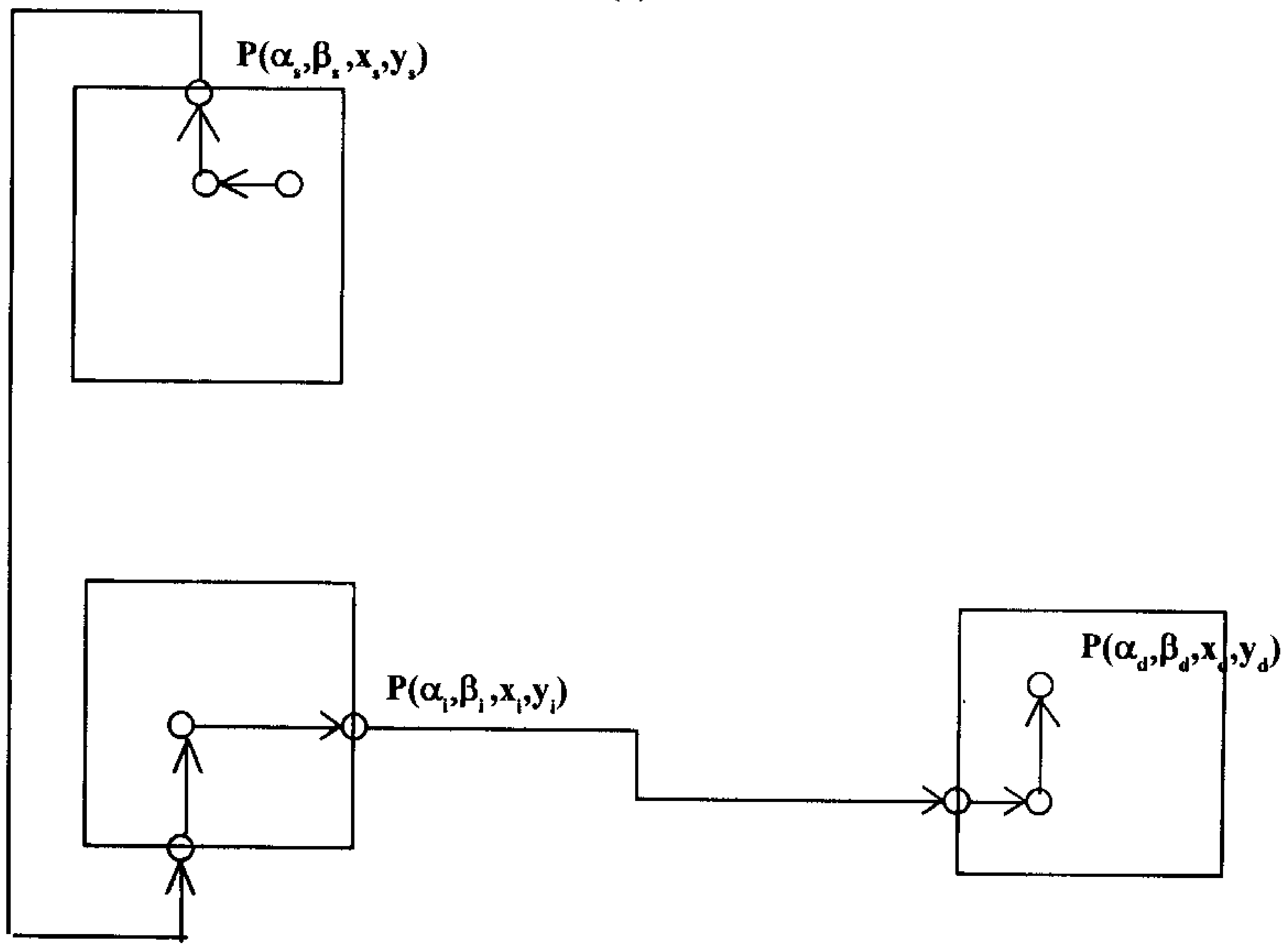
```
In each processor $(\alpha, \beta, x, y)$ do in parallel :
  for each arriving packet D do
    begin
      if (($\alpha, \beta, x, y$) is the source ) then
         make the first non-empty field current .Let us denote the current field as ($\alpha_c$ , $\beta_c$, $x_c$ ,$y_c$)
      else
        if (the packet has arrived following a interblock link ) then
           make the next non-empty field the current .
       if ($y_c$ = n -1) then
         if ($x_c \neq x$ and $y_c \neq y$ ) then
            put the packet in the queue of that vertical link which takes lesser number of steps to
            reach ($\alpha_c$ , $\beta_c$, $x_c$, $y_c$ ).
         else if ($y_c \neq y$ ) then
                put the packet in the queue of that horizontal link which takes lesser number of
                to reach ($\alpha_c$ , $\beta_c$, $x_c$, $y_c$ ).
              else
                if (current field is field number 3) then
                   stop
                else
                   put the packet in the queue of the horizontal interblock link
       else
          if ($x_c \neq x$ and $y_c \neq y$ ) then
             send the packet through that horizontal link which takes lesser number of steps to
             reach ($\alpha_c$ , $\beta_c$, $x_c$, $y_c$ ).
          else if ($x_c \neq x$ ) then
                 put the packet in the queue of that vertical link which takes lesser number of
                 steps to reach ($\alpha_c$ , $\beta_c$, $x_c$, $y_c$ ).
               else
                 if (current field is field number 3) then
                    stop
                 else
                    put the packet in the queue of the vertical interblock link
       end
    for each link do
       select the packet that has to go farther and forward it through that link .
```

It can be observed from the that two or more packets may arrive at a processor for which the next edge to be used is same . In that case , the packet that has to go farther , uses the edge first and the other(s) is(are) buffered in the processor .



$P(\alpha_s,\beta_s,x_s,y_s)$
$P(\alpha_i,\beta_i,x_i,y_i)$
$P(\alpha_d,\beta_d,x_d,y_d)$

(a)



$P(\alpha_s,\beta_s,x_s,y_s)$
$P(\alpha_i,\beta_i,x_i,y_i)$
$P(\alpha_d,\beta_d,x_d,y_d)$

(b)

Fig 4.1 Examples showing two possible routes taken by packets in greedy algorithm

4.3 Deterministic Algorithm with Queue Size 1

The algorithm presented here is a sorting-based algorithm and runs without requirement of any extra buffer in the processor in the MM network . The basic idea is to assign a tag to each packet based on its destination address and appropriate to the indexing scheme with respect to which the sorting is done . Next , a sorting algorithm is run which sorts the packets according to the tag assigned it . The result of this is that each packet reaches its destination .
Let the destination of the packet which starts at $P(\alpha_1, \beta_1, x_1, y_1)$ be $P(\alpha_2, \beta_2, x_2, y_2)$. If the permutation is partial , then dummy packets are introduced destination of which is the source itself .

The sorting algorithm for MM network [6] sorts with respect to snake-like ordering as shown in the Fig . 4.1 . The snake-like indexing scheme in MM network order the packets in following manner :

Let $D(\alpha, \beta, x, y)$ denote the data element residing in processor $P(\alpha, \beta, x, y)$. In a block $B(\alpha, \beta)$ where $\alpha + \beta$ is some even number , the snake-like sorted sequence starts from the leftmost end of the 0-th row and ends at the (n - 1)-th row . Such a block is referred to as even block . In a block $B(\alpha, \beta)$ where $\alpha + \beta$ is some odd number , the snake-like sorted ordering starts at from the (n - 1)-th row and ends at the left most end of the 0-th row . In snake-like ordering , $D(0, \beta, *, *) \leq D(1, \beta, *, *) \leq \cdots \leq D(n-1, \beta, *, *)$ if β is even and $D(0, \beta, *, *) \geq D(1, \beta, *, *) \geq \cdots \geq D(n-1, \beta, *, *)$ if β is odd . And finally , $D(*, 0, *, *) \leq D(*, 1, *, *) \leq \cdots \leq D(*, n-1, *, *)$.

In each processor $P(\alpha_1, \beta_1, x_1, y_1)$ for $0 \leq \alpha_1, \beta_1, x_1, y_1 \leq n-1$, the following computation is done to append tag to each packet appropriate to snake-like ordering shown in Fig 4.1 :

```
begin
   if ( $\beta_2$ is even ) then
      begin
         $t_1 = \alpha_2 * n^2$ ;
         if ( $\alpha_2$ is even ) then
           begin
              $t_2 = x_2 * n$ ;
              if ( $x_2$ is even ) then
                  $t_3 = y_2$ ;
              else
                  $t_3 = n - 1 - y_2$ ;
           end
```

```
          else
            begin
              $t_2 = (n - 1 - x_2) * n$ ;
              if ( $x_2$ is even ) then
                  $t_3 = n - 1 - y_2$ ;
              else
                  $t_3 = y_2$ ;
            end
        end
  else
      begin
          $t_1 = ( n - 1 - \alpha_2 ) * n^2$ ;
          if ( $\alpha_2$ is odd ) then
            begin
              $t_2 = x_2 * n$ ;
              if ( $x_2$ is even ) then
                  $t_3 = y_2$ ;
              else
                  $t_3 = n - 1 - y_2$ ;
            end
          else
            begin
              $t_2 = (n - 1 - x_2) * n$ ;
              if ( $x_2$ is even ) then
                  $t_3 = n - 1 - y_2$ ;
              else
                  $t_3 = y_2$ ;
            end
      end
  $t_4 = \beta_2 * n^3$ ;
  tag = $t_1 + t_2 + t_3 + t_4$ ;
end
```

The tags are assigned to each packet in such a way that , if i is the value of the tag for some packet , then the destination of the packet comes in the i-th position in the snake-like ordering in MM network . If packets are sorted according the tag assigned in the above manner with respect to snake-like indexing scheme , then packets reach their corresponding destination .

This algorithm completes in $58n + o(n)$ time which is the time complexity of the sorting algorithm in MM network .

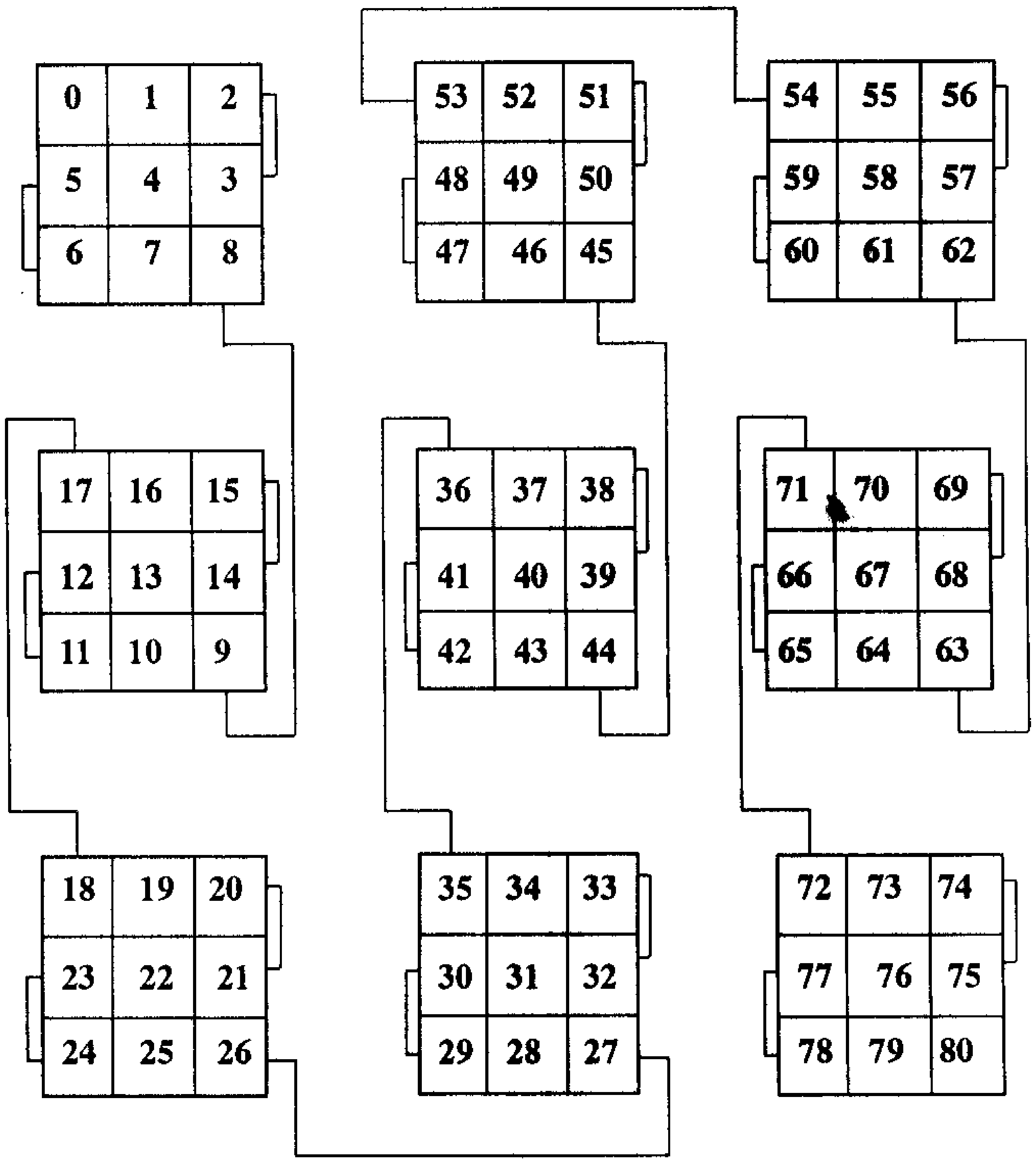


Fig. 4.1 Snake- like Indexing Scheme in MM network

4.4 A Deterministic Algorithm with Constant Queue Size

In this section , we present an algorithm without giving proof of correctness which requires buffer of size c to be present in each processor in the network . Before we give the algorithm , we give few definition :

T-permutation : A T-permutation performs a permutation among each set of processors represented by $P(*, \beta, x, y)$ in parallel . It can be noted that there exist no direct connections among the processors in a set . Hence , this operation is completed in three stages:

(i) Perform n shifts along the vertical interblock links , so that the packets $D(*, \beta , x, y)$ are brought into $P(y , \beta, x , *)$ i.e., to a row of processors .The situation is depicted in Fig. 4.1. This stage needs n steps .
(ii) Perform permutation in each row of processor in each block . This stage needs n -1 steps.
(iii) This stage is just the reverse of the stage (i) in which n shifts are performed along the vertical interblock links in the reverse direction as stage (i). This step needs n steps .
The T-permutation operation needs 3n - 1 steps to complete .

F-permutation : An F-operation performs a permutation among each set of processors represented by $P(\alpha, * , x , y)$ in parallel . Here also , there exist no direct connections among the processors in a set . Hence , this operation is completed in three stages:

(i) Perform n shifts along the horizontal interblock links , so that the packets $D(\alpha, * , x , y)$ are brought into $P(\alpha , x , * , y)$ i.e. to a column of processors .The situation is depicted in Fig. 4.2.
(ii) Perform permutation in each column of processor in each block .
(iii) This stage is just the reverse of the stage (i) in which n shifts are performed along the horizontal interbock links in the reverse direction as stage (i).
The F-permutation operation also takes 3n - 1 steps to complete .

The algorithm is as follows :
Let the destination address of the packet D starting at $P(\alpha_1 ,\beta_1 ,x_1 ,y_1)$ be $P(\alpha_2 ,\beta_2 ,x_2,y_2)$. The idea is to first correct the y-coordinate , next the x-coordinate , then the β-coordinate and finally the α-coordinate .
For this,
Step (1) : First, the packets are rearranged in the entire network in such a way that the correction of the α-coordinate can be done using T-permutation operation without congestion . Let the position of D after this step be $P(\alpha_i ,\beta_i ,x_i,y_i)$.
Step (2) : Next , the packets are rearranged within each row of blocks in such a way that the correction of the b-coordinate can be done using F-permutation operation without congestion . Let the position of D after this step be $P(\alpha_i ,\beta_j ,x_j,y_j)$.
Step (3) : Each block is divided into subblocks of size $k \times k$ where $2 \le k \le c^{1/3} \le n^{1/4}$.

(i) The packets are sorted in each subblock in column-major order based on their x and y-coordinates as follows : if $(x_p, y_p) \le (x_q, y_q)$ if $y_p n + x_p \le y_q n + x_q$.
(ii) In each row of processors in each block , the packets transmitted to the column that corresponds to the y-coordinates of their destination address . In particular , D is transmitted to the y_2-th column .
(iii) In each column of processors in each block , the packets are transmitted to the row that corresponds to x-coordinate of their destination address . In particular , D is transmitted to the x_2-th row .

After step 3 , the position of D is $P(\alpha_i, \beta_j, x_2, y_2)$ i.e. , x and y-coordinates of each packet is corrected . It can be shown that this step completes in $2n + O(n/c)$ steps and rpquires buffer of size c to be present in each processor .

Step(4) : The packets are transmitted to the correct column of blocks using F-permutation operation . After this step , the position of D is $P(\alpha_i, \beta_2, x_2, y_2)$.
Step(5) : The packets are transmitted to the correct row of blocks using T-permutation operation . After this step all packets reach their destination .

The rearrangement operation of step (1) is as follows :
Divide the entire network into k^4 subnetworks consiting of $(n/k)^4$ processors each — $(n/k)^2$ processors each from $(n/k)^2$ different blocks . The subnetwoks are given by , $S(k_1, k_2, k_3, k_4) = [k_1 n/k, \cdots, (k_1+1)n/k] \times [k_2 n/k, \cdots, (k_2+1)n/k] \times [k_3 n/k, \cdots, (k_3+1)n/k] \times [k_4 n/k, \cdots, (k_4+1)n/k]$

Step(1.1) : In each such subnetwork , the packets are sorted according to the following indexing scheme: $(\alpha_p, \beta_p, x_p, y_p) \le (\alpha_q, \beta_q, x_q, y_q)$ if $y_p n^3 + x_p n^2 + \beta_p n + \alpha_p \le y_q n^3 + x_q n^2 + \beta_q n + \alpha_q$
Step(1.2) : The packets are transmitted in such a way that the packet at (α, β, x, y) moves to $((\alpha + (y \bmod k)n/k) \bmod n, \beta, x, y)$. This can be achieved by using T-permutation operation .
Step(1.3) : In each subnetwork , the packets are sorted according to the following indexing scheme : $(\alpha_p, \beta_p, x_p, y_p) \le (\alpha_q, \beta_q, x_q, y_q)$ if $\alpha_p n^3 + \beta_p n^2 + x_p n + y_p \le \alpha_q n^3 + \beta_q n^2 + x_q n + y_q$

The rearrangement operation of step 2 is as follows :
Divide each row of blocks into k^3 sets of processors containing $(n/k)^3$ each — $(n/k)^2$ processors each from n/k different blocks . The sets of processors in each row of blocks α are given by , $SP(k_1, k_2, k_3) = [\alpha] \times [k_1 n/k, \cdots, (k_1+1)n/k] \times [k_2 n/k, \cdots, (k_2+1)n/k] \times [k_3 n/k, \cdots, (k_3+1)n/k]$

Step(2.1) : In each such set of processors , the packets are sorted according to the following indexing scheme: $(\beta_p, x_p, y_p) \le (\beta_q, x_q, y_q)$ if $y_p n^2 + x_p n + \beta_p \le y_q n^2 + x_q n + \beta_q$
Step(2.2) : The packets are transmitted in such a way that the packet at (α, β, x, y) moves to $(\alpha, (\beta + (y \bmod k)n/k) \bmod n, x, y)$. This can be achieved by using T-permutation operation .

Step(2.3) : In each set of processors $SP(k_1, k_2, k_3)$, the packets are sorted according to the following indexing scheme : $(\beta_p, x_p, y_p) \le (\beta_q, x_q, y_q)$ if $\beta_p n^2 + x_p n + y_p \le \beta_q n^2 + x_q n + y_q$
Here , we have assumed that sorting in multi-mesh can be done according to the indexing in schemes described in steps 1.1,1.3,2.1 and 2.3

4.5 An Off-line Algorithm

The off-line algorithm we present here is an extension of the off-line algorithm given in Chapter 3 for mesh . The idea behind the off-line algorithm we present here is to perform some precomputation to augment each packet with some intermediate destination addresses , so that the packet can be routed through these addresses without requirement of extra buffer in the processors in the MM network .

The precomputation required is described below . Let P_d be the destination of the packet D which is starting at P . We consider D as the representative of the packets to show how intermediate destination addresses are augmented to the packets .

First , find a permutation $perm_1$ within each column of blocks such that in each set of processors represented by $P(\alpha, *, x, y)$ for particular values of α, x , y , there is at most one packet destined for each column of blocks . Let P_1 be the destination of D in $perm_1$.

For routing $perm_1$ without requirement of extra buffer , we need that each packet be (i) first , routed to the correct block and then (ii) to P_1 within that block .

For effecting step (i) , a permutation $perm_2$ within each block such that in each set of processors represented by $P(*, \beta, x, y)$ for particular values of β, x , y , there is at most one packet destined for each block in the β-th column of blocks . Let P_2 be the destination of D in $perm_2$.

For routing $perm_2$ without requirement of any extra buffer , we need that each packet be first routed to the correct column of processor and then to P_2 which lies in that column of processor .

As discussed in the previous chapter , if this is to be done without using extra buffer , we need to find a permutation $perm_3$ within each column of processors so that each row there is at most one packet destined for each column of processor . Let P_3 be the destination of D in $perm_3$. At this stage of computation , addresses of P_3 and P_2 are appended to D in that order .

This completes the computation required for routing $perm_2$.

For effecting (ii) , a permutation $perm_4$ within each column of processor is to be found such that each row there is at most one packet destined for each column of processor . Let P_4 be the destination of D in $perm_4$. At this stage of computation , addresses of P_4 and P_1 are appended to D in that order .

This completes the computation required for routing $perm_1$.

For routing the packets in the destination column of blocks , exactly similar computation is to be done . Here , D is augmented with another four addresses including that of P_d .

Once this precomputation is over , the packets can be routed through the intermediate addresses without any congestion as follows :

Phase I :

Route D

(i) to P_3 which lies in the same column of processors as D . This step requires n - 1 steps .

(ii) to the column in which P_2 lies . This step requires n -1 steps .
(iii) to P_2 . This takes another n - 1 steps .
(iv) to the block in which P_1 lies . This can be completed in 3n - 1 steps using T-permutation operation .
(v) to P_4 which lies in the same column of processor as it is now . This takes n - 1 steps .
(vi) to the processor lying in the same column as P_1 . This takes n - 1 steps .
(vii) to P_1 . This takes n - 1 steps .

Number of steps required to complete these seven steps is 9n - 7 .

Phase II :

Route D to the column of blocks in which P_d lies . This can be completed in 3n - 1 using F-operation .

Phase III :

Next , D is to be routed to the destination P_d for which exactly similar steps as Phase I is to be taken . This takes 9n -7 steps .

Hence the entire routing can be completed in 21n - 15 number of steps to complete .

Next , we discuss how to generate the permutations required in finding intermediate destinations :

The permutation $perm_1$ can be generated by constructing bipartite routing graph as follows :

Given an arbitrary permutation routing problem , bipartite routing graph G= (U , V , E) is constructed with $U=\{u_1 , u_2 ,....,u_n\}$ and $V= \{v_1 , v_2 ,....,v_n\}$ and n^4 edges $E=\{e_0 ,e_1 ,....,e_{n^4-1}\}$ such that kth edge e_k connects u_{i_k} to v_{j_k},where i_k is the starting column of blocks number of the kth packet and j_k is the the destination column of blocks number of kth packet ($0 \leq k \leq n^2-1$) .

The bipartite graph G is n^3 - regular i.e. n^3 number of edges are incident on each vertex of both U and V . This is because of the fact that n^3 number of packets are to be routed from each column of blocks and each column of blocks has n^3 of number destination . By Hall's Theorem [1],unique labels in the range [$0,n^3 - 1$] can be assigned to each of the edges . For examples , let the edge e_k connecting u_{i_k} and v_{j_k} gets a label r , then for one of the packets whose source is in the column of blocks numbered u_{i_k} and destined for column of blocks numbered v_{j_k} , the destination in the generated permutation is (r / n^2 , u_{i_k} , (r mod n^2) / n , (r mod n^2) mod n) .

The permutations $perm_2$ and the similar permutation required for determining the routes of Phase III can be generated as follows :
A bipartite routing graph $G' = (U' , V' , E')$ is constructed with $U' = \{u_1 , u_2 ,\dots,u_n\}$ and $V'= \{v_1 , v_2 ,\dots,v_n\}$ and n^3 edges $E' =\{e_0 ,e_1 ,\dots,e_{n^3-1}\}$ such that kth edge e_k connects u_{i_k} to v_{j_k} ,where i_k is the starting block number of the kth packet and j_k is the the destination block number of k-th packet ($0 \leq k \leq n^2 - 1$) .

The bipartite graph G is n^2 - regular i.e. n^2 number of edges are incident on each vertex of both U and V . This is because of the fact that n^2 number of packets are to be routed from each block and each block has n^3 of number destination . By Hall's Theorem ,unique labels in the range [0, n^2 - 1] can be assigned to each of the edges . For examples , let the edge e_k connecting u_{i_k} and v_{j_k} gets a label r , then for one of the packets whose source is in the block numbered u_{i_k} and destined for the block numbered v_{j_k} , the destination in the generated permutation is (u_{i_k} , β , r / n , r mod n) where β is the column of blocks number which is being considered .

The permutations $perm_3$, $perm_4$ and the similar permutations required for routing in Phase III can be generated as described in previous chapter by constructing n-regular bipartite routing graph and then labeling the edges .

An MM network out-perform a mesh of same size in terms of an off-line algorithm when number of processor in the network N is greater than 2400 . We get this result by solving the equation : $3N \geq 21N^{1/4}$.

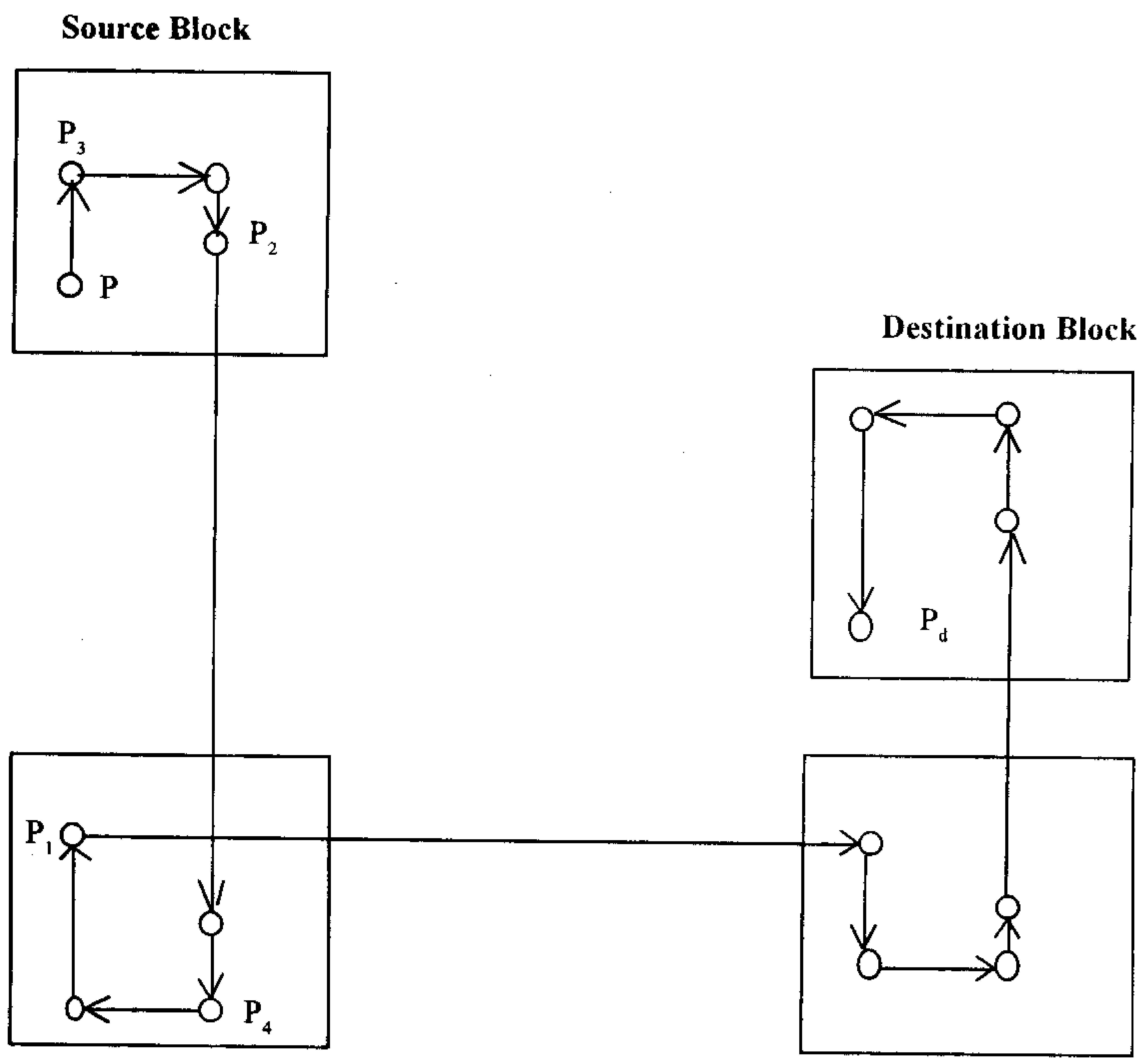


Fig. 4.3 An example showing the routes taken by a packet starting at processor P and destined for P_d

Chapter 5

Conclusion and Further Work

The algorithm presented for permutation routing on multi-mesh which requires buffer of constant size to be present in each processor is incomplete because the proof of correctness and time complexity analysis is not given . Also , it has been assumed that sorting on Multi-Mesh can be done with respect to the indexing scheme described in the algorithm .

Average-case and worst-case behaviour of the greedy algorithm presented can be studied.

Finding an algorithm for permutation routing which runs in optimal time without requirement of extra buffer space can be studied . As the diameter of MM network of size N is $2N^{1/4}$, an optimal algorithm should run in time close to this value .

The permutation routing problem in presence of faulty link / node can be studied . A robust shear-sort algorithm which runs in asymptotically same time as normal shear-sort algorithm has been proposed . Construction of a sorting algorithm on MM network similar to this one can be attempted . Based on this sorting network permutation routing on faulty network can be studied .

pendix

The shear-sort algorithm proposed in [17] has some mistakes . The correct algorithm is as follows :

An operation , Rrotate[w,h,s,r] is defined as follows : the mesh is partitioned into vertical regions of r columns each . Each of these regions is further partitioned into horizontal stripes of h rows . The first such horizontal stripe is given a clockwise cyclic shift by s positions . The next stripe is shifted by an amount w mod r relative to the first stripe . The third stripe is shifted relative to the second by the same amount and so on . A similar operation Crotate can be defined by interchanging horizontal with vertical direction .

Let $k = n^{3/4}$ and $l = n^{1/4}$.

The algorithm :

1. Divide the mesh into l^2 number of blocks of size $k \times k$.
2. Rrotate[k,1,0,n]
3. Rrotate[1,k,0, l]
4. Crotate[-k,1,0,n]
5. Rrotate[1,k,0,l]
6. Sort each of the $k \times k$ blocks in column major order in parallel .
7. Inverse(step 5)
8. Inverse(step 4)
9. Inverse(step 3)
10. Divide the mesh into horizontal l^3 stripes and within each such stripe divide every column into l segments , each consisting of l^2 elements . In odd-numbered stripes , route the l segments in such a way that every segment occupies a position corresponding to its mirror image about the horizontal axis within the l^3 stripes .
11. Crotate[-l^2 , k, k - l^2 ,k]
12. Rrotate[k, l^2 , k, n]
13. If two $k \times k$ blocks are adjacent in row-major snake-like ordering , then sort them into row-major ordering in the following way : if the blocks are at the edge of the mesh then sort the combined $2k \times k$ block ,otherwise, sort the combined $k \times 2k$ block .
14. Sort each of the $k \times k$ blocks in parallel with respect to row-major snake-like ordering .

Reference :

1. F. T. Leighton , Introduction to parallel algorithm and architecture : arrays , trees ,hypercubes . *Morgan Kaufmann Publishers* .

2. Leighton , F. Makedon and Tollis . A 2n - 2 algorithm for routing in an n × n array with constant size queues . *Proc. of Symp. Parallel Algorithms and Architecture ,1989* .

3. F. Makedon and A. Symvonis. An efficient heuristic for permutation packet routing on mesh with low buffer requirements . *IEEE Transaction on Parallel and Distributed Systems, Vol. 4, No. 3 ,March 1995* .

4. D. Das and B.P. Sinha . Multi-Mesh - an efficient topology for parallel processing . *Proc. of International Parallel Processing Symposium , 1995* .

5. D. Das and B.P. Sinha. A new network topology with multiple meshes . *Tech. Rep. no. T/ Rep/E-94/01 ,Indian Statistical Institute ,1994* .

6. M. De , D. Das , M.Ghosh and B.P. Sinha , Efficient sorting al on the multi-mesh topology. *Proc. of the International Conference on High-Performance Computing , 1995*

7. M. Kunde . Routing and sorting on mesh-connected arrays . *VLSI Algorithms and Architectures (AWOC '88) , Lecture Notes Computer Science , No. 319 ,1988* .

8. M. Kunde . Packet routing on grid of processors . *Proc. Optimal Algorithms , International Symp. 1989* .

9. D. Krizanc , S. Rajashekaran and T. Tsantillas . Optimal routing algorithm for mesh-connected processor arrays . *VLSI Algorithms and Architectures (AWOC '88) , Lecture Notes Computer Science ,No. 319 ,1988* .

10. S. Rajashekaran and T. Tsantillas , An optimal packet routing algorithm for mesh . *Proc. of 7-th Conf. of Foundation of Software Technology and Theoretical Computer Science ,Lecture Notes of Computer Science . Vol. 287 . 1987* .
11. C.R. Dyer , A VLSI pyramid machine for hierarchical parallel image . *Proc. IEEE Conference Pattern Recognition Image Processing ,1981* .
12. R. Miller and Q. F. Stout . Data movement techniques for the pyramid computer . *SIAM Journal of Computing , Vol. 16, No.1, February 1987* .
13. D. Nath, S.N. Maheshwari and P. C. Bhatt. Efficient VLSI network for parallel processing based on orthogonal trees . *IEEE Transaction on Computer , Vol. C-32, No. 6 . 1993* .
14. Q.F. Stout . Mesh connected computers with broadcasting . *IEEE Transaction on Computer , Vol. C-32, No. 2 . 1983* .
15. S.H. Bokhari . Finding maximum on an array processor with broadcasting . *IEEE Transaction on Computer ,Vol. C-33,No. 2 ,1984*
16. M. De , B.P. Kundu and B.P. Sinha . Worm-hole routing for complete exchange on Multi-Mesh . *Proc. High Performance Computing , 1997 , to appear* .
17. S. Sen and I.D. Scherson . Parallel sorting in two-dimensional VLSI models of computation . *IEEE Traansaction of Computers Vol. 38 No. 2 ,1989*
18. D.E. Knuth . Sorting and Searching . *Addision-Wesley*

INDIAN STATISTICAL INSTITUTE LIBRARY
C 23797
24 SEP 1997