CHALMERS

A Scalable Interconnection Architecture for Future Many-Core Systems

Master of Science Thesis in the Programme Networks and Distributed Systems

Pawan Acharya

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Göteborg, Sweden, June 2012

Supervisor:

Professor Per Stenstrom

CHALMERS UNIVERSITY OF TECHNOLOGY

Abstract

In this thesis, a scalable interconnection architecture for many-core systems is proposed and the tradeoffs in the design of this architecture are described. Using this model we investigate how the other aspects of network architecture like routing strategy, routing algorithms, buffer space and traffic pattern impact the performance of interconnection network. This report also discusses the simulation environment and the trace-driven simulation approach used for the simulation. The traces are obtained from NAS parallel benchmark using the MPICH application.

The simulation is performed for two network models 2-d MESH and CMESH. These models are then subjected to two different approaches of traffic injection: inject as fast as you can, and follow casual order for the injection. The performance of these network models is then compared for the worst case scenario: inject as fast as you can, network backpressure maintains the flow of traffic into network. The variable message size, buffer space and routing algorithm are the variables for which the network performance is measured. This study shows that the choice of network architecture depends upon the priority of the network design. Had it been just achieving lower average message latency regardless of area occupied, power consumed and requirement of buffer space, MESH is a better choice. On the other hand, for the on-chip interconnection architecture where space, power constraints and bufferspace plays a major role; CMESH presents a better choice with the expense of little more router computation complexity and greater average message latency.

Acknowledgement

First and foremost, I would like to offer my sincere gratitude to my supervisor Professor Per Stenstrom, who has constantly supported and guided me in completing my thesis work. Without his insights and encouragements I would not have been able to complete my work. One simply could not wish for a friendlier and a better supervisor.

I would also like to thank my family who supported me all this time both financially and morally in completing my work and with this Masters Degree from Chalmers. With this I also like to mention my friend Mr. Mod Nath whose moral support has been of great help in completing my work.

Table of Contents

1. Introduction

Multicore architectures promise to track the projections of Moore's Law by a doubling of the processor-core count every eighteen months. This roadmap will make it possible to build chips with a core count in the hundreds in the next 5-10 years. While one of the advantages of the multicore paradigm shift is that one can potentially double the performance by doubling the core count, the memory system infrastructure must scale from a performance and power perspective. With this paradigm shift arise numerous opportunities and challenges at both hardware and software level. With the promise of delivering scalable performance by scaling the number of cores, the major challenge will be delivering computation performance that scales linearly with number of cores within the constraints of underlying technology within acceptable power budget[32].Another challenge is understanding the specific hardware and software abstraction that can enhance the performance of parallel software development and then finding the appropriate implementation approach to realize it. The multicore architectures will have heterogeneous computing cores; the reason is mainly that the homogeneous architectures will most likely deliver scalable performance only with embarrassingly parallel systems [32]. The major challenge for this heterogeneous multicore architecture will be to offer scalable programming model and the hardware support to realize it. Thus with the shift towards scalable parallel programming and hardware efficient communication becomes the center for high performance. Use of dedicated ad-hoc wires to interconnect cores on the chip is not a feasible solution for the multi-core era because they are costly in terms of area and delay and inefficient to realize parallel communication. This has opened door towards on-chip networks, where communication between the cores are multiplexed on shared wires[30]. The on-chip network (interconnect) is able to provide the scalable solution for the multi-core architectures.

1.1. Aims and Objectives

The aim of this thesis work is to set up the requirements to accommodate a scalable interconnect architecture for the multi-core systems, address implementation issues and build a model of it. And access the performance in context of standard benchmark NAS [12] using a set of parallel application, MPICH application [11] suite is used in this work. The objective here is to model two network architectures using 2-dimensinal mesh and concentrated Mesh technology, use the traces obtained from the NAS benchmark using MPICH for the performance evaluation of these developed models. These network architecture models are

designed for evaluation under two routing algorithms deterministic and adaptive. In addition to algorithms the effect of network traffic and the impact of design parameters are also studied. Based on these evaluation criteria a scalable interconnect architecture is proposed.

1.2. Thesis Outline

This report is organised as follows: in the second section of this report there is an overview of the related works in the interconnect network architecture and the challenges within the design of interconnection network and the areas where most research has been carried out. The description of the architectural model used in this thesis work is presented in the third section. In this section the detailed network topologies, the wormhole switching strategy, buffer management techniques and the routing algorithms are discussed in detail. 2-D Mesh and CMesh are the two topologies studied here, and deterministic and adaptive routing algorithms are adapted for the wormhole switching scheme in the network architecture. Followed by the detailed picture of implementation architecture in the section 4, the router architectural setup used for the implementation of wormhole switching is explained here. And the experimental setup for the development of realistic simulation is presented in section 5; this section presents the overview of the traffic model used for communication in the interconnection network model thus developed. NAS parallel benchmarks are used for obtaining the traffic traces using MPICH. Section 6 presents the simulation results obtained using the traffic obtained from a real parallel application. The results presented in three categories: impact of algorithm, impact of message size and the impact of buffer space, Discussion and study of the results obtained from the simulation are studied in section 7.Section 8 presents the conclusion of this research work. This is followed by a brief overview for the further work in this area in the section 9and finally with section 10 for References.

2. Related Works

With the necessity of high performance processor architecture, researchers have been putting efforts on featuring multiple processing cores on a single chip. For a given number of cores, the best interconnect architecture in the multiprocessing environment depends on factors like power/area budget, bandwidth requirements, technology and performance objectives[1,2,12]. Choice of topology, switching strategy, flow-control mechanism and routing algorithms are such in which the system architect can play on in the design of interconnection network

architecture for many-core systems. For relatively fewer numbers of cores buses, rings, and crossbar provide a reliable solution which are utilized in IBM Cell [31] and Sun's Niagara [7]. For future many-core architectures topologies like meshes and tori provide the scalability needed [30].On-chip networks consume a considerable portion of the total on-chip power [6, 2]; up to ~30% for Intel's 80 core teraflops network [5,3] and 36% for the RAW con-chip network [4,7]. In the paper published by Intel it is suggested that the design support of on-chip networks consume almost 25% area of each tile in the many-core design [5]. Each tile contains a processing core with primary caches, a slice of L2 cache, and a connection to the on-chip network [29].

There have been several studies and evaluation in the design of power efficient, scalable interconnection network architecture with reduced area. This section presents a brief overview of research work done in the design of scalable interconnection architecture.

Mutsunori Igarashi et. al. in [31] have designed a diagonal-interconnect which is characterized by pervasive use of diagonal wiring. Their diagonal-interconnect achieves 20% path delay reduction and 10% area reduction.

J. Balfour and William J, Dally in paper [17] have proposed a concentrated mesh architecture which offers a improvement by 24% in area and efficiency and improvement by 48% in energy efficiency over other architectures like Mesh, Tree and Torus evaluated in their study.

A Multi-commodity flow (MCF) is proposed on paper [25] to find the throughput for multiple different routing architectures. It explained "the throughput is limited by the capacity of middle row and column in the mesh and a flexible chip shape provides around 30% throughput improvement over a square chip of equal area."[25]

Study by Magnus Ekman and Per Stenstrom in their paper [26] explores the trade-offs between issue width of cores and the number of cores with respect to performance and energy. They have found that with the increase in number of cores, the reduced power consumption in the cores is overwhelmed by the increased power consumption in the on-chip memory system.

A dragonfly topology [27] proposed by William J Dally et.all. a group of high-radix router is used as virtual router in order to increase the effective radix of the network. The dragonfly reduces the cost by 20% in comparison to flattened butterfly and by 52% in comparison to flooded Clos network with configuration > 16K nodes.

Jingcao Hu and Radu Marculescu in their paper [28] propose a deadlock-free deterministic routing algorithm which minimized the total communication for solving problems with mapping and path allocation routing in regular tile-based NoC architectures. They "formulate

the problem of energy/performance aware mapping, in a topological sense, and show how the routing flexibility can be exploited to expand the solution space and improve the solution quality."[28]

As the tori and mesh promise to give the scalability needed for the future multi-core systems, this study is to take the CMESH and 2-D Mesh network topologies and study the impact of routing algorithms, buffers space and packet size in the network in their overall performance. This study takes the measure of average network latency over the throughput using the worst case scenario i.e. the network is congested with traffic. The only control mechanism is the network backpressure which puts a limit to the traffic that can be injected into the interconnection network underneath.

3. Model Description

This section is used to describe and analyze components of the network design. The focus will be on the theoretical aspects and develops the notation that will be used in the subsequent development and the analysis of the model. The following components of the network design are the ones:

- Topology
- Routing Algorithm
- Switching Strategy
- Flow Control Mechanism

3.1. Topology

The topology determines the physical layout of the connections between the communicating nodes in the network. The proposed model has been designed using both 2 Dimensional Mesh and Concentrated Mesh topologies.

In a mesh network, the nodes are arranged in a k dimensional lattice of width w, giving a total of w^k nodes. Here k = 2 for the 2 dimensional array. Communication is allowed only between the neighbouring nodes. All the interior nodes are connected to 2k other nodes.

3.1.1. A8x8 2-Dimensional Mesh:

The 2D mesh Topology[17] is an example of a direct network or point-to-point network. It consists of a set of nodes, each node being directly connected to a subset of other nodes in the network. In a 2-Dimensional Mesh network, nodes at the boundaries are connected to immediate 3 neighbouring nodes whereas the ones in the middle are connected to 4 neighbouring nodes. The nodes represent the cores. Each node has a local router associated with it where the router handles the message communication among the nodes. The topology used in this model has a bidirectional channel connection neighbouring nodes. Each router supports a number of input and output channels. Internal Channels connect the local processor to the router. External Channels are used for communication between the routers. By connecting the internal channel to the output channel of the other nodes the direct network is defined. Figure 1 presents the graphical representation of the 2-Dimensional Mesh.

3.1.2. A Concentrated Mesh Topology:

CMESH topology[17] provides a network with a more compact overview of nodes thus reducing overall network space. CMESH takes the benefit of improving network performance by reducing maximum distance across the network, and increasing the buffer size. The large router services more than one node and thus it reduces the number of routers in the network. CMESH comes with the benefit of reducing effective distance within the network, as with each jump on router the packet moves two nodes close to the destination. CMESH has a compact layout with a larger router servicing multiple nodes, it has reduced wire length and overall reduced network space as well as wider channel width.

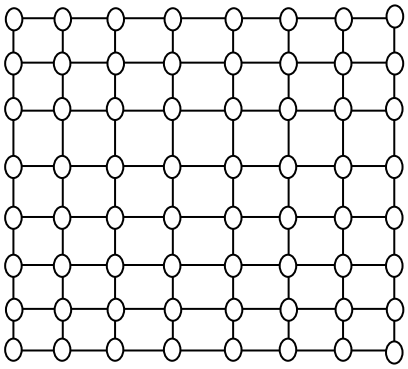

Figure 1: 2D Mesh Architecture

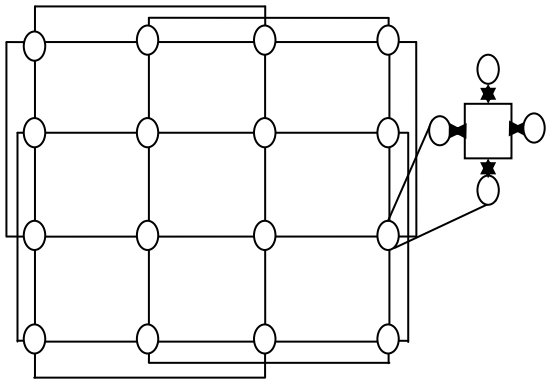

Figure 2: CMesh Architecture

The CMESH architecture as shown in Figure 2 comes with each router having four local connections with neighbour nodes. Thus the neighbours can communicate through a single router which reduces the communication latency. However it comes with the price of increased computation latency for the router with each router having to perform the routing and arbitration for the packets from all the four local cores

3.2. Routing Algorithm

The network architecture is experimented using both deterministic routing and Adaptive Routing algorithm.

Deterministic routing algorithms always choose the same path between *x* and *y* even if there are multiple available for the routing. These algorithms ignore the path diversity of the underlying topology and thus are inefficient at load balancing. However due to the fact of easy implementation and easy to make deadlock-free network they are quite common in practice. Dimension-order routing is one example of deterministic algorithms which is studied here. This routing algorithm routes packets by following strict increasing or decreasing order while crossing the dimension in the XY network, reducing the offset to zero before starting routing in the another offset of other dimension.

Deterministic (Dimension Order) Routing in a 2 D Mesh:

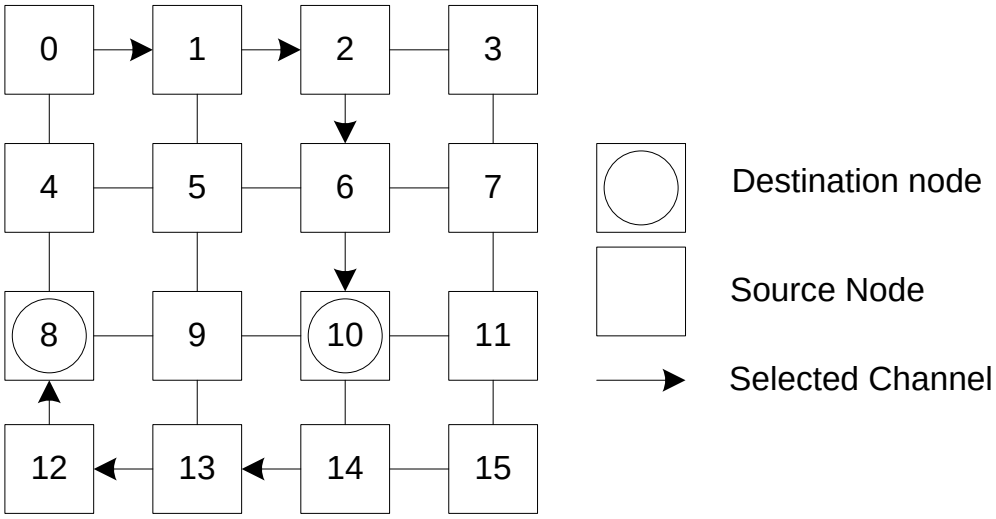

Figure 3: Dimension Order routing algorithm in 2-D Mesh [8]

In this algorithm, the packets are first routed in the X-direction and then in the Y-direction. Figure 3 depicts the dimension order routing. The packets source node 0 routes packets in the X-direction towards destination node 10. The packet first traverses in the X-direction till the x-offset reaches to zero, then from node 2, the packet is routed in the Y-direction till it reaches destination node 10.

Algorithm: XY Routing for 2D Meshes [8]

Inputs: Coordinates of current node (Xcurrent, Ycurrent) and destination node (Xdest, Ydest)

Output: Selected output Channel

Procedure:

Xoffset = Xdest - Xcurrent
Yoffset = Ydest – Ycurrent
If *Xoffset < 0* ***then***
    Channel = X-
Endif
If *Xoffset > 0* ***then***
    Channel = X+
Endif
If *Xoffset = 0* ***and*** *Yoffset < 0* ***then***
    Channel = Y-
Endif
If *Xoffset = 0* ***and*** *Yoffset > 0* ***then***
    Channel = Y+
Endif
If *Xoffset = 0* ***and*** *Yoffset = 0* ***then***
    Channel = Internal (the local node)
Endif

In the psuedocode above the xoffset and yoffset are calculated at each node as the packet traverses through the network. The path selection for the packet is made first in the X-direction till xoffset reaches zero and the packet traverses in the Y-direction till yoffset reaches zero. When both the xoffset and yoffset are, the packet has reached the destination.

Duato's Fully Adaptive Routing Algorithmin a 2-D Mesh[8]

For the Adaptive routing algorithm, Duato's fully adaptive routing algorithm [8] is used in this thesis work. As shown in the psuedocode below each physical channel is split into two virtual channels *a* and *b*. Xa+ and Xb+ denote the virtual channels a and b respectively in the positive direction of X dimension. Similarly, other Xa- and Xb- for the negative direction of X dimension. Similar notations are used for the other Y dimension.

Inputs*: Coordinates of current Node (Xcurrent, Ycurrent) and destination node (Xdest, Ydest)*
Output*: Selected Output Channel*
Procedure*:*
    Xoffest = Xdest – Xcurrent
    Yoffset = Ydest – Ycurrent

```
If Xoffset < 0 and Yoffset < 0 then
        Channel = Select (Xa-, Ya-, Xb-)
endif
if Xoffset < 0 and Yoffset > 0 then
        Channel = Select(Xa-,Ya+, Xb-)
endif
if Xoffset < 0 and Yoffset == 0 then
        Channel = Select (Xa-, Xb-)
endif
if Xoffset > 0 and Yoffset < 0 then
        Channel = Select(Xa+, Ya-, Xb+)
endif
if Xoffset > 0 and Yoffset > 0 then
        Channel = Select (Xa+, Ya+, Xb+)
endif
if Xoffset > 0 and Yoffset == 0  then
        Channel = Select(Xa+, Xb+)
endif
if Xoffset == 0 and Yoffset < 0 then
        Channel = Select (Ya-, Yb-)
endif
if Xoffset == 0 and Yoffset > 0 then
        Channel = Select ( Ya+, Yb+)
endif
if Xoffset == 0 and Yoffset == 0 then
        Channel = Internal
endif
```

In the pseudocode above for the adaptive routing algorithm, the xoffset and yoffset are calculated at each node as the packet traverses through the network. If xoffest and yoffset are not equal to zero, the choice for path selection is first looked into the two virtual channels in X-direction. If these virtual channels are not free, then the selection is made towards the Y-direction. When the xoffset is equal to zero, the path selection is made towards the virtual channels in Y-direction. When yoffset is equal to zero, the path selection is made towards the virtual channels in X-direction.

3.3. Switching Strategy

Wormhole switching strategy[16] is used in the proposed model development. In wormwhole switching, a packet is divided into flits, *flit* is the unit of message flow control [8]. The head flit contains the routing information, Body flits contain the data to be transferred and the tail flit to notify the end of the worm transfer. This switching technique helps to achieve low latency as the routers forward the header of the packets as soon as it is received without waiting for the tail packet to arrive. Packet flits are pipelined through the network in

wormhole switching. This technique is extensively used in high-performance parallel computer networks [28].

3.4. Flow Control Mechanism

In this network architecture both bandwidth and the buffers are allocated in units of flits and thus the *Wormhole Flow Control* [9] mechanism is used. When the head flit of the packet arrives at a node it acquires three resources a channel for the packet, one flit buffer and one flit of channel bandwidth before it can be forwarded to the next node along the route. Body flits of the packet use the channel already acquired by the head flit and so it needs to only acquire the flit buffer and the flit of the channel bandwidth to advance. The tail flit is handled just like the body flit; also it releases the channel as it traverses through the network.

The channel holds the state needed to co-ordinate the handling of the flits of a packet over a channel. This state identifies the output channel of the current node for the next hop of the route and the state of the channel. The state of channel can be in three states idle, waiting or active.

3.5. Buffer Management

The network architecture uses ack/nack based flow control for the buffer management and backpressure. With this kind of flow control, there is no recordkeeping of the individual buffer state of the upstream or downstream router. The router in the upstream sends flit optimistically to the downstream router. If the bufferspace is available in the downstream router, it accepts the flit and send an acknowledge (ack) to the upstream node. If the buffer is full, the flit is dropped and a negative acknowledgement (*nack*) is sent to the upstream router. The upstream router holds each flit until an *ack* is received from the downstream router. For the flit who received *nack* , the router retransmits the flit.

4. Model Development

4.1. Router Architecture

The router functionality is divided into two sections: datapath plane and the control plane. The Datapath plane handles the storage and movement of the data flits and it consists of the input buffers, switch and output buffers. The control buffer controls the flow of the flits through the components of the datapath plane, it also performs route computation, channel allocation and switch allocation. The Figure 4 is the block diagram of router architecture for wormhole router architecture:

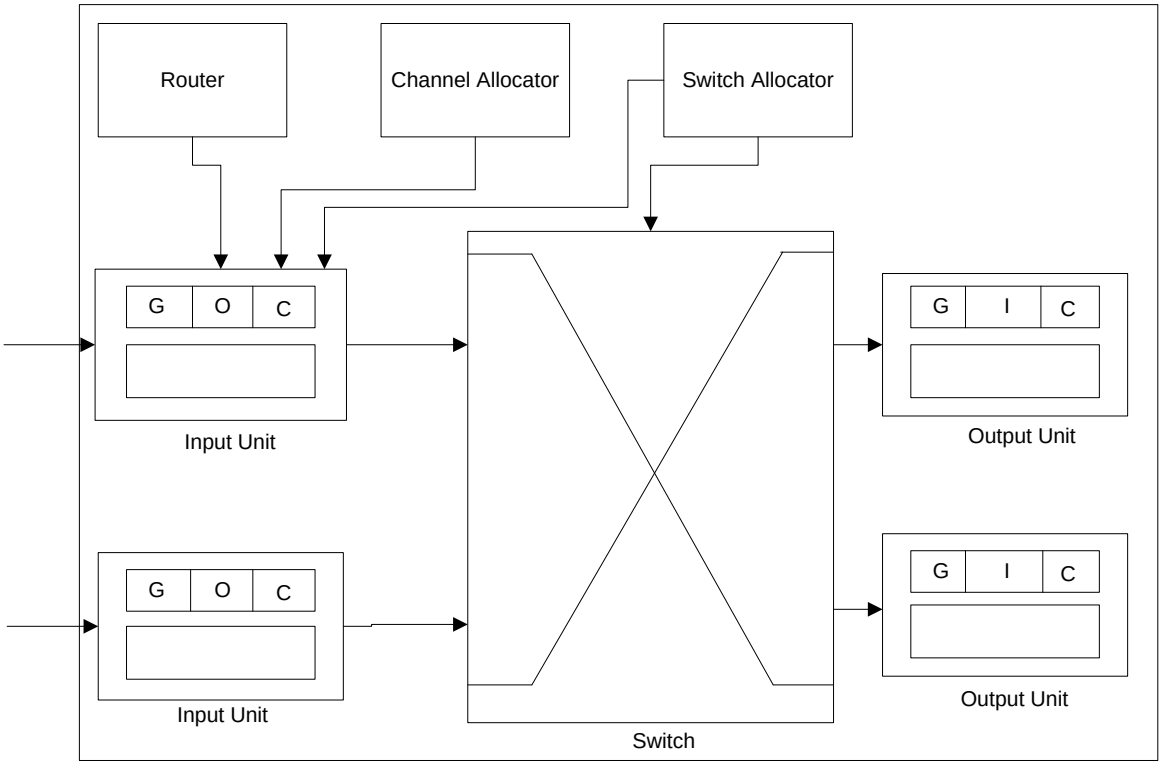

Figure 4: Router Architecture[9]

The flits arrive in the input unit of the router and are stored in the buffer allocated in the input units. The input unit also contains the information about state of the output channel allocated for it, there are three states fields allocated to track state of the output channel allocated for the particular input channel. The three states are: Global State, Output Channel, and Credit Count.

Before advancing the flit, first the route computation is performed to determine the port for the routing. After the output port is determined, the flit then requests an output channel from the channel allocator. Once the route is determined and the channel is allocated, the flit is then forwarded over this output channel using the switch allocator. At last, the output unit

forwards the flit to the next router in the destination path. Route computation is done for every head-flit received which corresponds to every packet also the channel allocation is done only for the head flit. The reserved channel is released after the tail flit is routed from the router.

4.2. The Router Pipeline

The Gantt chart demonstrating the pipelining of the typical wormhole router channel is shown in Figure5.

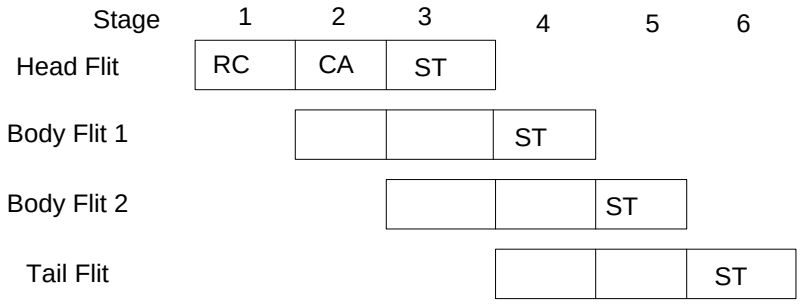

Figure 5: Pipelining routing of a packet[9]

The router pipeline begins upon the arrival of the head flit into the input unit of the router. Upon the arrival of the headflit into the particular input unit of the router, the state of the input channel for that particular unit is changed to occupied, i.e. 'active' from previously 'idle' state. In the next stage, the head flit undergoes the router computation and then Channel Allocation. The other stages are Switch Allocation and Traversal. The route computation is performed using the information stored in the headflit. After the route is calculated, the headflit now goes through output channel allocation stage. If the channel is unoccupied and is ideal then only the head flit is allocated the channel. Otherwise, it is stalled and waits for the next cycle till the channel is empty and ideal. When the channel is allocated to the head flit, the flit traverses the switch and is stored in the buffers allocated for the output channel, i.e. output unit. Whereas, for the body flit and tailflit, they just follow the route already created by the headflit in a worm like structure until they encounter a stall, which is the case when the channel buffer is not empty. When the tailflit leaves the input unit, on the way out the channel state is set to ideal.

4.3. Router Datapath Components

The Router Datapath components include input buffer, switch and output unit. Input buffers hold the flits till the output channel; switch bandwidth and channel bandwidth are available. Input buffers in this architecture are partitioned across the physical channels. For Each buffer partition, storage is dynamically allocated using lists from the C++ Standard Template Library [10]. Crosspoint switches are implemented for this architecture. Crosspoint switches provide the excess bandwidth or speedup to simplify the allocation problem [9].

4.4. Buffer Organization

Buffers are used by the flow control protocol for the allocation of space to store the flits awaiting the channel bandwidth to depart the routers.

4.4.1. Buffer Data Structures

A data structure is used to keep track of where the flits and packets are located in the memory and manage memory. Lists from the C++ Standard Template Library are used here for the store of flits and packets. Lists are implemented as doubly-linked lists. Doubly linked lists can store each of the elements they contain in different and unrelated storage locations. The ordering is kept by the association to each element of a link to the element preceding it and link to the element following it.

The advantages of these structures are[10]:

- Efficient insertion and removal of elements irrelevant of the location in the list
- Efficient moving elements and blocks within the list or between different list

4.5. Switches

Crossbar switches are used in this study. The nxm cross bar switch can connect any of the n inputs to each of the m outputs. Speedup is the major advantage and the purpose of using cross bar switching technique used in this project.

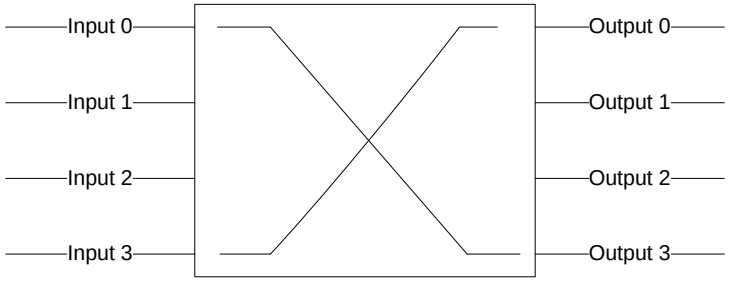

Figure 6: Symbol for 4x4 Crossbar Switch[9]

5. Development of Realistic System Simulation

In order to evaluate the performance of the interconnection network architecture it is important that the simulation is performed using realistic data samples. The aim here is to design the interconnection network architecture for the multi-core systems thus the traffic patterns of a real parallel computer system is necessary to draw the realistic conclusion about the performance evaluation of the proposed architecture model. For the realistic simulation of the proposed model, here the traces obtained from the actual parallel application are used. The target parallel application here MPICH [11] is run in a computer and the traces thus obtained from the application running are feed into the simulation of the network model developed.

5.1. Traces from MPI Applications

MPICH[11] is a standard programming interface for parallel applications whose process communicate using message passing. The profiling mechanism provided by the MPI standard allows users to intercept all calls to the MPI functions. The traces of applications are generated using this profiling mechanism as defined in the MPI standard.

5.2. Generating Trace file

The traces are obtained using the MPE. MPE is a collection of libraries and tools, which is included in the MPICH distribution package that is used to generate and analyze the traces of any parallel applications. The profiling mechanism defined in the MPICH standard that allows intercepting all calls to MPI functions is used to generate the traces of parallel applications. The parallel application used here are the standard NSA parallel benchmark.

- **Why NAS Benchmark?**

  The CG Kernel of NAS parallel benchmark solves an unstructured sparse linear system by the conjugate gradient method. CG benchmark uses the inverse power method to find the approximate of the largest Eigen value of a symmetric positive definite sparse matrix with a random pattern of non-zero[12]. The NPB suite can be compiled and run for various problems sizes which are defined by the several classes: S, W, A, B, C, D of which S is the smallest and D is the largest. Larger problems use larger data structures and thus resulting in heavy message size for the node to inject into the network.

Using trace-enabled MPICH application a trace file is created which consists of a set of time-stamped records describing the behaviour of the application during its execution. The trace record consists of two records for each MPI function invoked by the application process, one

when the process calls the function and the other when the function returns. Only the point-to-point operations are taken for the record of message interchanges. Two record entries are created for each message generated, one when it is created and the other when it is received. The basic message structure in the trace-records looks like this:

- Identifier
- Timestamp
- Record Type (send or receive)
- Identifier (which consist of either destination or the source)
- Message Tag
- Message Size

Analysis of Application Traces: The data trace obtained from the application consists of long and short messages:

- 4 are long 14 KB messages
- 3 are short 8 B messages

Visualization of the trace of Conjugate Gradient Class A with 64 processes (CG.A.64) shows that it consists of series of iterations, each of them with the following phases [14]:

1. An Entry of messages in the network: Each node sends and receives 7 messages. The first 4 being the very long messages each 14 KB and the remaining 3 being short messages of size 8B. The first two long messages traverse to the nearby neighbour nodes and the remaining two to the nodes at a longer distance. As with the short messages, the first one travels to the distant node and the remaining two travel to the nearby nodes.
2. A minimum duration of computation stages.
3. Another phase involves interchange of very short messages.
4. A longer computational stage.

Conjugate gradient shows long communication-synchronization chains [14]. A message sending event is executed only after the reception of another message by the node. The chains are of varying lengths 7 in phase 1 and 3 in phase 3 [14].

At the same time, the network simulation never has a very large number of messages travelling simultaneously. For the CG suite, any delay in injecting message into the network results in further delay in the injection of messages that are related to it.

5.3. Using Traces to Feed Simulation

In order to provide a real communication workload to the interconnection network simulator, the following assumptions are made:

- The trace file should contain the exactly same number of communicating process as the number of nodes simulated in the interconnection network. In this work, the processes the trace and the nodes of the interconnect network are arranged in the consecutive order. The process n goes to the node n of the simulated interconnected network.
- Interconnection network simulator deal with the exchange of packets. The Application generates messages of variable size and now it is the job of simulator to split those messages into packets of certain sizes and interject into the network. The size of the packets is determined by the type of networking technology used in the network simulator. For this interconnection network we are considering the fixed size packets.

5.3.1. Approach one: Inject as Fast as you Can

This is an unrefined and crude approach to feed network simulation with the events taken from the trace file.

- All the states recorded in the trace events are ignored except for the MS (Message Sent).
- The trace events from the trace file are injected into the list created for each node which is arranged with the timestamp order. Each node is then subjected to inject messages from its list into the network as fast as network can accept them. Timestamp is used to impose an order while the injecting packets into the network. In this case only the Network backpressure modulates the injection of message into the network.

This approach is suitable for the measure of network performance. The network is stressed with the packets and thus making it the bottleneck for the network. It can then be measured how fast the network can deal with the given workload.

5.3.2. Approach two: Follow Casual Order

The first approach which injected the message packets into the network as fast it can but it did not take care of the casual relationships between the messages. In real parallel application the process stalls while waiting for the message to be delivered. The application resumes the execution only after the message arrives.

- The trace events in the trace file contains the state records, MS(Message Sent), MR(Message Receive), timestamp, destination process address and the source process address. We use MS and MR for the order of message interchange in the network, and the timestamp is used to impose the order of message injection into the network.
- The event traces from the trace file is fed into the node record list for each node #n. The record list (an "event queue") consists of <n timestamp MS destination size> and <n timestamp MR origin size> records.
- In each node, a receipt list is created to hold the message delivered to it by the network.
- At each node, the following course of action takes place:
- If the first message in the record list is a Message Sent, it is removed from the list and is injected into the network.
- If it is a Message Receive, first check if a corresponding message with the matching origin, destination, size and tag is in the receipt list of the node. If the message is there, remove both entries else nothing is done.
- When the network delivers a message to the node, it is stored in the receipt list.

The purpose of MR record is to get the simulator network to work like a real parallel application which stalls until the corresponding message is actually received from the network. The process is depicted in the Figure 7: In the Figure, the node 7 cannot advance because it is waiting for the message from the node 1 even if the message from the node 1 is already received. Whereas the node 1 can advance further with sending message to node 2from its send event queue because the message from node 7 has already been delivered to it. This process of communication simulates the actual process of message communication in the real application which complies with the casual relation between the receipt of message and the subsequent message send triggered by it.

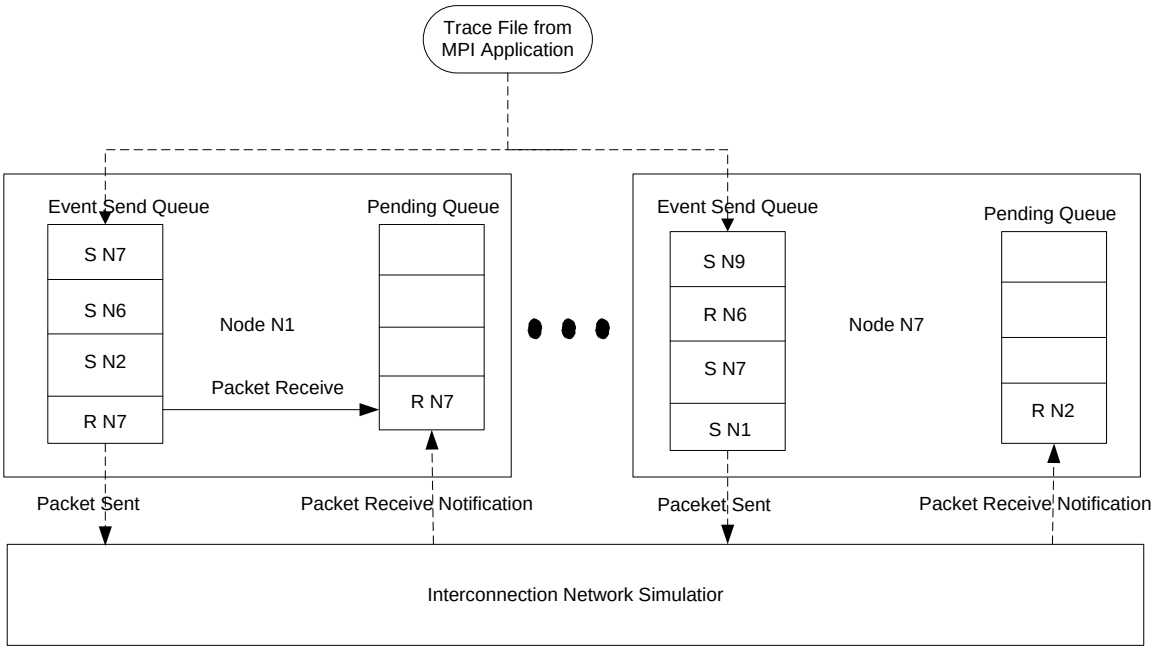

Figure 7: Casual Order flow Control Data Structure[14]

5.4. Interconnect Network Simulator:

The implementation of the network simulator was done in C++ since the object oriented prototypes mapped very well in defining different modules of the simulator. The interconnect networks consisting of both CMESH and MESH were implemented using the two dimensional array structure in C++. The simulate class creates a mesh of two dimensional array for both switches and nodes(here local core are represented using the nodes):

```
class simulate{
        int xpos,ypos,Npos,Wpos,Epos,Spos; // N,W,E,S represent North, West, East
        int count,pos, k, j, neighbor;          // South Neighbour Switches
public:
        simulate(node* [],switches* []); };
simulate::simulate(node *node_p[],switches *switch_p[])
```

The class called ***simulate*** then maps the local nodes with its corresponding router in the network and similarly defines the neighbouring router for every router in the network depending upon the topology used. The ports *switch-to-switch, node-to-switch, switch-to-node* are created using C++objects which are the C++ implementation for the physical channels of the real interconnect architecture. The Class definitions for the ports are as follows:

```
class switch_port{
  switches *owner; // in this code switches corresponds to the router
```

```
 public:
        switch_port(switches *own);};
class node_to_switch_port{
switches *owner;
 public:        node_to_switch_port(switches *own);};
class switch_to_node_port{
 node *owner;
public:switch_to_node_port(node *own);};
```

Each node and switches has their individual implementation of ports classes.

For the objects of type **switch_to_node_port** which is created for every router, owner is the local node which is used by the local router (here switch) to inject the packets received from the network.

For objects of type **node_to_switch_port**every node creates objects of this type with owner pointing to its local router which then it uses to send the packets into the network for routing to the designated node in the network.

For objects of type **switch_port**, every router creates objects of this type with owner of type router pointing to the neighbouring router in the desired position which then it uses to route the packets within the network.

The main function ***sim_start()*** starts the simulation; creates the mesh, initialize nodes, routers and switches; parses the trace file obtained from MPI Application. The trace file is split into timestamp-order lists, one list per simulated node.

The input and output buffers in the routers are implemented using lists in the C++ Library. The list takes two template parameters in their implementation in the C++ Standard Template [10]:

Template<*class* T, *class* Allocator = allocator<T>>*class* list;

Where the template parameters have flowing meaning:

T: Type of the element

Allocator: Type of the allocator object used to define the storage allocation model. By default, the allocator class template for type T is used, which defines the simplest memory allocation model and is value-independent.

Inserting to the list is performed using push_back()function, removal using pop_front() function. The C++ code snippet for the input data structure looks like following:

```
if(buffer_queue.empty() && Output_Channel->Status == -1){
```

```
msg_waiting_queue.pop_front();
Headflit = new flit_pkt();
Tailflit = new flit_pkt();
Headflit->next = Tailflit;
Tailflit->next = NULL;
buffer_queue.push_back(Headflit);
while(i < (msg_size / flitSize )){
        Bodyflit = new flit_pkt();
        Bodyflit->type = 0;
        if(i == 1){
                Bodyflit->next = Headflit->next;
                Headflit->next = Bodyflit;
                Previousflit = Bodyflit;
        }else{
                Bodyflit->next = Previousflit->next;
                Previousflit->next = Bodyflit;
                Previousflit = Bodyflit;
        }buffer_queue.push_back(Bodyflit);
        i++;
}buffer_queue.push_back(Tailflit);
```

The simulator runs in two stages:

1. The update phase:

Node: The packets received from the trace file which are queued in the Event queue are serviced in the FIFO order. If the queued message first in priority is a Send Message it is sent for packtization which divides the packet into flits. The flits are then queued in the output buffer queue for injection into the local router. Similarly, if the input buffer queue if it contains any packet flits received from the local router, the router checks if the flit arrived to the right destination and removes the flit from the input buffer and store in its local buffer till the tailFlit is received. Upon the receipt of tailFlit the Message Received event is logged into the Event Reception List.

If the Message queued in the Event Send queue contains Message Receive the node then checks if the desired message is already there in the Reception list. If the message is already received then the Messages from both the Reception List and the Event Send queue both are removed and it increments the message received log by 1.

The flits are of three types:

HeadFlits: contains the routing information such as address of destination node, message size, timestamp and the sender identification.

BodyFlits: Contains the data information to be carried to the destination

TailFlits: Contains the end of packet notification.

Router (Switch): The routerfirst checks if its output buffer has any packet flits queued in it. If there are any flits stored into the output buffer it then checks if it is the headflit, bodyflit or the tailflit.

***In case of a head flit*:** it is sent for channel selection which does the routing calculation and the arbitration of the flit to the selected channel. If the channel in the desired direction is available the flit is then pushed into the buffer of output channel in the desired direction and the status is set to 1. The buffer status 1 indicates that the channel is occupied and any other flits trying to access that channel are stalled. The buffer status -1 indicates that it channel associated to it is free.

***For the case of a bodyflit**:* only arbitration is performed since the channel is already reserved for this packet. If the buffer in the output channel in the desired direction is available it is pushed into that output buffer.

***For the case of the tailflit*:** it is pushed into the output buffer if the buffer in that desired location is free no routing is required as with bodyflit the channel is already reserved for this packet flit. Once the flit is pushed to output buffer, he lock on the input channel is freed the idea here is to set the status of the input buffer into -1.

***Code Snippet*:**

```
If (Input_Channel->Status == 1]){ //checking if the channel is active
If(flit_buffer.front() == 1) // checking if headFlit
Channel_Selection(flit_buffer.front(); // send the flit for routing and arbitration
else if(flit_buffer.front() == 0) //check for bodyFlit
Output_buffer.push_back(flit_buffer.front() // perform arbitration
else if(flit_buffer.front() == -a) //check for tailFlit
Output_buffer.push_back(flit_buffer.front() // perform arbitration
Input_Channel->Status = -1 // Release the input channel free
```

2. The communicate phase:

Node: In the communicate phase of the node, the node ejects the packet flits queued in the buffer of output channel into the local router for routing into the destination node. First the node checks if its output buffer contains any packet flits, it then ejects the flits from its output channel's buffer into input buffer of its local switch through one of its node_to_switch port.

Router: In the communicate phase of the router, the packet flits queued up in the buffer of the output channel are then ejected into the corresponding neighbouring routerof each output channel that is connected to neighbouring router through the switch_to_switch_port. The switch_to_switch_port upon the receipt of flit for delivery into the neighbouring switch of the respective direction pushes the flit into the buffer of the input channel of the respective neighbour switch.

The simulator runs the simulation until all the input and output buffers of both nodes and switch in the network are empty, i.e. all the message are delivered to the destination nodes.

6. Analysis of Performance Measures

The Performance for the proposed model is measured on the basis of:

- Impact of Algorithm
- Impact of Buffer Size
- Impact of Message Size

6.1. Impact of Algorithm:

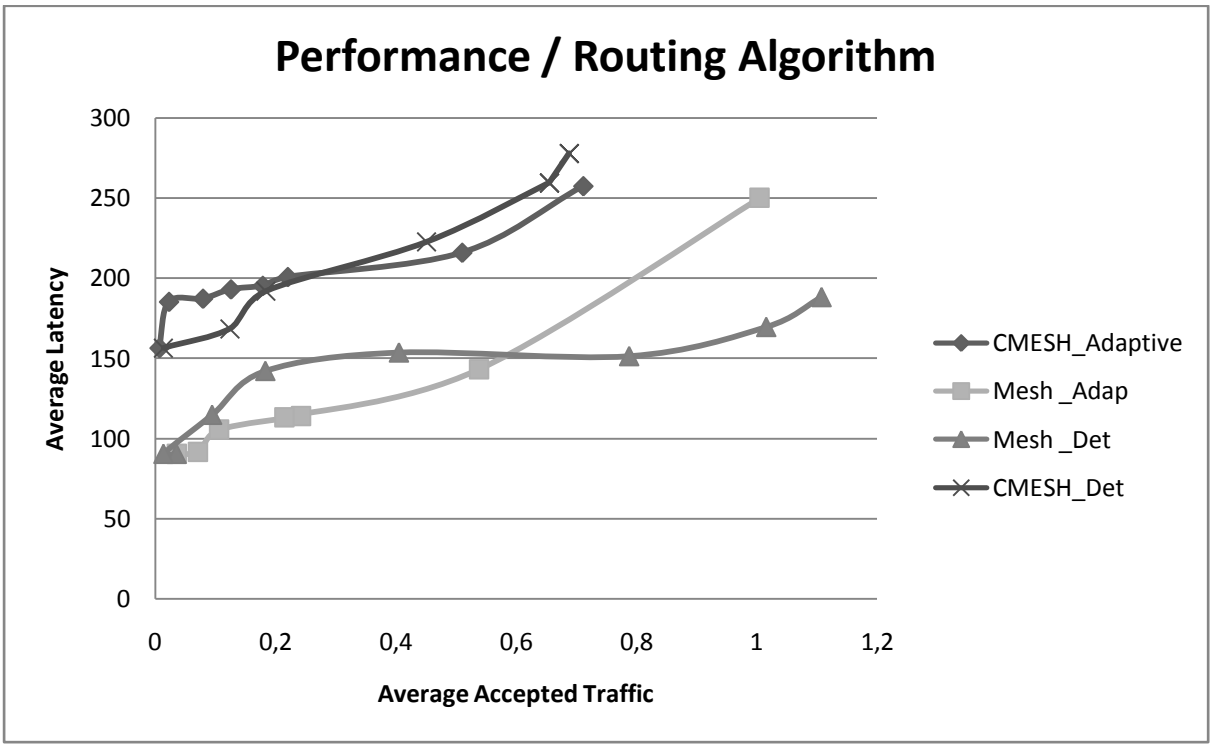

Figure8: Performance of Different network topology under deterministic and adaptive routing algorithms

The simulation was carried out for the 2-D Mesh topology 8x8 matrixes and 8x8 Concentrated Mesh matrixes both for deterministic and adaptive routing algorithm. From the above graph, the performance of Adaptive routing algorithm for the 2-dimensional deterministic algorithm appears to exceed the deterministic routing algorithm till the accepted traffic per node was 0.6 flits per cycle. On the other hand, for the Concentrated Mesh network, the deterministic routing algorithm appears to perform better than the adaptive algorithm up until the accepted traffic was 0.2 flits per cycle. But then the average latency for the deterministic routing steeply rises for the deterministic routing, whereas it remains considerably constant for the wider range of average accepted traffic with adaptive routing. From above, to achieve the best performance with the controlled the average accepted traffic the adaptive routing is the better solution.

The adaptive routing algorithm in 2-D Mesh gives better performance in comparison to the CMesh; trade-off is with area and buffer-space. The 8x8 matrix in 2-D Mesh has a total of 64 routers which cover the area 4 times that in an 8x8 CMesh which has only 16 routers in the network. With the increase in number of routers the demand for the buffer-space simultaneously increases. Thus CMESH in our interconnection network, provides a huge advantage in terms of area and minimizing buffer-space in comparison to 2-dimensional Mesh topology.

1.2 Impact of Buffer Space:

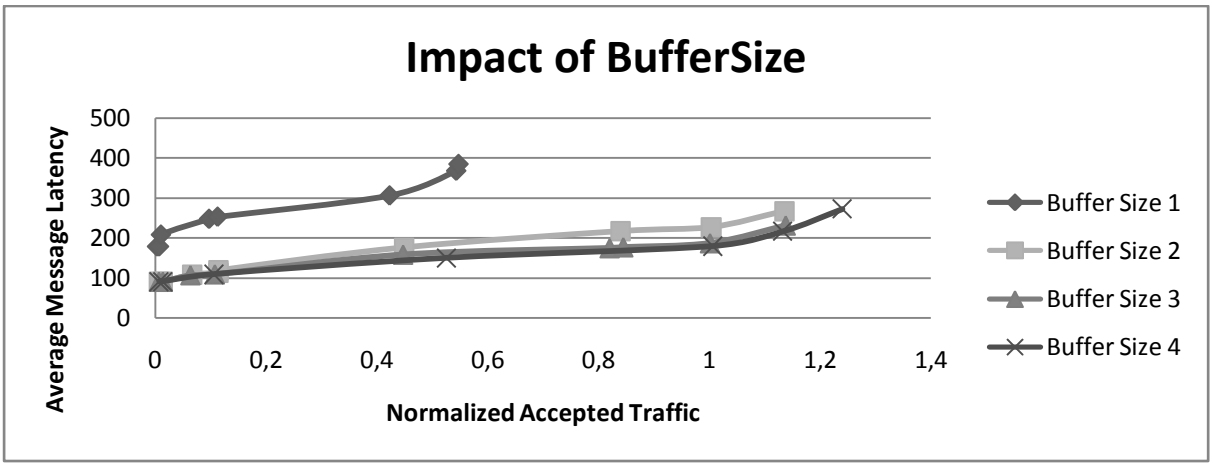

Figure 9: Effect of Buffersize. Plot shows the average message latency versus normalized accepted traffic for the fully adaptive Routing Algorithm

In this graph, as expected, the average message latency decreases with the increase in buffer size. However, the impact of buffersize on performance is small when the buffersize is kept at 2, 3 and 4. These higher buffersize though show significant change in message latency as compared to that when it is 1. The buffersize is designed to maintain equal distribution for both input and output buffer space.

Increasing buffer capacity does not increase performance significantly if message packets are longer than the diameter of the network times the total buffer capacity of the virtual channel. The reason is that the blocked packet keeps all the channels it previously reserved regardless of the buffersize [8].

1.3 Impact of Message Size:

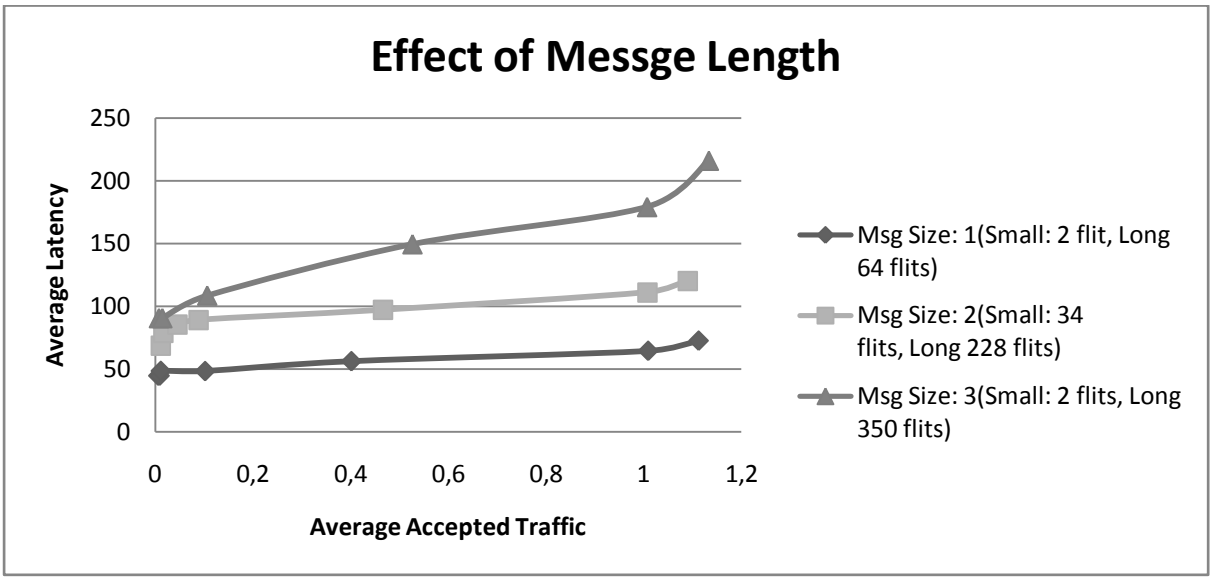

Figure 10: Effect of Message Length, the plot shows the average accepted Traffic versus normalized accepted traffic for fully adaptive routing algorithm in 2-dimensional Mesh network

It was observed that the average message latency is smaller for the traffic with longer message size than the smaller ones. It is seen that the fully adaptive routing algorithm performs comparatively better for the longer messages. The reason is because that the messages are pipelined. Data flits can advance faster than the header flits because headers have to undergo routing computation by the route allocator, wait for the output channel to be allocated , and also be stalled in case of an occupied channel. Thus when the header reaches the destination node, the data flits can transfer faster which in turn favours the longer messages[9].

7. Discussion and Analysis

This section of the document presents the discussion and results of the above performance analysis. The focus here is to contrast the performance of the networks architecture for different algorithms and provide the insight into the merit of each. Also the effect of design patterns and the type of traffic on the network is also discussed.

The effect of Routing Algorithm on regular topologies with uniform traffic is rather small [11] since this study involves the irregular traffic pattern obtained from the real parallel application the effects are quite distinct. Figure 8 presents the relative performance model for the CMESH and 2D Mesh network topology for adaptive and deterministic routing algorithm. The adaptive routing algorithm considerably improves performance over the deterministic routing algorithm for both CMESH and 2-D Mesh topologies. However it can be noted that adaptive routing in above Figure.8 does not reduce latency as compared to the deterministic routing when the traffic is low to moderate. This is because the traffic contention is low and the average latency is little better for the deterministic algorithm. The reason for this is that both the algorithms opt for the minimal paths, the slight increase in latency for the adaptive routing can be explained for its routing complexity. The choice of routing algorithm in real parallel applications is determined by the application requirements. Thus if the traffic produced by the application does not saturate the network then the choice of adaptive routing algorithm over deterministic algorithm does not produce such increase in performance. This is explained when the network traffic follows casual order. In this approach, the network traffic is maintained by the strict order of message receive and send i.e. message send in the event queue is blocked until the receive request that is in top of the event queue is not fulfilled as shown in the Figure 7.

Another limiting factor for the performance is the injection limitation mechanism [9]. As it can be observed in the above graph in Figure 8 for the measure of algorithmic performance measure, the average latency for the adaptive routing algorithm for both the topologies degrades rapidly when the injection of traffic or the average throughput reaches a limit. This saturation limit in the applied load should be carefully watched and maintained to achieve the best performance out of the adaptive algorithm for both topologies. This can be achieved by following some kind of injection control mechanism which does not allow the injection of traffic in the network beyond the saturation limit.

The effect of buffer size for wormhole switching does not show such significant changes after a certain threshold buffer value as seen from the above graph in Figure 10. Especially with

short messages this is even more evident [8, 9]. The trace file used for the traffic injection in the above simulation however has heterogeneous traffic with varying message sizes the details are already mentioned in above section 5.2. In the case of very long messages, the increase in buffer size can show effect only when the buffers are deep enough to allow the message packets leave the source node and release the occupied channels. The distribution of the packet source-destination distribution also affects the occupancy of the buffer-space, the localised packets, i.e. the ones near to the source node, are prevented from leaving the source node before reaching the destination even while deep buffers are used. Thus it can be noted that in most of the cases while using the wormhole switching strategy, small buffers are enough to achieve the good performance.

The adaptive routing algorithm in 2-D Mesh network topology gives better performance in comparison to the CMesh network topology; trade-off is with area and buffer-space. The 8x8 matrix in 2-D Mesh has a total of 64 routers which cover the area 4 times that in an 8x8 CMesh network which has only 16 routers in the network. With the increase in number of routers the demand for the buffer-space simultaneously increases. With the implementation of CMESH network topology in our interconnection network, provides a huge advantage in terms of area and minimizing buffer-space in comparison to 2-dimensional Mesh topology.

8. Conclusion

This project work has been aimed at developing a scalable interconnection architecture for the future multi-core systems. Two network topologies have been chosen for this study, and are evaluated for their performance with two routing algorithms: deterministic and adaptive routing algorithms. The wormhole switching strategy is chosen for the interconnection networks due to its advantage over other switching schemes like circuit switching, packet switching which require more dedicated resource allocation. The interconnection network model is simulated using the traffic taken from the traces obtained from a real parallel application MPICH.

The study showed that the 2-dimensional Mesh topology performed better in achieving lower average message latency compared to CMESH network, in both deterministic and adaptive routing algorithms. While for both topologies, the average message latency was lower for the adaptive routing algorithm than the deterministic routing. But also it is observed that for the adaptive routing algorithm, the average message latency was consistent till the average throughput of the network stayed below a certain point as shown in Figure 8. Thus it was concluded that for the best results, the average traffic injected into the network should not be beyond that critical level.

In this observation, however there is a trade off between the choices of lowering average latency over the area, power and memory requirement put forth by the two different topologies. With the 2-dimensional Mesh, the area of the network increases, so with power and the buffer space required by each router associated with each node in the network. Whereas with the CMESH with the compromise in little over average message latency and more computation complexity for the router, there is advantage of area occupied, power and the memory requirement.

9. Future Work

The study here is concerned with the study of two 8x8 matrixes, 2d Mesh and Concentrated – Mesh topologies. There many more topologies which can be studied and compared to come to an optimum choice for the selection of topology. The performance of any interconnect network is greatly determined by the routing and switching schemes used in it. The two algorithms: deterministic and adaptive can be re-designed and updated to produce even more efficient scheme. The network performance evaluation in this study are made studying the effect of message size in the network, length of buffer space and the impact of routing

algorithm. There are other factors like effect of virtual channels, deadlock handling technique, hardware and software support for collective communication that can be considered to get an even more detail picture of the performance study. The power-consumption, area occupied by the interconnect is not measured in this study for the evaluation of performance. These are few areas where this study leaves space for the future work.

10. References

1. Rakesh Kumar, Victor Zyuban, Dean M. Tullsen "Interconnections in Multi-core Architectures: Understanding Mechanisms, Overheads and Scaling" in*Proceedings. 32nd International Symposium on Computer Architecture, 2005. ISCA '05Pages 408 - 419*

2. H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitecture in on-chip networks," in *MICRO 36 Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, Page 105,.2003*

3. S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS," in *Proceedings of the IEEE International Solid State Circuit Conference, 2008..Volume: 43 , Issue: 1 Page(s): 29-41*

4. M. B. Taylor, W. Lee, S. Amarasinghe, and A. Agarwal, "Scalar operand networks: On-chip interconnect for ILP in partitioned architectures," in *Proceedings of the International Symposium on High Performance Computer Architecture, February 2003. Page 341-353.*

5. Jim Held, Jerry Bautista, Sean Koehl,*"From a few cores to many: A Terascale computing research overview,"* 2006.Intel

6. S. Borkar, "Networks for multi-core chips: A contrarian view," *Special Session at International Symposium on Low Power Electronic Devices (ISLPED) 2007*, 2007.

7. Kongetira, P. Sun MicroSystems Inc., Sunnyvale, CA, USA ,Aingaran, K. ; Olukotun, K., "Niagara: A 32-way multithreaded Sparc processor," *JournalIEEE Micro, vol. 25, no. 2*, pp. 21–29, 2005.

8. José Duato, Sudhakar Yalamanchili, Lionel Ni *"Interconnection: an engineering approach"* 2003, Morgan Kaufmann Publishers Inc.

9. William james Dally, Brian Towles *"Principles and Practices of Interconnection Networks"* Morgan Kaufmann Publishers 2004

10. DOI: www.cplusplus.com/reference/

11. Gropp, W., Lusk, E., Doss, N., Skjellum, A. "A high-performance, portable implementation of the MPI Message-Passing Interface standard."in*Journal Parallel Computing archive Volume 22 Issue 6, Sept. 1996 Pages 789 – 828.* Elsevier Science Publishers B. V. Amsterdam

12. D. Bailey, E. Barszcz, J. Barton, D. Browning, R. Carter, L. Dagum, R. Fatoohi, S. Fineberg, P. Frederickson, T. Lasinski,R. Schreiber, H. Simon, V. Venkatakrishnan and S. Weeratunga. "The NAS Parallel Benchmarks", *Publisher: {NASA} Ames Research Center - Volume: 5, Pages: 63, Issue: March 1995.*

13. F.J. Ridruejo, J. Navaridas, J. Miguel-Alonso, Cruz Izu"Realistic Evaluation of Interconnection Network Performance at High Loads"*Proceedings of the Eighth International Conference on Parallel and Distributed Computing, Applications and Technologies, Pages 97-104. IEEE Computer Society Washington,* DC, USA. 2007

14. Javier Navaridas Palma *"Performance Evaluation of Interconnection Networks using Simulation: Tools and Case Studies"*, Universidad del Pais Vasco, Euskal Herriko Unibertsitatea September, 2009 PhD Dissertation.

15. Mohammad Ali Jabraeil Jamali, Ahmad khademzadeh, "MinRoot and CMesh: Interconnection Architectures for Network-on-Chip Systems": *World Academy of Science, Engineering and Technology 54 2009*

16. C. Seitz et al. Wormhole Chip Project Report. Winter, 1985

17. James Balfour, William J. Dally Computer Systems Laboratory Stanford University "Design Tradeoffs for Tiled CMP On-Chip Networks"*Proceeding ICS '06 Proceedings of the 20th annual international conference on Supercomputing. Pages 187 - 198,* 2006

18. R. Mullins, A. West, and S. Moore. "Low-latency virtual-channel routers for on-chip networks. In *ISCA '04: Proceedings of the 31st annual international symposium on Computer architecture, page 188-197*, Washington, DC, USA, 2004.IEEE Computer Society.

19. H. Wang, L.-S. Peh, and S. Malik. "Power-driven design of router microarchitectures in on-chip networks". In MICRO 36: *Proceedings of the 36th annual IEEE/ACM International Symposium on Microarchitecture, page 105-116,* Washington, DC, USA, 2003. IEEE Computer Society.

20. A. Radulescu, J. Dielissen, S. G. Pestana, O. P. Gangwal, E. Rijpkema, P.Wielage, and K. Goossens, "An efficient on-chip network interface offering guaranteed services, shared-memory abstraction, and flexible network configuration," in *ProceedingsDesign, Automation and Test in Europe Conference and Exhibition, 2004. Page 878- 883.Vol 2.*

21. Pestana, S.G.; Rijpkema, E.; Radulescu, A.; Goossens, K.; Gangwal, O.P., "Cost performance trade-offs in Networks on Chip: A simulation-based approach." In *Proceedings:. Design, Automation and Test in Europe Conference and Exhibition, 2004. Page 764-769 Vol.2.*

22. Goossens, K.; Dielissen, J.; Gangwal, O.P.; Pestana, S.G.; Radulescu, A.; Rijpkema, E.,"A design flow for application-specific Networks on Chip with guaranteed performance to accelerate SOC design and verification." In *Proceedings: Design, Automation and Test in Europe Conference and Exhibition Munich, 2005 Page(s): 1182 - 1187.*

23. J. Miguel-Alonso · J. Navaridas · F. J. Ridruejo "Interconnection network simulation using traces of MPI applications" Published in *Journal International Journal of Parallel Programming,Volume 37 Issue 2, April 2009 Pages 153-174*

24. Ridruejo, F.J., Miguel-Alonso, J. "INSEE: an interconnection network simulation and evaluation environment" Published in *International Journal Of Parallel Programming. Volume 37Page 153-174, 2009*

25. Hongyu Chen, Bo Yao, Feng Zhou, and Chung-Kuan Cheng "Physical Planning Of On-Chip Interconnect Architectures" In *Proceedings: 2002 IEEE International Conference on Digital Object Identifier: Page(s): 30- 35, 2002.*

26. Magnus Ekman and Per Stenstrom "Performance and Power Impact of Issue-width in Chip-Multiprocessor Cores" In *Proceedings: 2003 IEEE International Conference on Parallel Processing. Page: 359 – 368, 2003*

27. W. Dally, C. Seitz, "Deadlock-free Message Routing in Multiprocessor Interconnection Networks", in *Proceedings :IEEE Transactions on Computers, Vol.36 Issue 5. Page 547-553, 1987.*

28. Jingcao Hu and Radu Marculescu "Exploiting the Routing Flexibility for Energy/Performance Aware Mapping of Regular NoC Architectures" *Proceedings of the conference on Design, Automation and Test in Europe Conference and Exhibition - Volume 1..Pages: 688-693.IEEE Computer Society 2003*

29. Antonio Flores, Juan L. Aragón, Manuel E. Acacio "Efficient message management in tiled CMP architectures using a heterogeneous interconnection network" In *Proceeding: HiPC'07 Proceedings of the 14th international conference on High performance computingPages 133-146*.Springer-Verlag Berlin, Heidelberg 2007

30. Natalie D. Enright Jerger "*Chip Multiprocessor Coherence And Interconnect System Design*" A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy (Electrical Engineering).University Of Wisconsin-Madison 2008

31. Igarashi Mutsunori (Toshiba corp.), Mitsuhashi Takashi (toshiba corp.), Le a (Artile microsystems, inc., California), Kazi s (artile microsystems, inc., california),Lin y-t (simplex solutions, inc., california),Fujimura a (simplex solutions, inc., california),Teig s (simplex solutions, inc., california) "A Diagonal Interconnect Architecture and Its Application to RISC Core Design" In *Proceedings:2002 IEEE International Solid-State Circuits Conference, Vol.102, Page.19-23 (2002)*

32. European Network of Excellence on High Performance and Embedded Architecture and Compilation*Multicore architecture roadmap*Submitted By: Per Stenstrom on Sun, 08/06/2008,DOI: http://www.hipeac.net/content/multicore-architecture-roadmap