

Permutation Routing on Mesh model Parallel Computers

Karthik Sivashanmugam
Email: kaart@cs.uga.edu
Dept. of Computer Science
University of Georgia

Submitted to: Dr. Suchendra M. Bhandarkar

ABSTRACT:

This paper studies the problems of permutation routing on different mesh models of parallel computation and simulates algorithms for different mesh models using C. Simple algorithms for permutation routing mesh models with and without bus have been considered, analyzed and simulated. In mesh models with bus, models with fixed bus and reconfigurable buses are dealt. For problems in permutation routing on meshes with bus, a technique has been considered that can be used to develop deterministic algorithms that are better than the simpler algorithms. This paper also analyzes one deterministic algorithm derived using the technique for permutation routing on two dimensional networks and implements this algorithm using C. For permutation routing with reconfigurable bus the computational aspects of reconfigurable mesh model has been analyzed and simulating the problem in a simulator called RMSIM. A hardware for the reconfiguration switch is also suggested and implemented in VHDL. The design of the switch is based on restricted configuration model that involves ten different possible configurations. VHDL test pattern analysis for the switch, simulation results for the above-mentioned models are also provided.

Keywords: Parallel Algorithms, Routing, Sorting, Reconfigurable Mesh, Mesh, Buses, Permutation routing, Reconfigurable bus.

1. MOTIVATION:

Why do we need parallelism when the performance of a single processor has increased dramatically over the last few decades?

The answer lies in the computational demands of contemporary science and engineering. Many of the scientific and engineering problems require enormous computational power. Following are the few fields to mention:

- Quantum chemistry, statistical mechanics, and relativistic physics
- Cosmology and astrophysics
- Computational fluid dynamics and turbulence
- Material design and superconductivity
- Biology, pharmacology, genome sequencing, genetic engineering, protein folding, enzyme activity, and cell modeling
- Medicine, and modeling of human organs and bones
- Global weather and environmental modeling
- Machine Vision

The upper bound for the computing power that can be obtained from a single processor is limited by the fastest processor available at any certain time. On the other hand, the upper bound for the computing power available can be dramatically increased by integrating a set of processors together. In order for all these processors to cooperate on solving problems, synchronization and exchange of partial results among processors are usually unavoidable. As a result, it is necessary to supply some kind of interconnection scheme so that they can communicate with each other. Various kinds of interconnection schemes have been proposed in the papers [4], [7], [10](Refer appendix). In this paper we will deal the simplest and representative of these models namely "Mesh model". Taking this model

we will continue with one of the most common problems in computation and communication, the permutation problem. In multi processor environments performance of parallel computing is heavily influenced by packets transferring between different nodes. Fast and low cost permutation algorithms are important to increase the performance of multiprocessors.

2. INTRODUCTION:

Recent advances in VLSI have made it possible to build massively parallel machines featuring many thousands of cooperating processors. Interprocessor communications and simultaneous memory accesses typically act as bottlenecks in parallel machines, hampering any increase in computational power from translating into increased performance of the same order of magnitude. To overcome the inefficiency of long distance communications among processors, bus systems have been recently added to a number of parallel machines.

In some realistic models of parallel computation the memory is distributed among the processing devices and each processing device can only communicate with a small, fixed set of neighbors in a single step, while several communication steps are necessary in order to send a message between processing devices that are not directly connected to each other by a communication link. We refer to such models of computation as having a fixed topology.

A fixed-connection network consists of a collection of sequential processors connected by a sparse system of communication links. These processors operate in synchronous fashion, and communicate by sending messages over the communication links. In a single step a processor can read a constant number of messages that were sent to it in previous step, perform some fixed amount of internal computation and send a constant number of messages across its communication links to neighboring processors. If such a system has bus that can be dynamically changed, under program control, to suit communication needs among processors, it is referred to as reconfigurable. These multiprocessor systems also arises need for sufficient algorithms that implement a variety of communication patterns between processing devices not directly connected to each other. Such algorithms are commonly referred to as routing algorithms. Another problem that is closely related to this is sorting problem. Various hardware devices for these problems have been analyzed in the papers [2], [3], [5], [6], [9], [11].

A single step of interprocessor communication in a fixed connection network can be thought of as the following task (also called packet routing): Each node in the network has a packet of information to be sent to some other node. The task is to send all the packets to their correct destinations as quickly as possible such that at most one packet passes through any more any time.

A special case of routing problem is called the partial permutation routing. In partial permutation routing, each node is the origin of at most one packet and each node is the destination of no more that one packet. A packet routing algorithm is judged by its run-

time and queue length. It is assumed that the packet not only contains message but also source and destination names.

The organization of remainder of this paper is as follows: In section 3, the problem, Permutation routing is defined and in section 4 the significance of the problem in a multi-processor environment is stated. In section 5, relevant works of other researchers in tackling this problem is mentioned. Section 6 gives the description of the project undertaken, that is the models chosen for analysis, their definitions, Permutation routing algorithms on these models and the simulation of these algorithms in C. Section 7 gives results of the project. In section 8 the conclusions and future directions are given.

3. DEFINITION OF PROBLEM:

The routing problem is the problem of rearranging a set of packets in a network, such that every packet ends up at the processor specified in the destination address. A routing problem in which each processor is the source and destination of at most k packets is called a k - k routing problem. The goal is to rearrange packets such that the packet with the key of rank i is moved to the processor with index i , for all i and in all at least $\theta(n)$ steps [10] are required for this. The routing problem most extensively studied in the literature is the 1 - 1 routing problem, also called permutation routing problem. An algorithm for routing is specified by the path to be taken by each packet and priority scheme.

Here the problem is assumed to be dealing with atomic packet (store and forward model) and these packets can be transmitted along the communication channel in one unit time.

4. SIGNIFICANCE OF THE PROBLEM:

An important feature of parallel computing that is absent in sequential computing is the need for inter-processor communication. For example, given any problem, the processors have to communicate among themselves and agree on the sub-problems each will work on. Also, they need to communicate to see whether every one has finished its task and so on. Hence the speed of any parallel computer is primarily decided by the power of individual processing elements and efficiency of inter processor communication[2]. Since the power of processors has increased it becomes essential to concentrate more on refining the speed of communication. Performance of parallel systems depends on efficiency of communications. A theoretical goal is to solve problems in a constant time interval, independent of problem size, for a large number of problems.

5. LITERATURE REVIEW

A considerable amount of research has been done on Mesh models with respect to Permutation routing. Some papers [11], [13] analyzed algorithms for Permutation routing on Meshes with and without buses. Regarding an interesting work a paper [12] discusses a C simulator named RMSIM for Reconfigurable Meshes. Some papers [1], [2], [7], [9] suggested different Reconfigurable models. My work is work based on the collective idea of all these papers. In this paper I am analyzing different algorithms proposed for different Mesh models suggested and implementing the algorithms using C. For

reconfigurable meshes I am using a simulator called RMSIM. Also I am suggesting a hardware that can be used for building switches to be used in reconfigurable meshes.

6. PROJECT DESCRIPTION

6A. MODELS UNDER CONSIDERATION:

The model of computation assumed in this paper is a mesh with row and column buses. We consider both fixed and reconfigurable buses. All algorithms designed for such a model also run on more powerful machines such as Polymorphic Taurus, the RMESH and PARBS dealt in papers [7], [1], [9] respectively. We will consider the PARBS, one of the supposedly superior architectures for comparison.

This paper deals following architectures:

1. Mesh with fixed row and column buses
2. Reconfigurable Mesh (Figure 1)
3. Mesh without bus

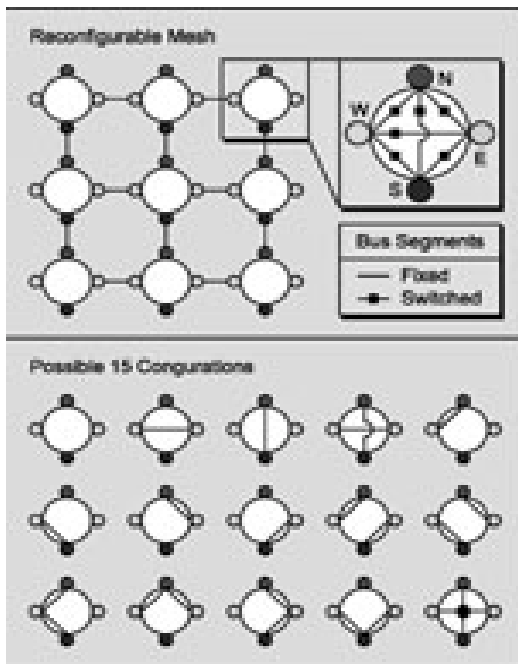


Figure 1 [e]

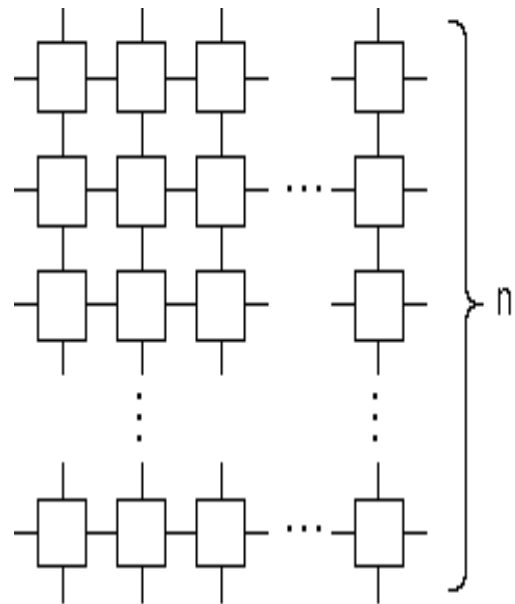


Figure 2 [f]

Fig 2: conventional Mesh architecture with only links and without bus

The combination of low cost and high speed for near-neighbor interactions makes Mesh connections quite attractive for implementation and analysis. We will analyze Mesh model for this reason. Apart from their advantages, the most serious disadvantage [8] is that Meshes do not support global communication and synchronization directly. The

overall speed of a parallel calculation on a mesh-based structure will depend on the proportion of the global operations that have to be performed. If a mesh structure is supported by a second interconnection structure for global operations, the two structures together can provide a computer system that is well suited to a broad class of scientific applications. We consider such structures, the meshes with fixed row and column buses.

The limitations posed by fixed bus model arise the need for a better model and hence the introduction of reconfigurable bus systems, where processors can dynamically take part in forming bus segments, enable the mesh to solve many scientific problems in constant time. Hence we consider this model also in this paper.

The main goal of this paper is to consider the computational aspects of variants of mesh models and the focus will be on the routing problem. The models under consideration are Meshes with and without buses and reconfigurable meshes.

Simpler algorithms for permutation packet routing will be analysed for the models mentioned above. According to [6] in fixed bus designs every permutation can be routed offline in $n+1$ steps. In this paper we will discuss the exact time complexity for doing this. This is achieved by using Off-line routing scheme and graph theory concepts. It is assumed that in reconfigurable models only one processor can write in the bus at a time.

Considering an algorithm that is used for network packet routing, the aim is to find a lower bound and approximate upper bound for the algorithm in these models.

We will start with routing on meshes without buses and then the paper will deal a scheme to derive a deterministic algorithm for meshes with buses. The work will continue with analysis of meshes with fixed buses, analysis and simulation of a simpler algorithm in RMSIM (a simulator called RMSIM that would simulate the algorithm to be running in a Reconfigurable Mesh and would produce images of the interconnection patterns during the execution of the algorithm) for meshes with reconfigurable buses. The ultimate purpose is to compare the time bound of these meshes and to find the validity of claim that meshes with buses show better performance than other conventional meshes without buses. It will also analyze the difference in time complexities of fixed and reconfigurable buses.

6B. DEFINITION OF MODELS:

Hypercube and mesh-based architectures are the most commonly used interconnection networks for parallel computers with distributed memory. Recently, mesh-based architectures have attracted more attention on the parallel computing community due to the following reasons:

- The wiring cost is cheaper and wiring complexity is lower comparing to hypercube.
- The network topology has a close match to the problem regularity (e.g., image processing, matrix multiplication, etc.).

A regular mesh is an $N \times N$ square grid with one processor per grid point. Except for the processors at the boundary, every other processor is connected to its neighbors to the left, right, above and below through bi-directional links. The instruction stream is MIMD (that is each node can send and receive a packet from all its (four or less) neighbors in one unit time). It has a communication diameter (maximum of the minimum distance between any two processors in the network) which equals to $2(N-1)$ (N for a wraparound mesh) and hence even computations require only a very limited amount of computation still require at least $N-1$ steps. As a result, a lower bound for the time complexity for problems that involve comparing or combining data that reside in different processors is $O(N)$. When wrap around connections are present, the mesh is called a Torus. Meshes with buses have also been proposed in which each processor has two bus ports, one vertical and the other horizontal. This type of structure is to augment mesh architecture with high-speed buses that allow communication between processors located in different areas of the mesh. But this requires each processor to manage two buses, and to handle traffic that is to switch from one to the other. Furthermore, the local buses still do not take advantage of locality in communicating with neighbors. Exact time complexity depends heavily on the properties of the bus system. It has been shown that meshes and buses be combined (mesh with row and column broadcast buses) to provide a combination of good local communication and low-diameter communication. The low diameter means that latency is low for sending single values across the network. However, the row and column buses end up being choked when all processors want to send a single value

To improve the time complexity for these problems, researchers have studied special architectures based on a 2 or 3-dimensional mesh called Reconfigurable networks. The basic idea of a reconfigurable network is to enable flexible connection patterns, by allowing nodes to connect their adjacent edges in various patterns. This yields a variety of possible topologies for the network and enables the program to exploit this topological variety in order to speedup the computation. The edges of the network are viewed as building blocks for larger bus components. The network dynamically reconfigures itself at each time step where an allowable configuration is a partition of the network into paths or a set of edge disjoint buses. The crucial point is that the reconfiguration process is carried out locally at each processor (or switch) of the network. That is at the beginning of each step during the execution of the program on the Reconfigurable network, each switch of the network fixes its local configuration by partitioning its collection of edges into any combination of pairs and singletons. Two adjacent edges grouped by a switch into the same pair are viewed as connected and the processor may choose to listen to any of incoming or passing message.

Reconfigurable mesh architectures (RMESH [7], PARBS [1]) set an excellent example for such machines. The Reconfigurable Mesh is a data-parallel architecture that extends traditional processor meshes with a set of reconfigurable buses. A bus is called reconfigurable if it can be partitioned into sub buses such that each sub bus can be used as a separate independent bus. These buses implement a fast interconnection network that decreases the network diameter and allows for fast algorithms that cannot be run on a traditional mesh. The buses are typically implemented via switches associated with the different mesh processors, allowing buses to be subdivided into sub-buses. Proper

configuration of switches and sub-buses allows partitioning the mesh in many different ways, which is useful in recursive, divide-and-conquer, and other kinds of parallel algorithms. Both the models, Meshes with Reconfigurable Buses and Meshes with Buses are popular models of computing because of the absence of diameter consideration[2].

Each processor on an N-dimensional regular mesh will have $2N$ external ports. Neighboring ports in neighbor processor nodes are connected to each other, to form the mesh interconnect skeleton. Inside the node, ports communicate to each other via a network of wires and switches that can be opened and closed to implement different connectivity patterns.

In a two-dimensional reconfigurable mesh, each processor connects to its four neighbors. The four external ports of a node are usually called "North", "South", "East" and "West", to reflect their geographical location. Neighbor ports are connected; for example, a processor's North port connects with the South port of its northern neighbor.

6C. ASSUMPTIONS

In this paper all analyses are done based on the following assumptions

The project is assumed to be dealing with atomic packet (store and forward model), these packets can be transmitted along the communication channel in one unit time and the packet not only contains message but also source and destination names.

6D. MESHES WITHOUT BUSES

For SIMD models $4N-4$ steps is the lower bound (with respect to the connection availability parameter) for almost any problem. This matches with the Simple Greedy algorithm for Permutation Routing. Following is the greedy algorithm that satisfies this bound and this paper implements this algorithm using C.

Algorithm for Permutation routing on mesh without buses[13]:

To understand why we need meshes with buses:

Considering each node is identical and connected to four nearest neighbors with buffer and routing decision circuit, routing is done as follows:

- a. Every node has status, available or disabled.
- b. If a node is full of buffers, it disables its links that receive packets.
- c. If the node is not full and if it can receive packets then it chooses one of its neighbors which can reduce the destination distance upon following condition
 1. if the destination is in same row or column then the node can have only one node which can reduce the destination distance.
 - if the link it chooses is available then it chooses that link
 - else if the link is disabled, do nothing
 2. Else there are two of its neighbors which can reduce the distance
 - if both the links are available then choose one randomly
 - if only one is available choose that link
 - if both are disabled then do nothing.

This algorithm could be modified to deal nodes without buffer also. This algorithm is successfully implemented using C. Refer appendix for source code and results of the simulation.

6E. MESHES WITH BUSES

This paper assumes CREW (Concurrent read-Exclusive read) model and according to this model all the PEs connected to a bus can read the bus concurrently but only one can write data on the bus. This model is taken and the following algorithm and off-line routing computing scheme are implemented using C.

Every Permutation can be routed in in $n+1$ steps after scheduling has been computed.

Analysis of off-line routing scheme and a resulting algorithm:

An off-line routing scheme has been selected that is used to write deterministic routing algorithms. In the offline routing scheme [6], every packet is routed to its destination by routing on a column bus to its destination row, and then routing it on a row bus to its destination column in the following step. Thus the algorithm does not use mesh edges at all. It has been also shown that for any input permutation, a schedule for the above routing scheme can be computed in time $O(n^{7/2})$ by computing a sequence of n maximum matching which runs in time $O(n^{5/2})$. Once the maximum matching has been computed (refer appendix) it can be executed in $n+1$ steps. According to the authors this runs in time $O(n^{7/2})$.

To get the running time $n+O(n)$ the size of the problem should be reduced. This is done by partitioning the mesh into a smaller number of processes and resources (refer appendix for details) and by treating sets of packets with similar sources and destinations as if they were a single packet.

To do this [11] the network is partitioned into blocks B_i , $0 \leq i \leq n^{2-2\alpha}$, of size $n^\alpha \times n^\alpha$ where α is some constant that is smaller, but sufficiently close to 1. We now interpret each of $n^{1-\alpha}$ columns of blocks as a process, and each of $n^{1-\alpha}$ rows of blocks as resource. Each process has an exclusive ownership of the column buses while each resource consists of row buses. At most one process is allowed to access a single resource at a time. Then the packets are arranged inside the processes in such a way that we can make an optimal use of this new configuration. Rather than requiring each packet to be at its final destination after execution of the schedule as dealt in [6] the paper [11] is content with routing each packet to some position in the $n^\alpha \times n^\alpha$ block that contains its destination. After completion of the schedule, it brings the final packets to destination by routing locally inside each block.

To arrange the packets for the schedule, the blocks are sorted into row-major order, where the packets are sorted by their destination blocks. The row of a block is said to be clean if all its packets have the same destination block. Otherwise it is dirty. All the packets in a clean row of a block are transmitted across the row buses to their destination block in a single step, after they have been routed to the correct row of blocks in the preceding steps. If a row of block is dirty, then the packets in the row are transmitted across the row buses in r steps where r is the number of distinct destination blocks that

occur among the packets in the row. This means that row is treated in the same way as r separate rows and this increases the number of steps required to route this row by $r-1$. Since there are only $n^{2-2\alpha}$ blocks, this increases the number of steps required to route this packets of a single block across the row buses by at most $n^{2-2\alpha} - 1$. This results in the required length of the schedule. It has also been proved that this runs in time $O(n)$.

Thus the following algorithm is arrived at:

- (1) Partition mesh into blocks of size $n^\alpha \times n^\alpha$. Sort the packets in each block into row-major order by destination blocks. This takes $O(n^\alpha)$ steps.
- (2) In each block, compute the m_i , $0 \leq i \leq n^{2-2\alpha}$, m_i – number of packets with destination block B_i , Send the m_i to a block of side length $n^{2-2\alpha}$ in the center of the mesh. This takes $O(n^{2-2\alpha})$ steps.
- (3) Compute the schedule and broadcast it to all blocks of the mesh. This takes $((n^{1-\alpha})^{9/2})$ steps.
- (4) Execute the computed schedule of length $n + n^{3-3\alpha}$.
- (5) Perform local routing inside each block to bring the packets to their final destinations. This takes time $O(n^\alpha)$.

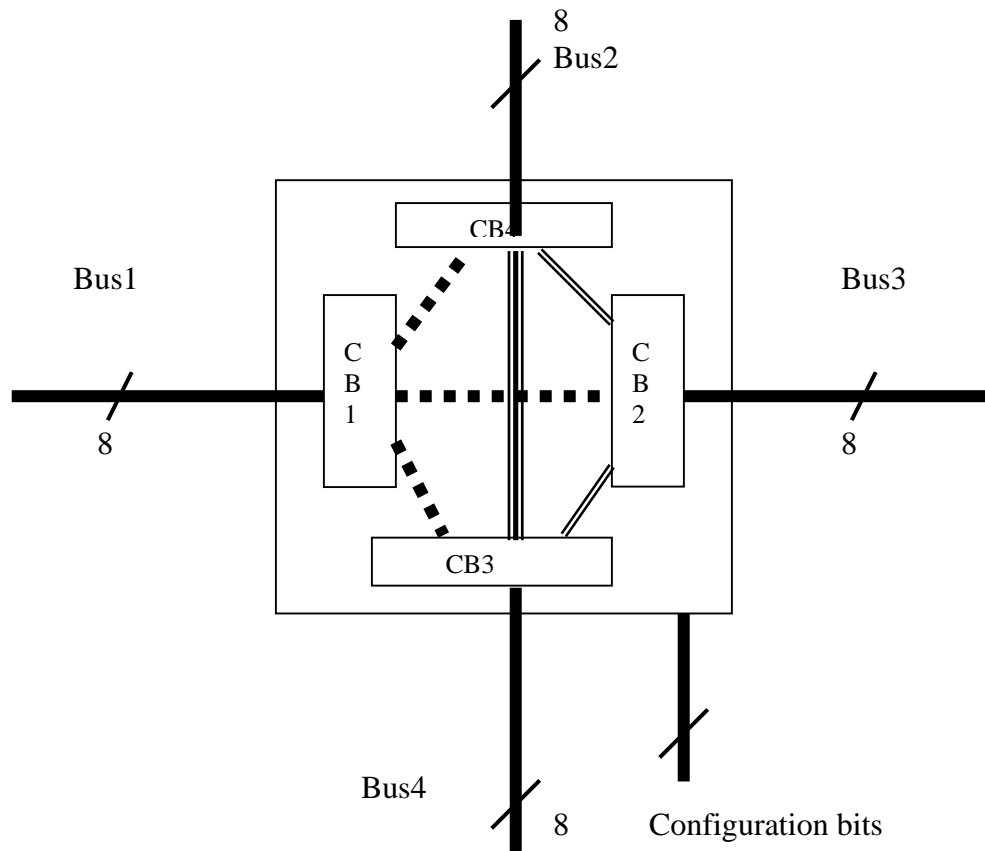
The above algorithm is only analysed and the algorithm given in appendix A is implemented in C.

6F. MESHES WITH RECONFIGURABLE BUSES

Mesh with reconfigurable buses are considered as EREW (Exclusive read and Exclusive write) where only one of the PEs connected to the bus can read the bus and only one of the PEs connected to the bus can write data on the bus. Here all PEs connected to a single bus which can be segmented / disconnected by switches and made to act like several small buses. Trivial bound is n for any problem in this model because the bus can be segmented into a maximum of n disconnected parts. The same bound is applied to Permutation routing also. To simulate Permutation routing this paper uses a simulator called RMSIM [12]. Using this simulator Permutation routing is successfully done. Results of the simulation are enclosed in appendix.

6G. SWITCH DESIGN

The switch is designed with 4 internal blocks named Configuration Blocks made of multiplexers that operate on buses. These blocks are used to establish different configurations among the bus connections. These configurations are made possible by configuration bits that are supplied by the control of the Processing Element in which the switch is to be embedded. These bits are sent to a ‘mux driver’ which outputs ‘select bits’ for the muxes. The schematic below gives possible bus configurations.



7. RESULTS

The Permutation routing algorithms are successfully simulated using C in Mesh without buses and Mesh with row, column buses. The simulator RMSIM is used successfully to do Permutation routing in a reconfigurable mesh environment. The hardware switch implemented in VHDL obeys the test patterns. The results of the simulations and trial run samples are enclosed in appendix.

8. CONCLUSIONS AND FUTURE DIRECTIONS

This paper ignored the problem of queue size in Reconfigurable meshes simulation. This simulation can be improved by considering queue size. Another possible improvement could be with respect to VHDL implementation. The implementation is done using bit vectors instead of buses. This can be improved by using actual buses. Another improvement is to optimize the instruction set for configurations. Since only ten configurations are implemented this can be done with 4 bits instead of 5 bits. The whole simulation could also be done for actual packet routing instead of limiting to Permutation routing. Only RMSIM simulation code needs to be changed because other simulations are already done keeping queue size and general packet routing in mind.

REFERENCES:

- [1] R.Miller, V.K.P.Kumar, D.Reisis, and Q.F.Stout, Parallel Computations on Reconfigurable Meshes, IEEE Transactions on Computers, 42 (1993), 678-692
- [2] S.Rajasekaran, Mesh-connected computers with fixed and reconfigurable buses: Packet routing, sorting and selection. In Proc. 1st Annual European Symposium on algorithms, September 1993, 309-320
- [3] S.Rajasekaran and T.McKendall, Permutation Routing and sorting on the reconfigurable mesh, Technical Report MS-CIS, Dept of Computer and Information Sciences, Univ. of Penn, May1992
- [4] Y.Ben-Asher, D.Peleg, R.Ramaswami, A.Schuster, The power of reconfiguration, 18th International colloquium on Automata, Languages and Programming, July 1991, Madrid.
- [5] G.F. Lev and N.Pippenger and L.G.Valinat, A Fast Parallel Algorithm for Routing in Permutation Networks, IEEE Trans. Comput. 30 (1981) 93-100
- [6] J.Y.Leung and S.M.Shende, On multidimensional packet routing for meshes with buses, J. Parallel and Distrib. Systems (1990)
- [7] H.Li and Q.Stout, Reconfigurable massively parallel Computers, Prentice-Hall, 1991
- [8] H. Stone, High performance computer Architecture
- [9] B.Wang and G.Chen, Constant time algorithms for the transitive closure and some related graph problems on processor arrays with reconfigurable bus systems, IEEE Trans. Parallel. and Distrib. Systems. 1 (1990)
- [10] J. Sibeyn, Solving Fundamental Problems on Sparse-Meshes, IEEE Transactions on Parallel & Distributed Systems. 11(12): 1324-1332, 2000 Dec
- [11] T.Suel, Permutation routing and sorting on meshes with row and column buses, 8th International symposium on parallel computing
- [12] M.Manzur Murzed, R.P.Brent, RMSIM: a Serial Simulator for Reconfigurable Mesh Parallel Computers, Technical report TR-CS-9706, Joint Computer Science Tech. Report Series, The Australian National University, April 1997.
- [13] Y.W.Lu, James Burr, Allen M. Peterson, Permutation on the Mesh with Reconfigurable Bus: Algorithms and Practical considerations, (1993)

Reference websites:

- [a] <http://bwrc.eecs.berkeley.edu/Classes/CS252/Notes/67>
- [b] http://www.krellinst.org/UCES/archive/classes/ASC/tableofcontents2_1.html
- [c] <http://www-unix.mcs.anl.gov/dbpp/text/node1.html>
- [d] <http://www.mv.com/org/rmesh/Tutorial.html>
- [e] www.rsise.anu.edu.au/annrep/1998/researchesl.html
- [f] www.iti.fh-flensburg.de/lang/papers/trans/transcl.htm
- [g] www.cs.umass.edu/~weems/CmpSci635/635lecture14.html

APPENDIX:

A. Off-line routing technique in [6]

The columns of the mesh are interpreted as *processes* $p_0, p_1, p_2 \dots p_{n-1}$. Every process p_i has exclusive ownership of its column bus, and has to route the n packets initially located in its column. To do so, a process needs to use the row buses, which are interpreted as *resources* $r_0, r_1, r_2 \dots r_{n-1}$. Before each packet can be transmitted on a row bus to its final destination, it has to be routed within its column to the correct row; this can be done in the preceding step using the column bus. If k packets in the column i gave a destination row j , then the process p_i needs to access resource r_j for k time steps. These k steps can be scheduled in any arbitrary order, provided that in any step, each resource is accessed by at most one process, and each process uses at most one resource. The problem of finding a minimum time schedule that satisfies these demands is known as *Open shop scheduling problem*[13].

A simple algorithm for finding a minimum time schedule computes a sequence of maximum matchings in the bipartite graph $G = (U, V, E)$ defined by $U = \{ p_0, p_1 \dots p_{n-1} \}$, $V = \{ r_0, r_1 \dots r_{n-1} \}$, and $E = \{ (p_i, r_j) \mid D_{ij} > 0 \}$ and D_{ij} is the demand of process p_i for resource r_j for $0 \leq i, j < n$ and this gives the number of packets in column i that have a destination row j . This algorithm first computes a maximum matching M of G , and schedules each process with its matched resource for D_{min} time steps, where $D_{min} = \min \{ D_{ij} \mid (p_i, r_j) \in M \}$. Next D_{min} is subtracted from all D_{ij} with $(p_i, r_j) \in M$, a new bipartite graph G' is constructed corresponding to the new values of D_{ij} , and a new matching M' is computed. This procedure is repeated until all D_{ij} are equal to zero. At most n matchings can be computed, since every matching increases the length of the schedule by at least one.

B. VHDL Implementation:

Tool used: asimut- Alliance tool.

This tool has some limitations like limited functionality, needs others Alliance tools for advance design (example: Bus design needs 'buseg' tool).

Design:

Instead of buses this implementation has used bit-vectors and showed how connections will be established.

| Configuration: | Instruction set: |
|----------------|------------------|
| 1-2: | 00110 |
| 1-3: | 01110 |
| 1-4: | 10110 |
| 3-2: | 11000 |
| 3-4: | 11100 |
| 2-4: | 11111 |
| (1-3)+(2-4): | 01111 |
| (1-2)+(3-4): | 00100 |
| (1-4)+(2-3): | 10000 |
| No connection | 11110 |

