

# **BOOKSIM: ADDING SUPPORT FOR MULTI-LAYERED NETWORK-ON-CHIPS**

## **Report**

Submitted in Partial Fulfilment of the Requirements for the

Degree of

BACHELOR OF TECHNOLOGY in

COMPUTER ENGINEERING

by

**Bishal Thingom (12CO98)**

**Shubham Raizada (12CO90)**



COMPUTER SCIENCE AND ENGINEERING  
NATIONAL INSTITUTE OF TECHNOLOGY KARNATAKA  
SURATHKAL, MANGALORE -575025

18<sup>th</sup> April, 2016

# C E R T I F I C A T E

This is to *certify* that the B.Tech. Project Work Report entitled “**BOOKSIM:  
ADDING SUPPORT FOR MULTI-LAYERED NETWORK-ON-CHIPS**” submitted by:

(1) Bishal Thingom (12CO98)

(2) Shubham Raizada (12CO90)

as the record of the work carried out by him/her/them, is *accepted as the B.Tech  
Project Work Report submission* in partial fulfilment of the requirements for the award  
of degree of **Bachelor of Technology** in **Computer Science & Engineering** from  
**National Institute of Technology Karnataka, Surathkal** for the academic year 2015-  
16.

Guide

Examiner

H.O.D.

**Basavaraj Talawar**

**K. Vinay Kumar**

**Vani M.**

## CONTENTS

INTRODUCTION.....	1
LITERATURE SURVEY .....	3
2.1 Performance of Network-on-Chips .....	3
2.2 BookSim: A Cycle-accurate NoC Simulator .....	5
2.3 Topology: k-ary n-cube .....	6
2.4 Routing Algorithm: Dimension Order Routing.....	9
PROBLEM DEFINITION AND OBJECTIVES.....	10
3.1 Problem Definition.....	10
3.2 Problem Description .....	10
3.3 Objectives .....	11
METHODOLOGY .....	12
4.1 Simulation flow.....	12
4.2 Important Source Code Changes .....	17
RESULTS AND ANALYSIS.....	20
Comparison of BookSim2 and Modified BookSim .....	22
Comparison of Results on Modified BookSim for Various Configurations .....	27
CONCLUSION AND FUTURE WORK.....	30
REFERENCES .....	31

## INTRODUCTION

An interconnection network is a programmable system which transports data between nodes. A programmable system network is able to make different connections at different points in time. The network is a system because it is composed of many components such as channels, switches, and controls that work together to deliver data.

Network-on-Chips (NoCs) is an interconnection network configured on an integrated circuit (chip). The wires in the links of the NoC are shared by many signals which allows a high level of parallelism because all links in the NoC can operate simultaneously on different data packets. Due to this, NoCs are starting to replace traditional communication architectures such as P2P signal wires, shared buses, etc.

Booksim is a cycle accurate simulator for NoCs which can also be used for modelling interconnection networks for other kinds of systems as well. It was originally developed for generating performance graphs in the interconnection network textbook by Dally and Towles. Since then, it has been widely used for research in many network contexts, including study of various aspects of network design, including topology, routing, flow control, router microarchitecture and quality-of-service.

In its current form, BookSim is able to simulate k-ary n-cube topologies, in addition to butterfly, concentrated mesh, fat tree, flattened butterfly, dragonfly and quad tree. However, all these topologies are planar and unable to simulate layered NoCs. The objective of this project was to add this functionality to the simulator allowing it to simulate 3-dimensional architectures. This allows replication of layered models

where vertical connections (connections across the layers) are treated differently to the on-chip connections.

# LITERATURE SURVEY

## 2.1 Performance of Network-on-Chips

An SoC (System-onChip) involves integrating many cores like processor cores, DSP cores, memories and many other hardware components.

However, SoC has certain limitations:

- The communication among the blocks delays the development at the system level, enhancing the complexity. The wire delays globally do not match up with the scaling of the technology even if several repeaters are utilized.
- Since SoC brings together varying hardware cores which are being utilized for different applications on the same chip, but these applications correspond to different requirements such as the standards used for communication and the methodology and constraints with respect to the design. Thus we can see that there are numerous problems if the purpose is a common SOC for a large number of applications.

Thus for resolving the above mentioned issues, NoC (Network-on-chip) is a viable alternative.

NoC ensures communication between IP blocks using routers. The communication methods ensure data transfer between the blocks as in real life situations on a chip. In order to ensure proper integration the size and structure of routers need to be optimised. Routers in NoC systems generally consist of buffers, arbiters and switches for performing core functions.

A connection between nodes in NoC is point to point. The methodology of communication between the two nodes relies on packet-switching, though certain other NoC utilize circuit switching techniques. Since the channels utilized are duplex and shared between the participating nodes, the technique used in shared-bus to send over the messages between the nodes.

In a NoC system, the power consumed, throughput, latency are the attributes which are used to measure the performance of the system. The latency stands for the time that passes between the appearances of the message header from the source node to the reception of the tail message at the receiver node. Thus the latency can be calculated as below:

$$L = \text{sender overhead} + \text{transport latency} + \text{receiver overhead} \quad (1.1)$$

The power consumed is also one of the factors that affects the performance of the system. Since the information passes between several nodes multiple times and there is a cost associated with each hop, the multi hopping system will cost us a lot of total energy. Router is the source of the hop and thus considered the source of the power consumption. The power dissipated in a NoC can be calculated as follows:

$$P = \Sigma Pr_{buf} + Pr_{arbiter} + Pr_{crossbar} + Pr_{link} \quad (1.2)$$

$Pr_{buf}$  is the average power consumption in buffers including both dynamic and static

power;  $Pr_{arbiter}$  is the average power consumption in routing computation;

$Pr_{crossbar}$  is the average power consumption in switches used in data transmission from input to output through a router; and  $Pr_{link}$  is the average link power consumption between two neighboring routers. Therefore, the total power consumption will be high if there are numerous hops in the transmission path.

Throughput is another performance parameter, defined by the amount of data which is passed successfully to the destination in one unit of time. In general, the throughput  $T$  is defined as:

$$T = (\text{Total msgs completed}) * (\text{Avg. msg. length}) / [(\text{No. of IP blocks}) * (\text{Total time})] \quad (1.3)$$

Where *total messages completed* is the number of messages that have arrived at the destination node successfully; the *average message length* is in its (in this report, we assume a it width of 20 bits); *number of IP blocks* refers to the number of routers that the message passes by and the *total time* is the whole time the message spends in transmission, which is equal to the time latency in 1.1. Thus, we see that if the *number of IP blocks* and *total time* increase, the throughput will decrease.

## **2.2 BookSim: A Cycle-accurate NoC Simulator**

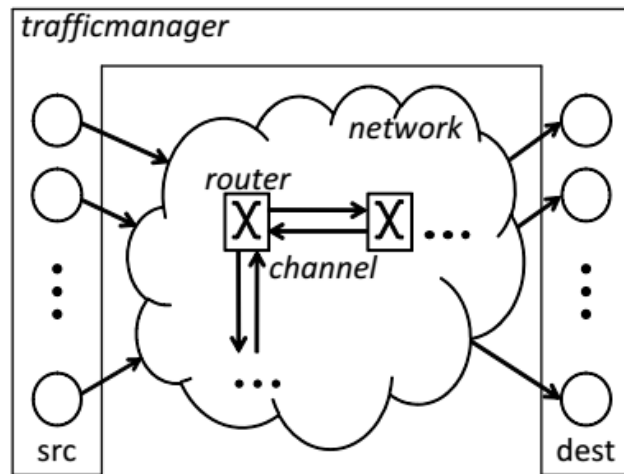
BookSim is a cycle accurate simulator for NoCs which can also be used for modelling interconnection networks for other kinds of systems as well. Developed at Stanford University, it was originally developed for generating performance graphs in the interconnection network textbook by Dally and Towles.

In the context of system simulations, BookSim is very flexible, making it a good high-level simulator. All major network components are parameterized which allows speedy replication of the network architecture. BookSim also provides detailed modeling of all key components of a network router.

The BookSim simulator has a modular structure which allows easy modification and addition of new network features. It has also been designed to replicate actual network design—for example, communication between adjacent routers has to occur through a “channel,” instead of using a global variable or a data structure.

BookSim’s flexibility comes from this highly modular implementation. The simulator consists of a hierarchy of modules which execute various functionalities of the network and simulation environment. Each of these modules has a well-defined interface which allows for easy replacement and modification of module implementations without influencing other parts of the simulated system.





At the top level of the simulator, we have sections for the trafficmanager and the network.

The trafficmanager is the wrapper around the network which is getting tested/simulated and models the source and destination endpoints. It injects packets into the network according to the configuration specified by the user, consisting of the traffic pattern, packet size, injection rate, etc. The trafficmanager is also ejects packets from the destination endpoints, collects appropriate statistics, and terminates the simulation.

The network top level module is composed of a collection of routers and channels, with the topology the structure of the interconnection of these modules. All communication between neighboring routers occurs through explicit send and receive operations across connecting channels, instead of updating global variables or data structures.

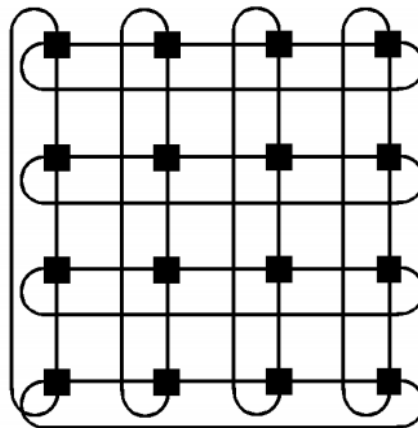
### 2.3 Topology: k-ary n-cube

Torus and mesh networks, k-ary n-cubes, consist of  $N = k^n$  nodes in a regular n-dimensional structure with each dimension consisting of k nodes and adjacent nodes having channels between them. They compass a variety of networks from rings ( $n = 1$ ) to binary n-cubes ( $k = 2$ ). These networks are attractive for following reasons:

- This regular physical arrangement is well matched to packaging constraints.

- Logically minimal paths in these grids are usually also physically minimal. This physical conservation facilitates torus and mesh networks to make use of the physical locality between communicating nodes for faster routing.
- For local communication patterns, such as each node sending to its neighbor in the first dimension, latency is much lower and throughput is much higher than for random traffic.

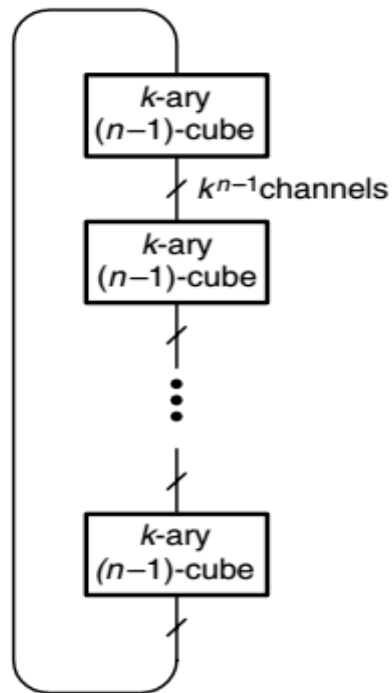
Being a direct network, each of the  $N$  nodes serves as an input terminal, output terminal, and a switching node of the network, all three at the same time. Each node is provided an  $n$ -digit radix- $k$  address and is connected by a pair of channels (outgoing and incoming) to all adjacent nodes. This requires 2 channels in each dimension per node.



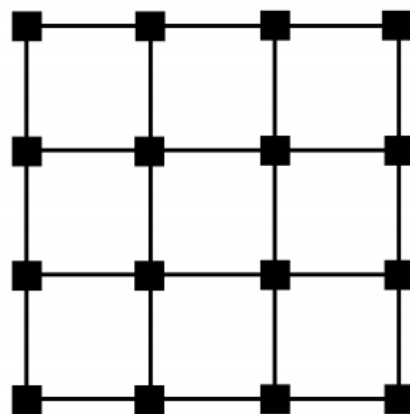
Torus Network

As a generic method, a  $k$ -ary  $n$ -cube can be constructed by adding dimension after dimension. A  $k$ -ary 1-cube is simply a  $k$ -node ring. We can make connections among  $k$  of these 1-cubes in a cycle incrementing the structure by a dimension, forming a  $k$ -

ary 2-cube. The process iterates one-dimension at a time, combining  $k$   $k$ -ary  $(n - 1)$ -cubes to form a  $k$ -ary  $n$ -cube.



A mesh network is a torus network with the connection from address  $a_{k-1}$  to address  $a_0$  omitted in each direction.

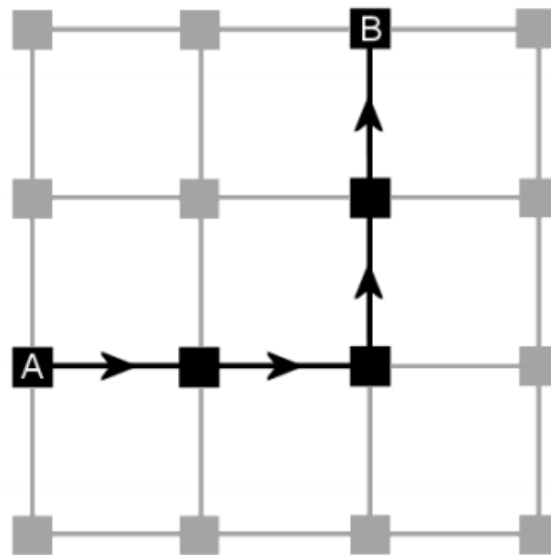


Mesh Network

## 2.4 Routing Algorithm: Dimension Order Routing

Dimension order routing (DOR) is an example of a minimal turn algorithm. The algorithm determines to which direction the packets are routed during every stage of the routing and routes packets in the minimum possible turns.

XY routing is a dimension order routing which routes packets first in x- or horizontal direction to the correct column and then in y- or vertical direction to the receiver. XY routing can be implemented on planar networks.



XY Routing

## **PROBLEM DEFINITION AND OBJECTIVES**

### **3.1 Problem Definition**

Making changes in the BookSim code to remove the specialization of equal weights of all links and equal nodes in all dimensions. At present, the simulator does not consider the vertical links any different from the horizontal links and there is no provision to make a network in which the number of the nodes in each dimension is different.

### **3.2 Problem Description**

Though a NoC has many important advantages, it has a serious performance limitation due to the planar metal interconnects essentially requiring multi-hop communication between any non-neighbor nodes. This causes high latency and power consumption. In other words, the large number of functional nodes is the main reason that limits the performance of a traditional NoC with wired communication. In the future, the number of processors and memories in a single multi-core system will increase to hundreds or even thousands [34].

To be specific, the longer the path, more interference from other crossing paths will increase latency and power to unreasonable amounts. In addition, the throughput will decrease.

In a multi-layered NoC, the distance between two adjacent layers is significantly smaller compared to distance between nodes on an integrated circuit which allows for reduced latency. However BookSim in its current configuration lacks the ability to reflect this property of vertical communication channels. While it is able to simulate multi-dimensional networks, all the dimensions are treated equally, with each of the dimensions having the same number of nodes and same latency for the communication channels in that dimension.

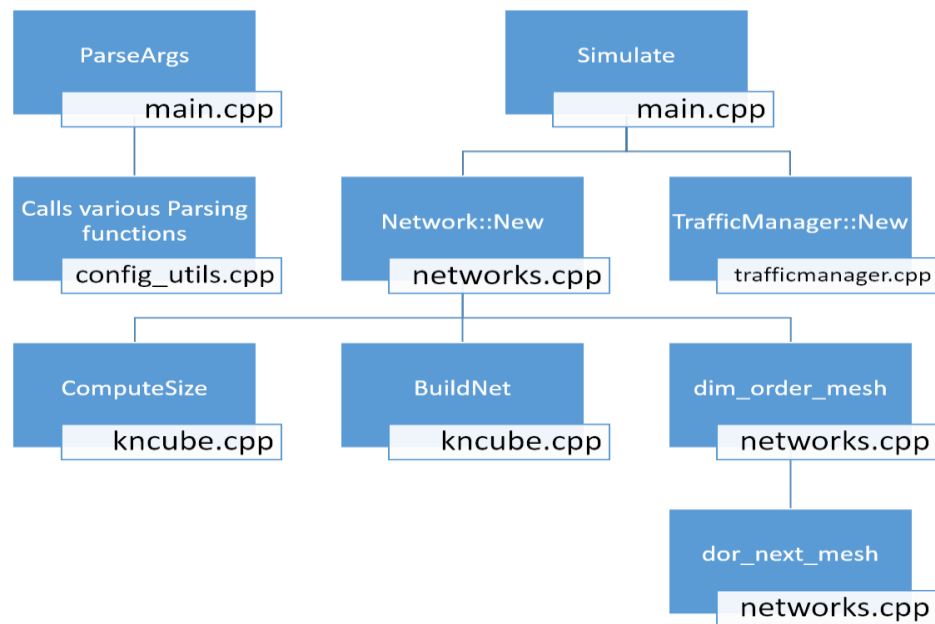
### 3.3 Objectives

Modifying the BookSim code to add the following features to it:

1. For a 3-D configuration, BookSim must be able to simulate a mesh network with different number of nodes in each dimension.  
**Current:** k, n input where k is the number of nodes in each dimension and n is the number of dimensions.  
**Modified:** k0, k1, k2 input where kN represents the number of nodes in dimension N.
2. BookSim must be able to implement a lower latency for vertical channels to reflect the distance between layers compared to the distance between nodes on a particular layer.

## METHODOLOGY

### 4.1 Simulation flow



The simulator is invoked using the following command line: `./booksim [configfile]`. The parameter `configfile` is a file that contains the configuration information for the simulator. So, for example to simulate the performance of a simple 8x8 mesh (8-ary 2-cube) network on a uniform traffic, a configuration such as the one shown below can be used.

```
// Topology
topology = mesh;
k = 8; n = 2;
// Routing
routing_function = dor;
```

```
// Flow control
num_vcs    = 8;
vc_buf_size = 8;
```

```
injection_rate = 0.005;
```

This particular example configuration file can be found in the examples/mesh88\_lat directory.

In addition to specifying the topology, the configuration file also contains the basic information about the routing algorithm, flow control, and traffic. This simple example uses dimension-order routing and four virtual channels. The `injection_rate` parameter is added to tell the simulator to inject (on average) 0.005 packets per simulation cycle per node. Packet size defaults to a single flit. Also, any line of the configuration file that begins with `//` is treated as a comment and ignored by the simulator. Any parameters not specified by the user will take on default values. The default values for every parameter in the simulator is specified in the file `booksim_config.cpp`.

The detailed list of configuration parameters in the configuration file is as follows:

k	Network radix, the number of routers per dimension
n	Network dimension
mesh	A k-ary n-mesh (mesh) topology. The k parameter determines the network's radix and the n parameter determines the network's dimension.
num_vcs	The number of virtual channels per physical channel



.  
vc\_buf\_size            The depth of each virtual channel in flits.

wait\_for\_tail\_credit    If non-zero, do not reallocate a virtual channel until the tail flit has left that virtual channel. This conservative approach prevents a dependency from being formed between two packets sharing the same virtual channel in succession.

input\_speedup           An integer speedup of the input ports in space. A speedup of 2, for example, gives each input two input ports into the crossbar. Access to these ports is statically allocated based on the virtual channel number: virtual channel  $v$  at input  $i$  is connected to port  $i \cdot s + (v \bmod s)$  for an input speedup of  $s$ .

output\_speedup          An integer speedup of the output ports in space. Similar to input speedup

internal\_speedup        An arbitrary speedup of the internals of the routers over the channel transmission rate. For example, a speedup 1.5 means that, on average, 1.5 flits can be forwarded by the router in the time required for a single flit to be transmitted across a channel. Also, the configuration parser expects a floating point number for this field, so integer speedups should also include a decimal point (e.g. "2.0").

sw\_allocator            The type of allocator used for switch allocation. See Section 4.6 for a list of the possible allocators.

sw\_alloc\_delay          The delay (in cycles) of switch allocation.

vc\_allocator            The type of allocator used for virtual-channel allocation.

vc\_alloc\_delay          The delay (in cycles) of virtual-channel allocation.

`alloc_iters` For the `islip`, `pim` and `select` allocators, allocation can be improved by performing multiple iterations of the algorithm; this parameter controls the number of iterations to be performed for both switch and VC allocation.

The simulation starts from the file `"booksim2-master/src/main.cpp"`. The arguments which are mentioned in the configuration file are then processed by calling the `ParseArgs` function. The function is defined in the file `"booksim2-master/src/config_utils.cpp"`. The function has return type `Boolean` which signifies whether the configuration file has been successfully parsed or not. The default value for the return variable is set to `false` and then the function starts processing the configuration file. When the configuration file is started a message is printed so as to notify the users of the process and a similar message is popped once the reading of the file is complete. If there are no errors the value of the return variable is set to `true`, and the control returns back to `main.cpp`.

The `Simulate` function in `main.cpp` with return type is `Boolean` then takes the input values and initiates the simulation. The function starts off with a vector of network class objects. The `Network` class is defined in `"booksim2-master/src/network.cpp"`. Once the vector has been declared all the elements in the vector are assigned values by iteratively calling the `Network::New` function defined in `"booksim2-master/src/network.cpp"`. `Network::New` function identifies the topology being used and then calls the corresponding `RegisterRoutingFunctions()`. In our example the function called is `KNCube::RegisterRoutingFunctions()`. The `KNCube` class and its associated functions and variables are defined in `"booksim2-master/src/kncube.cpp"`. The constructor includes functions

1. `_ComputeSize()`
2. `_Alloc()`
3. `_BuildNet()`

`_ComputeSize` method calculates the total number of nodes in the network. As for the standard BookSim simulation (k-ary n-cube) it would be k raised to n. Since we modified the BookSim code to implement layered networks, we then take different number of nodes in each dimension, with the value of n being fixed as 3. Thus the new size would be  $k_1 * k_2 * k_3$  where  $k_1$ ,  $k_2$  and  $k_3$  are the number of nodes in each dimension. The number of channels are also calculated in this method with the reasoning that the number of channels required would be twice the number of total nodes. This is because each node would have one incoming and one outgoing channel.

`_Alloc` method is from the Network class which is called for each of the nodes iteratively. This method is needed to simulate the channel latency. BookSim used arrays of flits as the channels, which has capacity of one. To simulate channel latency, `flitchannel` class has been added which are first in first out with `depth=channel latency` and in each cycle the channel shifts by one. Credit channels are the necessary counterpart.

`_BuildNet` function is responsible for building the entire network of the nodes in alignment of the topology mentioned. The method includes numbering the nodes, defining the left and right node for each node in the network, adding the corresponding left and right channel for each node/router. The method then differentiates the input channels from the output channels and also sets the latency for each channel. This arrangement builds the entire network as all the nodes/routers in the network now have a path with any other node based on their position in the network and the data can be transmitted using the channels, while the delay could be calculated in accordance with the channel latency value. The cost of the communication, used to depend only on the number of channels or hops the data has to pass through. But now since we differentiate the communication in the

vertical direction from data transfer in same plain, the cost would now also consider the link or the type of channel being utilised during the transmission of data between the nodes.

## 4.2 Important Source Code Changes

*Locating the Node to the Right in a Dimension (Left Node is Analogous)*

```
int KNCube::_RightNode( int node, int dim )
{
    int k_to_dim, loc_in_dim;

    if (dim == 0)
    {
        k_to_dim = 1;
        loc_in_dim = ( node / k_to_dim ) % _k0;
    }
    else if (dim == 1)
    {
        k_to_dim = _k0;
        loc_in_dim = ( node / k_to_dim ) % _k1;
    }
    else
    {
        k_to_dim = _k0*_k1;
        loc_in_dim = ( node / k_to_dim ) % _k2;
    }

    int right_node;
    // if at the right edge of the dimension, wraparound
    if ( loc_in_dim == ( _k-1 ) ) {
        right_node = node - (_k-1)*k_to_dim;
    } else {
        right_node = node + k_to_dim;
    }

    return right_node;
}
```

The variable **loc\_in\_dim** stores the position of the node in the required dimension.

For example, in a (4,4,4) configuration, node 63 [0-63] is the 4th node in Dimension

3.

The variable **k\_to\_dim** stores the numerical distance between the node and its right node. For example, in the vertical dimension, a node vertically immediately above the current node is the right node. For this scenario, k\_to\_dim is 16. (4\*4).

#### Insertion of Random Faults at Edges

```
for ( int i = 0; i < _size; ++i ) {  
    int node = i;  
  
    // edge test  
    bool edge = false;  
  
    if ( (node % _k0) == 0 || (node % _k0) == _k0-1 ) {  
        edge = true;  
    }  
    node /= _k0;  
  
    if ( (node % _k1) == 0 || (node % _k1) == _k1-1 ) {  
        edge = true;  
    }  
    node /= _k1;  
    if ( (node % _k2) == 0 || (node % _k2) == _k2-1 ) {  
        edge = true;  
    }  
    node /= _k2;  
  
    if ( edge ) {  
        fail_nodes[i] = true;  
    } else {  
        fail_nodes[i] = false;  
    }  
}
```

### Dimension Order Routing (dor\_next\_mesh: Identifies Output Channel for Routing)

```
if(descending) {
    for ( dim_left = ( gN - 1 ); dim_left > 0; --dim_left ) {
        if(dim_left == 2)
        {
            if ( ( cur * gK2 / gNodes ) != ( dest * gK2 / gNodes ) ) { break; }
            cur = (cur * gK2) % gNodes; dest = (dest * gK2) % gNodes;
        }
        else if(dim_left == 1)
        {
            if ( ( cur * gK1 / gNodes ) != ( dest * gK1 / gNodes ) ) { break; }
            cur = (cur * gK1) % gNodes; dest = (dest * gK1) % gNodes;
        }
    }
    cur = (cur * gK0) / gNodes;
    dest = (dest * gK0) / gNodes;
} else {
    for ( dim_left = 0; dim_left < ( gN - 1 ); ++dim_left ) {
        if(dim_left == 0)
        {
            if ( ( cur * gK0 / gNodes ) != ( dest * gK0 / gNodes ) ) { break; }
            cur = (cur * gK0) % gNodes; dest = (dest * gK0) % gNodes;
        }
        else if(dim_left == 1)
        {
            if ( ( cur * gK1 / gNodes ) != ( dest * gK1 / gNodes ) ) { break; }
            cur = (cur * gK1) % gNodes; dest = (dest * gK1) % gNodes;
        }
    }
    cur %= gK2;
    dest %= gK2;
}
```

To achieve Dimension Order Routing, we treat the 3D grid as a combination of 3 2D planes. Following this we apply XY Routing to each of the XY, YZ and ZX planes.

## RESULTS AND ANALYSIS

The BookSim code was modified to event for the different cost of communication between horizontal and vertical links. We then compared the results for the following attributes:

1. Packet latency
2. Network latency

For the original BookSim the simulations were done for

- $k=4$   $n=3$
- $k=8$   $n=2$ .

The configuration file was altered in each simulation so that the **injection rate** keeps on increasing by a margin of **0.005** in each simulation. Each link in this network is considered to be equal in terms of the latency and the cost involved in the communication. But we are aware of the fact that due to shorter links in the vertical direction both the latency and the cost of the communication is reduced. So the simulated performance could be improved if the changes suggested can be implemented in BookSim.

We removed the specialization where the number of nodes in each dimension must be the same and generalized the BookSim simulator for mesh topology where the number of the nodes in each dimension are independent of the number of the nodes in the other direction. This generalized the simulator and the simulations were closer to the actual performance to be expected from network on chip systems.

To compare the results obtained when the latency is not even for both vertical and horizontal channels, the default latency in the original BookSim was modified to be 2 instead of the previous value 1. Later the vertical links were assigned the latency

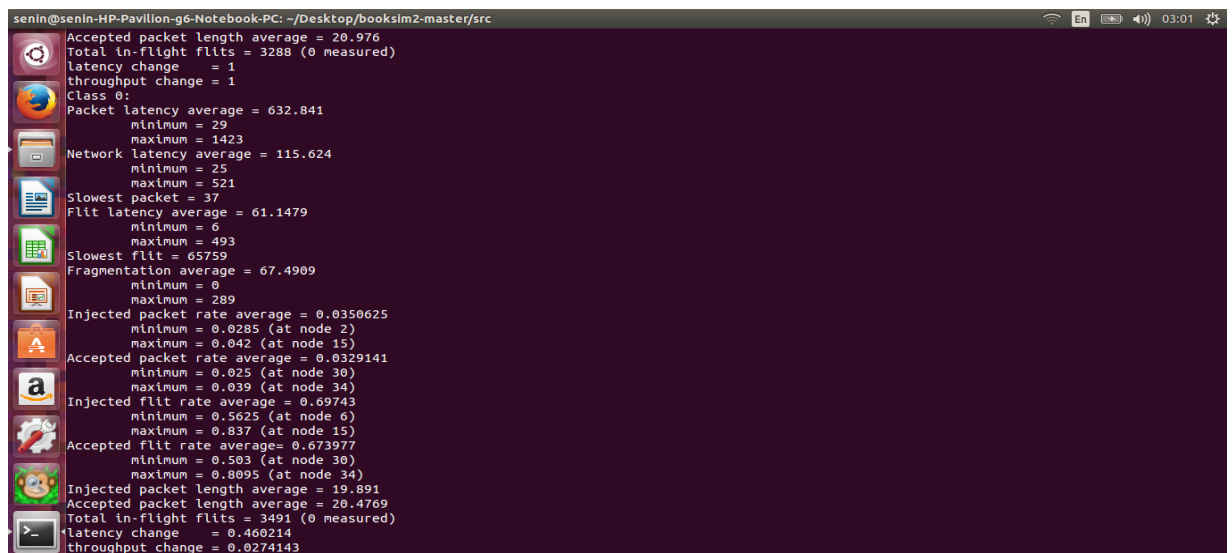
value of 1 and the horizontal links retained value 2. This was done so that the difference in throughput can be easily observed in both the cases.

A sample simulation showed the following results:

1. Total in flight flits
2. Latency change
3. Throughput change

Along with the traffic statistics

1. Packet latency average,min and max value
2. Network latency average,min and max value
3. Flit latency average,min and max value
4. Fragmentation average,min and max value
5. Injected packet rate average,min and max value
6. Accepted packet rate average,min and max value

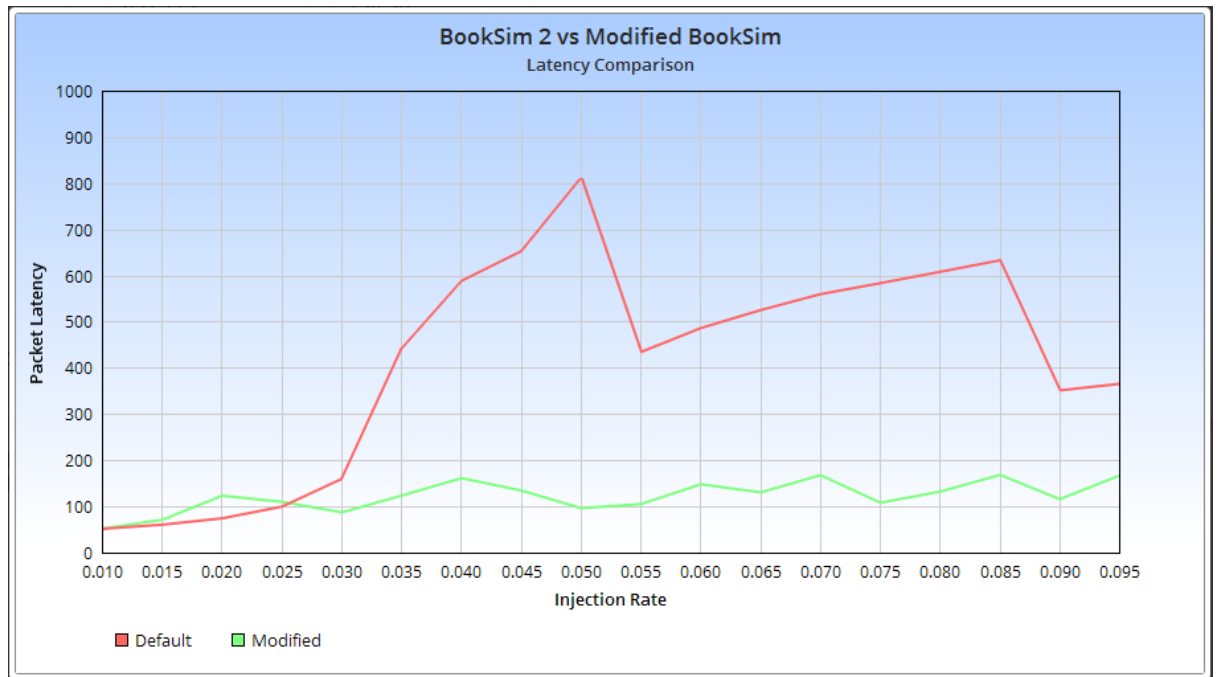


```
senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/books1m2-master/src
Accepted packet length average = 20.976
Total in-flight flits = 3288 (0 measured)
latency change = 1
throughput change = 1
Class 0:
Packet latency average = 632.841
    minimum = 29
    maximum = 1423
Network latency average = 115.024
    minimum = 25
    maximum = 521
Slowest packet = 37
Flit latency average = 61.1479
    minimum = 6
    maximum = 493
Slowest flit = 65759
Fragmentation average = 67.4909
    minimum = 0
    maximum = 289
Injected packet rate average = 0.0350625
    minimum = 0.0285 (at node 2)
    maximum = 0.042 (at node 15)
Accepted packet rate average = 0.0329141
    minimum = 0.025 (at node 30)
    maximum = 0.039 (at node 34)
Injected flit rate average = 0.69743
    minimum = 0.5625 (at node 6)
    maximum = 0.837 (at node 15)
Accepted flit rate average = 0.673977
    minimum = 0.503 (at node 30)
    maximum = 0.8095 (at node 34)
Injected packet length average = 10.891
Accepted packet length average = 20.4769
Total in-flight flits = 3491 (0 measured)
latency change = 0.460214
throughput change = 0.0274143
```

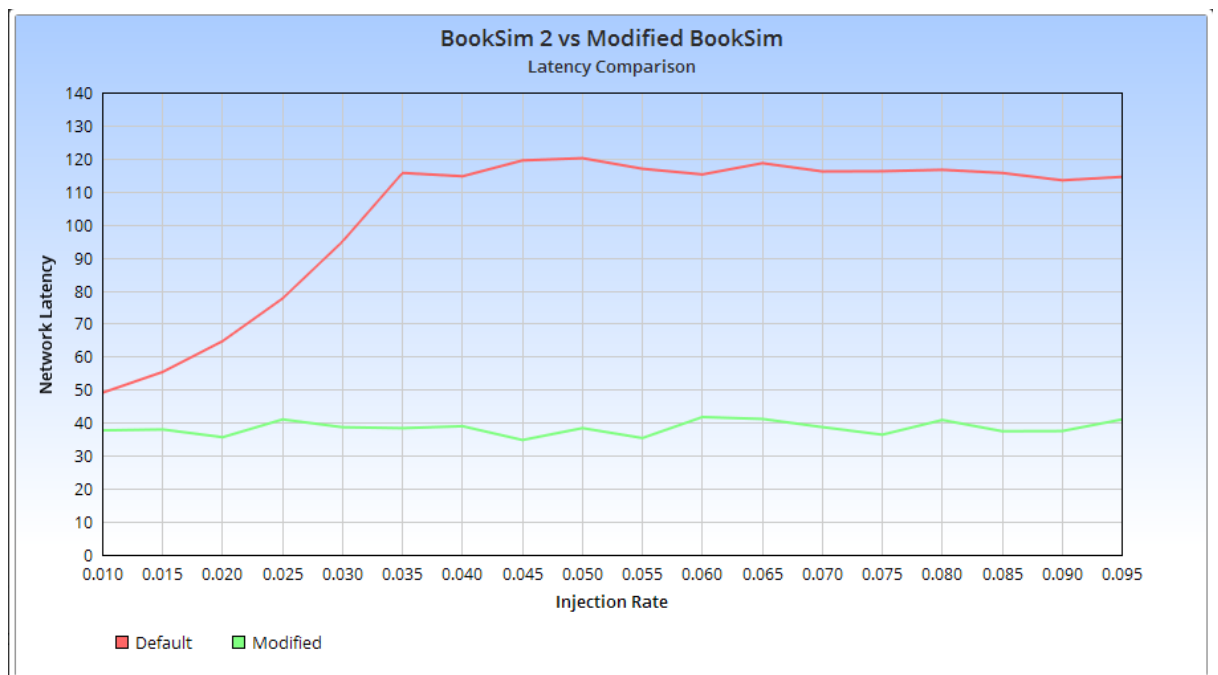


## Comparison of BookSim2 and Modified BookSim

### Configuration 1: 4 4 4



**Comparison of Average Packet Latency for (4,4,4) configuration**



**Comparison of Average Network Latency for (4,4,4) configuration**

## Results (4,4,4) configuration for different injection rates

senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksIm/booksim2\_vl/src

```

Class 0:
Packet latency average = 52.0571
    minimum = 27
    maximum = 205
Network latency average = 37.6857
    minimum = 27
    maximum = 63
Slowest packet = 67
Flit latency average = 17.2443
    minimum = 8
    maximum = 41
Slowest flit = 1359
Fragmentation average = 2.54286
    minimum = 0
    maximum = 26
Injected packet rate average = 0.0056875
    minimum = 0.002 (at node 8)
    maximum = 0.01 (at node 4)
Accepted packet rate average = 0.000546875
    minimum = 0 (at node 3)
    maximum = 0.004 (at node 12)
Injected flit rate average = 0.107188
    minimum = 0.036 (at node 11)
    maximum = 0.196 (at node 22)
Accepted flit rate average = 0.0109375
    minimum = 0 (at node 3)
    maximum = 0.08 (at node 12)

```

senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksIm/booksim2\_vl/src

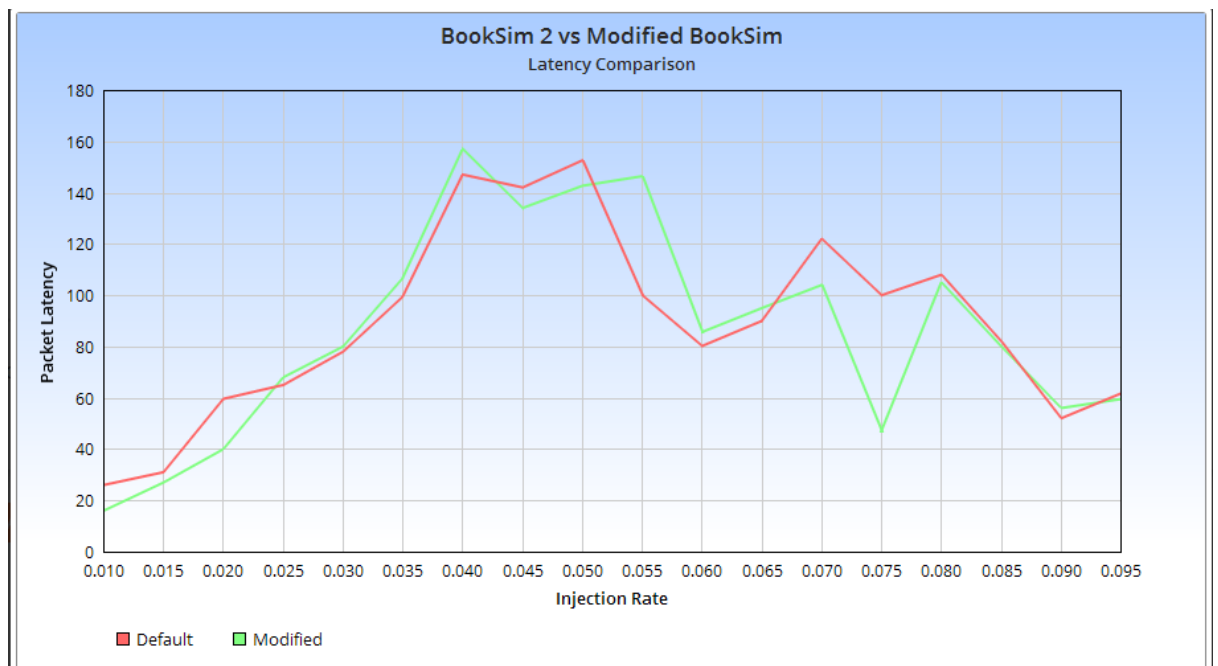
```

Class 0:
Packet latency average = 96.1818
    minimum = 27
    maximum = 256
Network latency average = 38.3636
    minimum = 27
    maximum = 60
Slowest packet = 55
Flit latency average = 16.5295
    minimum = 8
    maximum = 35
Slowest flit = 3039
Fragmentation average = 4.31818
    minimum = 0
    maximum = 18
Injected packet rate average = 0.0056875
    minimum = 0.002 (at node 5)
    maximum = 0.011 (at node 12)
Accepted packet rate average = 0.00034375
    minimum = 0 (at node 0)
    maximum = 0.003 (at node 45)
Injected flit rate average = 0.107125
    minimum = 0.028 (at node 26)
    maximum = 0.216 (at node 12)
Accepted flit rate average = 0.006875
    minimum = 0 (at node 0)
    maximum = 0.06 (at node 45)
Injected packet length average = 18.8352
Accepted packet length average = 20

```

```
senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBookSim/booksim2_V1/src
Class 0:
Packet latency average = 167.077
    minimum = 37
    maximum = 582
Network latency average = 41
    minimum = 27
    maximum = 65
Slowest packet = 83
Flit latency average = 18.3423
    minimum = 8
    maximum = 43
Slowest flit = 1676
Fragmentation average = 6.30769
    minimum = 0
    maximum = 23
Injected packet rate average = 0.00565625
    minimum = 0.003 (at node 17)
    maximum = 0.01 (at node 0)
Accepted packet rate average = 0.000203125
    minimum = 0 (at node 1)
    maximum = 0.001 (at node 0)
Injected flit rate average = 0.105875
    minimum = 0.048 (at node 17)
    maximum = 0.196 (at node 0)
Accepted flit rate average = 0.0040625
    minimum = 0 (at node 1)
    maximum = 0.02 (at node 0)
```

## Configuration 2: 8 8 1



### Comparison of Average Packet Latency for (8,8,1) Configuration

Analyzing these results, we can see that for a planar configuration (8,8,1), the performance of the two are quite similar.

However, for a (4,4,4) configuration, which has a **layered configuration**, there is an improved performance in the modified configuration due to reduced latency in the vertical channels.

### Results (8,8,1) configuration for different injection rates

```
senin@senin-HP-Pavillon-g6-Notebook-PC: ~/Desktop/MajorProjectBooksim/booksim2_v
Class 0:
Packet latency average = 139
    minimum = 139
    maximum = 139
Network latency average = 27
    minimum = 27
    maximum = 27
Slowest packet = 172
Flit latency average = 8
    minimum = 8
    maximum = 8
Slowest flit = 3440
Fragmentation average = 0
    minimum = 0
    maximum = 0
Injected packet rate average = 0.00375
    minimum = 0.001 (at node 11)
    maximum = 0.008 (at node 15)
Accepted packet rate average = 1.5625e-05
    minimum = 0 (at node 0)
    maximum = 0.001 (at node 56)
Injected flit rate average = 0.066875
    minimum = 0.016 (at node 11)
    maximum = 0.148 (at node 15)
Accepted flit rate average = 0.003125
    minimum = 0 (at node 0)
```

senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksim/booksim2\_vl/src

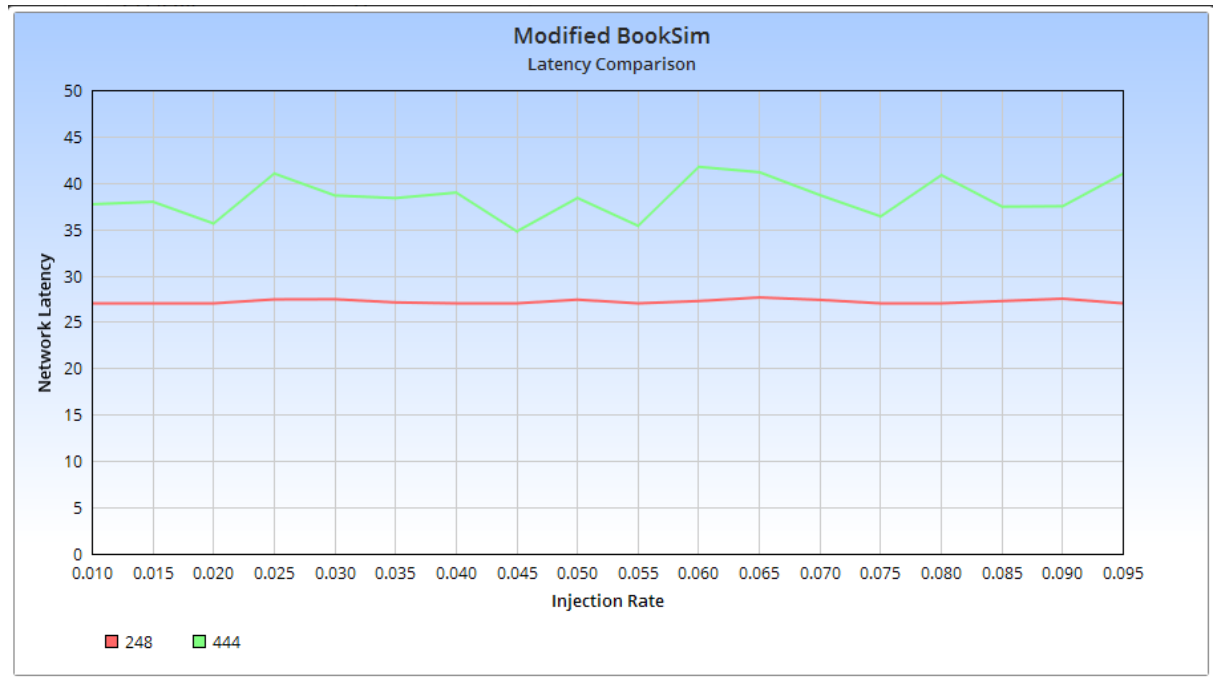
```
Class 0:
Packet latency average = 142.667
    minimum = 43
    maximum = 332
Network latency average = 27
    minimum = 27
    maximum = 27
Slowest packet = 69
Flit latency average = 8
    minimum = 8
    maximum = 8
Slowest flit = 1380
Fragmentation average = 0
    minimum = 0
    maximum = 0
Injected packet rate average = 0.00375
    minimum = 0.001 (at node 8)
    maximum = 0.006 (at node 9)
Accepted packet rate average = 4.6875e-05
    minimum = 0 (at node 0)
    maximum = 0.001 (at node 15)
Injected flit rate average = 0.06775
    minimum = 0.016 (at node 8)
    maximum = 0.108 (at node 9)
Accepted flit rate average = 0.0009375
    minimum = 0 (at node 0)
```

senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksim/booksim2\_vl/src

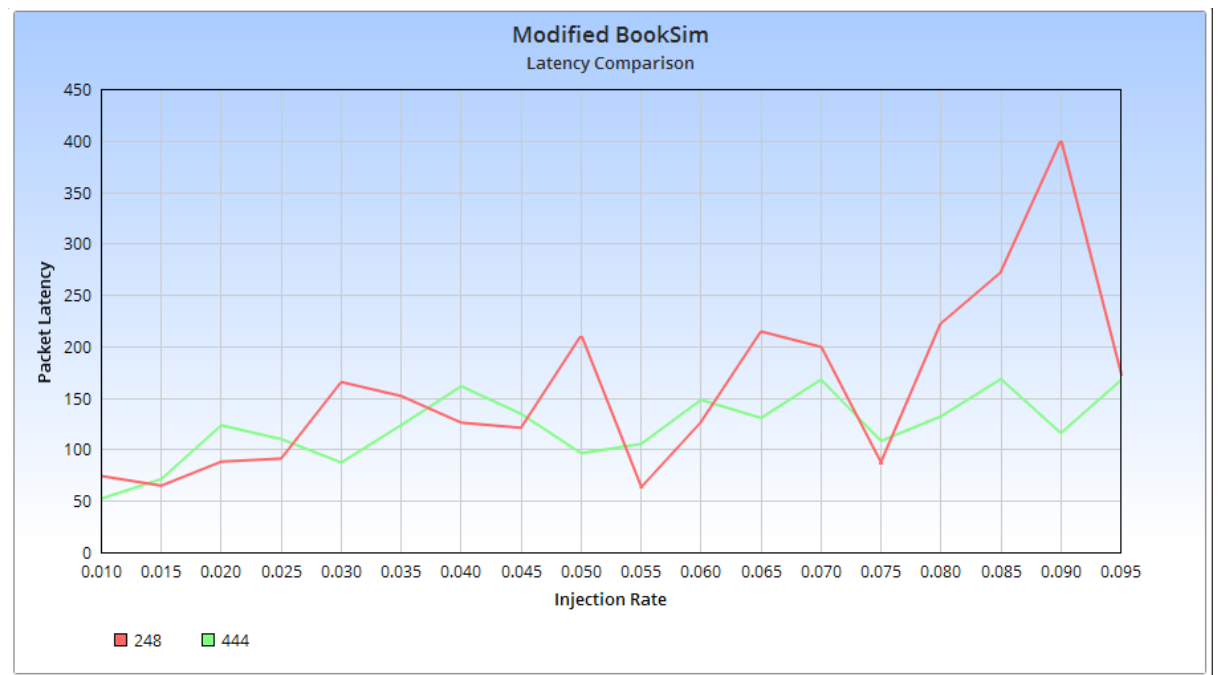
```
Class 0:
Packet latency average = 59.5
    minimum = 38
    maximum = 81
Network latency average = 27
    minimum = 27
    maximum = 27
Slowest packet = 100
Flit latency average = 8
    minimum = 8
    maximum = 8
Slowest flit = 2000
Fragmentation average = 0
    minimum = 0
    maximum = 0
Injected packet rate average = 0.00370312
    minimum = 0.002 (at node 10)
    maximum = 0.005 (at node 5)
Accepted packet rate average = 3.125e-05
    minimum = 0 (at node 0)
    maximum = 0.001 (at node 1)
Injected flit rate average = 0.0676875
    minimum = 0.036 (at node 10)
    maximum = 0.096 (at node 5)
Accepted flit rate average = 0.000625
    minimum = 0 (at node 0)
    maximum = 0.02 (at node 1)
Injected packet length average = 18.2785
Accepted packet length average = 20
```

## Comparison of Results on Modified BookSim for Various Configurations

The following graphs display a comparison of results of various configurations



Comparison of Network Latency



Comparison of Packet Latency



From the above results we can observe that **network latency is lower for (2,4,8)** configuration compared to (4,4,4) configuration.

This could be due to **more vertical channels in (2,4,8)** configuration. Vertical channels, as has been discussed earlier, are faster than the planar channels.

#### Results (2,4,8) configuration for different injection rates

```
senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksim/booksim2_vl/src
Class 0:
Packet latency average = 74
    minimum = 27
    maximum = 185
Network latency average = 27
    minimum = 27
    maximum = 27
Slowest packet = 185
Flit latency average = 8
    minimum = 8
    maximum = 8
Slowest flit = 3700
Fragmentation average = 0
    minimum = 0
    maximum = 0
Injected packet rate average = 0.00717188
    minimum = 0.003 (at node 53)
    maximum = 0.012 (at node 50)
Accepted packet rate average = 7.8125e-05
    minimum = 0 (at node 0)
    maximum = 0.001 (at node 7)
Injected flit rate average = 0.137063
    minimum = 0.048 (at node 53)
    maximum = 0.236 (at node 50)
Accepted flit rate average = 0.0015625
    minimum = 0 (at node 0)
    maximum = 0.02 (at node 7)
Injected packet length average = 19.1111
Accepted packet length average = 20
```

```
senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksim/booksim2_vl/src
Class 0:
Packet latency average = 210.4
    minimum = 27
    maximum = 599
Network latency average = 27.4
    minimum = 27
    maximum = 29
Slowest packet = 124
Flit latency average = 8.05263
    minimum = 8
    maximum = 10
Slowest flit = 4959
Fragmentation average = 0.4
    minimum = 0
    maximum = 2
Injected packet rate average = 0.00807813
    minimum = 0.003 (at node 42)
    maximum = 0.014 (at node 26)
Accepted packet rate average = 7.8125e-05
    minimum = 0 (at node 0)
    maximum = 0.001 (at node 3)
Injected flit rate average = 0.153391
    minimum = 0.056 (at node 42)
    maximum = 0.262 (at node 26)
Accepted flit rate average = 0.00178125
    minimum = 0 (at node 0)
    maximum = 0.02 (at node 3)
Injected packet length average = 18.9884
Accepted packet length average = 22.8
```

```
senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksim/booksim2_vl/src
Class 0:
Packet latency average = 173
    minimum = 37
    maximum = 377
Network latency average = 27
    minimum = 27
    maximum = 27
Slowest packet = 83
Flit latency average = 8
    minimum = 8
    maximum = 8
Slowest flit = 1660
Fragmentation average = 0
    minimum = 0
    maximum = 0
Injected packet rate average = 0.007875
    minimum = 0.004 (at node 21)
    maximum = 0.014 (at node 42)
Accepted packet rate average = 6.25e-05
    minimum = 0 (at node 2)
    maximum = 0.001 (at node 0)
Injected flit rate average = 0.150547
    minimum = 0.076 (at node 21)
    maximum = 0.268 (at node 42)
Accepted flit rate average = 0.00125
    minimum = 0 (at node 2)
    maximum = 0.02 (at node 0)
Injected packet length average = 19.1171
Accepted packet length average = 20
```



## CONCLUSION AND FUTURE WORK

BookSim Network Simulator is a valuable tool for research in various network contexts, including networks found in large-scale supercomputers and many-core processors. BookSim has been used to study many different aspects of network design, including topology, routing, flow control, router microarchitecture, quality-of-service, as well as new technologies such as Nano photonics.

BookSim can also be incorporated into other system simulators to model the network; for example, GPGPU-sim, a many-core accelerator simulator for evaluating GPGPU workloads, leverages BookSim to model the on-chip communication.

With the current networking paradigm shifting more and more towards layered NoCs from traditional architectures such as shared bus and crossbar, it would be very useful to have simulators able to reflect this shift. With the help of these simulators, students will be able to research and test network configurations without any financial expenditure.

It also allows students to explore how much vertical communication is feasible in a 3-D grid architecture and where the latency improvement regresses.

As a part of this project, we have only modified k-ary n-cube topologies. There are various other topologies such as CMesh, Quad Tree and Butterfly which can be modified to support layered NoCs.

## REFERENCES

1. Nan Jiang, Daniel Becker and William Dally, “A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator”
2. W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA: Morgan Kaufmann, 2004.
3. N. Bansal, A. Blum, S. Chawla, A. Meyerson: *Online Oblivious Routing. Proceedings of the fifteenth annual ACM symposium on Parallel algorithms and architectures*, 2003
4. BookSim User manual
5. BookSim 2.0. [Online]. Available: <http://nocs.stanford.edu/booksim.html>
6. J. Nurmi: *Network-on-Chip: A New Paradigm for System-on-Chip Design. Proceedings 2005 International Symposium on System-on-Chip*, 15–17 November 2005