2016

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

BISHAL THINGOM 12CO98

CONTENTS

INTRODUCTION

An interconnection network is a programmable system transporting data between terminals. The network is programmable in the sense that it makes different connections at different points in time. The network is a system because it is composed of many components: buffers, 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. Developed at Stanford University, 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

Many publications state that the emergence of SoC (System-on-Chip) platforms consisting of a larger number of embedded processors is an outstanding solution for some industrial tasks. An SoC is a type of micro system that integrates many components like processor cores, DSP cores, memories (or storage of control interface that may be out of the chip) and many other hardware cores, which otherwise perform specialized tasks on separate single dies. However, there are two limitations of SoC. First of all, the communication among the Intellectual Property (IP) blocks impedes the development at the system level, aggravating the complexity issue. The global wire delays are the most important factors that typically do not scale with technology scaling even after we insert repeaters. Another problem with SoC is that, SoC always integrates several different hardware cores for different applications on the same chip, but those applications always have diverse requirements such as the different communication standards and specific design constraints. Therefore, there may be difficulties when designing a common SoC for many applications. In order to solve the problems stated above, NoC (Network-on-chip) is a good paradigm.

NoC is an integrated network that uses routers to allow the communication among those blocks. It makes use of networking theory and methods for on-chip communication so that the blocks can exchange information on a chip just like what the terminals do in the actual world. The so called blocks here will always refer to processors and caches and, for simplify, a block will be renamed as node in the remaining report. In a system, the distribution of nodes complies with certain specific topologies. Some popular topologies have been studied, such as SPIN (Scalable, Programmable, Integrated Network), CLICH (Chip-Level Integration of Communicating Heterogeneous Elements), regular mesh, Torus, Fold torus, Octagon and BFT (Buttery Fat Tree) [26], [65]. Each node communicates through routers and since there must be a numbers of routers integrated on a centimeter-size chip, the size of router should be small enough and the structure should be as simple as possible. In general, a router in NoC system consists of

1) Buffers to store data temporarily

2) Arbiters to decide the sequence of data transmission

3) Switches to transfer data in the right direction.

So far, the wormhole router is the most popular router used in NoC systems. The advantage of this type of router is that data can be transmitted more uently. Corresponding to the wormhole router, the transmitted data unit in NoC is usually called \it". A it contains dozens of bits. Each it has a header, which contains the address of the destination, and it is followed by a string of data bits. When being transmitted, then it runs like a stream through the routers. A link in NoC is point to point. The common communication between two nodes is generally based on packet-switching, although there exist other NoC proposals utilizing circuit-switching techniques. As each channel that connects nodes is duplex and shared by only two nodes, the NoC improves the performance over the traditional system, which uses shared-bus to transmit data.

In a NoC system, time latency, power consumption and throughput are the main parameters used to evaluate its performance. Latency is defined as the time (in clock cycle) that elapses between the occurrence of a message header injection into the network at the source node and the reception of the tail of message at the destination node. Therefore,for a given message, the latency *L* is given by:

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

The transport latency is tiny so that we can ignore it. Send overhead and receiver overhead mainly depend on the message waiting time in a router, which is in proportion to the number of nodes on a single chip times the injection rate. For example, if a node requires to transmit data to another node, the data must pass through all routers along the path between those two nodes. Although a path can be readily determined by any kinds of routing protocol, the transmitted data must encounter every router on the path. Assuming a standard mesh network, if node *A*, positioned is at (1, 1), intends to transmit data to node B at position (*N;M*), where (*N* > 2*; M* > 2), with some high probability there can be other transmitting paths that cut across the long path between nodes *A* and *B*. That means the routers in the path from node *A* to node *B* will be used by other transmissions at the same time. Since a router can process only one transmission at a time, when several transmissions come to the same router, only one of them can be processed immediately while the others wait until the router is free. In such cases, a high latency in the NoC system is encountered. Power consumption is another parameter that affects the performance of an NoC system. Since there are many node-to-node hops in a path and each hop consumes some energy, the multi-hop must cost more total energy. For each hop, router is the main source of the power consumption. When a it passes by a router, it will be stored in an input buffer to wait for being processed. If there are more than one bits stored in different input-buffers, an arbiter should decode the destination information of each it and decide which it to send first according to the routing protocol being used. Then, the chosen it is transmitted from input-buffer to output-buffer through a crossbars switch. If there is only one it arrives at the output of the router, it can be sent directly. But if there are several bits they have to be stored in the output buffer temporarily until their turn. The power dissipated in a NoC can be calculated as follows:

$$P = \Sigma Pr\ buf + Pr\ arbiter + Pr\ crossbar + Pr\ link \qquad (1.2)$$

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, it is obvious that the total power consumption will be large if there are many hops in the transmission path.Throughput is another performance parameter, defined by the amount of data arriving successfully at the destination in one unit of time. There are many ways to define it depending on the specifics of the implementation. In general, the throughput *T* is defined as:

***T* =(*Total msgs completed*) * (*Avg. msg. length*) / [(*No. of IP blocks*) * (*Total time*)]**

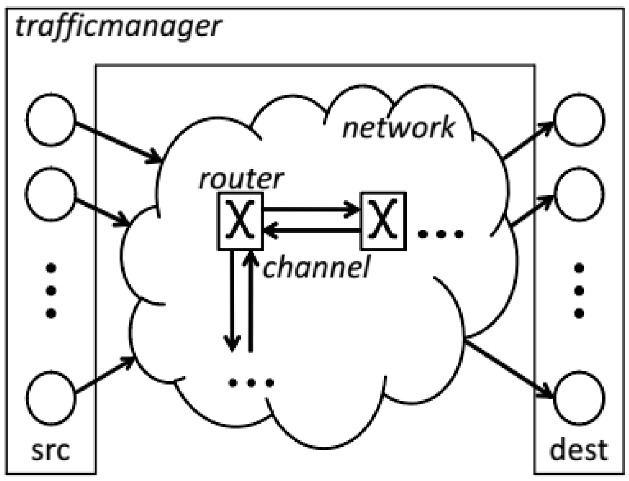
(1.3)

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.

2.2BookSim: 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.
From a system simulation perspective, BookSim provides the flexibility that is needed in a high-level simulator. All major network components are parameterized to allow rapid sweeps of the network design space. From a network design perspective, BookSim provides detailed modeling of all key components of a network router. The simulator is designed to be modular and to facilitate modifications and the addition of new network features. BookSim is also structured to reflect actual network design—for example, communication between neighboring routers needs to occur through a “channel,” rather than a global variable or a data structure. This approach forces simulator users to think about the network as a physical entity before implementing new features in the simulator.

BookSim’s flexibility comes from its highly modular implementation. The simulator is composed of a hierarchy of modules that implements different functionalities of the network and simulation environment. Each of these modules has a well-defined interface that facilitates replacement and customization of module implementations without affecting other parts of the simulated system.



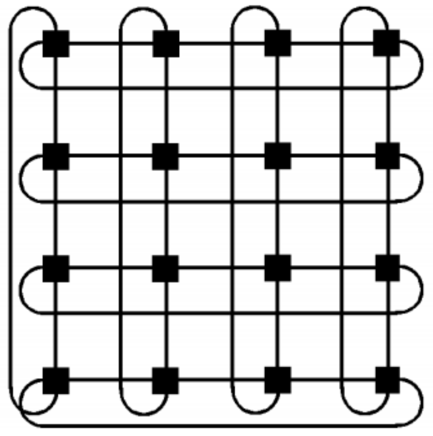
The top level modules of the simulator are the trafficmanager and the network. The trafficmanager is the wrapper around the network being evaluated and models the source and destination endpoints. It injects packets into the network according to the user-specified configuration, including the traffic pattern, packet size, injection rate, etc. To properly model network behavior beyond the point of saturation in open-loop simulations, an infinite source queue1 is implemented at the injection nodes to ensure that latency measurements properly account for source queuing delay and head-of-line blocking effects. The trafficmanager is also responsible for ejecting packets from the destination endpoints, collecting appropriate statistics, and terminating the simulation.

The network top level module comprises a collection of routers and channels, with the topology defining how these modules are interconnected. All communication between neighboring routers occurs through explicit send and receive operations across connecting channels, rather than by updating global variables or data structures. The simulator assumes that credit-based flow control is used for buffer management between adjacent routers and uses a separate, dedicated channel to communicate credit information; i.e., each network channel is accompanied by a credit channel in the opposite direction.

2.3 Topology: k-ary n-cube

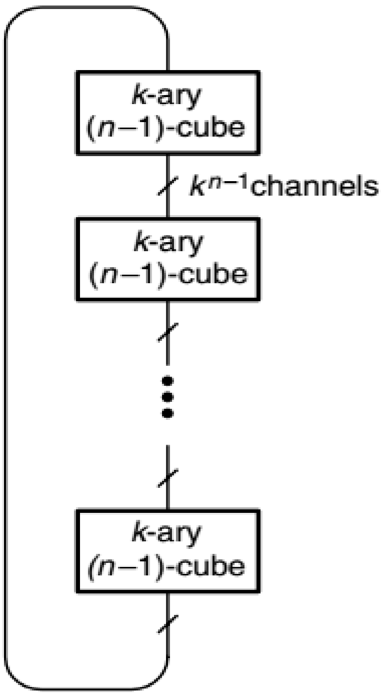
Torus and mesh networks, k-ary n-cubes, pack $N = k^n$ nodes in a regular n-dimensional grid with k nodes in each dimension and channels between nearest neighbors. They span a range of networks from rings ($n = 1$) to binary n-cubes ($k = 2$), also known as hypercube. These networks are attractive for several reasons. This regular physical arrangement is well matched to packaging constraints. At low dimensions, they have uniformly short wires allowing high-speed operation without repeaters. Logically minimal paths in these cubes are almost always physically minimal as well. This physical conservation allows torus and mesh networks to exploit physical locality between communicating nodes. 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 simultaneously as an input terminal, output terminal, and a switching node of the network. Each node is assigned an n-digit radix-k address and is connected by a pair of channels (one in each direction) to all nodes with addresses that differ by $\pm1 (\bmod k)$ in exactly one address digit. This requires 2 channels in each dimension per node or 2nN channels total. Tori are regular (all nodes have the same degree) and are also edge-symmetric, which helps to improve load balance across the channels.

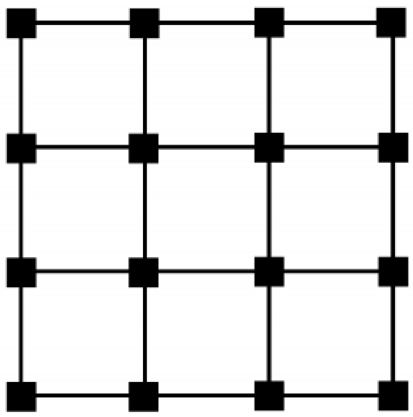


Torus Network

In general, an arbitrary k-ary n-cube can be constructed by adding dimensions iteratively. A k-ary 1-cube is simply a k-node ring. Connecting k of these 1-cubes in a cycle adds a second dimension, forming a k-ary 2-cube. The process continues 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. A mesh network has the same node degree, but half the number of bisection channels as a torus with the same radix and dimension. Althought he mesh has a very natural 2-D layout that keeps channel lengths short, it gives up the edge symmetry of the torus. This can cause load imbalance for many traffic patterns, as the demand for the central channels can be significantly higher than for the edge channels.

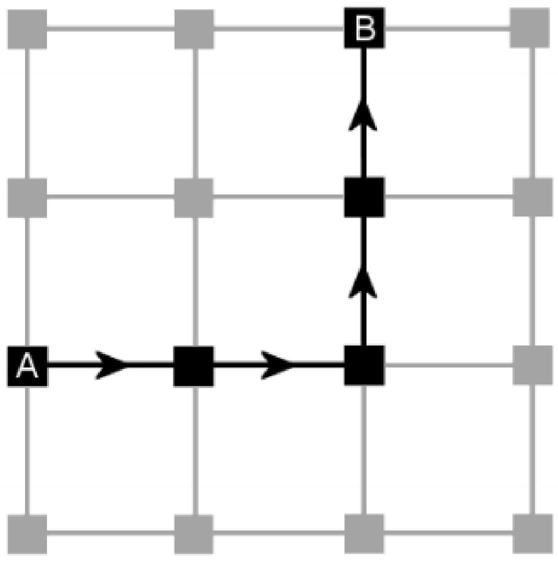


Mesh Network

2.4 Routing Algorithm: Dimension Order Routing

Dimension order routing (DOR) is a typical minimal turn algorithm. The algorithm determines to what direction packets are routed during every stage of the Routing. It routes packets using as few turns as possible.

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 suits well on a network using mesh or torus topology. Addresses of the routers are their xy-coordinates. XY routing never runs into deadlock or livelock.



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 an 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.

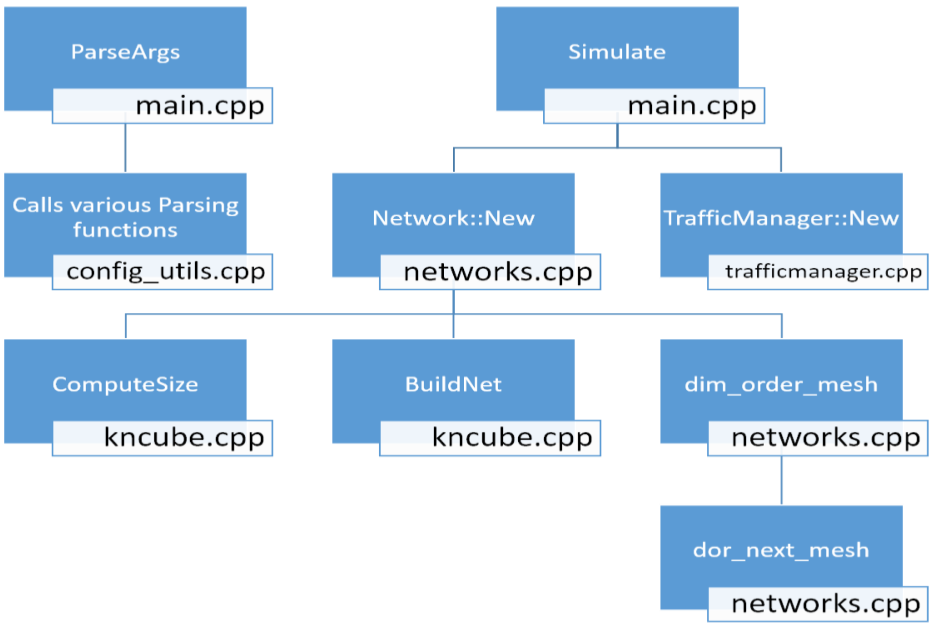
3.3 Objectives

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 · s + (v mod 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()

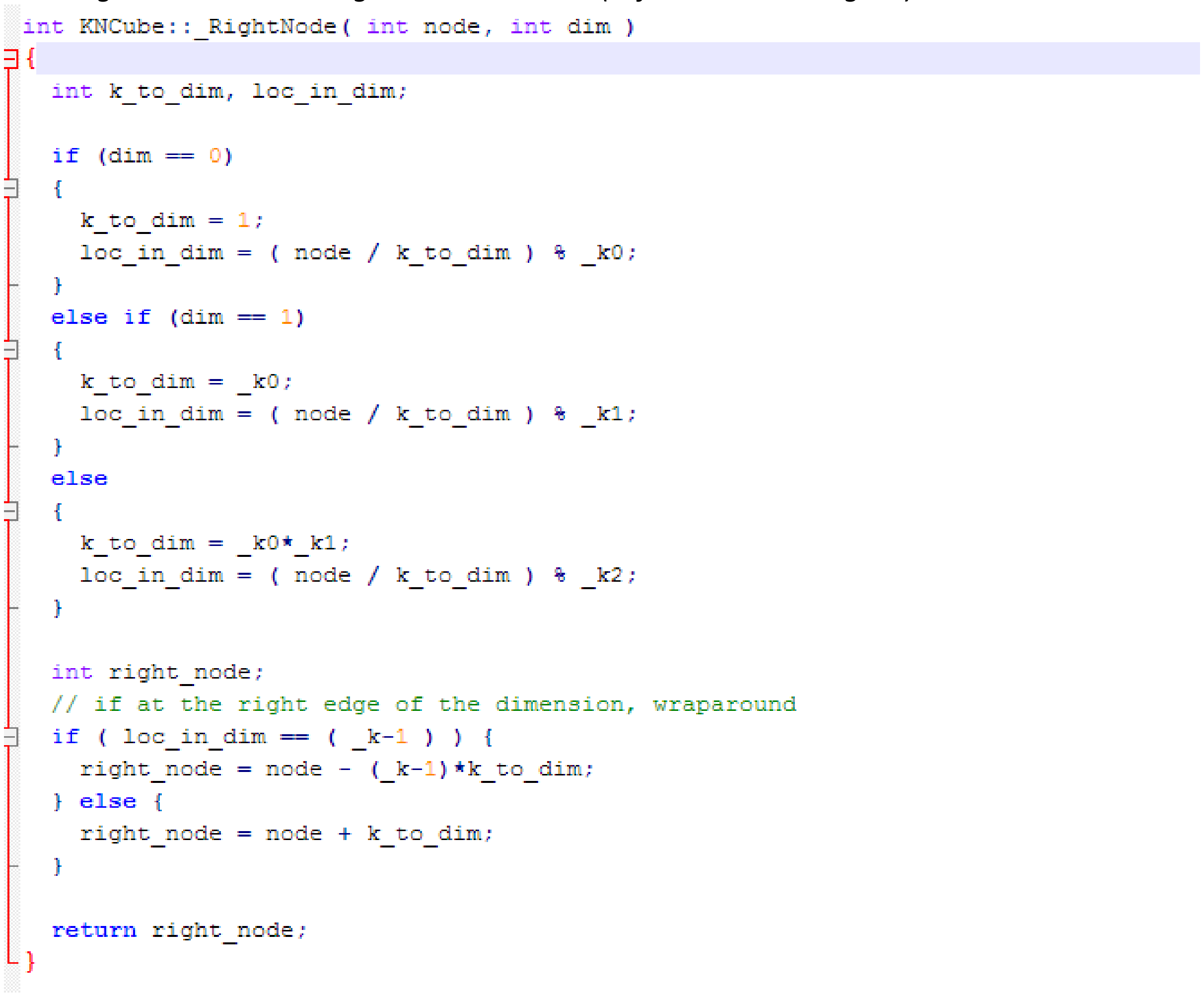
_ComputeSize method calculates the total number of nodes in the network. As for the standard BookSim simulation (k-ary n-cube) it would be k raised to n. Since we modified the BookSim code to implement layered networks, we then take different number of nodes in each dimension, with the value of n being fixed as 3. Thus the new size would be k1*k2*k3 where k1, k2 and k3 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)

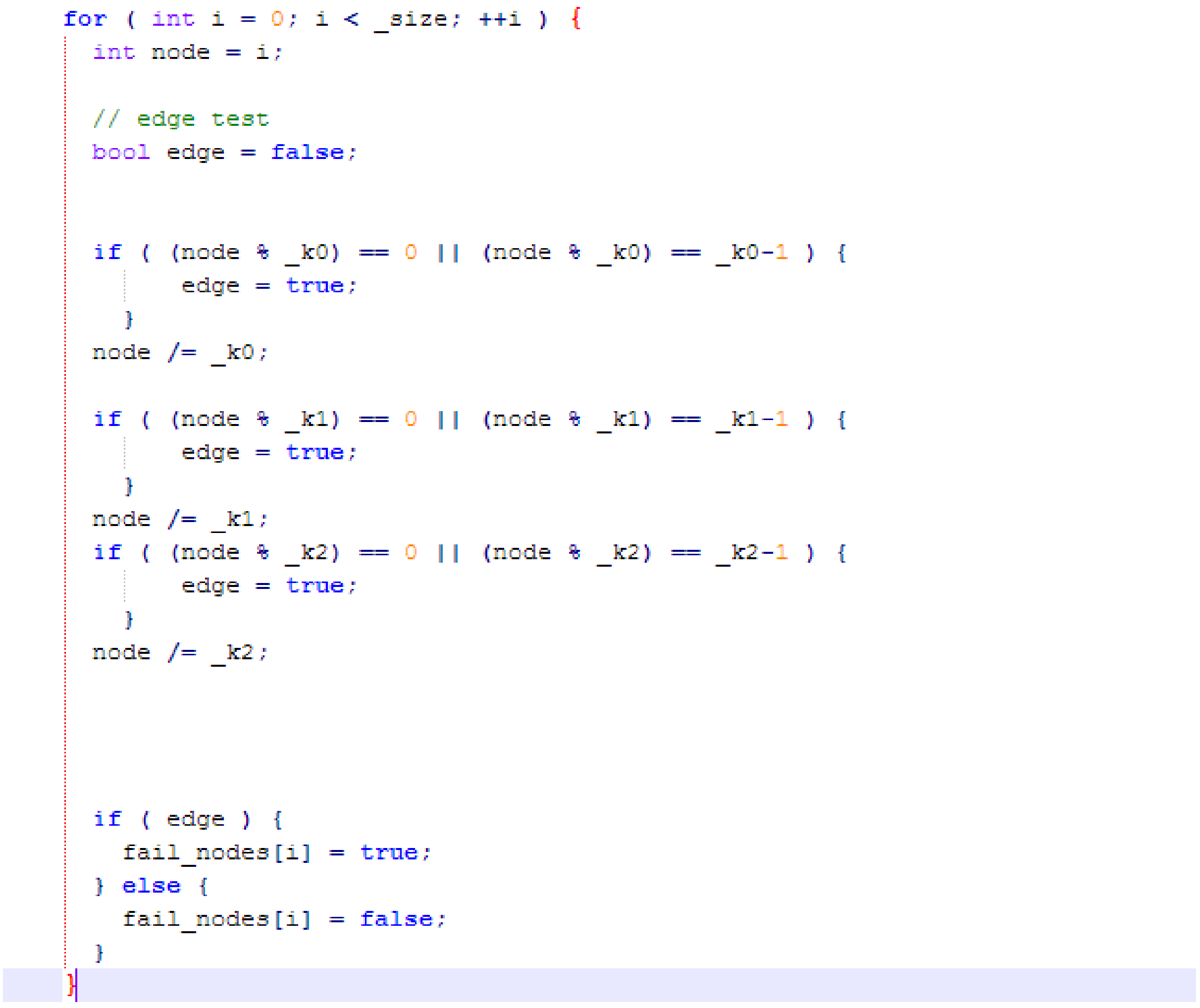


```
int KNCube::_RightNode( int node, int dim )
{
  int k_to_dim, loc_in_dim;

  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.

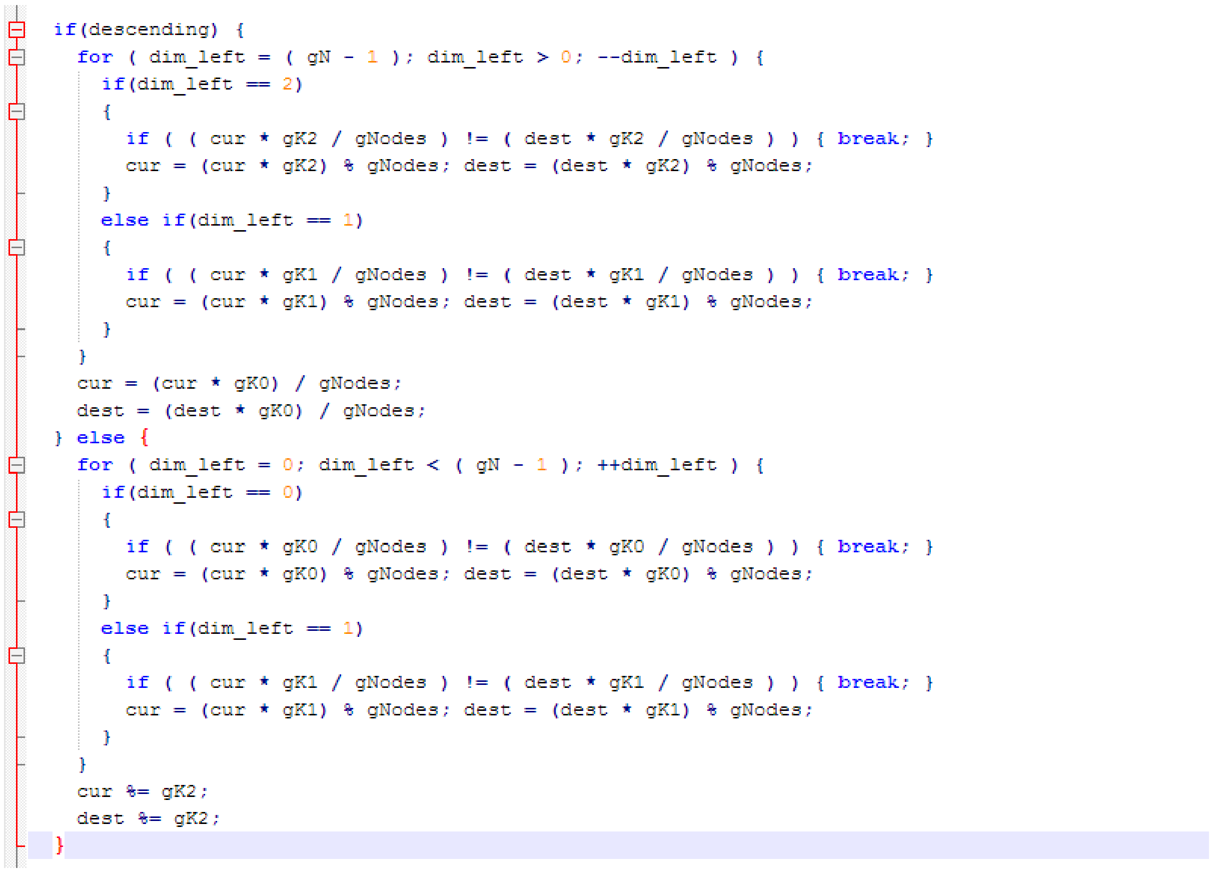
The variable **k_to_dim** stores the numerical distance between the node and its right node. For example, in the vertical dimension, a node vertically immediately above the current node is the right node. For this scenario, k_to_dim is 16. (4*4).

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.

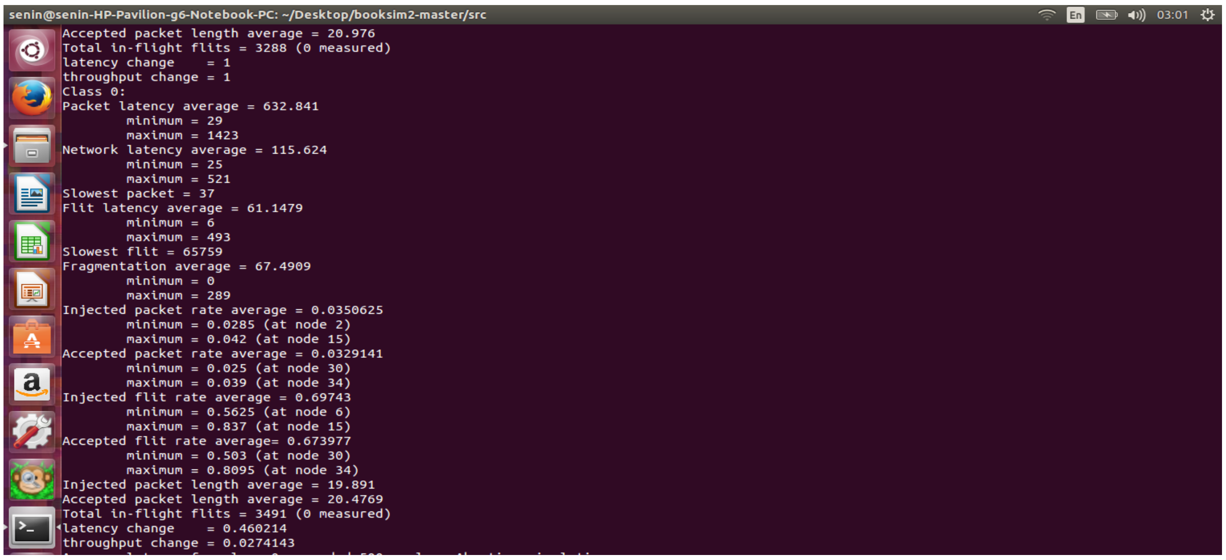
To compare the results obtained when the latency is not even for both vertical and horizontal channels, the default latency in the original BookSim was modified to be 2 instead of the previous value 1. Later the vertical links were assigned the latency value of 1 and the horizontal links retained value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

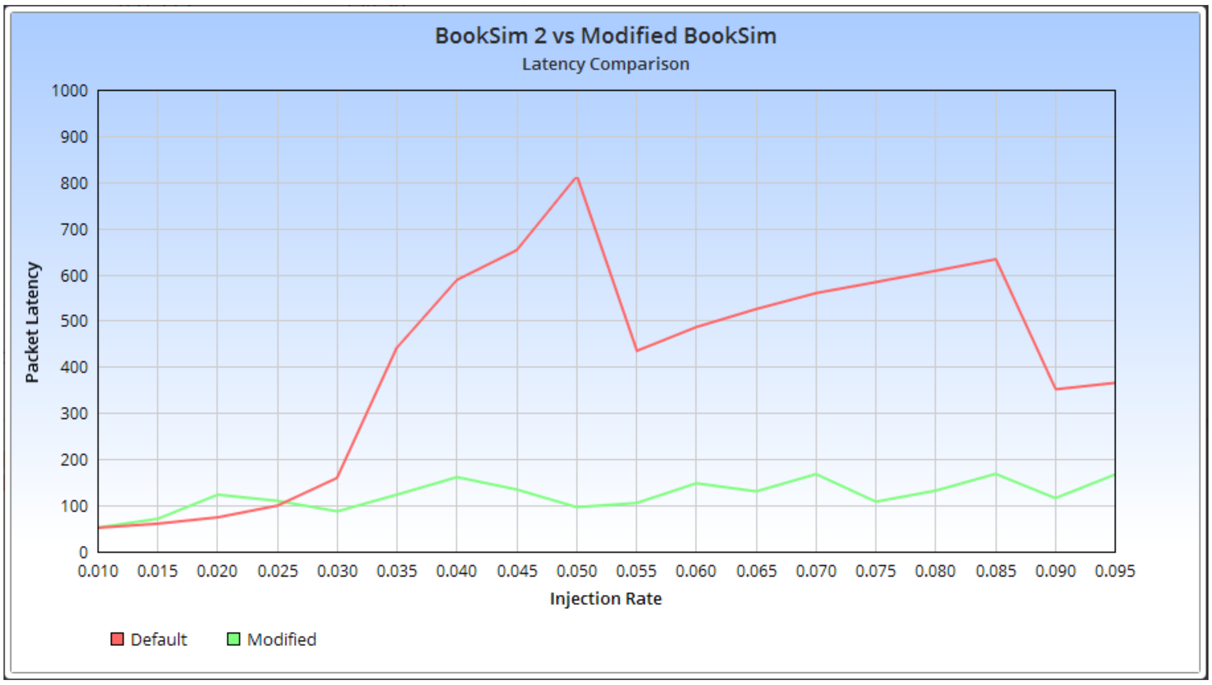
1. Packet latency average,min and max value
2. Network latency average,min and max value
3. Flit latency average,min and max value
4. Fragmentation average,min and max value
5. Injected packet rate average,min and max value
6. Accepted packet rate average,min and max value



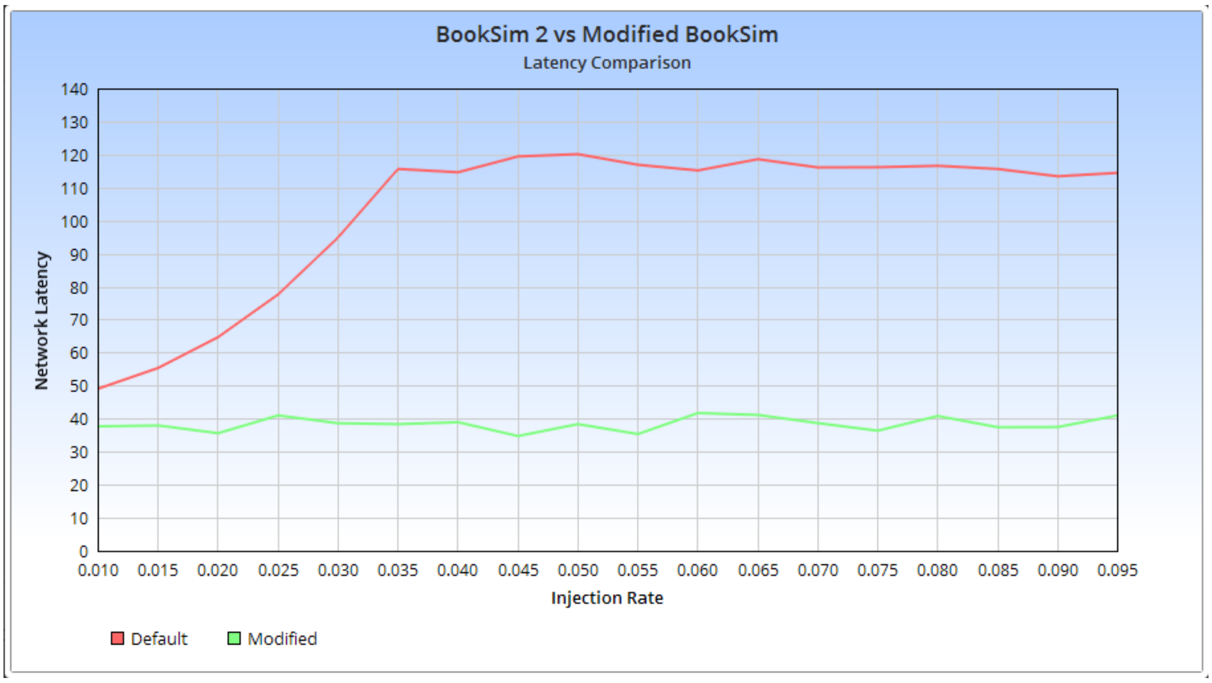
```
senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/booksim2-master/src
Accepted packet length average = 20.976
Total in-flight flits = 3288 (0 measured)
latency change    = 1
throughput change = 1
Class 0:
Packet latency average = 632.841
        minimum = 29
        maximum = 1423
Network latency average = 115.624
        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 = 19.891
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

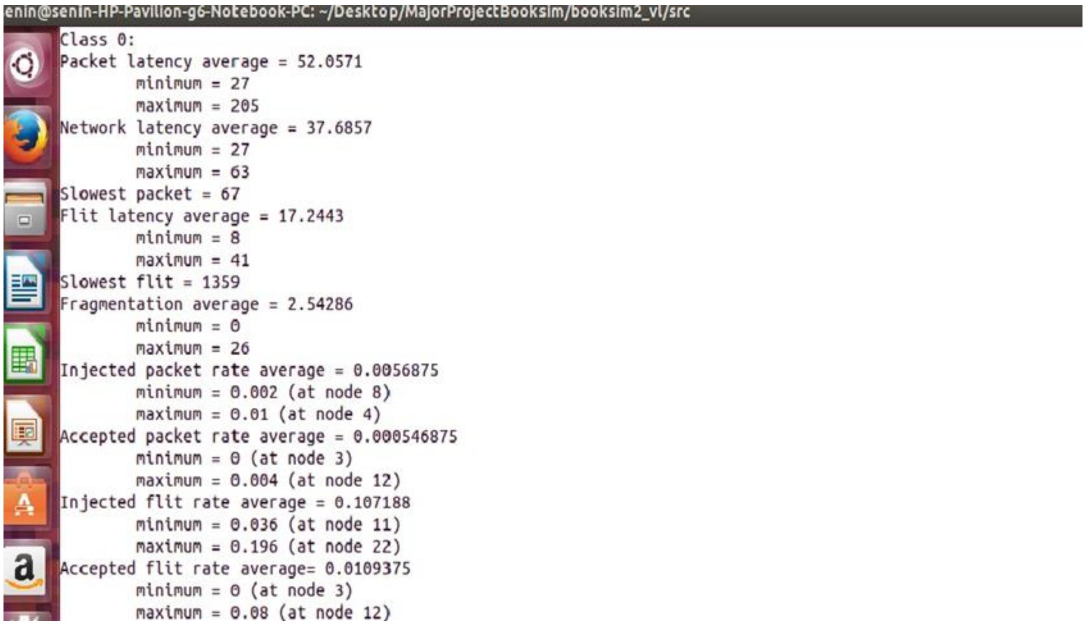


Comparison of Average Packet Latency for (4,4,4) configuration

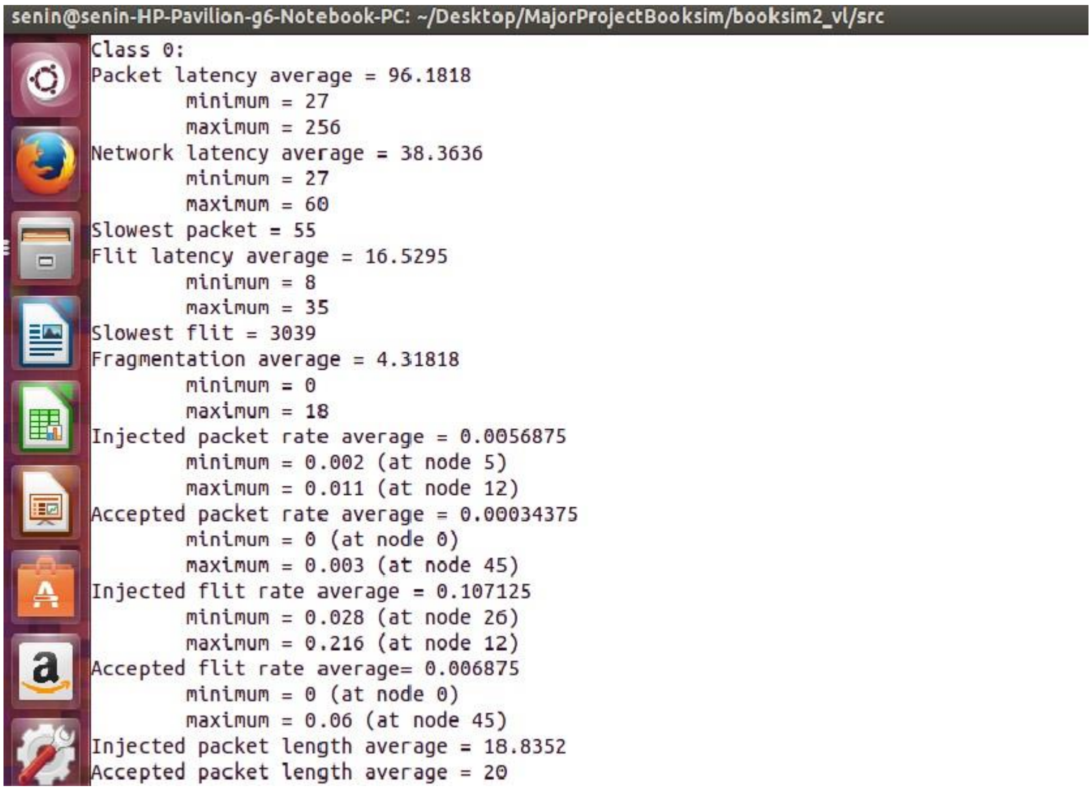


Comparison of Average Network Latency for (4,4,4) configuration

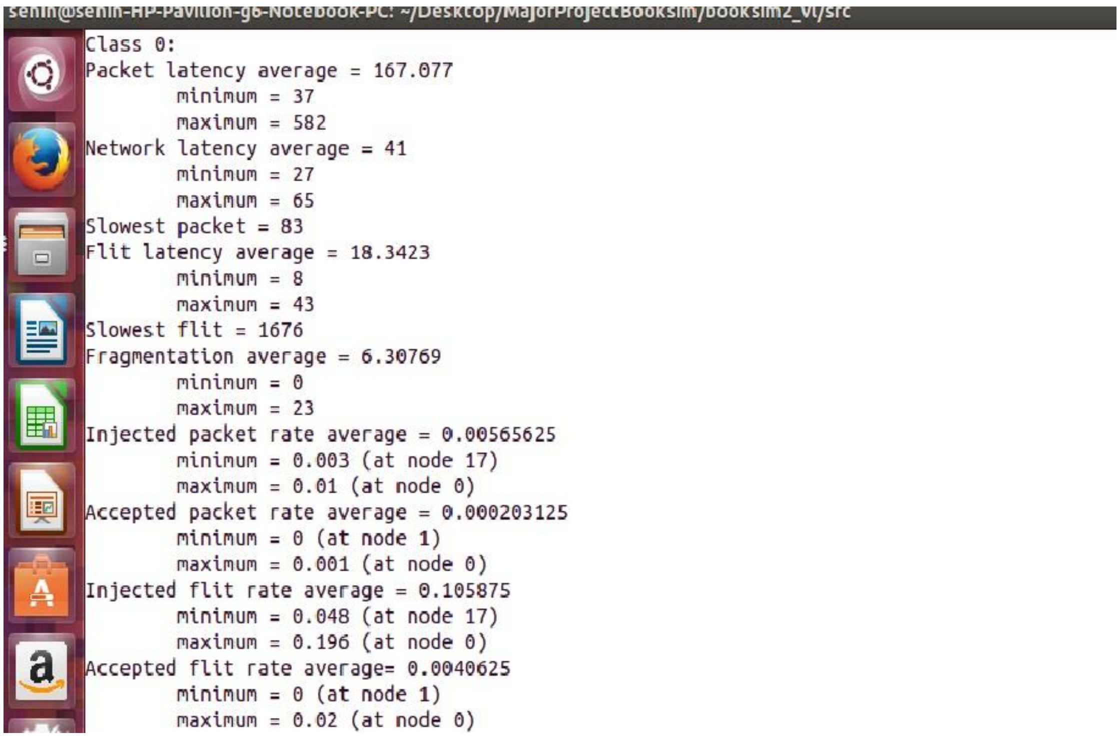
Results (4,4,4) configuration for different injection rates



```
enin@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)
```

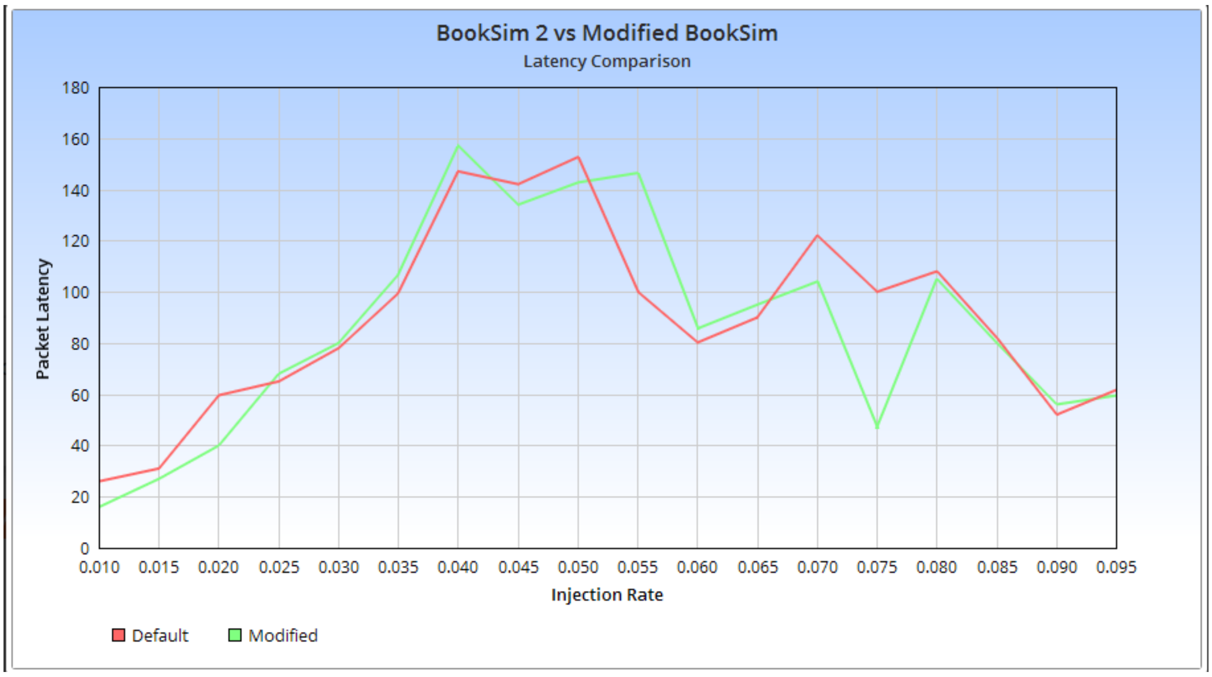


```
senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksim/booksim2_vl/src
Class 0:
Packet latency average = 96.1818
        minimum = 27
        maximum = 256
Network latency average = 38.3636
        minimum = 27
        maximum = 60
Slowest packet = 55
Flit latency average = 16.5295
        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
```



```
sehin@sehin-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

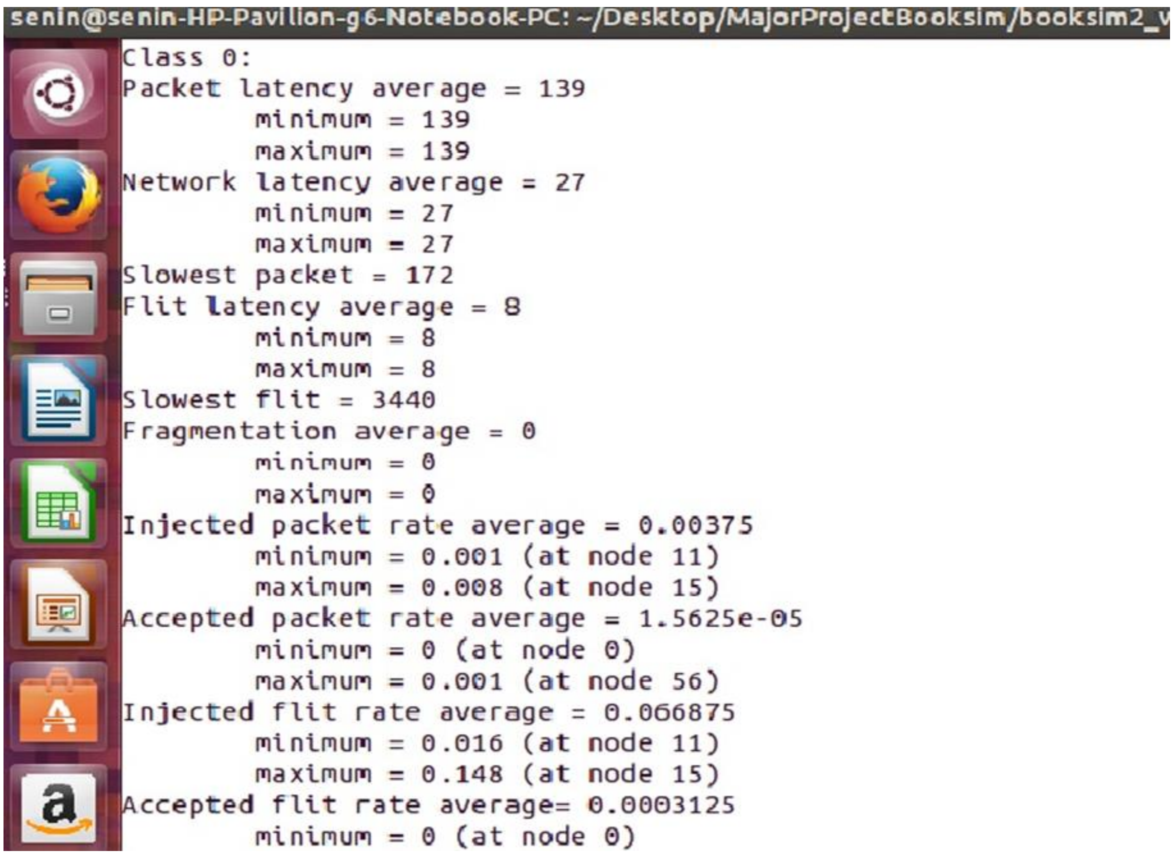


Comparison of Average Packet Latency for (8,8,1) Configuration

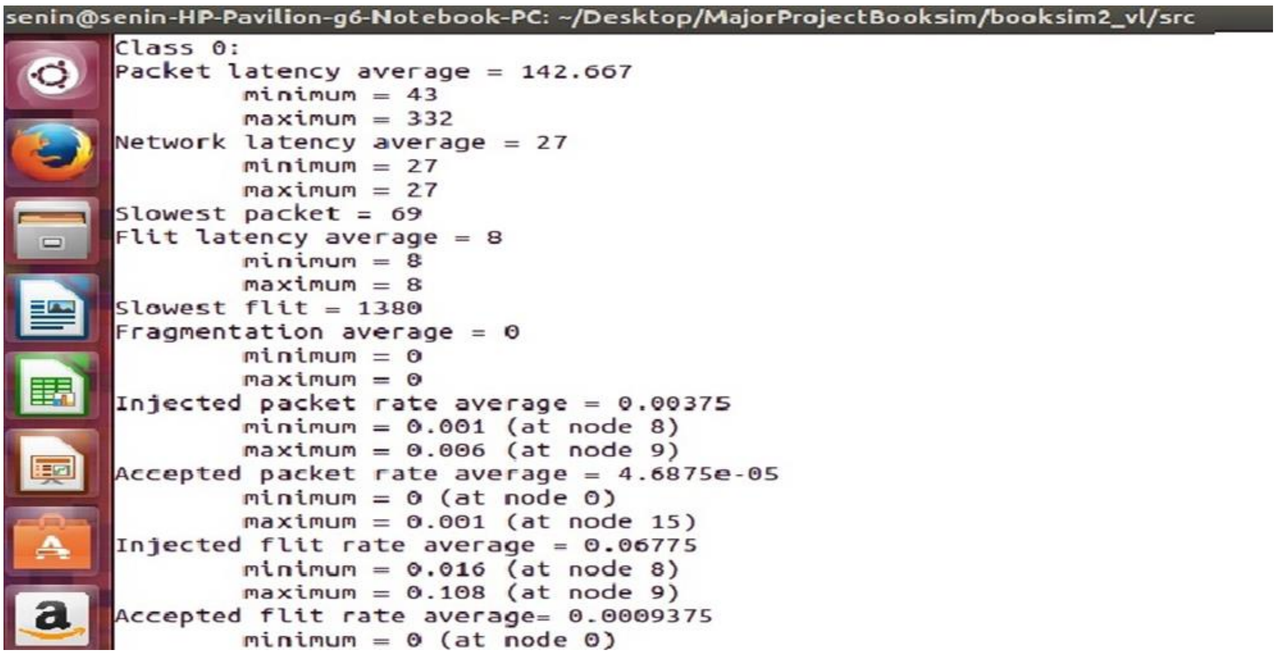
Analysing these results, we can see that **for a planar configuration (8,8,1), the performance of the two are quite similar.**

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.**

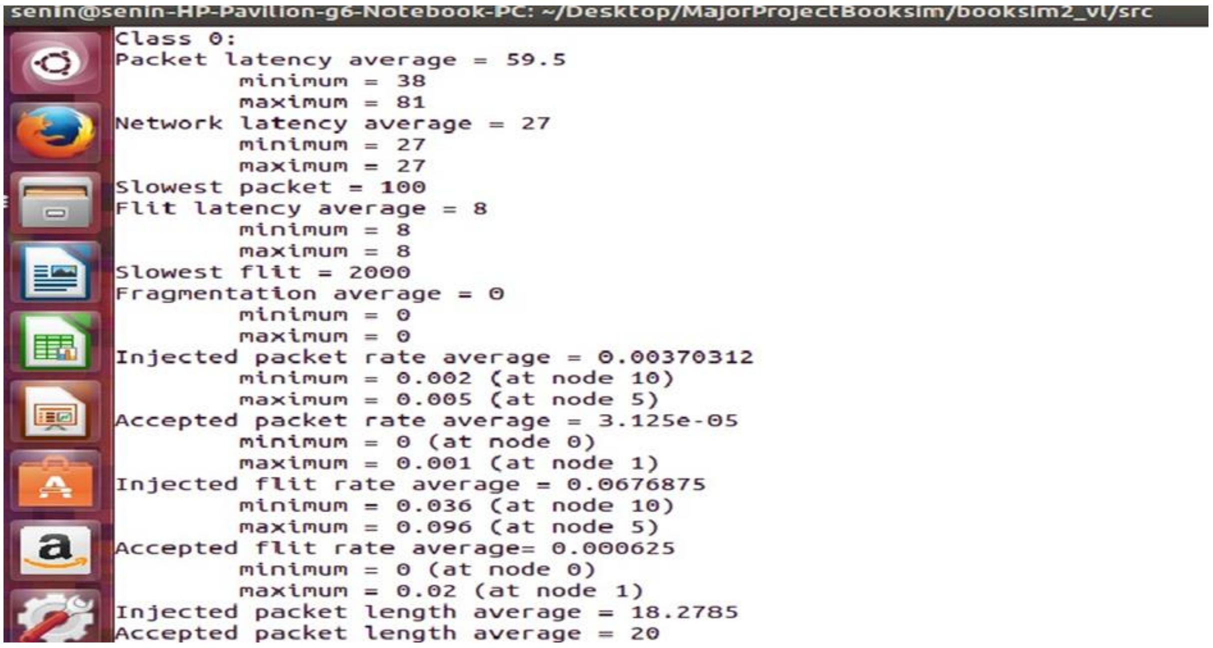
Results (8,8,1) configuration for different injection rates



```
senin@senin-HP-Pavilion-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= 0.0003125
        minimum = 0 (at node 0)
```


senin@senin-HP-Pavilion-g6-Notebook-PC: ~/Desktop/MajorProjectBooksim/booksim2_vl/src

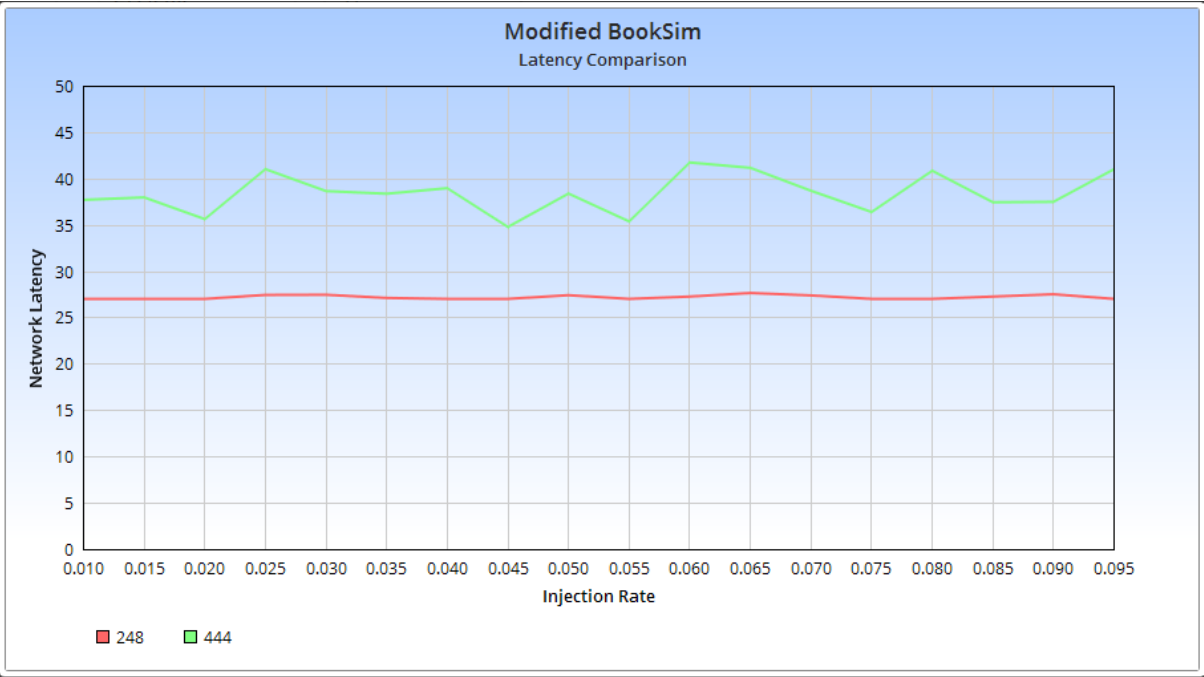
```
Class 0:
Packet latency average = 142.667
        minimum = 43
        maximum = 332
Network latency average = 27
        minimum = 27
        maximum = 27
Slowest packet = 69
Flit latency average = 8
        minimum = 8
        maximum = 8
Slowest flit = 1380
Fragmentation average = 0
        minimum = 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

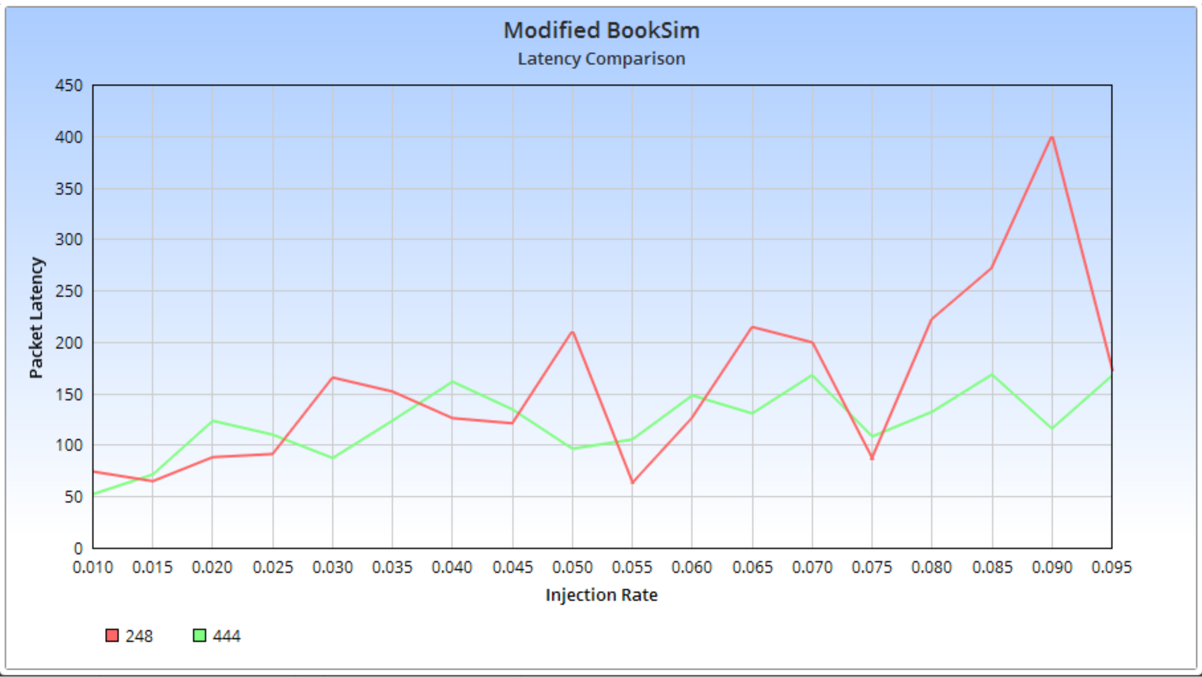
```
Class 0:
Packet latency average = 59.5
        minimum = 38
        maximum = 81
Network latency average = 27
        minimum = 27
        maximum = 27
Slowest packet = 100
Flit latency average = 8
        minimum = 8
        maximum = 8
Slowest flit = 2000
Fragmentation average = 0
        minimum = 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

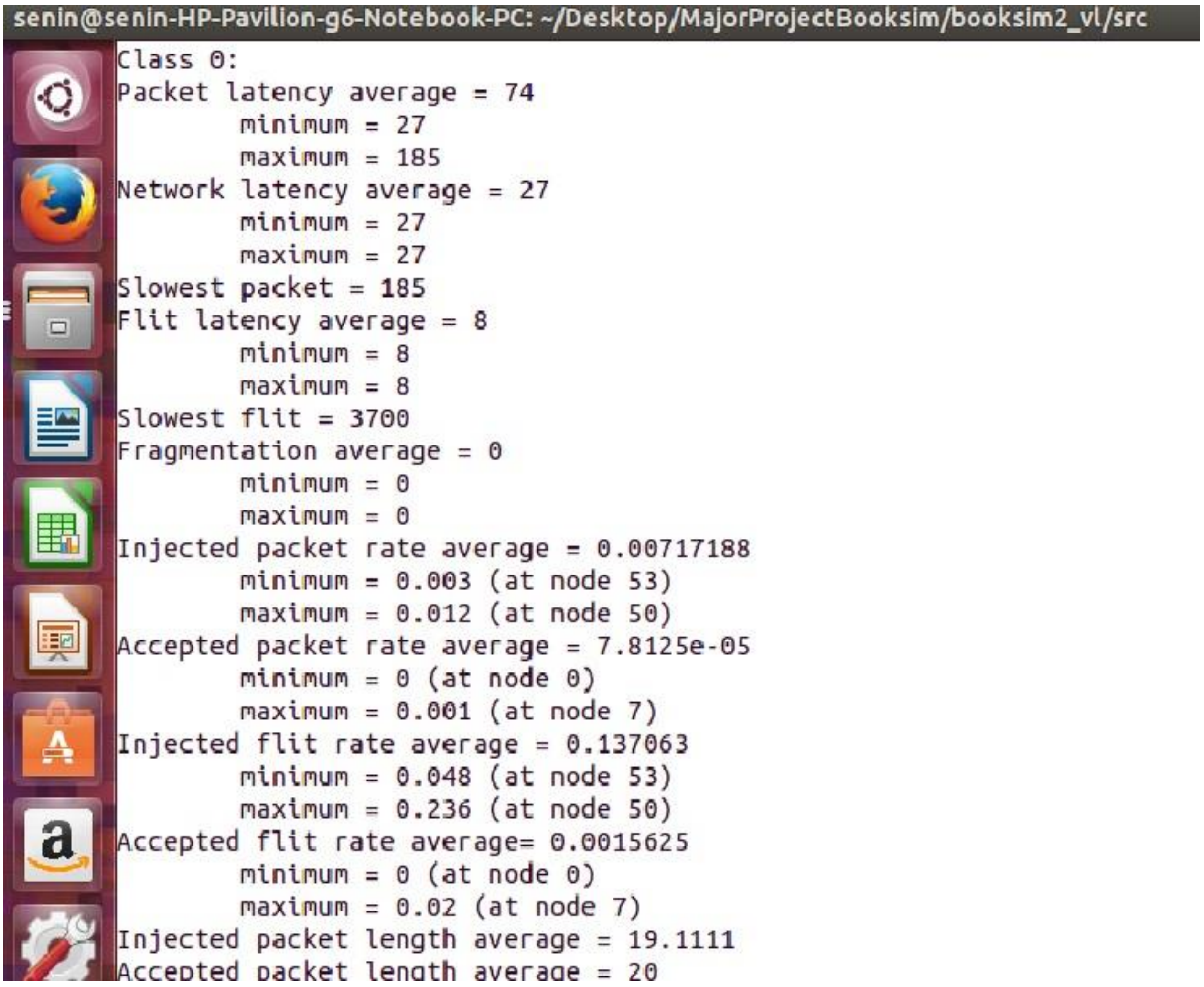


Comparison of Packet Latency

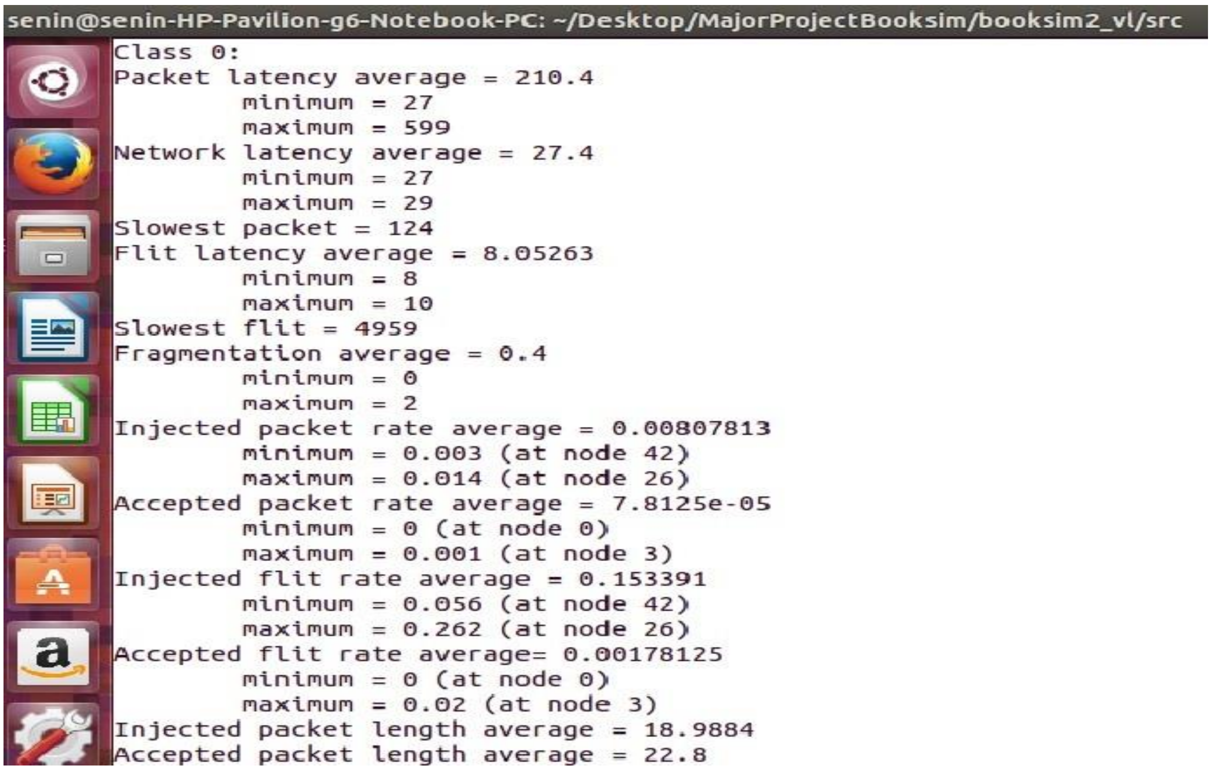
From the above results we can observe that **network latency is lower for (2,4,8)** configuration compared to (4,4,4) configuration.

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.

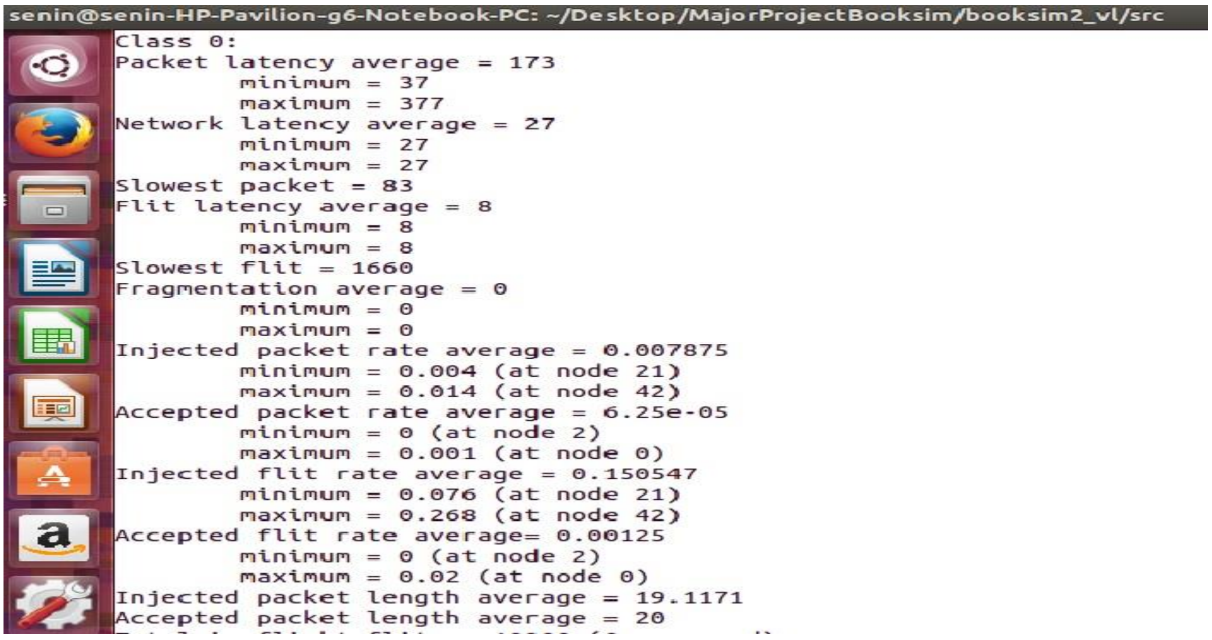
Results (2,4,8) configuration for different injection rates



```
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 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 nanophotonics.
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