

Electronic Theses and Dissertations, 2004-2019

2005

Off-chip Communications Architectures For High Throughput Network Processors

Jacob Engel
University of Central Florida

 Part of the [Computer Engineering Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Engel, Jacob, "Off-chip Communications Architectures For High Throughput Network Processors" (2005). *Electronic Theses and Dissertations, 2004-2019*. 550.
<https://stars.library.ucf.edu/etd/550>

OFF-CHIP COMMUNICATIONS ARCHITECTURES FOR HIGH THROUGHPUT
NETWORK PROCESSORS

by

JACOB ENGEL
BSCpE University of Central Florida, 2001
MSCpE University of Central Florida, 2003

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Engineering
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Fall Term
2005

Major Professor:
Dr. Taskin Kocak

© 2005 Jacob Engel

ABSTRACT

In this work, we present off-chip communications architectures for line cards to increase the throughput of the currently used memory system. In recent years there is a significant increase in memory bandwidth demand on line cards as a result of higher line rates, an increase in deep packet inspection operations and an unstoppable expansion in lookup tables. As line-rate data and NPU processing power increase, memory access time becomes the main system bottleneck during data store/retrieve operations. The growing demand for memory bandwidth contrasts the notion of indirect interconnect methodologies. Moreover, solutions to the memory bandwidth bottleneck are limited by physical constraints such as area and NPU I/O pins. Therefore, indirect interconnects are replaced with direct, packet-based networks such as mesh, torus or k -ary n -cubes. We investigate multiple k -ary n -cube based interconnects and propose two variations of 2-ary 3-cube interconnect called the 3D-bus and 3D-mesh.

All of the k -ary n -cube interconnects include multiple, highly efficient techniques to route, switch, and control packet flows in order to minimize congestion spots and packet loss. We explore the tradeoffs between implementation constraints and performance. We also developed an event-driven, interconnect simulation framework to evaluate the performance of packet-based off-chip k -ary n -cube interconnect architectures for line cards. The simulator uses the state-of-the-art software design techniques to provide the user with a flexible yet robust tool, that can emulate multiple interconnect architectures under non-uniform traffic patterns. Moreover, the simulator offers the user with full control over network parameters, performance enhancing features and

simulation time frames that make the platform as identical as possible to the real line card physical and functional properties.

By using our network simulator, we reveal the best processor-memory configuration, out of multiple configurations, that achieves optimal performance. Moreover, we explore how network enhancement techniques such as virtual channels and sub-channeling improve network latency and throughput. Our performance results show that k -ary n -cube topologies, and especially our modified version of 2-ary 3-cube interconnect - the 3D-mesh, significantly outperform existing line card interconnects and are able to sustain higher traffic loads. The flow control mechanism proved to extensively reduce hot-spots, load-balance areas of high traffic rate and achieve low transmission failure rate. Moreover, it can scale to adopt more memories and/or processors and as a result to increase the line card's processing power.

To my parents,

ACKNOWLEDGMENTS

I would like to dedicate this to my parents, Aaron & Ofra Engel, who always encouraged me to aim high and succeed in life. Without their support none of this would be possible. I am also grateful to my wife, April, who inspires me to do my best and has given me the emotional support to accomplish my goals.

I would like to thank my advisor, Taskin Kocak, who taught me the methodology, discipline, and direction to do my work. Dr. Kocak urged me to always excel and to focus on what is most important. I am also very appreciative to my committee members, Dr. Georgiopoulos, Dr. Chatterjee, Dr. Zhou and Dr. Necati for their suggestions and comments which helped me improve my work.

I would like to thank Danny Lacks, my close friend and former UCF Engineering student, who gave up a lot of his valuable time to assist me with implementing the network simulator used in this research. Danny was very dedicated to making sure that the simulator would result in the most accurate and quality product.

TABLE OF CONTENTS

| | |
|--|-------|
| LIST OF FIGURES | xi |
| LIST OF TABLES | xvii |
| LIST OF ACRONYMS/ABBREVIATIONS | xviii |
| CHAPTER 1: INTRODUCTION | 1 |
| 1.1 Motivation | 3 |
| 1.2 Design constraints issues (off-chip vs. on-chip) | 4 |
| 1.3 Interconnect systems | 7 |
| 1.3.1 State-of-the-art in interconnect networks | 9 |
| 1.4 Related work..... | 11 |
| 1.4.1 K-ary n-cube network..... | 11 |
| 1.4.2 Switching..... | 13 |
| 1.4.3 Routing mechanism | 15 |
| 1.4.4 Deadlock and livelock | 16 |
| 1.4.5 Adaptive routing algorithms..... | 18 |
| 1.4.6 Cray 3TE and Caltech cosmic cube | 20 |
| 1.4.7 NS-2, Qualnet and OPNET network simulation frameworks..... | 21 |
| CHAPTER 2: NETWORK PROCESSOR BACKGROUND | 24 |
| 2.1 Architectural design approaches | 24 |
| 2.1.1 Hardware-oriented techniques..... | 25 |
| 2.1.2 Software-oriented techniques | 28 |

| | |
|--|----|
| 2.1.3 NPU comparison tables | 29 |
| CHAPTER 3: K-ARY N-CUBE BASED ARCHITECTURES | 32 |
| 3.1 K-ary n-cube interconnect structures | 32 |
| 3.1.1 Routing mechanism | 33 |
| 3.1.2 Switching mechanism..... | 38 |
| 3.1.3 The traffic controller | 40 |
| 3.2 3D-mesh interconnect architecture..... | 44 |
| 3.3 The 3D-bus architecture..... | 48 |
| 3.3.1 Bus interfaces | 49 |
| CHAPTER 4: ANALYTICAL PERFORMANCE ANALYSIS | 51 |
| 4.1 Performance metrics..... | 51 |
| 4.1.1 Distribution of IP-packet length in core routers | 52 |
| 4.2 K-ary n-cube latency equation under uniform traffic load..... | 53 |
| 4.3 Latency of packet switched multi-processor shared-bus..... | 55 |
| 4.3.1 M/D/1 queue characteristic equations | 58 |
| 4.4 Performance results k-ary n-cube interconnect vs. shared-bus | 60 |
| 4.5 Average distance of 8-ary 2-cube network and 4-ary 3-cube interconnects with multiple configurations..... | 62 |
| 4.6 Analytical model of k-ary n-cube interconnect with hot-spot traffic..... | 66 |
| 4.7 Performance results of k-ary n-cube interconnect with hot-spot traffic..... | 71 |
| 4.8 3D-mesh performance analysis | 75 |
| 4.8.1 Cube notation | 75 |
| 4.8.2 3D-mesh average distance analysis under non-uniform traffic | 76 |

| | |
|--|-----|
| 4.8.3 3D-mesh latency equation | 80 |
| 4.8.4 3D-mesh interconnect vs. shared-bus | 81 |
| 4.8.5 3D-mesh interconnect vs. folded-torus network | 86 |
| 4.9 3D-bus performance analysis | 90 |
| 4.9.1 3D-bus vs. shared-bus | 90 |
| 4.10 Memory bandwidth | 91 |
| 4.11 Area analysis..... | 92 |
| CHAPTER 5: EXPERIMENTAL RESULTS..... | 95 |
| 5.1 K-ary n-cube simulation framework..... | 95 |
| 5.1.1 K-ary n-cube interconnect simulator architecture..... | 96 |
| 5.1.2 Simulator modeling approach..... | 98 |
| 5.1.3 Software components..... | 103 |
| 5.1.4 Optimization strategies..... | 106 |
| 5.1.4.1 The singleton class..... | 106 |
| 5.1.4.2 Pure virtual functions..... | 108 |
| 5.1.4.3 System design with Standard Template Library (STL) functions.. | 109 |
| 5.1.5 Simulator parameters..... | 111 |
| 5.2 3D-mesh, 4-ary 3-cube, & 8-ary 2-cube simulation results..... | 115 |
| 5.2.1 Latency and throughput analysis..... | 115 |
| 5.2.2 Worm allocation and distribution..... | 118 |
| 5.2.3 Routing accuracy..... | 121 |
| 5.2.4 Interconnect and bandwidth utilization..... | 125 |
| 5.2.5 Failure rate..... | 127 |

| | |
|---|-----|
| 5.2.6 Routing accuracy vs. hot-spot nodes..... | 131 |
| 5.2.7 K-ary n-cube interconnects performance comparison with common interconnects..... | 133 |
| 5.3 3D-bus Simulation Results..... | 135 |
| 5.3.1 Latency of 3D-bus vs. shared-bus..... | 135 |
| 5.3.2 Throughput of 3D-bus vs. shared-bus..... | 136 |
| 5.3.3 3D-bus routing accuracy..... | 137 |
| 5.3.4 3D-bus failure rate..... | 138 |
| 5.3.5 3D-bus latency with memory and PE interfaces..... | 140 |
| 5.3.6 3D-bus performance comparison with common interconnects..... | 141 |
| CHAPTER 6: RESEARCH CONTRIBUTIONS & FUTURE WORK..... | 143 |
| APPENDIX: NETWORK SIMULATOR MANUAL..... | 146 |
| Simulator menus..... | 148 |
| Choosing interconnect type..... | 151 |
| Setting network properties..... | 152 |
| Running simulation..... | 153 |
| Managing input/output files..... | 155 |
| LIST OF REFERENCES..... | 163 |

LIST OF FIGURES

| | |
|---|----|
| Figure 1.1: a) 4-ary 1-cube network b) 4-ary 2-cube network c) 4-ary 3-cube network | 11 |
| Figure 2.1: Block diagram for a pipeline based packet processing | 26 |
| Figure 2.2: Task-level parallelism | 27 |
| Figure 2.3: Packet-level parallelism | 27 |
| Figure 3.1: K -ary n -cube architecture on the line card | 32 |
| Figure 3.2: Message segmentation | 34 |
| Figure 3.3: Message timing (wormhole routing vs. parallel bus) | 36 |
| Figure 3.4: a) Routing directions and coordinates b) Node connectivity | 37 |
| Figure 3.5: Omega switch configuration | 39 |
| Figure 3.6: Cyclic deadlock prevention | 39 |
| Figure 3.7: Traffic controller | 41 |
| Figure 3.8: PE and Memory interfacing the TC | 42 |
| Figure 3.9: Channel partitioning to 4 sub-channels | 43 |
| Figure 3.10: Virtual channels | 43 |
| Figure 3.11: 3D-mesh structure on the network line card | 44 |
| Figure 3.12: 3D-bus structure on the network line card | 48 |
| Figure 3.13: PE interface | 49 |
| Figure 3.14: Memory interface | 50 |
| Figure 4.1: IP packet length distribution | 52 |
| Figure 4.2: Shared-bus multiple processors with arbitration | 56 |
| Figure 4.3: Shared-bus arbitration | 57 |

| | |
|--|----|
| Figure 4.4: Latency comparison between 4-ary 3-cube and 8-ary 2-cube..... | 61 |
| Figure 4.5: Latency comparison between 4-ary 3-cube and shared-bus..... | 62 |
| Figure 4.6: 8-ary 2-cube: a) Configuration 1 b) Configuration 2..... | 63 |
| Figure 4.7: 8-ary 2-cube: a) Configuration 3 b) Configuration 4 c) Configuration 5..... | 63 |
| Figure 4.8: 4-ary 3-cube network..... | 64 |
| Figure 4.9: 4-ary 3-cube: a) configuration 1 b) configuration 2 c) configuration 3..... | 64 |
| Figure 4.10: 4-ary 3-cube: d) configuration 4 e) configuration 5..... | 65 |
| Figure 4.11: 4-ary 3-cube: f) configuration 6 g) configuration 7..... | 65 |
| Figure 4.12: Latency comparison between SB and 8-ary 2-cube..... | 71 |
| Figure 4.13: 8-ary 2-cube with HS..... | 72 |
| Figure 4.14: 8-ary 2-cube with HS vs. shared-bus..... | 73 |
| Figure 4.15: 8-ary 2-cube vs. 4-ary 3-cube vs. shared bus (32 bits)..... | 73 |
| Figure 4.16: Latency comparison between 8-ary 2-cube and 4-ary 3-cube HS traffic with different D_{ave} | 74 |
| Figure 4.17: 3D-mesh interconnection notation..... | 75 |
| Figure 4.18: 3D-Mesh faces..... | 76 |
| Figure 4.19: Non-uniform traffic for PEs in configuration 1..... | 78 |
| Figure 4.20: Non-uniform traffic for M in configuration 1..... | 78 |
| Figure 4.21: 3D-mesh: a) Configuration 1 b) Configuration 2..... | 79 |
| Figure 4.22: 3D-mesh: a) Configuration 3 b) Configuration 4..... | 79 |
| Figure 4.23: 3D-mesh: a) Configuration 5 b) Configuration 6..... | 79 |
| Figure 4.24: Latency SB vs. 3D-mesh with $ch_w=32b$ (1 inch spacing)..... | 82 |
| Figure 4.25: Latency SB vs. 3D-mesh with $ch_w=64b$ (1 inch spacing)..... | 83 |

| | |
|--|-----|
| Figure 4.26: Latency comparison SB vs. 3D-mesh for $ch_w=32b, 64b$ (1 inch spacing)... | 83 |
| Figure 4.27: Latency ratio SB/3D-mesh for $ch_w=32b, 64b$ (1 inch spacing)..... | 84 |
| Figure 4.28: Latency SB vs. 3D-mesh with $ch_w=32b$ (0.5 inch spacing)..... | 85 |
| Figure 4.29: Latency SB vs. 3D-mesh with $ch_w=64b$ (0.5 inch spacing)..... | 85 |
| Figure 4.30: Latency ratio SB/3D-mesh for $ch_w=32b, 64b$ (0.5 inch spacing)..... | 86 |
| Figure 4.31: 3-Dimensional Torus network..... | 87 |
| Figure 4.32: 4-Dimensional Torus network..... | 87 |
| Figure 4.33: Latency Ratio 3D-mesh vs. folded Torus..... | 89 |
| Figure 4.34: 3D-bus latency vs. shared-bus..... | 91 |
| Figure 5.1: Simulation control modules..... | 96 |
| Figure 5.2: Class relationship diagram..... | 99 |
| Figure 5.3: Cyclical relationship..... | 100 |
| Figure 5.4: Simulation setup..... | 100 |
| Figure 5.5: UML class diagram of interconnect architecture..... | 102 |
| Figure 5.6: Dynamic model of routing algorithm..... | 104 |
| Figure 5.7: Sequence diagram of user-simulator interaction..... | 105 |
| Figure 5.8: <i>WormManager</i> singleton..... | 107 |
| Figure 5.9: <i>Interconnect</i> singleton..... | 107 |
| Figure 5.10: Structural hierarchy of classes: face, node, port..... | 110 |
| Figure 5.11: Generation rate..... | 113 |
| Figure 5.12: Maximum worms in the interconnect (MWII)..... | 114 |
| Figure 5.13: Generation rate and maximum worms in the interconnect..... | 115 |
| Figure 5.14: Latency comparison..... | 116 |

| | |
|---|-----|
| Figure 5.15: Latency vs. offered-load..... | 117 |
| Figure 5.16: Throughput comparison..... | 118 |
| Figure 5.17: Worm allocation & distribution..... | 119 |
| Figure 5.18: Worm allocation & distribution with VC=8KB..... | 120 |
| Figure 5.19: Worm allocation & distribution with SC=4..... | 121 |
| Figure 5.20: 3D-mesh worm deviation from shortest path..... | 122 |
| Figure 5.21: 3D-mesh routing accuracy with VC=8KB and SC=2..... | 123 |
| Figure 5.22: 8-ary 2-cube routing accuracy with VC=8KB and SC=2..... | 124 |
| Figure 5.23: 4-ary 3-cube routing accuracy with VC=8KB and SC=2..... | 124 |
| Figure 5.24: Bandwidth utilization rate..... | 125 |
| Figure 5.25: Interconnect utilization rate..... | 126 |
| Figure 5.26: Failure rate comparison with VC switched on/off..... | 128 |
| Figure 5.27: Worm failure rate with SC=2 switched on/off..... | 129 |
| Figure 5.28: Worm failure rate with SC=4 switched on/off..... | 130 |
| Figure 5.29: Failure rate vs. VC size..... | 130 |
| Figure 5.30: 3D-mesh routing accuracy vs. hot-spot nodes..... | 131 |
| Figure 5.31: 4-ary 3-cube routing accuracy vs. hot-spot nodes..... | 132 |
| Figure 5.32: 8-ary 2-cube routing accuracy vs. hot-spot nodes..... | 133 |
| Figure 5.33: Throughput comparison: k -ary n -cube interconnects vs. common interconnect technologies..... | 134 |
| Figure 5.34: Latency of 3D-bus vs. shared-bus..... | 135 |
| Figure 5.35: Throughput of 3D-bus vs. shared-bus..... | 137 |
| Figure 5.36: 3D-bus average routing accuracy..... | 138 |

| | |
|---|-----|
| Figure 5.37: 3D-bus worm failure rate..... | 139 |
| Figure 5.38: 3D-bus latency with interfaces attached..... | 141 |
| Figure 5.39: 3D-bus throughput comparison with common interconnects..... | 142 |
| Figure A1: a) Main simulator window b) The view menu..... | 148 |
| Figure A2: a) Selecting the properties menu b) Selecting the simulation menu..... | 149 |
| Figure A3: a) Selecting the help menu b) Help menu content..... | 150 |
| Figure A4: a) Worms manual settings b) Source/destination settings..... | 150 |
| Figure A5: a) Network type selection b) Network type and configuration..... | 151 |
| Figure A6: a) 3D-mesh b) 4-ary 3-cube c) 8-ary 2-cube..... | 151 |
| Figure A7: a) Network properties selection b) Network properties menu..... | 153 |
| Figure A8: a) Selecting the sampling menu b) Sampling menu content..... | 154 |
| Figure A9: a) Simulation pacing menu b) Simulation pacing setup..... | 155 |
| Figure A10: a) Simulation pacing overdrive b) Simulation pause on/off..... | 155 |
| Figure A11: a) Import worms from file b) Export worms to a file..... | 156 |
| Figure A12: a) Worm modeling data b) Simulation properties..... | 157 |
| Figure A13: Worm run-time data..... | 158 |
| Figure A14: Worm run time data – cont..... | 158 |
| Figure A15: Worm run time data – cont..... | 158 |
| Figure A16: Modeled worms data..... | 159 |
| Figure A17: Modeled worms data - cont..... | 160 |
| Figure A18: Modeled worms data - cont..... | 160 |
| Figure A19: Interconnects configuration file..... | 161 |
| Figure A20: Interconnects configuration file - cont..... | 162 |

Figure A21: Simulator configuration file - cont.....162

LIST OF TABLES

| | |
|---|----|
| Table 1.1: Interconnect Technology Overview | 10 |
| Table 2.1: NPU system architecture comparison..... | 30 |
| Table 2.2: NPU performance comparison..... | 31 |
| Table 3.1: Comparison between shared-bus, crossbar & 3D-mesh..... | 46 |
| Table 4.1: Average distance of 8-ary 2-cube for different configurations..... | 63 |
| Table 4.2: Average distance of 4-ary 3-cube for different configurations..... | 65 |
| Table 4.3: Average distance of 3D-mesh for different configurations..... | 80 |

LIST OF ACRONYMS/ABBREVIATIONS

| | |
|----------|-----------------------------------|
| NPU | network processor unit |
| PE | processing element |
| VC | virtual channels |
| <i>g</i> | generation rate |
| MWII | maximum worms in the interconnect |
| GPP | general purpose processor |
| TC | traffic controller |
| CDG | channel dependence graph |
| VLIW | very long instruction word |
| Co-P | co-processor |
| TM | traffic manager |
| HWA | hardware accelerator |
| QoS | quality of service |
| LRU | least recently used |
| RDC | rotating daisy chain |
| FCFS | first comes first served |

CHAPTER 1: INTRODUCTION

Today, routers and switches are the fundamental equipments for the overall network infrastructure. These devices provide the functionality to receive, decode, repack, and switch packets of data within the network. The basic hardware building block for a mid-end to high-end router or switch is the line card. On one end, the line card is linked to incoming traffic through a port, and on the other end, it is linked to central processor cards and/or line cards via the backplane. Currently, two major trends are impacting the architecture and design of line cards. First, line cards are handling more functions to support new services such as quality of service (QoS) and policy management, which increases the traffic overhead to incoming line rates by 40%-100%. This, in turn, raises memory, chip interconnect and back plane bandwidth requirements. Second, ever increasing memory capacity requirements which are due to higher link rates and unstoppable expansion in lookup tables.

Network line rates are constantly increasing, currently reaching 40 Gb/sec. For example, at this rate, a 40 byte packet arrives every 8 ns. At higher rates developers will face new memory challenges for packet buffering, network parameters storage in the form of long memory access times also known as the “memory wall”. The nature of packet transmission (such as randomness of arrival, variable packet size, and out of order transmission) has surpassed the I/O bandwidth capabilities of most conventional high speed DRAM chips.

Current network processors or network processing units (NPUs) use multithreading to hide memory latency. However, it is not clear whether this technique

will scale well at higher line rates. This is especially questionable for deep packet processing required by security and active networking applications. Stateful networking applications aggravate the problem since per-flow state information needs to be maintained through buffers to achieve per-flow bandwidth and delay guarantees.

The trend in network processing is towards layer 4-7 processing in the routers and switches. Seven-layer processing places a heavy strain on memory usage and aggravates the gap between the network and memory performance. Deep packet processing implies reading and writing significantly more data from/to the memory. More fields and headers are processed to access higher layers and more data are processed within each layer. The heavily stressed memory is used by network processors for two main tasks: packet storage (buffering) and lookup table searches. Packet buffer memory is accessed at least four times per packet. Therefore, in order to sustain wire-speed performance the buffer memory should be able to provide at least four times the bandwidth of the network link.

Theoretical bandwidth of memory devices is commonly accepted to be twice the effective bandwidth. This is mainly a result of using bursts for memory access and consequently, using bandwidth on irrelevant information. With DRAM technology, clock cycles may also be under utilized due to its refresh cycles. Accordingly, buffer memory should provide not four, but eight times the link bandwidth. In 10 Gigabit/OC-192 environments this implies a required buffer memory bandwidth of 80 Gbps. Yet, as networks are full duplex by nature, generating a total of 20 Gbps traffic, the total memory bandwidth needed for sustaining wire-speed is 160 Gbps [89]. Since network processors use memory for both packet storage and lookup tables searches, the total memory bandwidth required for 10 Gbps 7-layer packet processing is therefore approximately 320

Gbps (160 Gbps + 160 Gbps), presenting a tremendous challenge for network processor designs.

In this work, we present a network-on-a-board (NOB) to increase the bandwidth for chip-to-chip communications on printed circuit boards. Network line cards are chosen to demonstrate the application of the proposed scheme since a line card, probably, requires the fastest interconnect among its chips. NOB in this work is based on well-known k -ary n -cube interconnection networks. The originality of this work is the application of this class of networks on to boards. Mapping this type of networks into two dimensional board domain is not a straightforward task. Moreover, board-level implementation brings a lot of constraints such as space, wire length, manufacturability of small communications controllers or switches at the nodes, which are not considered in designing k -ary n -cube networks for multi computers.

1.1 Motivation

Our research deals with an off-chip interconnect that provides an effective solution, under certain constraints, to the increasing demand for memory bandwidth on line cards. In most line card architectures there exist a direct interconnect, such as busses or switches that connects different processing elements to memory modules. The heart of the line card is the network processor which performs different operations in order to analyze the flow of incoming packets. The nature of packet processing requires frequent read/write operations to memories which are distributed around the NPU.

As line-rate data and NPU processing power increase, memory access time becomes the main system bottleneck during data store/retrieve operations. The growing demand for memory bandwidth contrasts the notion of indirect interconnect methodologies and requires them to be replaced with direct, packet-based networks such as mesh, Torus or k -ary n -cubes.

A shared bus cannot scale well as the number of modules (processing elements or memories) connected to it increases. In addition, it requires an arbitration mechanism that becomes distributed (rather than centralized) as the number of modules connected to the bus grows. Solutions to the memory bandwidth bottleneck are limited by area on the line card and NPU I/O pins. Pin constraints bound the bus size that can be interfaced with the NPU. Hence, only a packet-based network-on-board can provide the required performance improvement between the NPU and off-chip memory modules.

1.2 Design constraints issues (off-chip vs. on-chip)

On- and off-chip interconnects are attracting more attention as chip scaling continues to shrink. Protocols, signaling and scaling technologies facilitate system-on-chip designs which can be incorporated into on-chip communication networks. Because multiprocessor networks require different network functionalities, tasks and data transfer properties, design space for new architectures is explored. On-chip networks have been researched substantially. Off-chip interconnects, on the other hand, lack innovative methods and performance analysis results that can improve and abbreviate their integration into large, scalable, network components such as core routers. Off-chip

interconnects are as crucial as on-chip networks for achieving high-performance, low-cost networks.

The area on PCB is limited by the physical dimensions and data lines routing and therefore, the total number of bus lines for any bus interconnect (or bus width in bits) has space limitations. Processing elements such as traffic manager, classification engine or the NPU itself are limited by their available I/O pins. Hence there exists an upper limit on the bus width (in bits) that can be connected to these chips. The physical and electrical characteristics of wires routed on PCB for off-chip communications are different than those of wires routed on-chip. Wires routed on board are longer and larger in size than those which are laid out on chip. Off-chip wires have higher capacitance and that results in higher latency.

All interconnects require switching elements that enable the connection of input to output devices. Switching elements are usually CMOS transistors embedded on a chip or as a stand alone chip. An on-chip switching mechanism takes less space and allows more switching elements than off-chip. Off-chip switches take more space since it cannot be embedded directly on the board and requires a separate package. In addition, adding switching modules will increase manufacturing costs to the design.

On-chip interconnects differ from off-chip networks mainly in their resource constraints (tighter on chip) and synchronization. In a network, data sent from a source to a destination may arrive out of order due to reordering in network nodes, following different routes, or retransmission after dropping. For off-chip networks out-of-order data delivery is typical. However, for on-chip network where no data is dropped, data can be forced to follow the same path between a source and a destination (deterministic routing)

with no reordering [44]. Off-chip networks typically use packet switching and offer best-effort services. When no flow control is provided, messages are dropped when buffers overflow. Multiple policies of dropping messages are possible such as the oldest message is dropped or the newest message is dropped.

Performance measures, in terms of latency and throughput, play a major role in selecting an off-chip interconnect. The interconnect is required to transfer large amount of data without incurring additional delays as a result of wire propagation, switching latency and contention among devices. Another factor in interconnect performance is choosing a routing algorithm that will select the best path for the packet flow to minimize latency. Under any design constraints, latency should not have a higher value than the memory access time.

Line cards are required to keep up with increasing data line rates and deep packet processing of incoming data. As a result, the line card continuously needs an upgrade in order to scale the system to growing performance demands. Scalability is not possible on-chip since the layout is set at the time of manufacturing. An off-chip interconnect can better scale in order to increase the number of modules connected to it. Scalability in an off-chip design is possible since the interconnect and its routing algorithm can dynamically be configured to adopt new devices and change its configurability to add new local bus links and switches.

1.3 Interconnect systems

Before we discuss the current interconnect systems, let us give a brief definition of terms. We follow the terminology given in [64]. There are four major characteristics of interconnect networks. The first is the interconnect topology. Topology is the physical structure of the network and depicts how nodes are being connected. Nodes can be connected directly, to every switch in the network, or indirectly, which means that a certain node is connected only to specific number of nodes.

The second interconnect characteristics is the network switching strategy. Switching strategy defines how a message traverses a route. Switching is the process of determining how source and destination will be coupled to allow data transfer between them through the interconnect structure. There are three types of switching mechanisms of data from source to destination: circuit-switching, message-switching and packet-switching flow control. In circuit-based switching the system reserves a dedicated circuit (path) and then transmits data through it. In message-based switching the system stores the complete message at each node (switch) and then continues to forward it to the next intermediate switch until the message reaches its destination. In packet-based switching, the message is partitioned into smaller, fixed size packets and sends them independently. Packets comprised of the same message follow the same path determined by the routing algorithm. Packets of the same message move in a pipelined-fashion through the interconnect switches. The main advantage that a packet-switching mechanism has over other methods is that it uses the available bandwidth efficiently by sharing the interconnect links at all times with packets of other messages.

Routing algorithm is the third characteristics of an interconnect network. Routing is the mechanism of forwarding packets inside the interconnect following certain paths or patterns. The availability of paths within the interconnect network determines its performance factors such as latency and throughput and is influenced by factors such as rate of packets delivered into the interconnect, total number of channels, channel width, number of nodes and more.

The fourth characteristics of an interconnect network is its flow control mechanism. Flow control mechanism defines the operations taken when the flow of packets encounters a deadlock, points of heavy traffic or link fault. Examples of major flow control processes include virtual channels (VC), deadlock avoidance mechanisms, and throttling.

Interconnects may be classified into three categories based on their purpose and where they reside: Processor, mezzazine, and local area. The processor interconnect is typically used to connect the processor to another component that contains a memory controller and one or more mezzanine level interconnection ports. A mezzanine interconnect generally employs an address/data read/write data model with memory-like semantics and is targeted for simple translation between processor bus memory operations and mezzanine interconnects transactions [37]. PCI bus is the most well known example to this category. Local area interconnects are used between the public access networks and mezzanine interconnects. In this work, we are primarily interested in mezzanine interconnects (MIs). There are three types of MIs: Shared parallel, switched parallel, and switched serial. The difference between shared and switched is based on an interface's support for multi-drop capability or point-to-point connection, respectively.

The difference between serial and parallel is based on whether an interconnect is self clocked or source clocked, respectively. Shared parallel interconnects are relatively easy to implement, however they pose major problems such as electrical limitation, speed, reliability, scalability, and physical distance (clock skew). Examples to this category include *VME* [14] and *PCI* [69]. Most of these problems except the signal skew can be addressed by switched parallel interconnects. Several new standards supporting this mode are introduced recently such as *HyperTransport* [72] and *RapidIO* [74]. Migration from parallel to serial solves one important issue, the physical reach. *Infiniband* [70] and *PCI Express* (PCIe) [73] are major examples to this category.

1.3.1 State-of-the-art in interconnect networks

In this work, we are primarily interested in interconnect systems that are used for network line card applications. In Table 1.1, we give an overview of current interconnects for these applications along with a comparison to one of our interconnect candidates, the 3D-mesh architecture. Note that we exclude *RapidIO* and *SPI* in this table. *RapidIO* is mostly used for signal processing and control-plane applications, and *SPI* [71] is a network interface but not a system interconnect. Also, Network Processing Forum is introducing several interfaces and interconnects which are built on *CSIX* [24]. Hence, we only cover *CSIX* in the table.

Table 1.1: Interconnect Technology Overview

| Technology | Hypertransport | PCI Express | CSIX | 3D-mesh |
|-----------------------------|--------------------------|-----------------------------|-------------------------|-------------------------|
| Channel width (bits) | 2, 4, 8, 16, 32 | 1, 2, 4, 8, 16, 32 | 8, 16, 32 | 1, 8, 16, 32 |
| Bandwidth | 102.4 Gbps | 128 Gbps | 10 Gbps | 452 Gbps |
| Signaling | Differential | LVDS | LVTTL | LVDS |
| Layers | Not layered architecture | Physical, link, transaction | Physical, link, logical | Physical, link, logical |
| PCI transparency | No | Yes | Yes | No |
| Switched arch. | No | Yes | Yes | Yes |
| Packet based | Yes | Yes | Yes | Yes |
| Payload size | 4-64 B | 1 KB | 256 B | 40 B-2 KB |
| Pins/channel (w=16b) | 103 | 90 | 35 | 35 |

1.4 Related work

The type of interconnect networks we include in this work are classified as direct networks. A direct network is defined as a network in which all nodes are connected together by links. Direct networks are classified according to their topologies and communication techniques (switching, routing, flow control).

1.4.1 *K*-ary *n*-cube network

A *k*-ary *n*-cube network is a directed network topology. It consists of $N = k^n$ nodes where *n* is the dimension of the network and *k* represents the number of nodes in each dimension of the structure. Each node in *k*-ary *n*-cube interconnect is uniquely labeled and elements of the same plane are connected to each other.

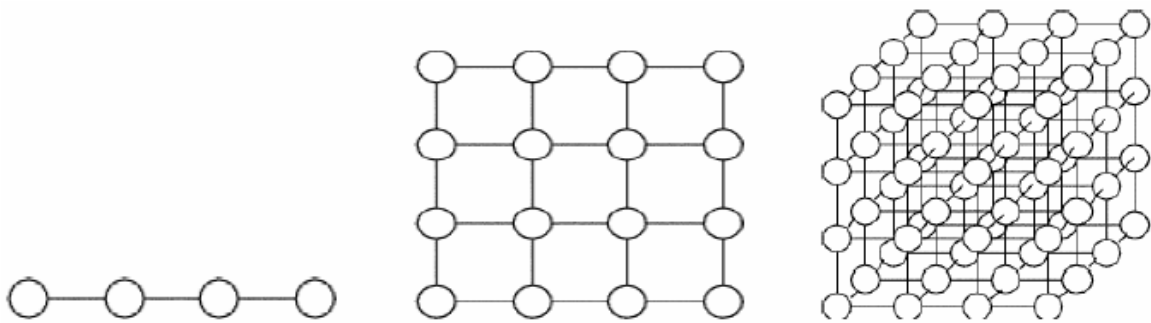


Figure 1.1: a) 4-ary 1-cube network b) 4-ary 2-cube network c) 4-ary 3-cube network

One of the most intriguing tasks in designing these networks is to find the optimal value of *n* and *k* to achieve the best performance. The optimal choice depends on many design constraints such as channel width/density, number of elements connected to the

network, and cost. The network's wire bisection width is defined as the minimum number of wires to be cut when the network is divided into two equal parts. The network wire bisection width is depended on the channel width and network size. Moreover, the wire bisection width factor has a great importance in determining the cost of the network since it is depended on the layout technology, power dissipation, and system cost.

For a k -ary n -cube network the wire bisection width is $B = k^{(n-1)}$. For example, in 4-ary 2-cube network the wire bisection width equals 4. It means that the number of wires cut when dissecting the network into two equal parts is 4 (assuming there is only one wire connecting two adjacent nodes). If the number of wires connecting two adjacent nodes is higher than one then it is defined as channel width. Practically, channel width size is constrained by multiple factors such as node size, network dimensions, number of pins available and layout size. The bisection width of 3D interconnect proposed in this work is equal to $B=2^{(3-1)}=4$. This is based on the fact that the interconnect is a 2-ary 3-cube network. In addition, in this particular calculation we assume that the channel size is only 1-bit. The interconnect model includes an option which allows variable channel widths (1, 8, 16, 32, 64, 128). The channel width is pre-configured by the user. For example, the bisection width for an 8-bit channel width is $B=4*8=32$. As the bisection width increases, routability and performance increase as well.

Torus network is an implementation of k -ary n -cube network with wraparound channels. Torus networks are symmetrical. The wraparound paths of Torus networks reduce channel congestion and average distance while introducing additional wiring and more chances for deadlock occurrences. Mesh networks are another type of k -ary n -cube interconnects. Mesh networks are structured with or without wraparound channels and

are simpler than Torus. In mesh networks not all edges are connected to neighbor nodes and therefore it allows easier integration to external processing elements. Channel utilization in mesh networks is lower than Torus since channels in the center of the mesh network are used more than corner nodes. Hypercubes are another instance of k -ary n -cube networks in which $k=2$. Hypercubes increase only in network dimensions, n .

A study done by Dally in [12], demonstrated that low-dimensional networks provide better performance than high-dimensional networks. It also stated that low-dimensional networks are favored since wire length increases with the network dimension. Dally introduced a low-dimensional k -ary n -cube structure called express cube [9]. Express cube is a superposition of mesh and Torus networks. Express cube provide short cuts for messages traveling long distances. A message destined to a far node is routed through high-level channels instead of being routed through intermediate nodes and as a result reduces the latency significantly. The main disadvantage of express cube is its reduced bisection bandwidth which affects its performance by locality of applications.

1.4.2 Switching

Switching is the mechanism which determines when a message flow along a certain path changes direction to new route with more available resources. The main switching techniques include circuit switching, message switching and packet switching.

In circuit switching the entire path from source to destination is reserved until the complete message has reached its destination. Circuit switching guarantees full

bandwidth usage per flow but increases latency and wastes network bandwidth when source/destination modules are idle. Message switching (also known as store-and-forward) each message contains its routing information. A message is a single entity which moves from one intermediate node to another until it reaches destination. This technique increases network latency since it requires a complete storage space for the message in each node. To overcome this, another switching technique was developed called: packet switching.

Packet switching scheme delivers a message in smaller units called packets. Packets of the same flow follow each other in a pipeline fashion following the first packet (header) which holds source/destination data. Multiple hybrid schemes were developed as well to utilize the advantages of two or more switching techniques. An example is the virtual-cut-through switching scheme presented by Kermani and Kleinrock [30]. In virtual-cut-through when a message arrives at a node where one or more of its output channels is available the message will not be buffered at all and will be immediately forwarded to continue without buffering overhead. Dally and Seitz [28] developed wormhole routing technique which reduces storage requirements even more. The main difference between virtual-cut-through and wormhole routing scheme is when a message path is blocked wormhole routing stops the message in place and does not buffer the complete message. More details about wormhole routing are given in chapter 3.

1.4.3 Routing mechanism

Routing algorithm is a restricted set of paths packets may follow in order to reach destination. There are two main types of routing techniques, adaptive and deterministic (non-adaptive) routing. In deterministic routing packets follow a fixed path between source and destination. This technique usually chooses minimal path to reduce latency and is deadlock free. The main disadvantage of this method is its inability to adapt to changes in traffic load. It is more prone to contention of flows from different sources and its performance degrades as more sources and destinations are added to the system. An example for deterministic routing is the e-cube routing (also called dimension-order routing) in which each message is routed in order of dimension. Change in dimension can be viewed as a change in 3D cube along different axis. Adaptive routing, on the other hand, is a dynamic technique of forwarding packets through multiple switches. Adaptive routing is further classified into minimal or non-minimal paths and maneuvers packets to avoid heavy congested spots. Its main disadvantage is that it has a potential to create deadlocks. Certain methods are using partially adaptive routing which limits the degree of algorithm adaptivity. Fully adaptive routing algorithms allows complete adaptivity (can select any path) to resource utilization within the network.

Dally's 2D mesh adaptive router [6] utilizes a fully adaptive routing scheme. By assigning virtual channels for each y -dimensional link and physical channels for x -dimension channels the network was virtually divided into two 2D mesh networks. Messages moving in the positive x -axis were labeled as eastbound messages and messages moving in the negative x -axis were labeled as westbound messages. Messages

were separately routed in each of the virtual channels. Since messages in each virtual network were routed in $+x$ or $-x$ directions there was no cyclic channel dependency and therefore it avoids deadlock.

The turn-model, introduced by Ni and Glass [8], is an example of partially-adaptive routing algorithm. An adaptive routing algorithm performance and behavior on k -ary n -cube networks was made. The conclusion was that prohibiting some turns on the k -ary n -cube network prevent channel dependency cycles. Wormhole routing was used with little addition, it used misrouting to avoid deadlocks and livelocks. Its main disadvantage is that it creates uneven channel utilization and causes degradation of performance.

1.4.4 Deadlock and livelock

Deadlock is the situation in which a message is stuck forever within the network and is unable to obtain the required resources. Deadlock occurs when a set of messages require a portion of network resources which is held by other messages in the set. Livelock occurs when messages are routed over the channels without deadlock but can never reach their destinations.

Most common deadlock avoidance methodologies include: abort-and-retry, the “connection-machine”, and multistage interconnect networks. Abort-and-retry methodology aborts messages which reach deadlock and resend them into the network after a certain delay. The “connection-machine” is similar to virtual cut-through in which packets are stored and therefore removed from the network for a limited time until

deadlock is cleared. Multistage interconnect networks outperform grids and hypercubes but incorporate more implementation restrictions (physical wiring). The main techniques used to avoid deadlocks include using virtual channels (VC), channel dependence graphs (CDG) and message flow misrouting to break the message cyclic nature.

Dally [1] created a routing function on Torus network which uses minimal paths and virtual channels. The deadlock avoidance approach eliminates arcs which make a packet movement acyclic and hence it prevents deadlock. If packet forwarding is discontinued as a result of its deadlock avoidance mechanism, then it duplicates the links and uses virtual channels instead. Duato [2] methodology is pretty similar to Dally's approach but works for any topology. First, the routing mechanism used is deadlock-free which can duplicate all channels by using virtualization. The routing algorithm can use the original (physical) channel or can take a new path (one of the virtual channels). Once a flow uses one of the paths available (physical or virtual) it cannot revert to new channels. The network eventually reaches a steady-state status where all channels are occupied.

Dally and Seitz [27] introduced another technique which utilizes channel dependency graphs to develop deadlock-free network based on wormhole routing. The channel dependence graph is a directed graph where vertices of the graphs are the communication channels of the network. Deadlock is avoided by breaking cyclic channel dependencies and using virtual channels wherever a channel is broken.

Another popular technique to avoid deadlocks is to allow misrouting of a message in order to break its cyclic movement. A waiting message takes an alternative arbitrary

route which does not conform to its usual routing mechanism. Misrouting has one major disadvantage, it can cause livelock.

Livelock avoidance techniques include a mix of deterministic and adaptive routing algorithms. The use of minimal routing helps to keep the network free from the livelock. For example, in [22], a model for designing livelock-free adaptive routing algorithm for meshes without virtual channels is presented. Chiu [22] proposed the odd-even turn model which restricts the locations where some types of turns can take place such that the algorithm remains deadlock as well as livelock free. The odd-even routing prohibits the east-to-north and east-to-south turns at any tiles located in an even column. It also prohibits the north-to-west and south-to-west turns at any tiles located in an odd column.

Glass and Ni [8], turn-model, provides deadlock and livelock free routing and was discussed in the routing mechanism sub-section.

1.4.5 Adaptive routing algorithms

Adaptive routing algorithms are based on the following characteristics: deadlock and livelock free, minimal hardware requirements and simple to implement. Many adaptive routing algorithms were developed the most successful ones are briefly explained here.

Priority-based non-minimal adaptive routing algorithm was introduced by Ngai and Seitz [7]. Packets are assigned priority based on their “age” (time the packets are routed within the network). Priority changes dynamically as the packet is traveling

through the network on its way to destination. Since packets may be rerouted and as a result must remain in the network longer they are assigned a greater priority and therefore, they are given higher forwarding priority than new packets. The disadvantage of this routing algorithm is its slow speed and complicated priority assignment to packets.

Chaos routing [31] is another non-minimal adaptive routing algorithm in which a packet is randomly selected at the node when misrouting is required. There is no guarantee that packets will reach destination in finite time.

Deflection adaptive routing algorithm [43] guarantees that packet arriving at a node will leave the node within the next routing cycle. It assigns preference to channels which are closer to destination but if the channels are busy it uses misrouting. Its main disadvantage is that it misroute more frequently than Chaos routing.

The *-channel algorithm is an example of minimal adaptive algorithm [5]. The *-channel algorithm was implemented in n -dimensional mesh or Torus networks. Each physical channel is associated with two virtual channels: *-channel and non *-channel. The *-channel is in the i^{th} dimension only and is used by messages which are moving along i dimensions. The non *-channel is used for adaptive routing in higher dimensions. If none of the channels are free it waits unconnected. The routing scheme is more complicated but significantly reduces buffer requirements.

The packet-based routing algorithms reviewed resolve deadlock by two methods: voluntary misrouting and virtual channels. In order for routing algorithm to prevent livelock conditions, it utilizes randomization of channel selection. Wormhole routing makes deadlock prevention more difficult because of resource coupling. The most

common technique used to avoid deadlocks is to split the network into several virtual networks or utilize virtual channels.

1.4.6 Cray 3TE and Caltech cosmic cube

Caltech cosmic cube is a high-speed network characterized by 64-nodes, point-to-point, packet-based, multi processor-memory interconnect infrastructure which utilizes message passing mechanism. Nodes are modeled by 64 small computers interconnected as a 6-dimensional hypercube. Each node is connected through a bidirectional, asynchronous, point-to-point communication channels. Each node there is an operating system kernel which allows to initiate transmission or reception of messages as well as forwarding messages to other nodes. The network does not employ any switching mechanism between nodes (processing nodes to storage nodes). The message passing mechanism is based on store-and-forward protocol. Messages do not “block” the channel but remain pending until the channel becomes idle. In addition, the volume of the system is 6 cubic feet, the power consumption is 700 watts and manufacturing costs were \$80,000 [21]. The Caltech high-throughput network [20] is a distributed grid-based system employing Cisco's 7600 and 6500 series switch routers connected to each other via 10 Gbps dedicated lines. The Caltech network communicates via new fast TCP algorithm developed at Caltech Netlab.

There are two well-known architectures that conceptually and structurally resemble somewhat the work we propose here: Cosmic Cube [21] and Cray 3TE [19]. However, there are a couple of major differences. First, the scope of the interconnect

architectures is different. Nodes in our case are at the chip-level (i.e., processors and memory devices), not boards or cases (i.e., computers). The physical dimensions of our interconnect must be smaller since area on the line card is limited. Second, the application workload shows very different behavior. Unlike data traffic generated in supercomputers, the network traffic is proven to be self-similar and exhibits burstiness. Third, the cost of implementing the interconnect architecture is lower, since its performance superiority and reliability are primarily depend on its routing algorithm and message flow-control not its hardware. This allows us to use simple logic and bus lines between chips making it an inexpensive interconnect to fabricate.

1.4.7 NS-2, Qualnet and OPNET network simulation frameworks

There are many discrete event network simulation and modeling tools available that contain some of the architectural features and functionalities we incorporate in our model. However, to the best of our knowledge, none of these simulation frameworks is capable of delivering the physical and functional attributes required to emulate off-chip communications on line cards. We would like to consider three of these simulators (NS-2, Qualnet and OPNET) and make the distinction between their applications and ours.

NS-2 is an object-oriented, discrete, event-driven network simulator developed at UC Berkeley, written in C++ and OTcl. NS2 is primarily useful for simulating local and wide area networks and supports simulation of TCP, UDP, routing, and multicast protocols over wired and wireless networks [91][92].

The Qualnet is a real-time simulation framework, developed by Scalable Network Technologies (SNT), to emulate the communications of multiple network models. Qualnet includes a rich 3D-visualization interface to provide the user with control over data packets, network topology and performance evaluation. Qualnet supports wireless and Ad-hoc networks as well as parallel and distributed architectures [93]. In addition, it supports multiple routing protocols such as BGP, SIP, RIP, ARP, and BRP. Some related applications that can benefit by using this network simulator include: microwave technologies, high frequency radio communications or satellite communications.

OPNET's network modeling and simulation environment delivers a scalable simulation engine that can emulate wireless, point-to-point and multi-point network links. It has the capability to support routing protocols such as voice, HTTP, TCP, IP, Ethernet, frame relay and more [94]. Some of the application best suit for this simulator are mobile, cellular, Ad-hoc, wireless LAN, and satellite networks. The OPNET simulator allows the user to custom design traffic models since it supports FSMs and object-oriented modeling [95].

These network simulators are not designed to emulate off-chip communication environment required for our application based on the following differentiations:

- Physical attributes: none of these simulators include specific PCB physical properties which have a great effect on the interconnect performance. Physical properties are crucial to meet the stringent area restrictions on line cards.
- Applications: all three simulators fit better for LAN, WAN, mobile and Ad-hoc communications, not small scale interconnects which require different routing

algorithms and flow control mechanisms. The line card simulator must include message flow enhancement features such as virtual channels and sub channeling.

- Message control: our interconnect simulator provides control of how to deliver messages, perform statistics, gather data, route the packets through the network and run auto test cases. Furthermore, the user has more control of how to save and re-run data using the simulator options menus, rather than learning OTcl or Parsec.
- Participants: while our simulator models communication among PEs and memories, the other simulators include other participants such as PCs, satellite communication, routers or other moving objects.
- Communication medium: most of communication mediums used in these simulators have different signal propagation characteristics and performance. Our off-chip interconnect model is a small scale network in which packets propagate from point-to-point via PCB buses no longer than 1 inch in length.

CHAPTER 2: NETWORK PROCESSOR BACKGROUND

This section provides an overview of network processor technology and architectures. In this work, we adopt the following definition: a network processor is a programmable processor that is optimized to perform one or more of the following functions: packet classification, packet modification, buffer management and scheduling, and packet forwarding. Note that this is not an all encompassing definition and excludes General Purpose Processors (GPP), which are still being used for packet forwarding in routers. It is expected that GPPs will be replaced by programmable network components for packet processing. However, GPPs will still find use for initializing, configuring and orchestrating the NPU control path. There is also a notion of specialized co-processor, which is being used hand-in-hand with a network processor. In general, a co-processor is used for more specialized tasks, and is more likely to be shared among multiple processing elements. Common networking functions that are implemented as co-processors are search engine, classification, buffer management, encryption/decryption, and traffic management.

2.1 Architectural design approaches

Most of the currently available NPU designs employ such hardware-oriented techniques as pipelining and parallel processing, as well as software-oriented techniques such as multi-threading and special-purpose instructions. In this section, we will summarize important features of these techniques and point out NPUs from industry that employ them [39][40].

Network processor designs can be historically divided into three main architectural trends: A general RISC-based processor, an augmented RISC processor / ASIC network-specific processor, and a programmable network processor. Programmable network processors are better suited for today's Gigabit/s data processing with many protocols, since they can cope with specialized data handling tasks, and adapt high data rates and packet diversity. Recently, networking products have become more modular and also more software-intensive. As a result, system tasks are divided such that a core processor manages complex global tasks, while single or multiple low-level processor(s) (i.e., NPUs) perform packet-processing operations.

2.1.1 Hardware-oriented techniques

A network processor may contain many individual processor cores, which range in complexity. This multi-processor type architecture can increase the speed and bandwidth capability of packet processing units. Currently available NPU designs are based on either highly parallel multi-processor (e.g., Clearspeed [76], IBM [77], Vitesse [78], etc.), or highly pipelined multi-stage architectures (e.g., Cisco [79], Mindspeed [80], etc.). In all cases, at least a RISC core is combined with specialized software to support packet-oriented functions. In the pipelined approach, shown in Fig. 2.1, packet processing is divided into stages. Each stage is responsible for performing a single task and takes the same amount of time. It is hard to load-balance the pipeline, since the pipeline rate is determined by the slowest stage. The advantage of this architecture is that specific processing tasks can be optimized for each stage. Examples of this architectural

technique include the NPUs from Vitesse [78], Cisco [79], and Xelerated Packet Devices [81].

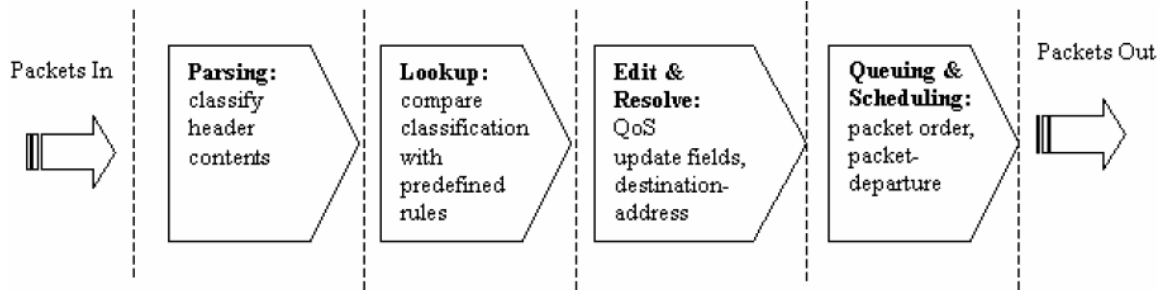


Figure 2.1: Block diagram for a pipeline based packet processing

In the parallel processing approach, multiple processor cores are embedded into one packet-processing unit to exploit the parallelism in networking functions. One significant limitation of this approach is the packet order management, which is concerned with packets leaving the processing unit in the same order in which they arrive. There are two common parallel packet processing architectures as illustrated in Fig. 2.2-2.3. The first is task-level parallelism, in which a packet is divided into smaller units and processed by multiple engines. Each unit requires a different task and includes a task-specific processing engine, which performs a unique operation to extract information from the particular components. All processed units are then integrated back and prepared for departure. This approach is used in the NPU designs from Clearspeed [76], Vitesse [78], Agere [82], Motorola [83], and Virata [84].

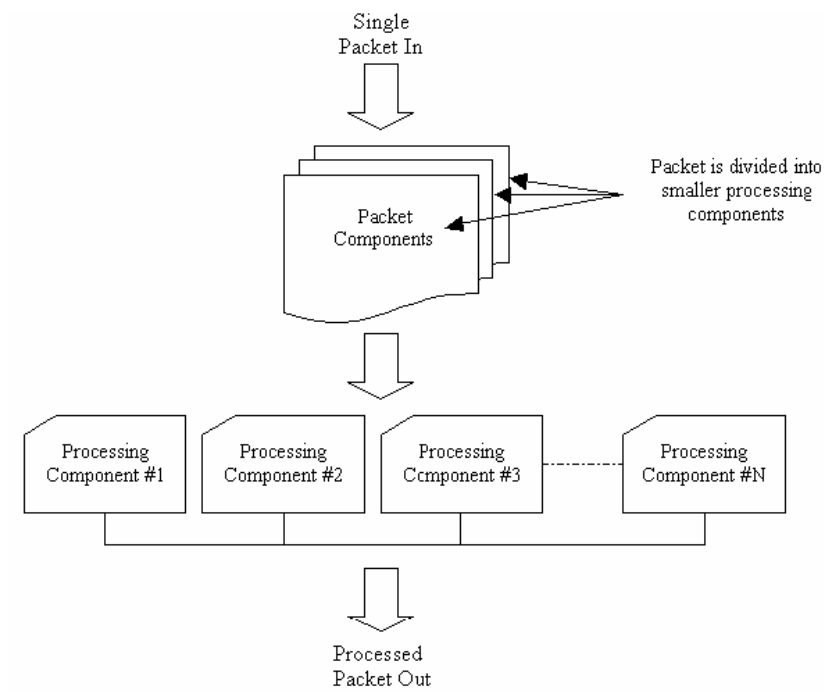


Figure 2.2: Task-level parallelism

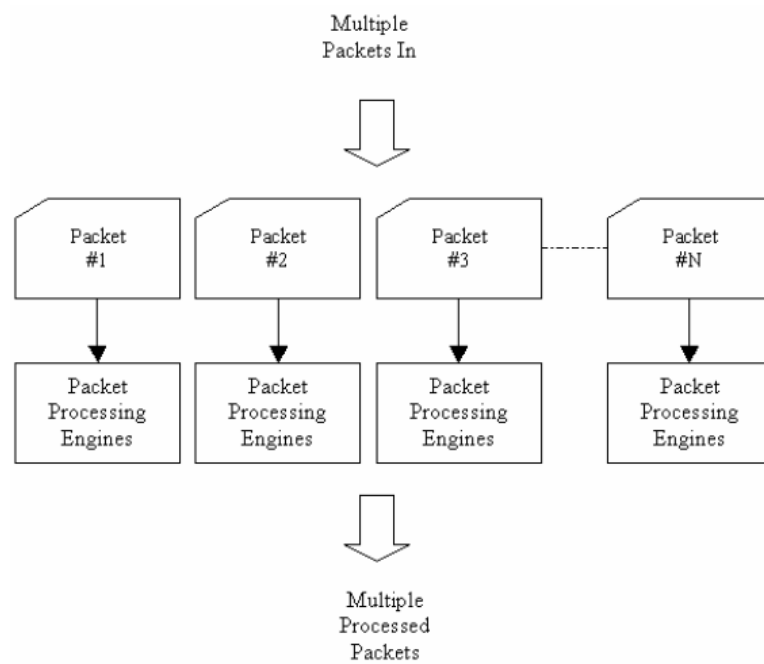


Figure 2.3: Packet-level parallelism

The second parallel architecture is called packet-level parallelism, which allows the processing of whole packets by many different engines. In this approach, multiple incoming packets can be processed at the same time like a super-scalar architecture. Companies that implement this approach in their NPUs include Mindspeed [80], Xelerated Packet Devices [81], Broadcom [85], EZchip [86], Juniper [87], Paion [88], and AMCC [90].

2.1.2 Software-oriented techniques

Many network processors are implemented by utilizing special software techniques that not only allow high performance, but also reduce the need for hardware. In VLIW (very long instruction word) architecture, for instance, the compiler can handle many functions such as instruction issue and scheduling, which would otherwise require additional hardware. This facilitates opportunities for low-power design; however, as a tradeoff, it requires sophisticated compiler technology. VLIW architectures are used in some of the example NPUs from Vitesse [78], Mindspeed [80], Agere [82], Broadcom [85], and Juniper [87].

Memory management and optimization techniques are employed to tolerate the latency of longer accesses. Most common is multithreading, which keeps multiple processing units busy at all times. While retrieving data from memory, packets are assigned to functional units that are free so that a particularly slow lookup does not create a bottleneck in the packet processing. Multithreading enables context to be switched from one packet to another while waiting for slower memory devices such as off-chip Content

Addressable Memory (CAM) and SRAM chips. Companies that use multithreading in their products include Clearspeed [76] and Vitesse [78]. A variety of techniques are also deployed both in hardware and software to reduce the number of memory accesses for lookup [38].

2.1.3 NPU comparison tables

Tables 2.1-2.2 compare NPUs from different manufacturers both from system architecture and system performance point of view. Some companies include their custom made NPU as part of a line card. The line card includes an NPU “block” that is an integrated unit within the system, which performs packet processing. On the other hand, some companies developed a stand alone NPU that can be easily integrated in a variety of network equipment, which requires packet processing and includes more functionality (switch fabric, packet processors, table lookup, buffers and more). Moreover, there are network-processing chips that perform tasks such as DSP functions required in wireless communication systems. For example, the Chameleon CS2112 is a reconfigurable communication processor, optimized for DSP applications. Application-specific network processors, like the CS2112, were not included in the tables.

NPU future designs will continue to be based on multiple processing engines, task specialized software, superscalar and/or superpipelined architectures. New techniques that were developed to cross the OC-192 line will be used to reach a greater scalability. Parallel packet processing among multiple processing engines together with software pipeline technology accelerated packet-processing rate dramatically. In addition, cache

memory is added to each processing engine for fast data and instructions storage. Technological developments in fabrication process, currently 0.09 μm , allow better utilization of wafer space and therefore, more transistors can be fabricated in a given area. As a result, more processing engines can be used and therefore, more packet processing power. The interconnects developed in this work maintain close coupling between processing engines. This allows high-speed packet data sharing among processing engines.

Table 2.1: NPU system architecture comparison

| NPUs \ Parameters | Capacity | OSI layers | Central Control | Processor Engines | Memory Size |
|------------------------------------|----------|------------|------------------|-----------------------|--|
| Acorn genFlow | 10 Gb/s | 2-4 | genFlow core | custom | 123 MB SDRAM |
| Agere PayloadPlus | 5 Gb/s | 2-4 | none | RISC | Int. 64 KB; Ext. SSRAM |
| AMCC nP | 5 Gb/s | 2-7 | nPeore | none | N/A |
| Broadcom BCM1250 | 2.5 Gb/s | 3-7 | none | MIPS | 32 KB L1; 512 KB L2 cache |
| Cisco Toaster-2 | 6.4 Gb/s | N/A | general cpu | custom | SDRAM or SRAM 256 Mb max |
| ClearSpeed MTAP | 40 Gb/s | N/A | N/A | RISC | N/A |
| Clearwater CNP810SP | 10 Gb/s | 4-7 | CNP810 | Superscalar RISC | 64 KB I+D cache; 256 KB pack. cache |
| Cognigine | 10 Gb/s | 2-7 | none | VISC | 512 KB buffer; 1 MB SSRAM |
| EZchip NP-1 | 10 Gb/s | 2-7 | N/A | custom | N/A |
| IBM PowerNP | 8 Gb/s | 2-5 | PowerPC | 16 Prog. protocol PEs | Int. 32 KB SRAM; Int. 32 KB SRAM |
| Intel IXP1200 | 2.6 Gb/s | 2-7 | StrongARM | RISC | 8 MB SRAM; 256 MB SDRAM |
| IPS SR10G | 10 Gb/s | 2-7 | none | FPGA | N/A |
| Juniper Internet Processor II ASIC | 20 Gb/s | 2-3 | N/A | custom | N/A |
| Lexra NetVortex | 10 Gb/s | 2-7 | Lexra LX8380 | RISC | 16 KB I+D cache; 128 KB SRAM |
| Mindspeed Edgemaker | 155 Mb/s | N/A | Conexant CX27510 | RISC+CISC like | Int. 512 KB SRAM; Ext. 1 MB SRAM |
| Motorola C-5 | 5 Gb/s | 2-7 | Executive proc. | RISC | Int. 512 KB; Ext. 16 MB; 128 MB buffer |
| Paion gigabitPLUS | 10 Gb/s | 2-7 | none | RISC | SDRAM 64b + SSRAM 32b |
| Silicon Access iFlow | 20 Gb/s | N/A | N/A | RISC | 2 MB SRAM; 72 KB TCAM |
| Xelerated X40/T40 | 10 Gb/s | N/A | X40 and T40 | RISC (PISC) | N/A |
| Vitesse IQ2000 | 10 Gb/s | 2-3 | none | N/A | 64 KB |
| Virata Helium 100 (ARM7TDMI) | N/A | 2-3 | ARM7TDMI | RISC | 8 KB cache; 16 KB SRAM |

Table 2.2: NPU performance comparison

| NPs \ Parameters | I/O BW | Max. Packet Rate | Multi-threading | Clock | Process | Power | SW support |
|------------------------------------|-----------------------|------------------|-----------------|-----------|--------------------|-------|--------------------|
| Acorn genFlow-10G | 6 Gb/s | 1 Mpps | N/A | N/A | N/A | N/A | N/A |
| Agere PayloadPlus | 5 Gb/s | 15 Mpps | 64/FPP | 133 MHz | 0.18 μm | 12 W | FPL |
| AMCC nP | 5 Gb/s | 15 Mpps | 8/cpu | 220 MHz | 0.18 μm | 4 W | Assembler or C/C++ |
| Broadcom BCM1250 | 25 Gb/s | N/A | N/A | 600M-1GHz | 0.15 μm | 10 W | C/C++ |
| Cisco Toaster-2 | N/A | 6 Mpps | 16/cpu | N/A | 0.20 μm | N/A | N/A |
| ClearSpeed MTAP | 10 Gb/s | 100 Mpps | yes | 400 Mhz | 0.13 μm | N/A | C |
| Clearwater CNP810SP | 12.8 Gb/s | 25 Mpps | 8/cpu | 300 MHz | 0.15 μm | 12 W | C |
| Cognigine | 200 Gb/s | 25 Mpps | 4/cpu | 200 MHz | 0.18 μm | N/A | C/C++, Asml. |
| EZchip NP-1 | 320 Gb/s | N/A | none | 200 Mhz | 0.18 μm | N/A | Assembler |
| IBM PowerNP | 8 Gb/s | 16.6 Mpps | 2/cpu | 133 MHz | 0.18 μm | 20 W | Assembler |
| Intel IXP1200 | 2.6 Gb/s | 6 Mpps | 4/cpu | 200 MHz | 0.28 μm | 6 W | Assembler |
| IPS SR10G | N/A | 25 Mpps | N/A | N/A | FPGA | N/A | N/A |
| Juniper Internet Processor II ASIC | 51.2 Gb/s half-duplex | 40 Mpps | N/A | N/A | N/A | N/A | Junos |
| Lexra NetVortex | 20 Gb/s | 60 Mpps | 4/cpu | 420 MHz | 0.13 μm | 12 W | JTAG |
| Mindspeed Edgemaker | N/A | N/A | N/A | 133 MHz | N/A | 2 W | Assembler |
| Motorola C-5 | 5 Gb/s | 16.7 Mpps | 4/cpu | 200 MHz | 0.18 μm | 15 W | C/C++ |
| Paion gigabitPLUS | 2.5 Gb/s | N/A | N/A | 250 Mhz | 0.18 μm | N/A | Assembler or C/C++ |
| Silicon Access iFlow | 115 Gb/s | 30 Mpps | 8/cpu | 333 MHz | 0.13 μm | N/A | C-like |
| Xelerated X40/T40 | 160 Gb/s | N/A | N/A | 166 Mhz | 0.13 μm | N/A | Xelerator soft. |
| Vitesse IQ2000 | 6.4 Gb/s | 8.4 Mpps | 5/cpu | 200 MHz | 0.25 μm | 7 W | none |
| Virata Helium 100 (ARM) | 75 Mb/s | N/A | N/A | 48 Mhz | 0.18 μm | N/A | ISOS Platform |

CHAPTER 3: K-ARY N-CUBE BASED ARCHITECTURES

In this chapter we present the k -ary n -cube architectures that we propose to be used as off-chip communication architectures for high-speed packet processing. We start with general k -ary n -cube structures and later present 3D-mesh interconnect, a specific type of k -ary n -cube (2-ary 3-cube), which might better suit to our needs.

3.1 K-ary n-cube interconnect structures

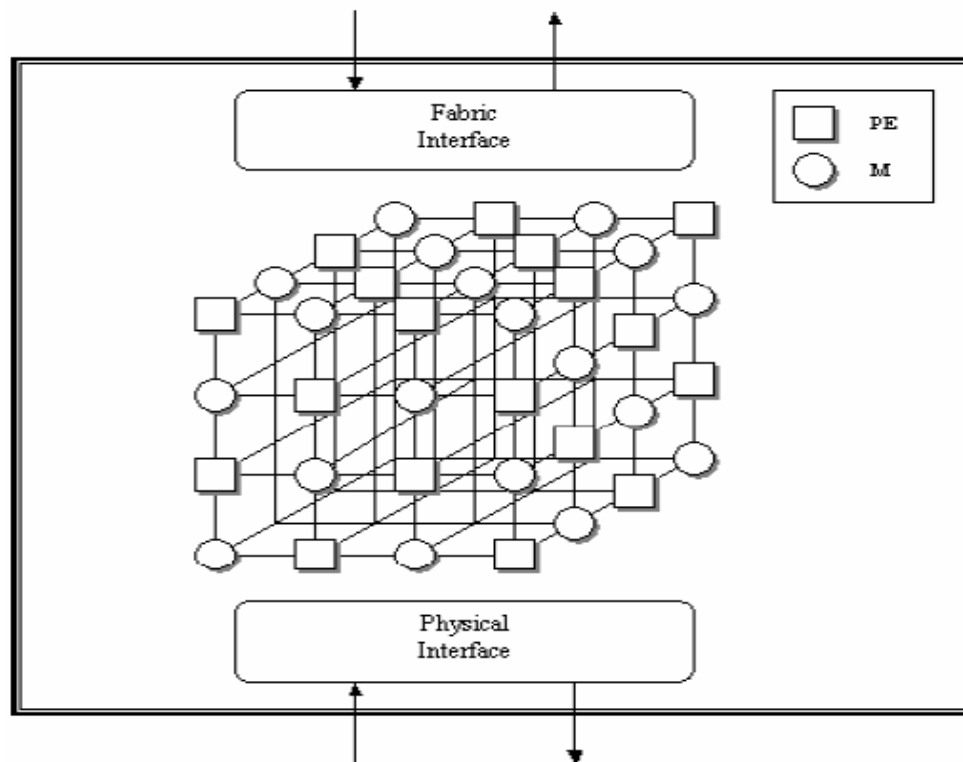


Figure 3.1: K -ary n -cube architecture on the line card

The k -ary n -cube architecture shown in Fig. 3.1 is a packet-based multiple paths interconnect that allows network packets to be shared by different processing elements (PE) and memory modules (M) on the network line card. Memories are distributed

around processing elements, such as traffic manager, QoS co-processor, classification processor, to allow data sharing among modules and direct processor memory storage. If a link goes down, not only should the fault be limited to the link, the additional links from the intermediate nodes should ensure the connectivity continues.

3.1.1 Routing mechanism

The routing algorithm routes a packet from a source device $s=s_1, s_2, \dots, s_m$ to a destination module $d=d_1, d_2, \dots, d_n$, by choosing a direction to travel in each of the n -dimensions to balance the channel load. A memory packet sent by a processing element will always attempt to take the shortest path as long as packets are admissible (accepted by ideal nodes). If a node is oversubscribed (i.e., all ports are occupied), packets in transit will take a different route using the traffic controller (TC) in each corner (node). The architecture protocol utilizes an efficient message-passing structure to transfer data. The architecture path diversity offers alternative paths between source and destination modules.

Before a node forwards a packet to one of its adjacent nodes, it polls the status of each node. The traffic controller at each node has a “sensor status” flag which determines if the node is currently processing a packet (i.e., busy), or if the node is idle and ready to accept a new packet (i.e., not busy). Depending on its direction preferences (some nodes may get higher preference than others if they are located closer to the packet's destination), TC will choose an admissible node to forward the packet. If at least one

adjacent node is available to forward a packet, it will require only one clock cycle to do so.

Wormhole routing is used since it is known for its improved latency and throughput measures. The message required to be sent to or from a memory module is segmented into smaller size packets, flow control digits (flits), as shown in Fig. 3.2.

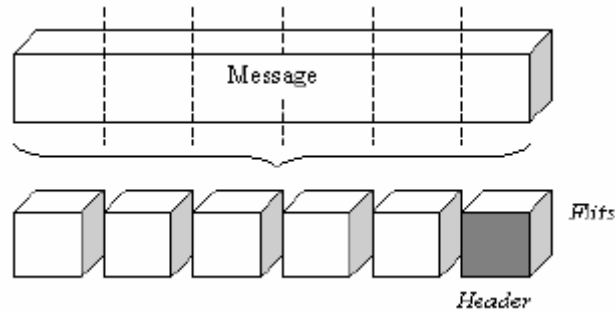


Figure 3.2: Message segmentation

In wormhole routing, the header flit is sent first. While the header propagates through the interconnect, it sets the node switches in a certain position corresponding to the traffic load on the node's channels. The rest of the packets comprising the message do not wait, but are transmitted in a pipeline manner following the message header (resembles a worm movement). The main advantage in using wormhole routing is that it diminishes the latency as the size of the message increases while increasing its throughput. The major part of the latency is hidden in the transfer of the first packet. The rest of the packets follow it and introduce only wire transfer delay. As the message size increases, the ratio of consecutive latencies decreases.

From a throughput viewpoint, packets can travel with every $T_w + T_s + T_r$ psec following their header. Where T_w is the propagation delay of one bit in a unit length which is equal to 254 psec per 1 inch using the current manufacturing technology [35]. T_s

is the switching delay within a node. Switching delay equals 100 psec and can increase if the message is required to be queued (if all node outputs are blocked). T_r is the routing decision latency and is equal to 500 psec. Therefore, ideally, without considering any extra queuing delays, the maximum throughput that can be achieved is equal to 90 Gbps per node.

Throughput is calculated by dividing the channel size by the sum of latencies per link ($T_w + T_s + T_r$) and then, it is multiplied by 2 since there can be two message flows transferred simultaneously through a single node. The aggregate throughput is a function of the number of PEs as well. The configuration of PE vs. memories within the k -ary n -cube architecture increases the network load since more communicating modules are available. This becomes a great advantage in achieving high throughput while a parallel bus can only send those packets like a store and forward type architecture (see Fig. 3.3).

Routing decisions are made for each message (also called a “worm”) at each node. A worm always tries to take the shortest path to its destination, if traffic conditions will allow it, otherwise, it takes an alternative route as close as possible to its original shortest path. Routing directions in a node are referenced along 3-dimensional axes, as shown in Fig. 3.4a. The default packet direction is straight forward to the next node (corner) or x -axis. When an alternative path is taken, the other two directions possible are the y or z axes. Any combination of movement along the z axis following a y movement, and vice versa, is called south/north turns. Fig. 3.4b depicts a node connecting multiple channels. RF and LF denote a movement from one face to another. In addition, each node contains virtual channels (resembles a long buffer within the node) and a message port (virtual port holding the incoming message).

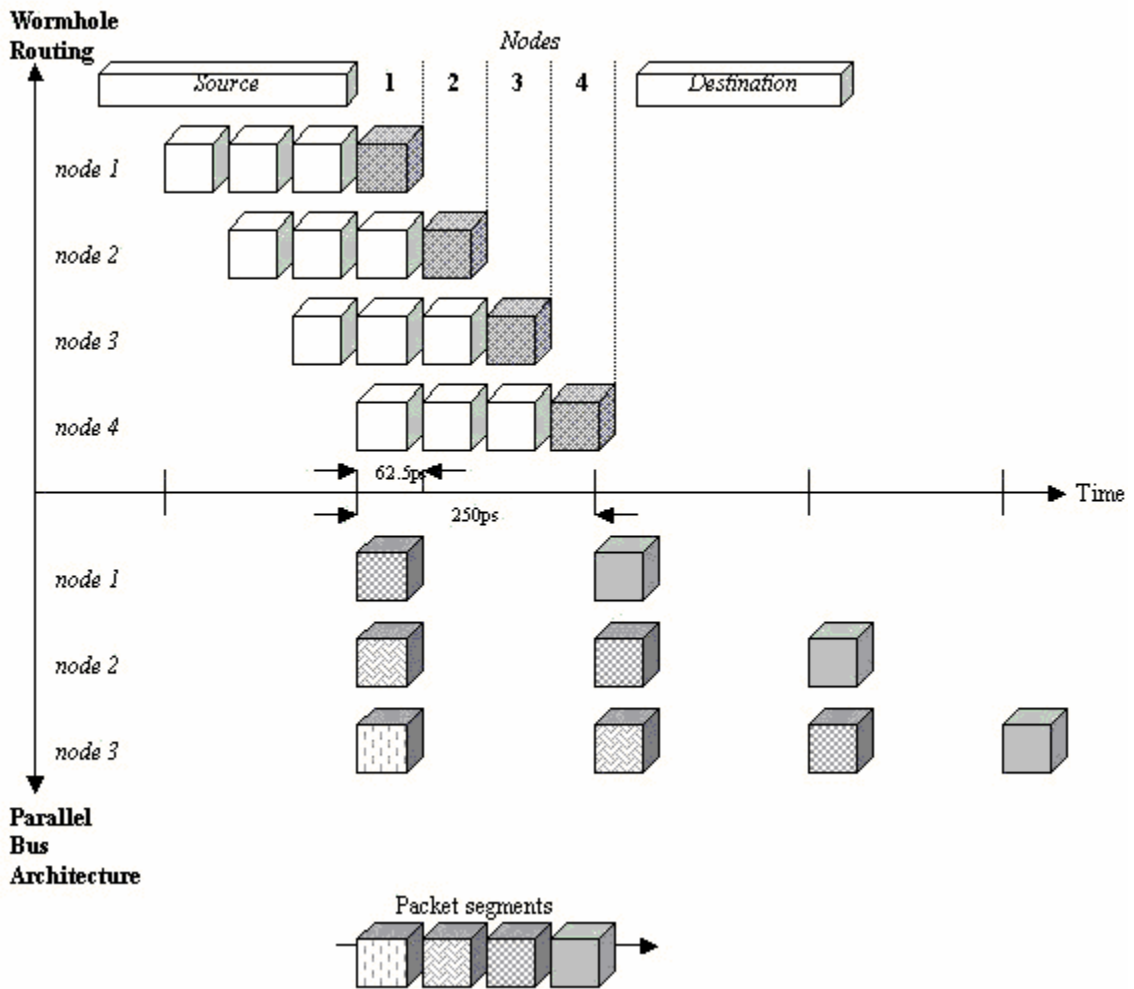


Figure 3.3: Message timing (wormhole routing vs. parallel bus)

The routing algorithm is a combination of previously developed algorithms [17], [22], and [23]. It tries to merge the best of each algorithm to achieve high maneuverability and adaptivity to traffic conditions within the interconnect. The following rules must be satisfied:

- Ensures the shortest path first, by comparing source and destination vectors (in terms of x , y , and z coordinates) and move forward by evaluating the variance in each dimension.

- If one of the chosen output ports is occupied (busy transferring other message), it samples the status of other ports in the following order (Fig. 3.4a): EW (East-West) a movement from one face to another, NS (North-South also up-down) resembles a clockwise vs. counter-clockwise movement on an individual face.
- Avoid certain consecutive turns. This rule seeks to avoid deadlocks. A worm following an EN, ES will not take west movement as the third direction. Similarly, if WN, WS movements are taken then, it will not take an east movement as the third direction.
- Since worms are generated either from PEs to memories and vice versa then the worm's relative direction is always towards its destination and will never move backward (towards its source). This step attempts to avoid livelocks.

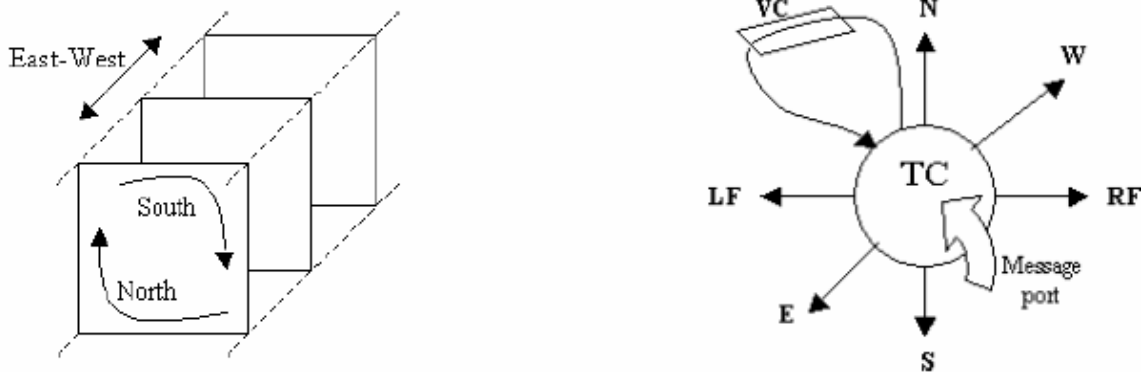


Figure 3.4: a) Routing directions and coordinates b) Node connectivity

The most common disadvantage of wormhole message passing mechanism is its tendency to form deadlocks and livelocks. There are multiple methods to minimize these adverse states. In our model, we deploy various techniques to reduce deadlocks/livelocks and avoid their occurrence. These techniques include virtual channels, adaptive routing

algorithm with turn prohibitions, node switch state control, channel partitioning and message retransmission management.

3.1.2 Switching mechanism

A channel to channel switching mechanism is required to toggle packets in different directions (i.e. to move from one channel to another). There are two common switching mechanisms used in networking components to switch the channel wires: a crossbar and a multistage Omega network.

A crossbar interconnect can simultaneously route any permutation of I/O pins. The main disadvantages of a crossbar switch are its physical dimensions and hardware resources it consumes. An Omega switch uses less hardware but it is limited in its switching flexibility. When we compare their switching complexity, a crossbar uses n^2 switches while omega network uses $\frac{n}{2} \log_k n$ switch boxes, where n is the number of channels and k is the number of switches in each box. The wiring complexity associated with a crossbar switch is $O(n^2w)$ while the Omega switch box is $O(nw \log_k n)$ [13]. Hence, the tradeoff between an Omega network switch and a crossbar is hardware resources vs. switching flexibility. We decided to use Omega switches to implement the interconnect architecture since space is very limited on the line card.

There are three configurations that a switch can have for a 4-channel node in the k -ary n -cube structure as illustrated in Fig. 3.5. However, an Omega switch, shown next to it, can accommodate only two of these configurations. This will minimize the number

of Omega switches in expense of restricted routing. An incoming packet from input A can be switched to output C or D and input B can be switched to C or D.

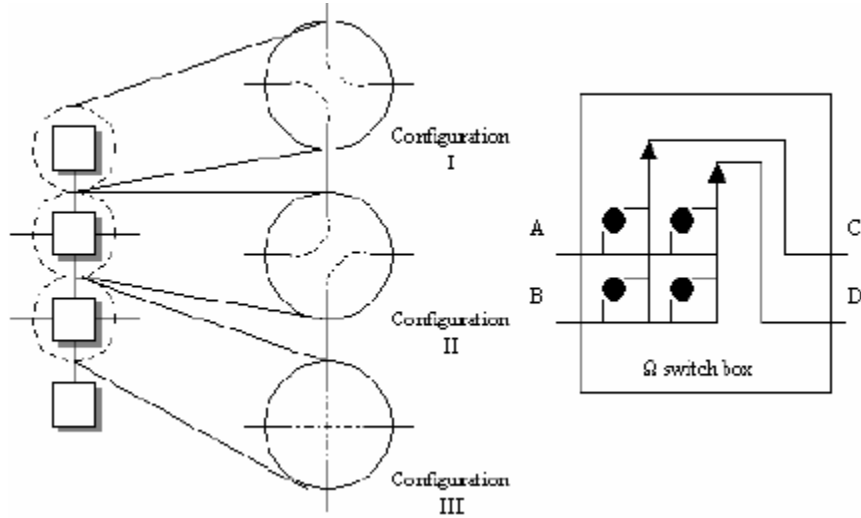


Figure 3.5: Omega switch configuration

Figure 3.6 shows a scenario of packets from two different messages traverse in 2D-mesh network. Here, the Omega switches can only be set in configurations I and III.

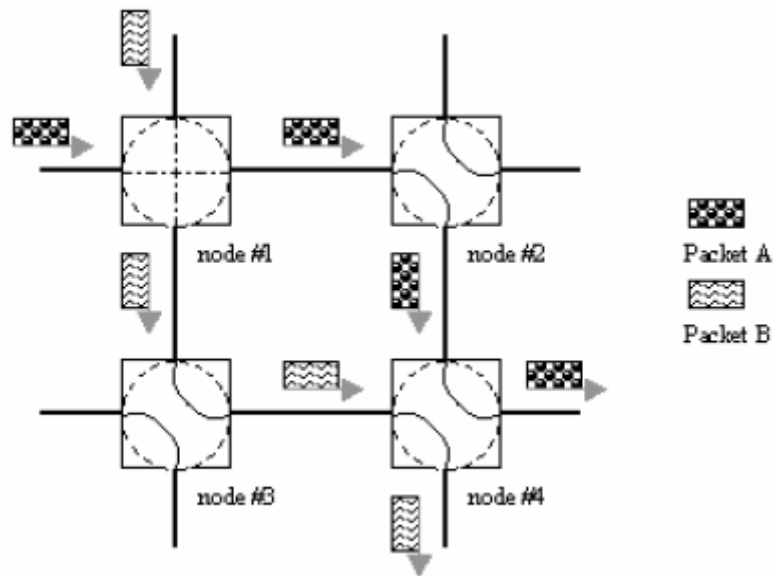


Figure 3.6: Cyclic deadlock prevention

Packets A and B enter node 1 which is set to configuration III. Therefore, packet A moves straight to right and B moves down. Nodes 2, 3 and 4 are set to configuration I. Note that node 4 would have started a cyclic movement if it was set to configuration II. Since it is set to configuration I, packet A is routed to the right direction, while packet B is routed down. This configuration avoids packets collision and cyclic movement by forcing the packets to take opposite routes.

3.1.3 The traffic controller

Each node employs a traffic controller (TC) to forward messages. The TC includes five components: the routing algorithm, multi-port switch, channel sampler, channel partitioning mechanism and virtual channels (Fig. 3.7) [18]. Each cycle, the channel sampler samples each port to determine its status (total of 4 ports). If a port is currently busy transferring a message, the channel sampler will not allow any new messages to be routed to it.

Fig. 3.8 shows the connectivity of memories and PEs using the TCs. Packets are transmitted between two communicating devices where they utilize nodes that are directly connected to a memory module. The PEs/memories are not responsible to forward the packets, the traffic controller resides within each node is. The memories and/or PEs (Fig. 3.8) are connected to only one port (out of 4 ports) available per node.

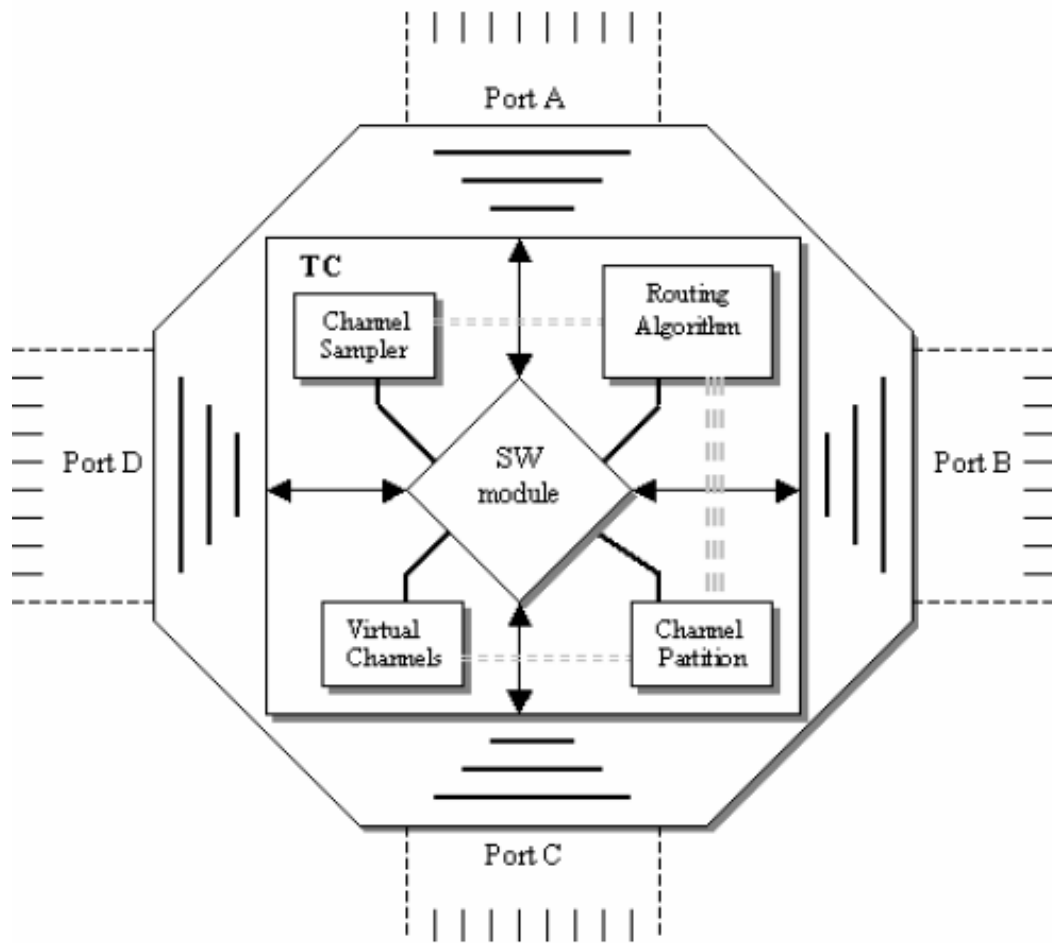


Figure 3.7: Traffic controller

The memory/PE can transmit data while the node, which contains micro switches, allows packet of different flow (different message) to pass through it to a different port, as long as there is no contention over that specific port.

The channel partitioning module can divide a unidirectional channel into two or four bidirectional sub-channels, as shown in Fig. 3.9. For example, a channel of 32 bits can be partitioned into 4 sub-channels of 8 bits each, and transfer 4 different messages simultaneously. It receives channel configuration information from the user interface and sets the TC's internal parameters accordingly. Each of the channels (connecting two

nodes) can be partitioned into smaller channel widths to allow more packets (each packet belongs to a different message) to share the same channel. As a result, the packet size of each message will decrease while its latency increases.

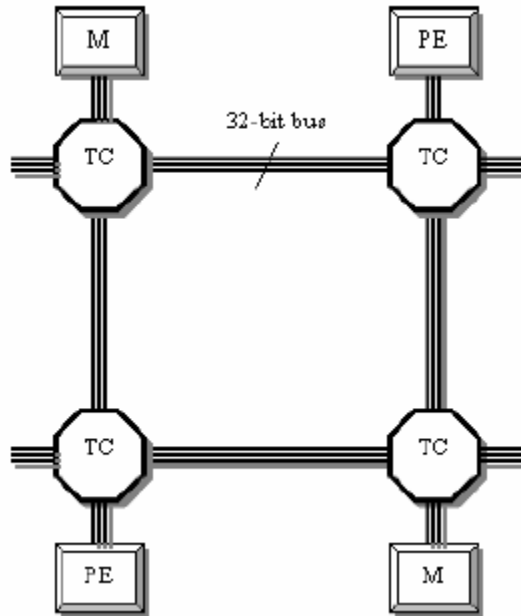


Figure 3.8: PE and Memory interfacing the TC

The main advantage of channel partitioning is the additional path selection flexibility which institutes a tradeoff between lower message failure rate to increasing message latency. Channels can be configured to one of the following partitions: single unpartitioned unidirectional channel, two bidirectional subchannels (the channels can contain two packets simultaneously) and four multidirectional subchannels.

The system load balances itself to handle momentary channel overloads and provides data transfer stability by highly utilizing the TC functionality in cooperation with its message passing mechanism and routing algorithm. System stability is measured by the routing mechanism's ability to perform when the system reaches saturation. A stable routing protocol does not degrade at saturation point, while an unstable routing

protocol does. The routing algorithm dynamically ensures that connectivity continues without significant performance degradation.

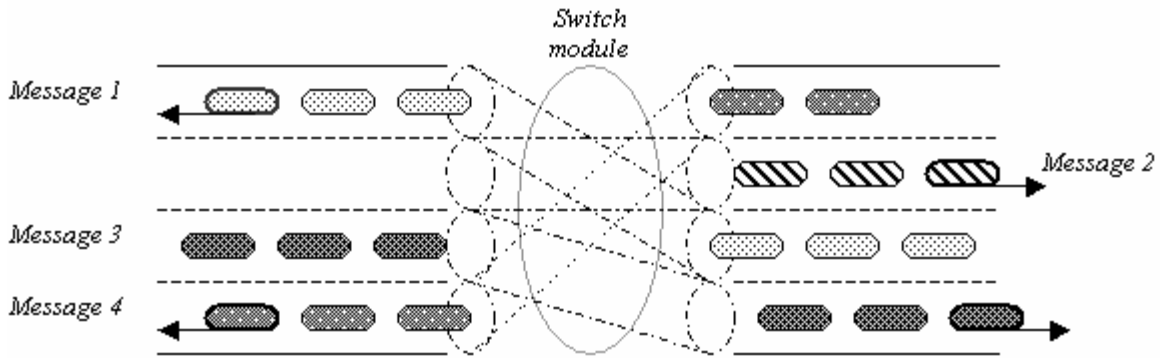


Figure 3.9: Channel partitioning to 4 sub-channels

Virtual channels (VC) are used when an incoming message cannot be routed to any output port since all output ports are busy transferring other messages. Fig. 3.10 depicts a situation where two messages compete over the same output port (West port). Since message 1 is granted permission to continue in its path, message 2 will have to be queued in one of the available VCs.

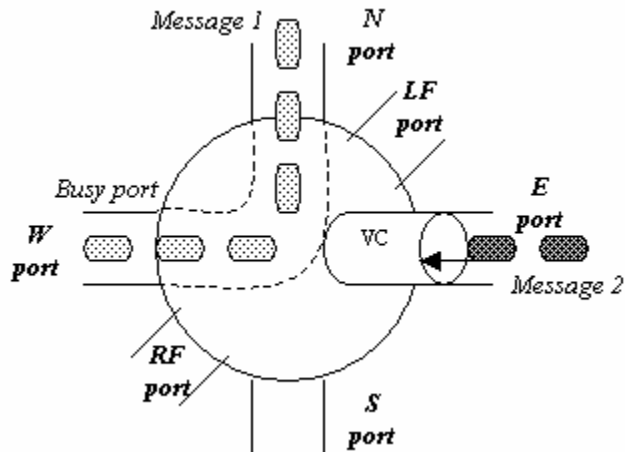


Figure 3.10: Virtual channels

Practically, a propagating message occupies two ports (out of four ports available in one node) when moving from input port to output port. There are only two VCs per node, since in the worst case only two messages can arrive and compete over the same output port. The VC module, within the TC, sets virtual channels to enabled/disabled status and if enabled, it also allocates its buffer size (in KB).

3.2 3D-mesh interconnect architecture

The 3D-mesh architecture shown in Fig. 3.11 is a packet-based multiple paths interconnect that allows packets to be shared and transferred by different processor and memory modules on the network line card simultaneously.

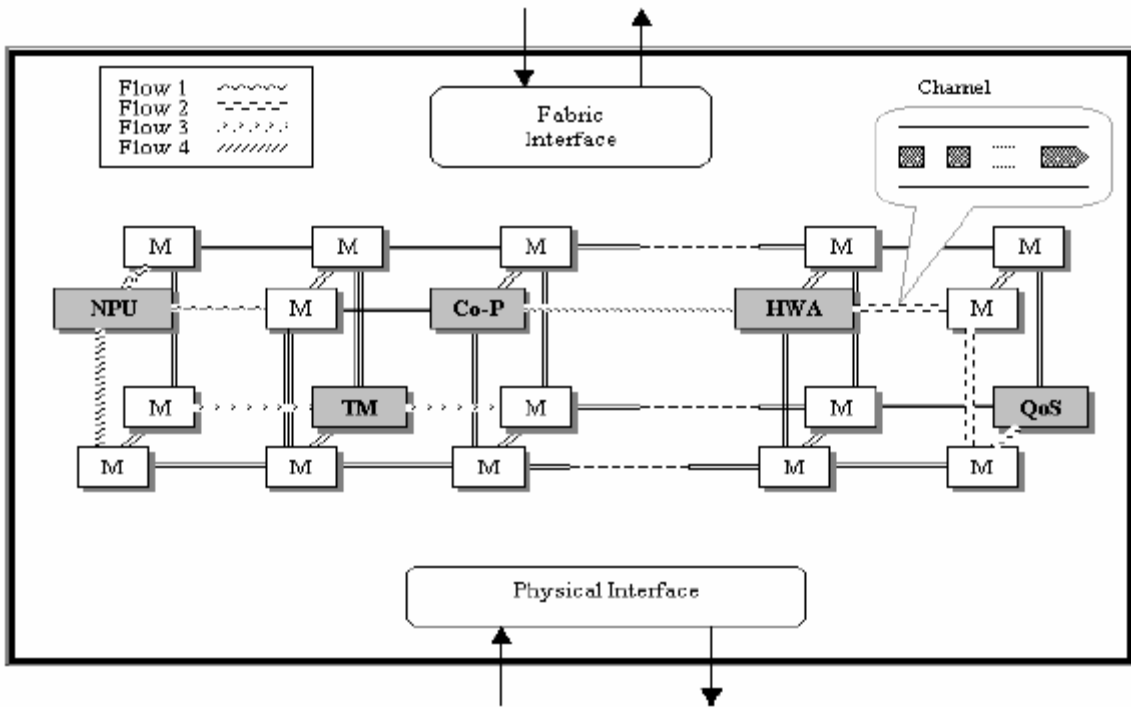


Figure 3.11: 3D-mesh structure on the network line card

Fig. 3.11 portrays a general line card architecture in which processing, communication and memory components can communicate via 3D-mesh interconnect. Besides the NPU several processing components are shown in the figure, such as co-processor (Co-P), traffic manager (TM), hardware accelerator (HWA), and quality of service (QoS) co-processor. The memory banks are distributed over the interconnect structure to allow data sharing among modules and direct processor memory storage. The fabric and physical interfaces handle incoming/outgoing traffic.

Each component, which requires memory access, sends its data encapsulated in packets. If there is a congested area (hot spot), packets in transit will take a different route, using predetermined and prioritized set of directions using the traffic controller (TC) located in each node. The proposed architecture protocol utilizes an efficient message-passing mechanism to transfer data. A faulty link will not discontinue the transmission of a message to its destination since packets will be rerouted through alternative paths using other available nodes. Routing ensures that a faulty link will be limited only locally and other links from the intermediate nodes should ensure that connectivity continues.

Each node contains two bidirectional VCs. VCs are used to avoid transmission failure. Transmission failure can occur when a packet (fraction of message) cannot take any of the output ports. Another feature in our model includes channel width partitioning. This feature allows a unidirectional channel to become bidirectional so that more than one message can share each channel. There are three available settings to channel partitioning, unidirectional channels (one message occupies the complete channel width), bidirectional channels (two messages can share the same channel and can move in either

direction), quad-directional channels which partition channels into four (each channel can contain four simultaneous messages in either direction). As a result of channel partitioning, the packet size of each message will decrease. The main advantage of channel partitioning is the additional freedom in paths that can be selected by propagating messages. Table 3.1 presents a comparison between 3D-mesh, crossbar and shared-bus. Other performance comparisons will be given in performance analysis section.

Table 3.1: Comparison between shared-bus, crossbar & 3D-mesh

| Category | Shared-bus | Crossbar | 3D-mesh |
|--------------------------|--|--|--|
| Simultaneous transfers | Yes $O(n/2)$ requires additional hardware for bus arbitration | Yes $O(n)$ as long as transfers are initialized between modules not currently communicating | Yes $O(n)$ data can be sent to busy modules as well. Nodes buffer data until busy module becomes idle |
| Scalability | No performance degrades as number of modules increases | No fix number of I/O | Yes can scale in all dimensions by connecting more faces |
| Cost | $O(n)$ number of bus data links (bus width) | $O(n^2)$ $n*m$ links + switches | $O(n)$ per face $O(n \log n/2)$ per cube 8 nodes = 12 links |
| Latency | $O(1)$ | $O(1)$ | $O(N)$ |
| Node degree | 1 one link between master and slave | $n-1$ all nodes but itself | 4 function of configuration |
| Bisection width | $O(1)$ | $O(n^2/4)$ | $O(4)$ x-direction $O(n/2)$ y-direction |
| Bandwidth per connection | $O(1/n)$ bus bandwidth is divided among sharing modules | $O(1)$ circuit is reserved for communication | $O(1)$ worm utilizes complete channel bandwidth |

Bus performance, in terms of latency and throughput, of a shared bus is affected by the following factors: 1. Number of processors or memories connected to it (as the number of modules connected to the shared-bus increase, performance degrades). 2. The shared-bus length (average is between 5-12 inches). 3. Shared-bus width (increasing width results in higher cost). Most shared bus systems do not have more than 30 processors / memories. Traffic on the shared bus is ideal when exactly one message is directed at each output. Communication protocol is trivial and is based on simple connection oriented mechanism. In addition, shared bus systems require arbitration mechanism to send/receive data from multiple modules. Bus arbitration adds delay to the overall system latency. Some systems use multiple buses to reduce the effect of the factors mentioned above. Although the bandwidth of the multiple bus architecture is higher than that using a single shared bus, the system is more costly and requires complex bus arbitration logic. As a result of the shared bus disadvantages many multi processor-memory systems include multiple shared buses and/or other type of interconnects.

A crossbar can support multiple simultaneous connections (up to the number of devices) as long as no contentions occur. Once contention occurs its performance degrades (can reach lower limit equal to single shared-bus). If the destination node is currently receiving a message, the request to this destination must wait until it becomes idle.

3.3 The 3D-bus architecture

The 3D-bus interconnect is a variation of the 3D-mesh. Processing modules as well as memories are not distributed in different locations on the interconnect, but placed on both sides of it. The interconnect switching and routing mechanisms are similar. The 3D-bus architecture shown in Fig. 3.12 is a packet-based multiple paths interconnect that allows network packets to be shared by different processor and memory modules on the network line card. In Fig. 3.12, the line card processing and communications components which have access or require access to the memory banks are shown on the left. The components in the figure are given as an example, and other functional components can be also interfaced to this bus. The memory banks are located on the other side of the 3D bus structure.

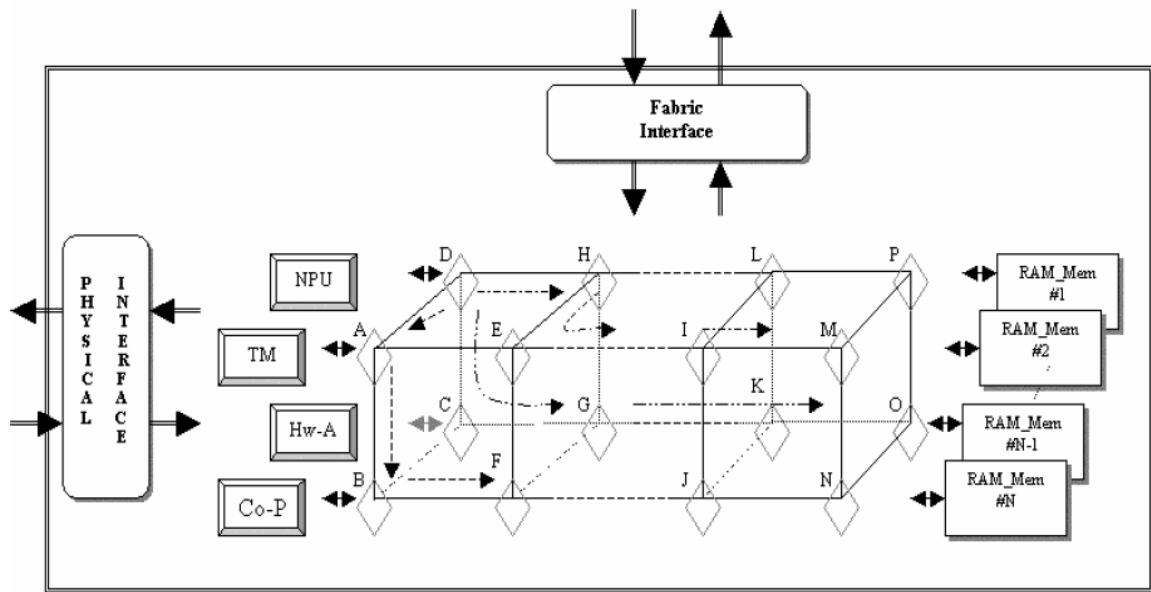


Figure 3.12: 3D-bus structure on the network line card

Each component, which requires memory access, sends its data encapsulated in packets. The default route is the x -axis direction. If there is a congested area (hot spot), packets in transit will take a different route in y -axis or z -axis directions using the traffic controller (TC) in each corner (node). The proposed architecture protocol utilizes an efficient message-passing mechanism to transfer data. If a link goes down, not only should the fault be limited to the link, the additional links from the intermediate nodes should ensure the connectivity continues.

3.3.1 Bus interfaces

The 3D interconnect system requires two interfaces from both end points (i.e., functional units and memory banks). Both interfaces are required to perform the following operations: distribute packets coming from PE/memory to bus interface and vice versa, congestion detection and control at the endpoints, handle arbitrary data transfer requests.

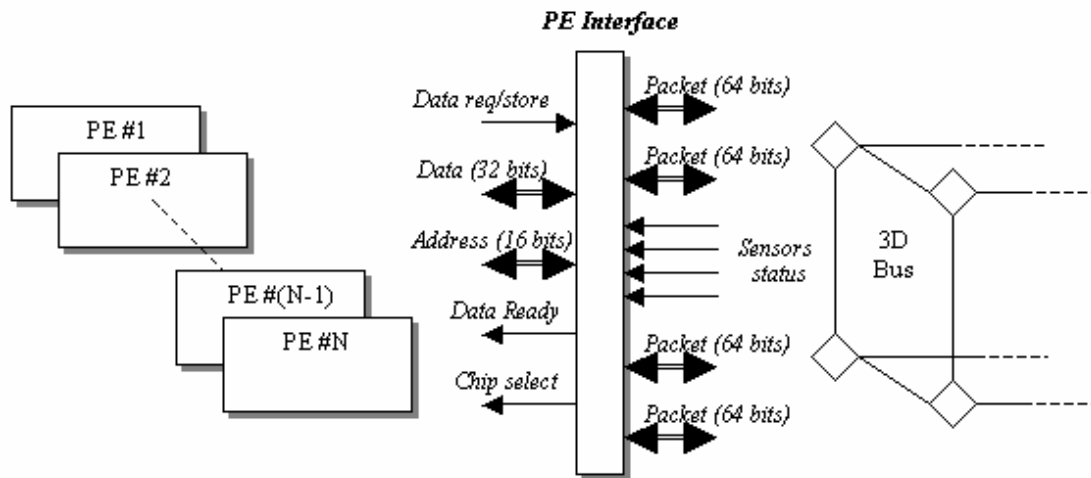


Figure 3.13: PE interface

The PE interface shown in Fig. 3.13 repeatedly samples the four input nodes to the bus and keeps track of which node is busy. Simultaneously, PEs send requests to the interface to allow data to be sent to memory. The interface must allow data transmission through the bus as soon as one of the nodes is idle. Moreover, during heavy traffic and congestion, the interface must balance the data request load.

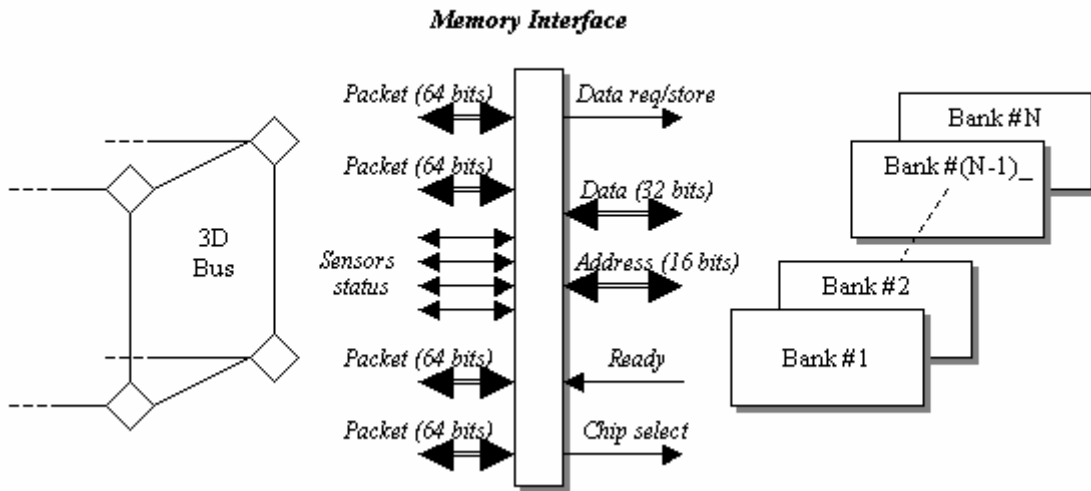


Figure 3.14: Memory interface

PEs requesting data from memory need to wait until the data return from memory. The interface must keep track of the PE which requested the data. When data are ready it signals the same PE to allow data load.

On the memory side, the interface performs same basic operations with little deviation (see Fig. 3.14). Data packets coming from the bus side must be stored in their destination memory banks. The memory interface monitors the utilization of the banks to avoid bank overflow. It also balances the traffic load to avoid congestion and collisions. Before any packet is placed on the bus interconnect the memory interface continuously samples the status of the bus nodes.

CHAPTER 4: ANALYTICAL PERFORMANCE ANALYSIS

4.1 Performance metrics

We use standard performance metrics such as latency and throughput. Each of these measures has different effect on system performance and is influenced by the network physical structure such as network dimensions, channel width, and switching delay. Latency is defined as the time it takes for a complete message to reach its destination. Latency can be determined using an analytical model and verified by using a network simulator. Throughput is defined as the rate in which the packets are exiting the bus for a certain message size per unit time. Throughput will be measured by simulating the entire system. Measuring throughput during simulation can demonstrate how it is being affected by the interconnect load, message size and deadlocks.

The load or offered-load is the number of packets injected into the interconnect network per second and depends on the processing elements which are generating them. The load can vary from starvation (small number of packets enter the network, while the network is empty, and as a result take only minimal paths) to saturation (the network is congested). Throttling is a method of slowing down packet transmission by placing a limit on the rate of injected packets. This limits the maximal network throughput but prevents excessive congestion.

The number of worm transmission failures is another routing measure. The number of faults sums the number of worms which were not able to reach their destination. A worm transmission failure can be a result of heavy congestion within the

interconnect in which all routes are busy and new worms cannot complete their path. The number of failed transmission attempts will be recorded and compared with offered load.

4.1.1 Distribution of IP-packet length in core routers

The k -ary n -cube interconnect will be an integrated part of a line card. It will need to perform under variable line rates and packet sizes. In order to design the interconnect to achieve highest performance, it is required to explore some statistics about IP packets such as average packet size and the distribution of its length. Fig. 4.1 describes the distribution of IP packet lengths in core routers. It illustrates that there are four types of common IP packet lengths: 40B, 52B, 576B and 1500B. 40B packets are most frequently used (56%) [66]. The same IP-packet length distribution will be further utilized, as the message length (M), when analyzing the performance of different interconnect architectures.

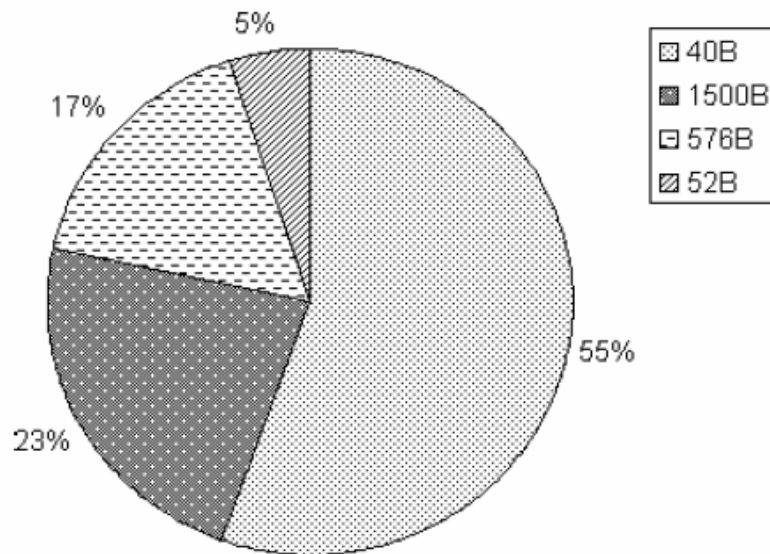


Figure 4.1: IP packet length distribution

4.2 K-ary n-cube latency equation under uniform traffic load

The latency model for k -ary n -cubes was initially modeled by M. O. Khaoua in [48] and was developed also in [50][51][52]. Under uniform traffic pattern, a message passes on average \bar{k} hops across the network

$$\bar{k} = \frac{(k-1)}{2} \quad (4.1)$$

The average distance of this message is

$$\bar{d} = n * \bar{k} \quad (4.2)$$

The mean message latency consists of two parts, the delay due to message transmission and the time a message spends if blocking occurs. For an average of \bar{d} hops from source to destination, latency can be expressed as

$$Latency_{k\text{-ary},n\text{-cube}} = M + \bar{d} + \sum_{i=1}^{\bar{d}} B_i + W_{ej} \quad (4.3)$$

The first term, M , is the message length in flits. B_i is the average blocking time seen by a message at any i^{th} hop, where $1 \leq i \leq \bar{d}$. W_{ej} represents the mean waiting time between message transmissions at the ejection node. In this equation, it is assumed that channel to channel transfer time is 1 unit cycle. V virtual channels are used per physical channel in the model introduced by [49][63]. Although virtual channels are divided into two types adaptive and deterministic, there is no distinction between them when computing virtual channels occupancy probabilities [62].

A message is blocked at any i^{th} hop channel when all virtual channels are busy. If W_b denote the average waiting time due to blocking and P_b represents the probability of blocking then the mean blocking time expression is

$$B_i = P_{b_i} * W_b \quad (4.4)$$

The blocking probability, P_b , is determined by calculating the probability that all virtual channels are busy. Virtual channels analysis and analytical model is discussed in [48]. Since multiple virtual channels share the bandwidth of the same physical channel, a multiplexer is required to select which virtual channel will use the physical channel. This multiplexer functions in a TDM manner and adds the following component to the latency equation

$$\bar{V} = \frac{\sum_{i=0}^V i^2 * P_i}{\sum_{i=0}^V i * P_i} \quad (4.5)$$

The mean waiting time at any node within the message's path is given as

$$W_b = \frac{m_c * S^2 * (1 + \frac{(S-M)^2}{S^2})}{2 * (1 - m_c * S)} \quad (4.6)$$

where, the latency of the k -ary n -cube network, S , is measured in cycles and m_c is the traffic rate on a given channel and is expressed as

$$m_c = \frac{m_g * \bar{d}}{n} \quad (4.7)$$

The mean waiting time of a message at any ejection channel is given as

$$W_{ej} = \frac{m_g * M^2}{2 * (1 - m_g * M)} \quad (4.8)$$

The mean waiting time in a source node is determined by modeling the injection channel at the source node as an M/G/1 queue gives

$$W_s = \frac{\frac{m_g * S^2 * (1 + \frac{(S - M)^2}{S^2})}{V}}{2 * (1 - \frac{m_g * S}{V})} \quad (4.9)$$

The term $\frac{m_g}{V}$ denotes the mean arrival rate.

The overall message latency is composed of the sum of the mean network latency, S , and the mean waiting time at the source node, W_s , multiplied by the multiplexing factor V to account for the virtual channels multiplexing that takes place over the physical channel. Thus,

$$L_{msg} = (S + W_s) * \bar{V} \quad (4.10)$$

4.3 Latency of packet switched multi-processor shared-bus

The system illustrated in Fig. 4.2 describes a multiple-processor (multiple-memory) packet-based shared bus. A processor-memory communication is only allowed when the bus is not in use by other devices.

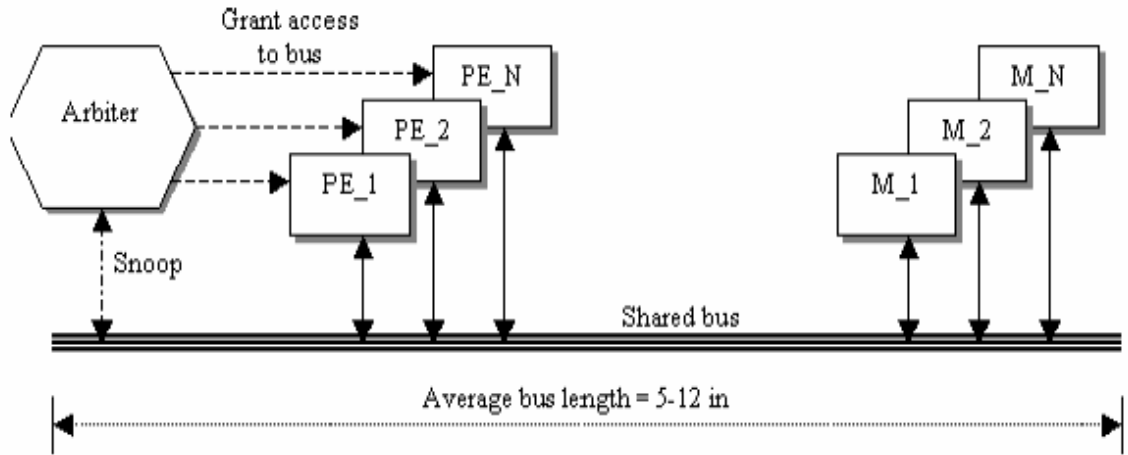


Figure 4.2: Shared-bus multiple processors with arbitration

A processor wishes to communicate with a memory receives permission to send data from the arbiter. The data is transferred using flits. The bus width determines the flit size. When multiple processors wish to use the bus it can cause contention. Therefore, the bus is being monitored (process called also “snooping”) by the arbiter and only the arbiter grants processors access to the bus. Bus of this type fabricated on a PCB line card has an average length between 5 inches to 12 inches. The bus length is an important factor in determining the packet propagation delay through the bus (not including queuing, arbitration or transfer time).

$$L_{sb} = \frac{M}{w} * T_w + T_r + T_q \quad (4.11)$$

Equ. 4.11 represents the latency of a shared bus (more detailed analysis will follow). The first component represents the delay associated with packet propagation from source to destination through the bus. $\frac{M}{w}$ represents the number of packets to send, T_w is the time it takes for a packet to propagate in 1 inch = 254 psec). T_r represents the

arbitration delay. Arbitration latency is determined by the arbitration algorithm used and can vary for each scheme. The most common arbitration schemes include Least Recently Used (LRU) in which arbitration = $(N-1)$ bus cycles, Rotating Daisy Chain (RDC) has the same arbitration latency as LRU and last, FCFS scheme (First-Comes First-Served) with arbitration = $(N-1)$ [54]. The arbitration component must be multiplied by the average time a packet spends within the processor queue before it is being transmitted (Fig. 4.3).

Both components provide the queuing latency. For example, in FCFS scheme the queuing latency is equal to $(N-1)*r$. The value of r varies in each system depending on the queuing model which represents the bus type and communicating modules connected to it. A packet-based multiple processors shared bus such as the one depicted above can be modeled as M/D/1 queue.

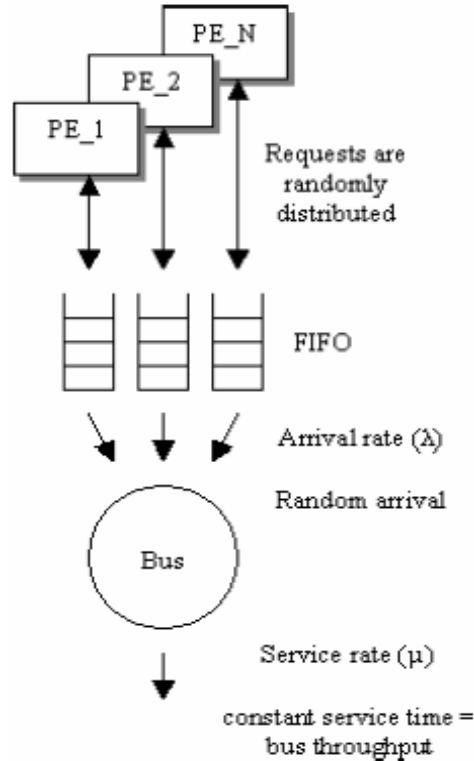


Figure 4.3: Shared-bus arbitration

M/D/1 queue model has a Poisson input (random bus access requests by processors) to a single-server queue with constant service times (this is the bus throughput), unlimited capacity and FIFO queue discipline (Fig. 4.3). Now that arbitration and queuing issues are covered, we can provide a more detailed latency equation:

$$L_{sb} = \frac{M}{w} * T_w * (2n-1) + (n-1) * r \quad (4.12)$$

Equ. 4.12 includes the arbitration cycles (n-1 cycles to wait) multiplied by the average queuing time per message. In addition, it incorporates the factor (2n-1) which counts for the spacing required (as a function of bus segments) between modules connected to the bus.

4.3.1 M/D/1 queue characteristic equations

The characteristic equations of M/D/1 queue are given below:

$$L_q = \frac{\lambda^2}{2 * \mu * (\mu - \lambda)} \quad (4.13)$$

$$\rho = \frac{\lambda}{\mu} \quad (4.14)$$

$$W_s = \frac{\rho}{2 * \mu * (1 - \rho)} \quad (4.15)$$

Equ. 4.13 represents the average queue length, equ. 4.14 symbolizes the queue utilization and equ. 4.15 corresponds to the message waiting time in the queue.

Arrival rate, λ , is the rate in which processor requires memory access. The arrival rate is a function of the incoming packet rate (line rate) and the number of memory accesses performed by the processors in order to complete all the packet analysis operations. To determine the arrival rate, we need to utilize the following data: message size = 40 Bytes (we use the lowest IP packet length for worst-case analysis) and line rate = 40 Gbps. Therefore, if the line rate is 40 Gbps it takes 25 psec per bit to arrive which converts into 0.2 nsec per Byte of data and therefore, 8.8 nsec per packet. This implies that every 8.8 nsec a new packet arrives.

$$\lambda = \frac{1}{8.8 \text{ nsec}} = 1.1 * 10^8$$

Service rate, μ , is the rate in which the bus can service a packet once a processor granted access to the bus. This rate is determined by the bus throughput (bits per second). For example, if the bus length is 10 inches and bus width is 64 bits, the service rate is equal to (message size = 40 Bytes),

$$\mu = \frac{1}{\frac{40 * 8}{64} * 100 \text{ psec} * 2.54 * 5} = \frac{1}{5} \text{ nsec} = 0.157 * 10^9$$

Note that,

$$\sum_{i=1}^{PE} \lambda_i < \mu \quad (4.16)$$

The sum of all arrival rates cannot exceed the service rate of the system. Moreover, λ is used in our model as an aggregate value of all issued communications by PEs to the shared-bus. Using previous example values, system utilization is equal to:

$$\rho = \frac{\lambda}{\mu} = \frac{1.1 * 10^8}{0.157 * 10^9} = 0.7 \quad (70\% \text{ utilization})$$

Therefore, the average wait time in the queue is equal to:

$$W = \frac{0.70}{2 * 2 * 10^8 (1 - 0.70)} = 5.833$$

The average packet waiting time in queue (W) is multiplied by $(N-1)$, where $N = \#$ proc. ($N=4$, for example), to calculate the sum of arbitration latency + queue waiting time.

Thus,

$$L = \frac{40B}{64} * 100 psec * 2.54 + (N - 1) * W = \frac{40B}{64} * 100 psec + (4 - 1) * 5.833 nsec = 18.877 nsec$$

4.4 Performance results k-ary n-cube interconnect vs. shared-bus

Performance of k -ary n -cube is compared against a shared-bus. Since line cards contain multiple network processing elements, which can reach 64 co-processors (including memory modules), we chose two combinations of k -ary n -cubes which comprised of 64 nodes and evaluate their performance. Both shared-bus and the k -ary n -cube have 32 bits channel width. Fig. 4.4 compares the latency of 8-ary 2-cube network with 4-ary 3-cube network while traffic load increases. Moreover, performance for both types of k -ary n -cubes were analytically computed using 32-flit and 64-flit messages.

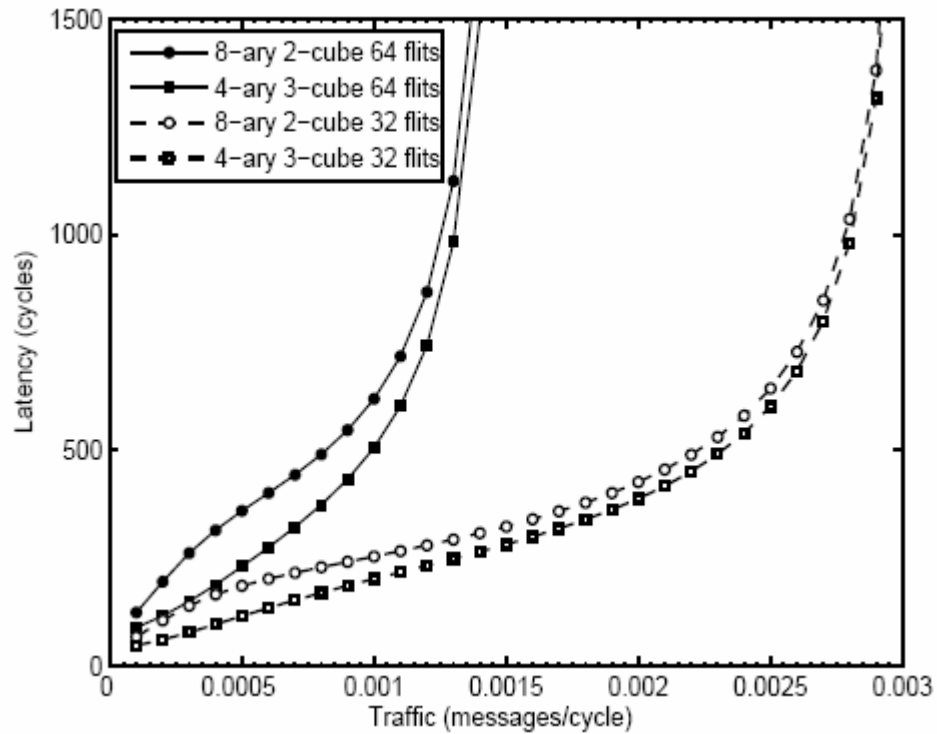


Figure 4.4: Latency comparison between 4-ary 3-cube and 8-ary 2-cube

Fig. 4.4 depicts that 4-ary 3-cube network is superior to 8-ary 2-cube network with respect to load vs. latency for both 32 and 64 flits message. Once the better k -ary n -cube was chosen we compared its performance against a shared-bus (Fig. 4.5).

Fig. 4.5 portrays latency comparison results for shared bus vs. 4-ary 3-cube network. In both interconnects the channel width is 32-bits. 4-ary 3-cube network was able to sustain much higher traffic rate while keeping lower latency than its competitor the shared-bus. Moreover, the 4-ary 3-cube network maintained its exceptional latency for both low as well as high traffic loads.

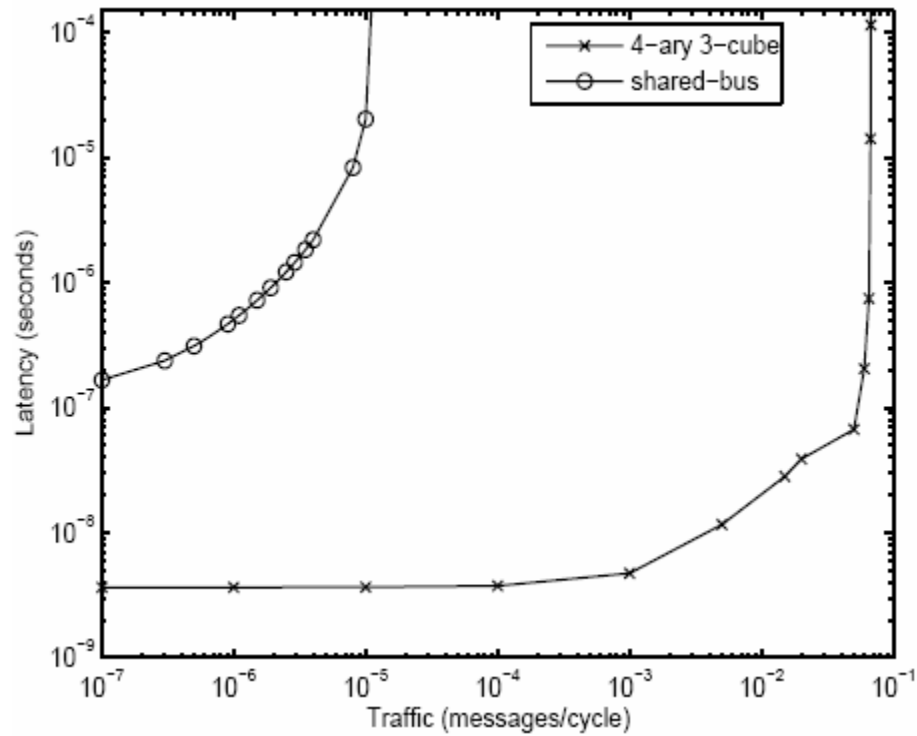


Figure 4.5: Latency comparison between 4-ary 3-cube and shared-bus

4.5 Average distance of 8-ary 2-cube network and 4-ary 3-cube interconnects with multiple configurations

Previous average distance was given for standard k -ary n -cube networks. However, for our application we need to consider certain type of configurations based on the PE and M locations. The average distance, \bar{d} , is calculated for both the 8-ary 2-cube and 4-ary 3-cube networks. The goal is to find which network will result in lower average distance. Moreover, after such network is explored a certain configuration will be chosen within the k -ary n -cube network with the lowest average distance.

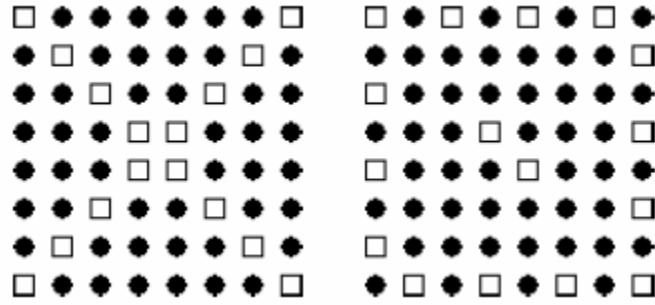


Figure 4.6: 8-ary 2-cube: a) Configuration 1 b) Configuration 2

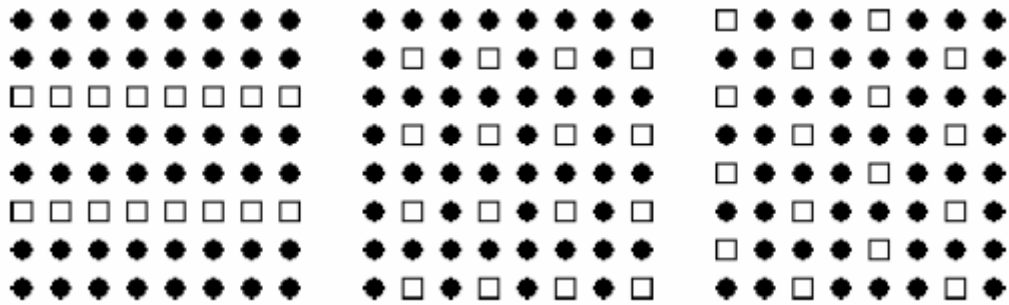


Figure 4.7: 8-ary 2-cube: a) Configuration 3 b) Configuration 4 c) Configuration 5

Since the latency of k -ary n -cube is dependent on the average distance, \bar{d} , as \bar{d} decreases the average latency should decrease as well (Equ. 4.37). The magnitude of the effect which \bar{d} has on the k -ary n -cubes will be investigated and presented visually later on.

Table 4.1: Average distance of 8-ary 2-cube for different configurations

| Configuration | \bar{d} |
|-----------------|-----------|
| Configuration 1 | 4.9492 |
| Configuration 2 | 4.9863 |
| Configuration 3 | 4.9824 |
| Configuration 4 | 5.0234 |
| Configuration 5 | 5.3105 |

First the average distance for 8-ary 2-cube network is calculated, the results are summarized in table 4.1. In the figures, squares represent PEs and circles represent memories. Fig. 4.8 shows a 4-ary 3-cube network nodes structure. Each face includes 4-by-4 nodes in 3-dimensions module. Each node is connected to its adjacent nodes with 4 I/O ports.

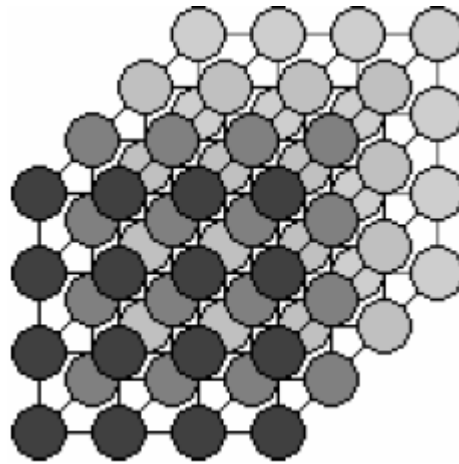


Figure 4.8: 4-ary 3-cube network

Average distance of 4-ary 3-cube network was calculated using the following configurations (Figs. 4.9-4.11). The first configuration includes 4 faces (4-ary) with the same structure as shown in Fig. 4.9a. The second configuration (Fig. 4.9b) has two faces of each of the shown structures. Same connectivity occurs in Fig. 4.9c, Fig. 4.10d, Fig. 4.10e and Fig. 4.11f. Configuration Fig. 4.11g has one face of each.

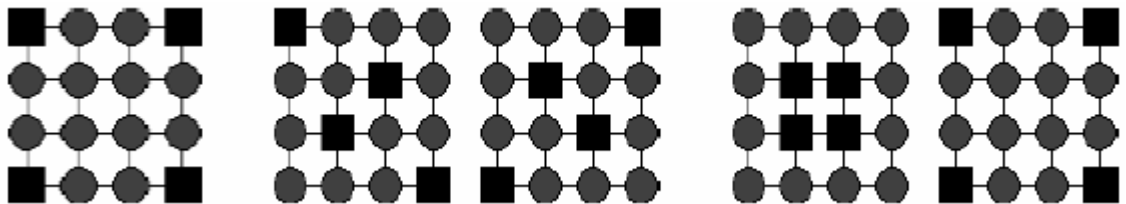


Figure 4.9: 4-ary 3-cube: a) configuration 1 b) configuration 2 c) configuration 3

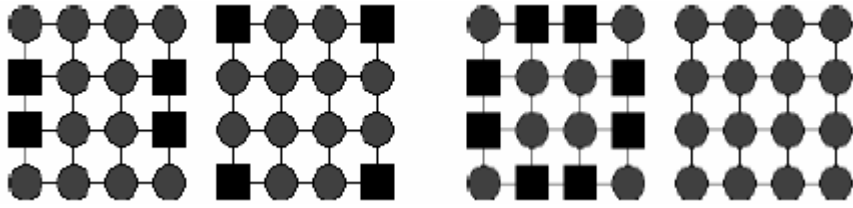


Figure 4.10: 4-ary 3-cube: d) configuration 4 e) configuration 5

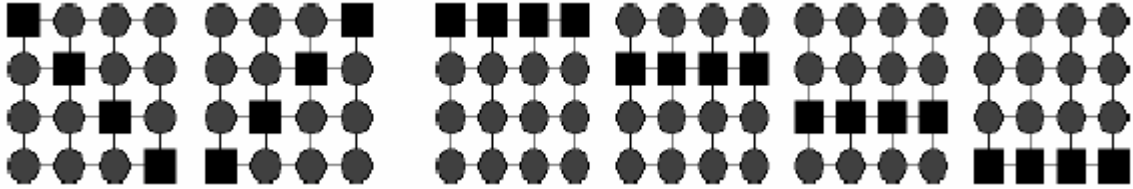


Figure 4.11: 4-ary 3-cube: f) configuration 6 g) configuration 7

Comparing table 4.1 with table 4.2 we conclude that the average distance diminishes as the dimensions of the k -ary n -cube network increase. Moreover, for each k -ary n -cube type, different configurations show very close \bar{d} value. Therefore, there is higher flexibility of choosing any one of the configurations to satisfy system needs and constraints.

Table 4.2: Average distance of 4-ary 3-cube for different configurations

| Configuration | \bar{d} |
|-----------------|-----------|
| Configuration 1 | 4.0143 |
| Configuration 2 | 3.5332 |
| Configuration 3 | 3.6074 |
| Configuration 4 | 3.5332 |
| Configuration 5 | 3.533 |
| Configuration 6 | 3.533 |
| Configuration 7 | 3.767 |

4.6 Analytical model of k-ary n-cube interconnect with hot-spot traffic

Hot-spot traffic refers to network nodes which experience high traffic load. A hot-spot is created in two scenarios. First, when multiple sources transmit messages to the same destination node, second, when multiple messages adaptively route itself through the same node increasing the traffic and result in local congestion. The latency model for k -ary n -cubes was initially modeled by M. O. Khaoua [48]. The traffic load level of a node, or hot-spot, is measured in terms of probability. Each generated message has a certain and finite probability, α , of being directed to a hot-spot node and probability of $(1-\alpha)$ to be directed elsewhere (through other nodes). The latency equation under hot-spot traffic is given as

$$Latency = (\bar{S} + \bar{W}_s) * \bar{V} \quad (4.17)$$

where \bar{S} is the mean network latency, \bar{W}_s represents the mean waiting time seen by the message and \bar{V} captures the effect of channel multiplexing through virtual channels at each node.

Regular and hot-spot messages see different network latencies as they pass through different channels since each experience different traffic rate and as a result different blocking latency depending from its distance from high traffic load, thus hot-spot. The mean network latency, while taking into account both message types (hot-spot and regular) is equal to,

$$Latency = (1 - \alpha) * S_r + \alpha * S_{hs} \quad (4.18)$$

where S_r represents the latency seen by regular messages which do not experience hot-spot traffic and S_{hs} symbolizes message latency as a result of hot-spot traffic. Given that N -nodes in k -ary n -cube are generating $\alpha * \lambda$ hot-spot messages per cycle, the rate of hot-spot traffic is

$$\lambda_{hj} = \alpha * N * \lambda * P_{hj} \quad (4.19)$$

where P_{hj} represents the probability that a message has used during its network journey a particular channel located j hops away from hot-spot node.

$$P_{hj} = \frac{n - \sum_{l=0}^{j-1} D_l}{C_j N} = \frac{\sum_{l=j}^{n(k-1)} D_l}{C_j N} \quad (4.20)$$

Unlike regular message, a hot-spot message encounters different blocking times at different channels due to non-uniform traffic rates. A hot-spot message may visit $1, 2, \dots, n(k-1)$ channels to reach the hot-spot node. Thus, C_j corresponds to the total number of channels which are j hops away from a given node, C_j , as

$$C_j = \sum_{l=0}^{n-1} \sum_{t=0}^{n-l} (-1)^t (n-l) \binom{n}{l} \binom{n-l}{t} \binom{j-t(k-1)-1}{n-l-1} \quad (4.21)$$

D_i represents the number of nodes that are i hops away from a given node and is calculated by

$$D_i = \sum_{l=0}^n (-1)^l \binom{n}{l} \binom{i-lk+n-1}{n-1} \quad (4.22)$$

The network latency seen by hot-spot message j hops away from the hot-spot node, is given by

$$S_{hj} = M + j + \sum_{m=1}^j B_{h_{mj}} \quad (4.23)$$

where $B_{h_{mj}}$ symbolizes the blocking time of a j -hop hot-spot message at its m^{th} hop channel. In order to obtain the probability that a node is j hops away from a given hot-spot node, we divide the total number of nodes that are j hops away from a hot-spot node by the total number of nodes in the network, that is

$$\theta_j = \frac{D}{N-1} \quad (4.24)$$

Hence, when averaging the network latency over all possible hops seen by a hot-spot message, S_h can be expressed as

$$\bar{S}_h = \sum_{j=1}^{n(k-1)} \theta_j S_{h_j} \quad (4.25)$$

When the message reaches the m^{th} hop channel, it is $(j-m+1)$ hops away from the hot spot node. Therefore, the mean blocking time is written as

$$B_{h_{mj}} = \varphi(j-m+1)\omega(j-m+1) = \varphi_{(\bar{k})}\omega_{(\bar{k})} \quad (4.26)$$

This expression is comprised of the probability of blocking, $\varphi(j-m+1)$ and the average waiting time, $\omega(j-m+1)$, for hot-spot message.

The mean waiting time is given by

$$\omega_j = \frac{\lambda_j S_j^2 \left(1 + \frac{(S_j - M)^2}{S_j^2}\right)}{2(1 - \lambda_j S_j)} \quad (4.27)$$

where λ_j represents the rate of messages (both regular and hot-spot) arriving at the channel and S_j corresponds to the mean service time of a node located j -hops away from a hot-spot message. Both parameters can be determined by

$$\lambda_j = \lambda_r + \lambda_{h_j} \quad (4.28)$$

$$S_j = \frac{\lambda_r}{\lambda_j} S_r + \frac{\lambda_{h_j}}{\lambda_j} S_{h_j} \quad (4.29)$$

By modeling a source node as M/G/1 queue we can extract the mean waiting time of messages at the source node (with mean arrival rate $\frac{\lambda}{V}$, where V is the number of virtual channels)

$$W_{S_j} = \frac{\frac{\lambda}{V} S_{s_j}^2 \left(1 + \frac{(S_{s_j} - M)^2}{S_{s_j}^2}\right)}{2\left(1 - \frac{\lambda}{V} S_{s_j}\right)} \quad (4.30)$$

The mean network latency for a message that originates at a source node that is located j hops away from a hot-spot node, S_{s_j} is

$$S_{s_j} = (1 - \alpha) S_r + \alpha S_{h_j} \quad (4.31)$$

Averaging over all possible values of j ($1 \leq j \leq n(k-1)$) gives the mean waiting time in a source node as

$$W_s = \sum_{j=1}^{n(k-1)} \theta_j W_{s_j} \quad (4.32)$$

The probability, p_{vj} , that v virtual channels are busy at the physical channel that is j hops away from hot-spot node is based on using a Markovian model. In the steady-state, the model yields the following probabilities

$$Q_{v_j} = \begin{cases} \lambda_j^v S_j^v & 0 \leq v \leq V-1 \\ \frac{\lambda_j^v S_j^v}{1 - \lambda_j^v S_j^v} & v=V \end{cases}$$

$$P_{v_j} = \begin{cases} (\sum_{i=0}^V Q_{ij})^{-1} & v=0 \\ P_0 Q_{v_j} & 1 \leq v \leq V \end{cases}$$

In virtual channel flow control multiple virtual channels share the bandwidth of a physical channel in a time multiplexed manner. The average degree of multiplexing of virtual channels (over all possible values of j) and which takes place at the physical channel is provided by

$$\bar{V} = \sum_{j=1}^{n(k-1)} \theta_j \frac{\sum_{v=1}^V v^2 p_{v_j}}{\sum_{v=1}^V v p_{v_j}} \quad (4.33)$$

4.7 Performance results of k -ary n -cube interconnect with hot-spot traffic

In this section we provide performance results of k -ary n -cube interconnects with or without hot-spots and compare it against shared-bus. Our goal is to evaluate the effect of hot-spots on the interconnect latency and to find which traffic load level brings the interconnect to complete saturation where it is unable to sustain higher traffic load and as a result its latency becomes infinite.

Figure 4.12 compares the latency between 8-ary 2-cube network with and without hot-spots vs. 32-bits shared bus. Although hot-spots increase the latency of the 8-ary 2-cube network, the network was able to sustain higher traffic rate than shared-bus.

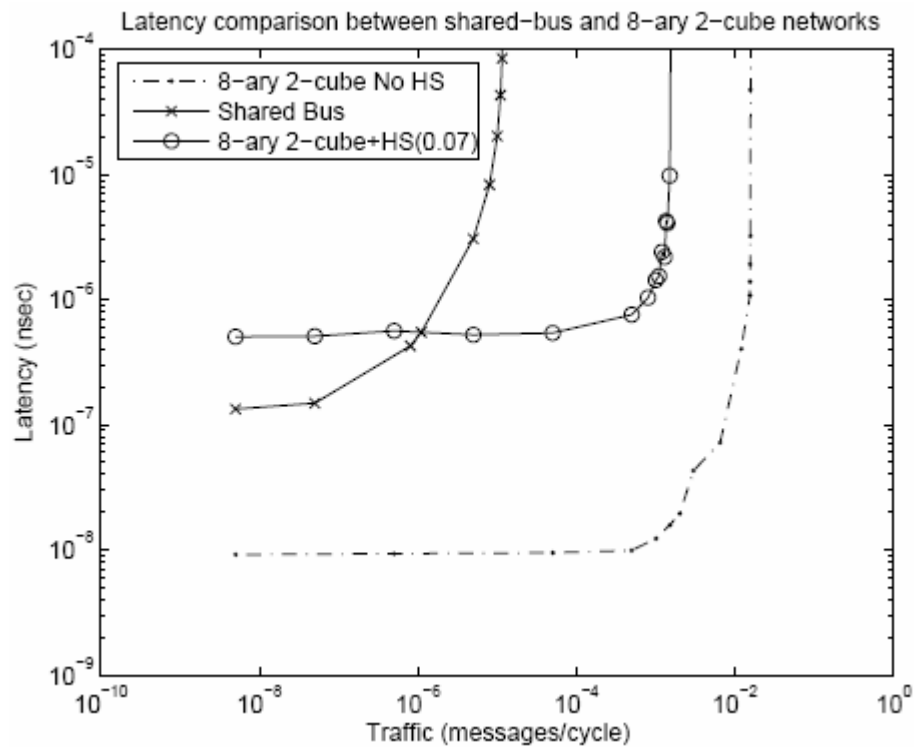


Figure 4.12: Latency comparison between SB and 8-ary 2-cube

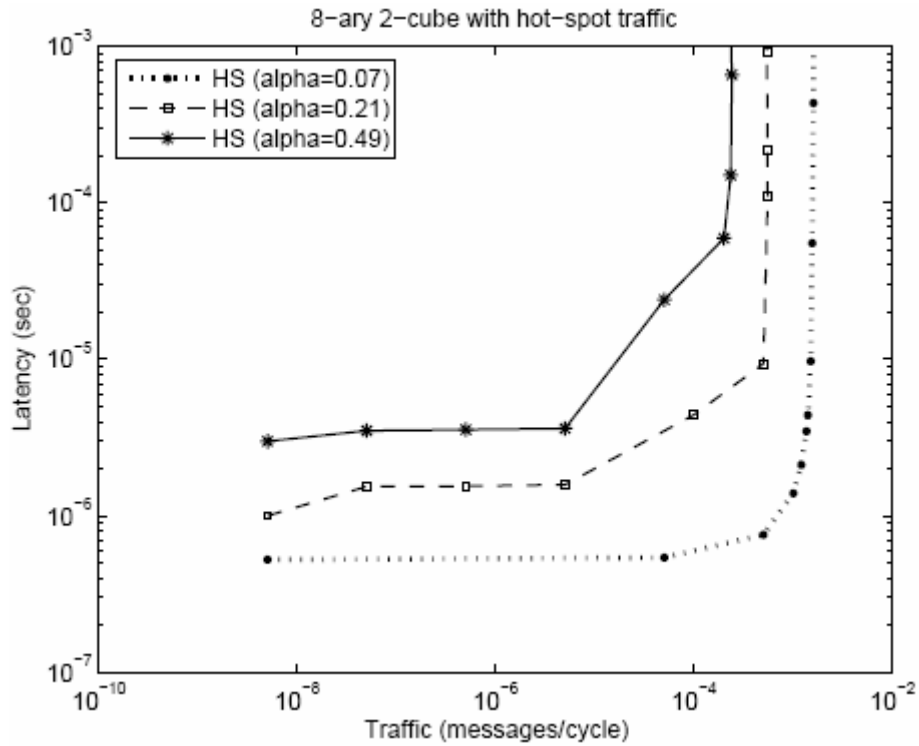
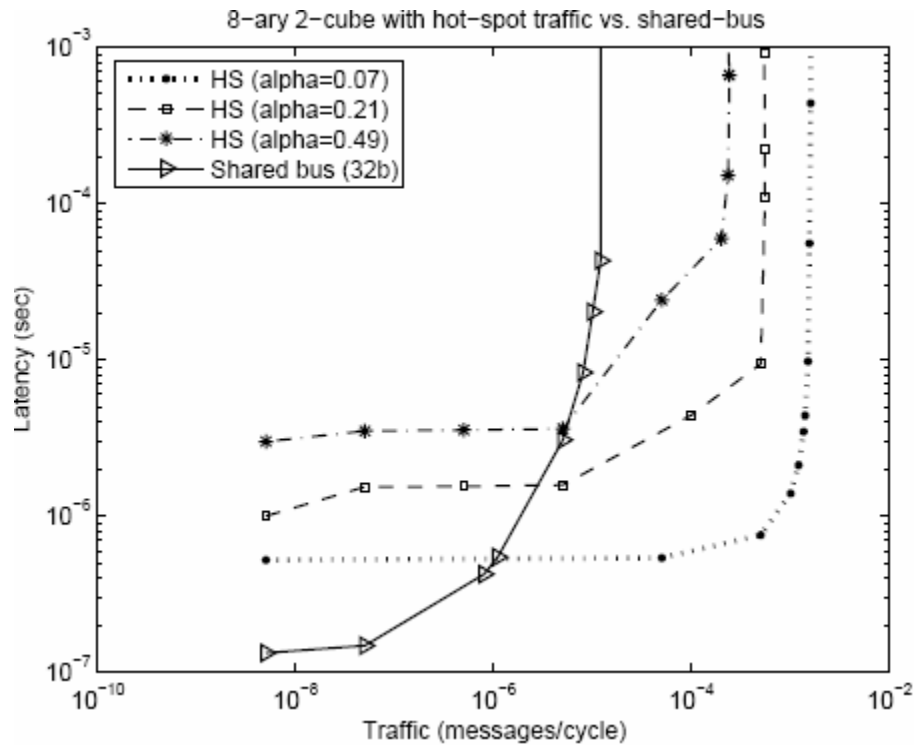


Figure 4.13: 8-ary 2-cube with HS

Figure 4.13 depicts the effect of hot-spot traffic on latency as the hot-spot probability, α , increases. This figure depicts that as hot-spots probability increases the interconnect latency increases as well. Moreover, there is an upward shift in the latency curve with respect to the hot-spots rate increase.

Figure 4.14 is a comparison between 8-ary 2-cube network latency with different levels of hot-spot traffic vs. shared bus (32 bits). We conclude that at low traffic rates shared bus latency is lower and therefore better. The reasoning behind this is that blocking has more influence on latency, in 8-ary 2-cube, at low rates while blocking latency does not exist in shared bus.



4.14: 8-ary 2-cube with HS vs. shared-bus

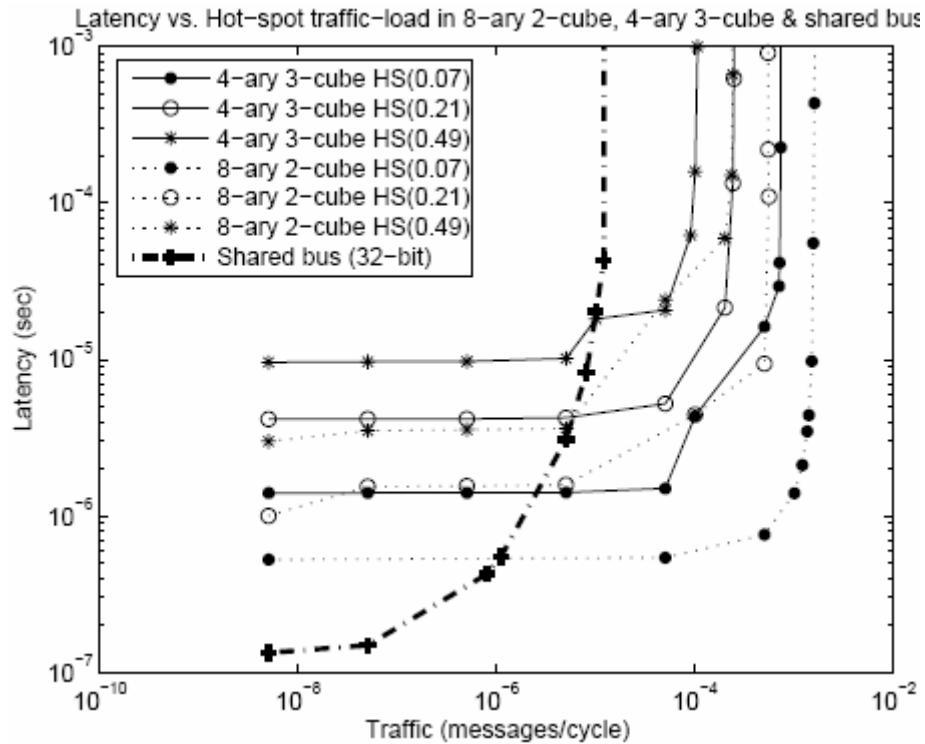


Figure 4.15: 8-ary 2-cube vs. 4-ary 3-cube vs. shared bus (32 bits)

As traffic rate increases, shared-bus reaches its maximum throughput and arbitration latency becomes dominant, while its counterpart, the 8-ary 2-cube network, sustains higher traffic rate at lower network latency.

Figure 4.15 includes the latency of 4-ary 3-cube network under non-uniform traffic. We chose 4-ary 3-cube network and 8-ary 2-cube network since both have 64 total nodes but different dimensions. The figure portrays that 8-ary 2-cube network sustained higher traffic rates than 4-ary 3-cube network. In addition, 4-ary 3-cube network has slightly higher latency than the 8-ary 2-cube. Both networks sustain higher traffic rates even with hot-spots than shared bus.

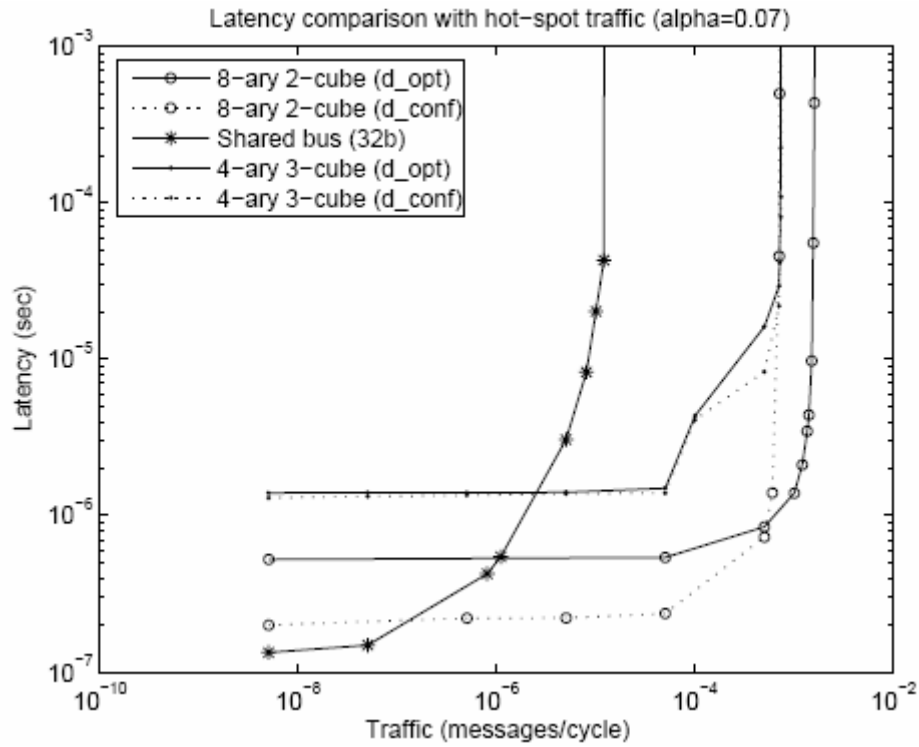


Figure 4.16: Latency comparison between 8-ary 2-cube and 4-ary 3-cube HS traffic with different D_{ave}

Figure 4.16 depicts the latency differences as d_{ave} changes. The dotted lines represent the latency as d_{ave} decreases. D_{ave} can be calculated by using the standard k -ary n -cube equation, $d = n * \bar{k}$, or the configured value which we obtain by configuring the placement of memories and processors within the network. D_{conf} , which represents the average distance for different M/PE configurations, is lower than the theoretical value since we do not assume communication between memories under non-uniform traffic. Moreover, d_{ave} for different configurations was very close in value and therefore, provides us with the flexibility of choosing one configuration that will best meet our constraints.

4.8 3D-mesh performance analysis

4.8.1 Cube notation

Cubes are connected in series to form the 3D-mesh. Each vertical cube plane is denoted by i and it is incremented along the right-hand side of the x -axis direction as depicted in Fig. 4.17.

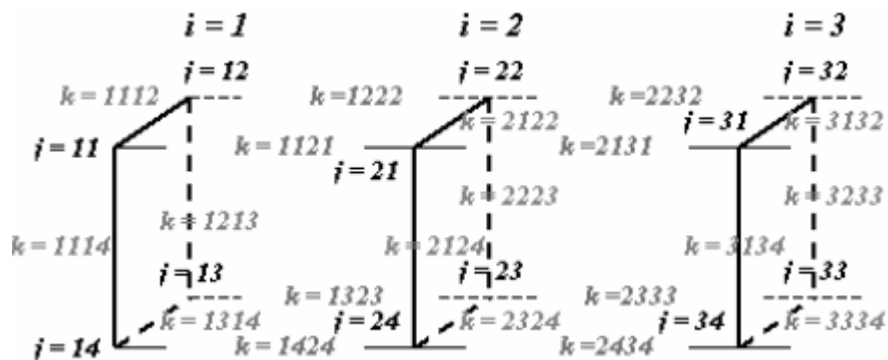


Figure 4.17: 3D-mesh interconnection notation

Within each i^{th} plane there are four corners denoted by j , moving in a clockwise direction. The k notation is used to distinguish the links on the cube and is formed by combining two js . We use minimal routing algorithm which favors shortest path for each packet.

4.8.2 3D-mesh average distance analysis under non-uniform traffic

Our goal is to find which processor-memory configuration will result in minimum average distance. One important issue is that communications are non-uniformly distributed within the cube-interconnect and as a result create more processor-memory traffic (vs. memory to memory). Each processor can communicate with all memories connected to it directly or indirectly (not adjacent to it). Memories, on the other hand, communicate with all processors and communication between memories is allowed only through a processor. Therefore, traffic is distributed non-uniformly creating “hot spots” in areas of higher congestion. The average distance equation must be adjusted to emphasize this property. The 3D-mesh interconnect consists of multiple faces ($i = 1, 2, 3, \dots$)

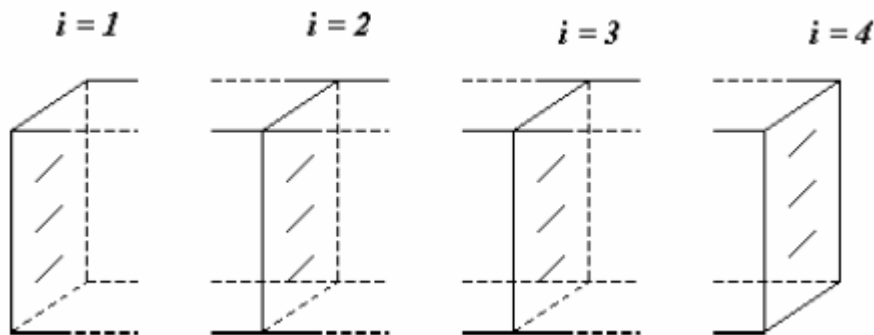


Figure 4.18: 3D-Mesh faces

Each face has a unique binary address starting at “000...0”. The number of bits representing this address is equal to $\log_2(\#faces)$. Within each face there are four nodes with addresses “00”, “01”, “10”, “11” assigned in clockwise direction. Average distance for different processor-memory configurations is calculated by comparing the source node address and destination node address in two steps. First, the two lower bits of the address, which represent the node address (location) within the face, are compared. The number of 1's resulting from XOR(source,destination) of the last two binary digits is summed up and stored in temporary variable. In the second step, the addresses of the two faces are subtracted from each other and the binary result (converted to integer) is added to the previously stored temporary value.

Equ. 4.34 is the general case of calculating average distance. Equ. 4.35 and Equ. 4.36 show the average case for PEs and memories respectively.

$$E_{\sigma} = \sum \frac{\sigma_i}{N} \quad (4.34)$$

$$E_{PE} = \frac{(\sum \sigma_i + \sum \sigma_j)}{P + M} \quad (4.35)$$

$$E_M = \frac{(\sum \sigma_i + \sum \sigma_j)}{P} \quad (4.36)$$

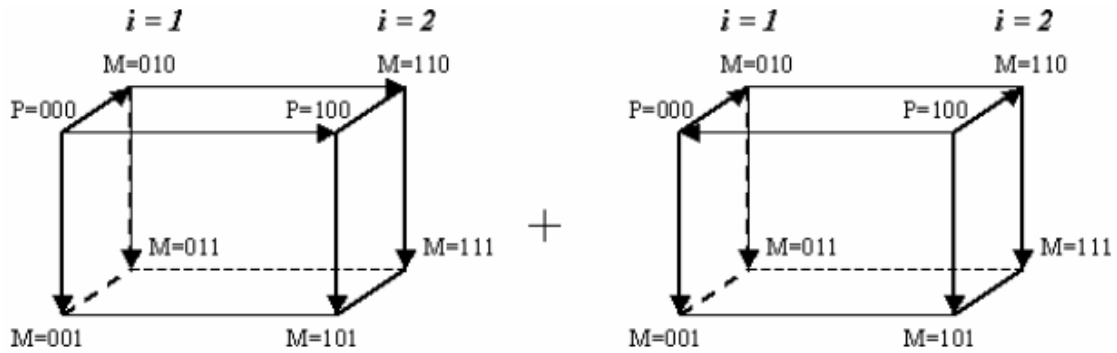


Figure 4.19: Non-uniform traffic for PEs in configuration 1

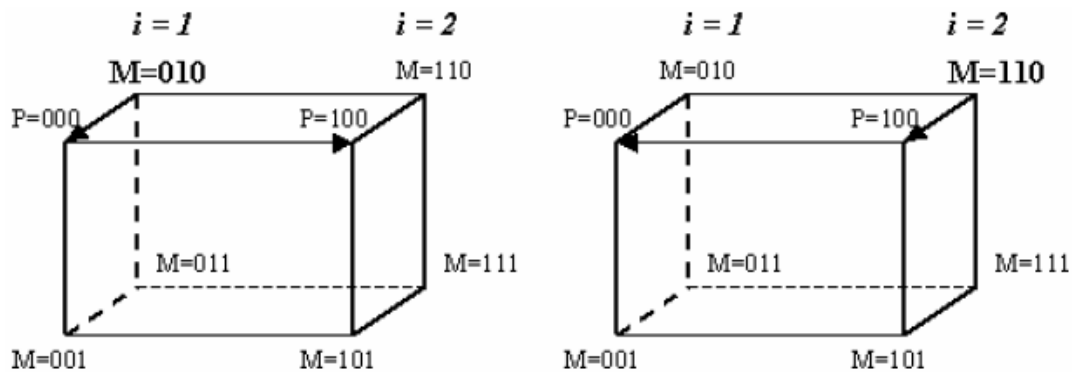


Figure 4.20: Non-uniform traffic for M in configuration 1

Under non-uniform communication all PEs can communicate with all other modules (PEs as well as memories). On the other hand, memories communicate only with PEs and therefore, it results in different average distance for memories.

The PEs average distance is equal to the sum of all distances from each one of the PEs to all other nodes divided by the total number of nodes in the interconnect. The memories average distance is equal to the sum of distances from each memory to each PE. This sum is divided by the total number of PEs in the interconnect.

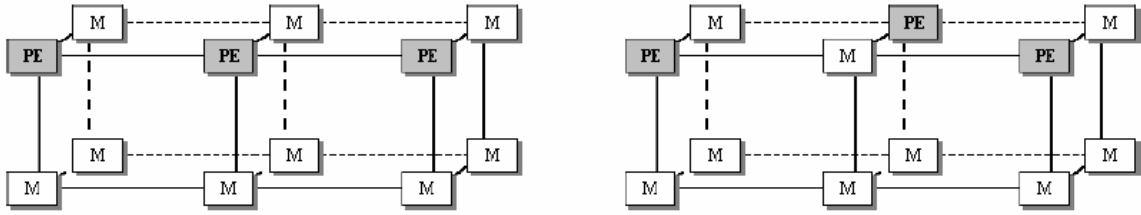


Figure 4.21: 3D-mesh: a) Configuration 1 b) Configuration 2

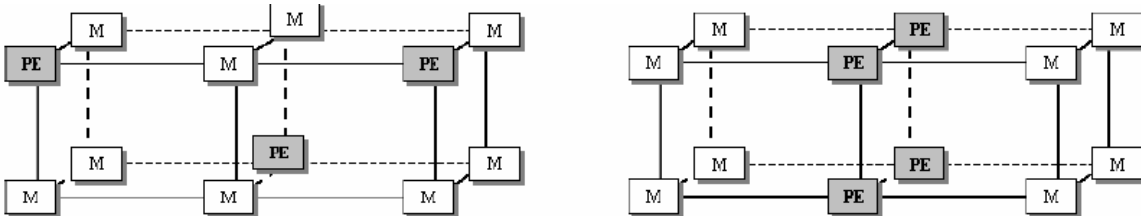


Figure 4.22: 3D-mesh: a) Configuration 3 b) Configuration 4

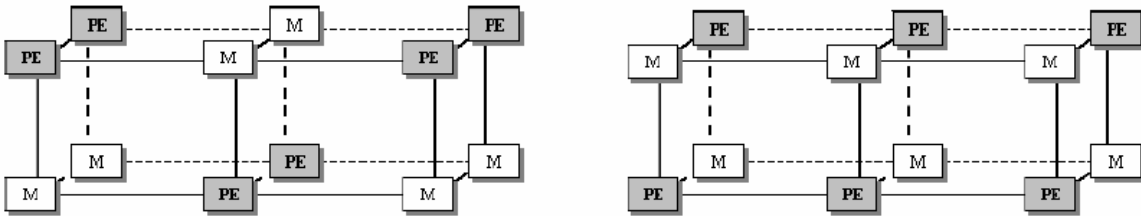


Figure 4.23: 3D-mesh: a) Configuration 5 b) Configuration 6

The number of memories in each configuration is always higher than the number of PEs to allow better distribution of data and reduce the load (and the bandwidth) per memory when peak traffic is achieved. The following figures illustrate the PE-Memory structure in each configuration.

Table 4.3 depicts that there are two configurations which can compete in order to reach lowest average distance, configuration 3 and configuration 4. Those configurations will be used to calculate latency in the following subsection.

Table 4.3: Average distance of 3D-mesh for different configurations

| Configuration/Faces | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---------------------|-------|-------|-------|-------|-------|-------|-------|-------|
| Configuration 1 | 1.333 | 1.804 | 2.182 | 2.538 | 2.884 | 3.226 | 3.566 | 3.904 |
| Configuration 2 | 1.333 | 1.679 | 2.071 | 2.413 | 2.764 | 3.101 | 3.444 | 3.779 |
| Configuration 3 | 1.333 | 1.554 | 1.959 | 2.287 | 2.644 | 2.976 | 3.321 | 3.654 |
| Configuration 4 | 1.333 | 1.857 | 1.939 | 2.283 | 2.895 | 2.986 | 3.267 | 3.464 |
| Configuration 5 | 1.416 | 1.607 | 2.003 | 2.325 | 2.678 | 3.008 | 3.352 | 3.683 |
| Configuration 6 | 1.166 | 1.607 | 1.975 | 2.325 | 2.668 | 3.008 | 3.347 | 3.683 |

4.8.3 3D-mesh latency equation

Our latency equation is based on five factors: routing algorithm (wormhole - packet based switching), interconnect physical characteristics (such as channel width, wire length, propagation delay, switching mechanism and switching delay), Message length, buffering scheme and average distance between nodes using non-uniform communication patterns.

$$Latency_{3D} = D * (T_s + T_w + T_r) + (T_s + T_w) * \frac{L}{w} \quad (4.37)$$

The first term multiplies the average distance, D , calculated earlier by the sum of propagation delay (T_w), switching delay, (T_s , also queuing if required) and routing delay (T_r). It assumes an input buffer in each node used to handle deadlocks by buffering the packets. Each additional queuing will result in additional delay of T_s . $\frac{L}{w}$ is the message length divided by the channel-width. This term determines the number of packets

required to be sent in order to complete the message transmission. Channel width can vary between 8-32 bits depending on chip I/O pins but it is also limited and should not exceed 32 bits per channel. A wider than 32 bits channel-width will increase the channel layout density on the PCB. The term $(T_s + T_w)$ represents the minimum cycle delay of transmitting a new packet.

4.8.4 3D-mesh interconnect vs. shared-bus

Performance evaluation of 3D-mesh is compared against a shared-bus. Our performance model includes statistical data (such as IP distribution) and physical measures (like PCB placement and spacing) in order to increase the accuracy of our calculations. since IP packet length distribution shows that most packets have 40B length, we use this data to evaluate our interconnect with the same packet length. Spacing and placement are another aspect affecting performance. The spacing between modules must include the physical dimensions of each module plus the routing of the interconnect among modules to reach every node connected to it. We are using two possible spacing values (1 inch and 0.5 inch) between modules and check the effect it makes on the interconnect performance.

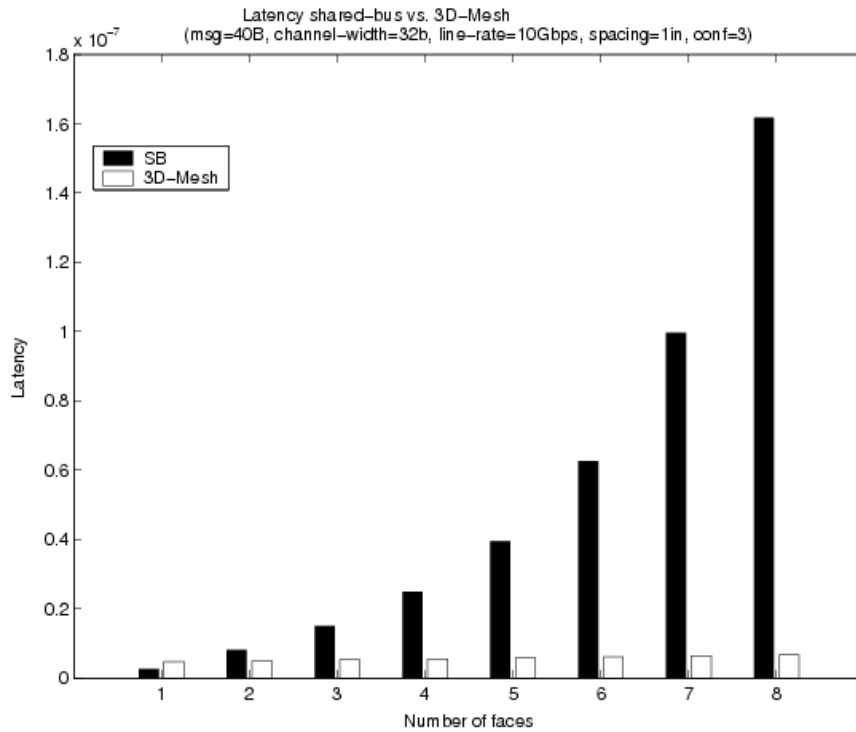


Figure 4.24: Latency SB vs. 3D-mesh with $ch_w=32b$ (1 inch spacing)

Fig. 4.24-4.25 show the latency of shared-bus vs. 3D-mesh interconnect with packet size 40B, channel width of 32 bits and 64 bits and spacing width of 1 inch among adjacent modules. Both figures show that shared-bus (SB) latency is much higher than that of the 3D-mesh interconnect.

Fig. 4.26 depicts a different view of latency comparison between 3D-mesh and shared-bus with 32b and 64b channel width. As the number of faces increase, shared-bus latency rapidly increases while 3D-mesh keeps almost constant latency. Fig. 4.27 emphasizes the differences in latencies by illustrating the latency ratio of 3D-mesh vs. shared-bus for both 32b and 64b channel width.

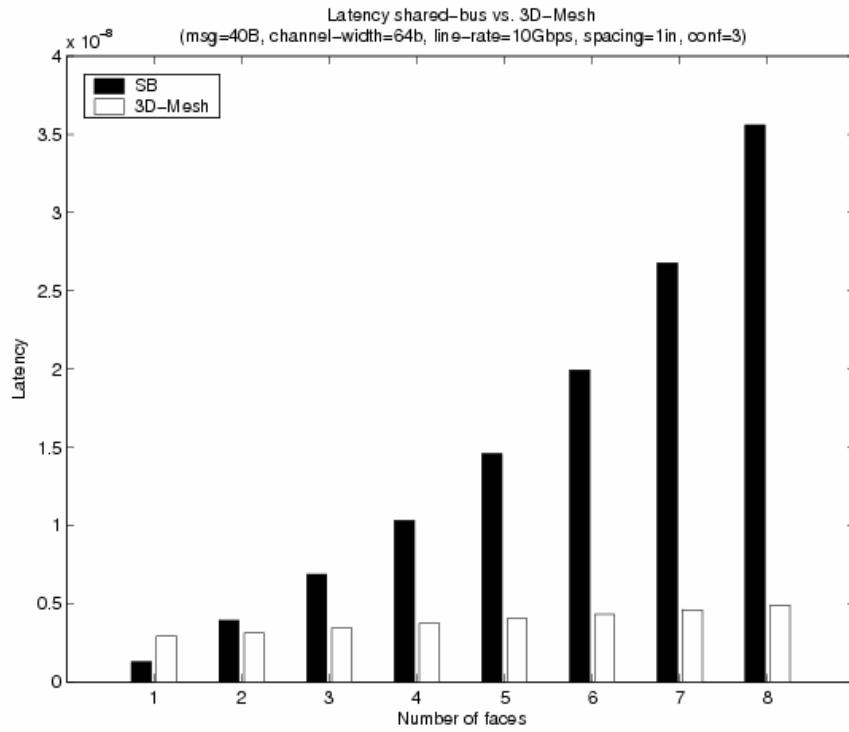


Figure 4.25: Latency SB vs. 3D-mesh with $ch_w=64b$ (1 inch spacing)

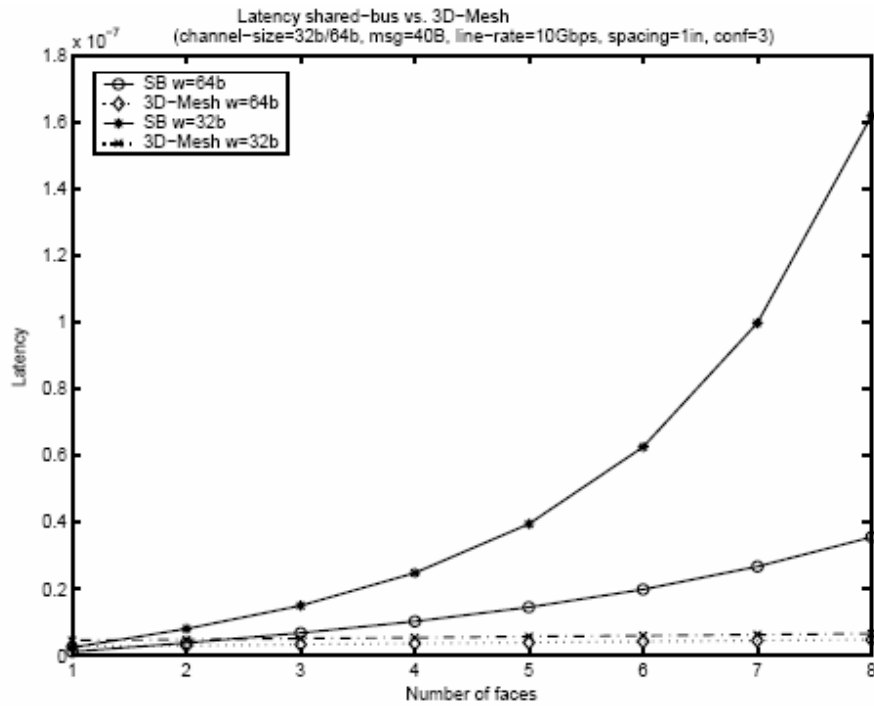


Figure 4.26: Latency comparison SB vs. 3D-mesh for $ch_w=32b, 64b$ (1 inch spacing)

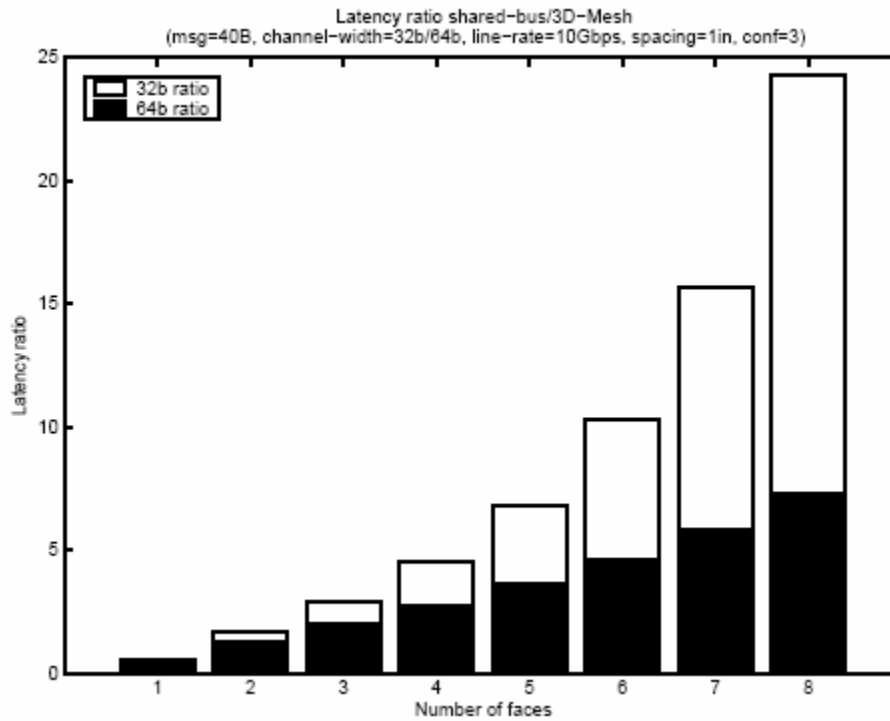


Figure 4.27: Latency ratio SB/3D-mesh for $ch_w=32b, 64b$ (1 inch spacing)

The latency calculations were repeated for channel width of 0.5 inches among adjacent modules. Fig. 4.28 shows that 3D-mesh latency slightly increased while the shared-bus latency decreased. For low number of faces, shared-bus had lower latency than 3D-mesh. As a result the ratio in latencies between shared-bus to 3D-mesh was reduced significantly (Fig. 4.30).

Moreover, both Figs. 4.27-4.28 show that even for closer placement (0.5 inch vs. 1 inch) between modules, shared-bus (SB) latency is higher than that of the 3D-mesh interconnect.

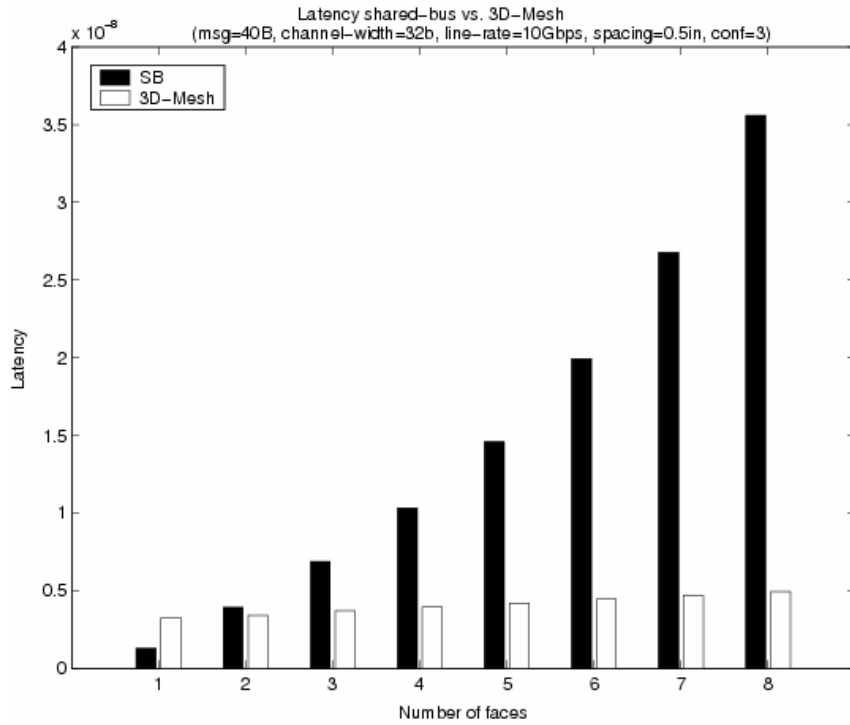


Figure 4.28: Latency SB vs. 3D-mesh with $ch_w=32b$ (0.5 inch spacing)

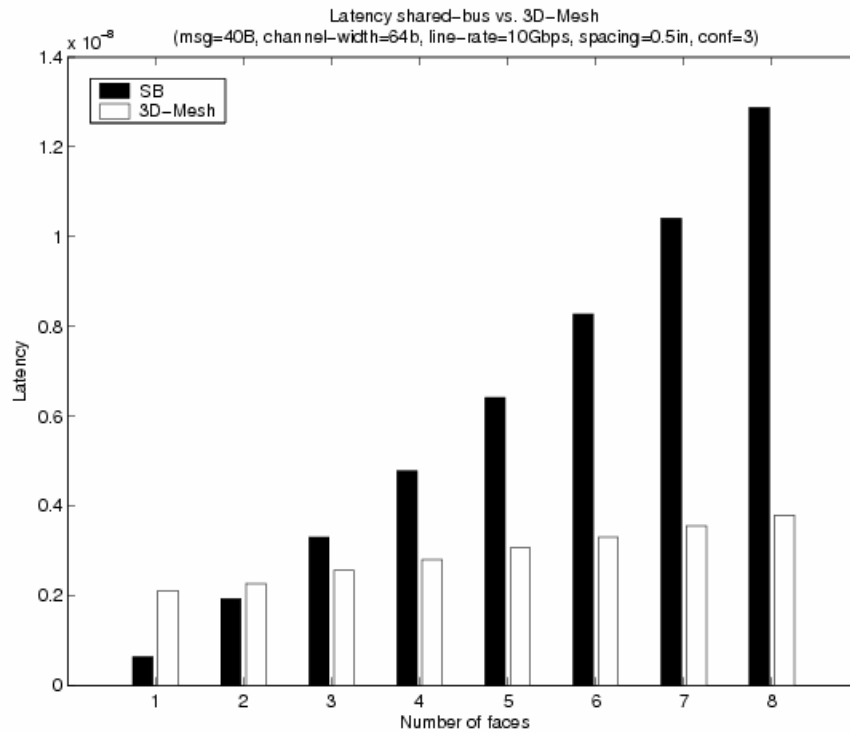


Figure 4.29: Latency SB vs. 3D-mesh with $ch_w=64b$ (0.5 inch spacing)

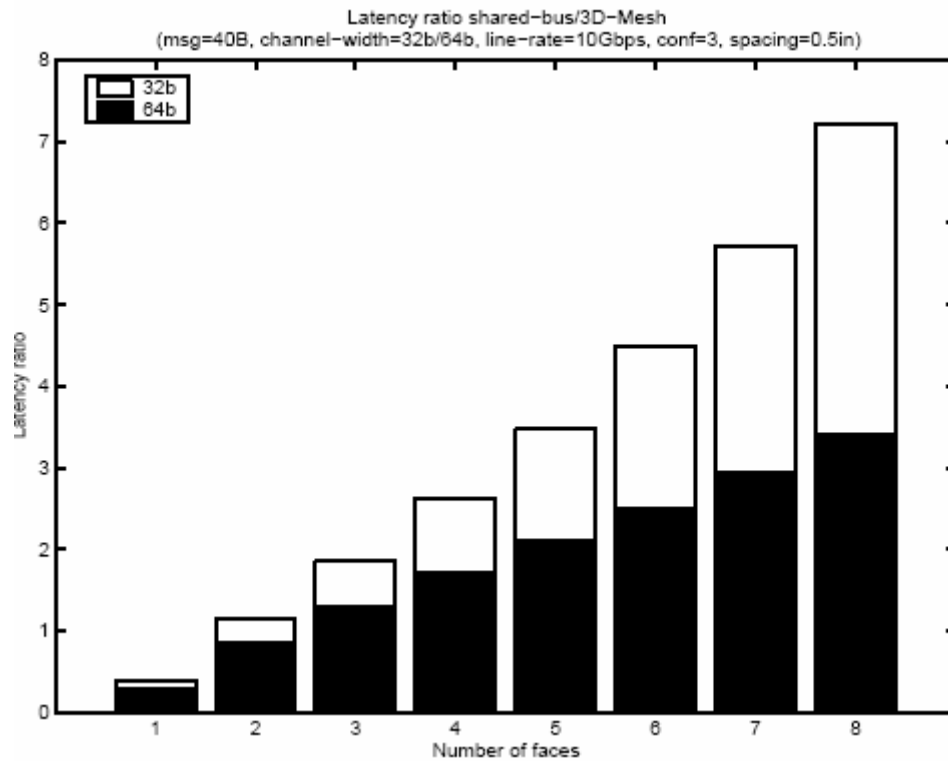


Figure 4.30: Latency ratio SB/3D-mesh for $ch_w=32b, 64b$ (0.5 inch spacing)

4.8.5 3D-mesh interconnect vs. folded-Torus network

The closest interconnect structure to our 3D-mesh is the folded Torus network. A folded 3D Torus is a k -ary 3-cube interconnect. The folded Torus includes additional connections linking the ends of each row and column of the 3-dimensional cube (see Fig. 4.31). It can be easily noticed that the shortest path of the folded Torus network is 2 bus links and the longest path takes maximum of 5 links from source to destination when compared with three 3D cubes connected in series. In Fig. 4.32 the number of cubes is based on the number of 3D-cubes appear in different dimensions of the Torus network. For example, in a 4-dimensional Torus network there are 2 3D-cubes connected as shown

in Fig. 4.32. Since our 3D-interconnect resembles 3D-cubes connected in series to each other, 3D-interconnect performance calculations use 3 cubes vs. 2 cubes for Torus. The same ratio was used for higher dimensions of Torus network vs. 3D-interconnect.

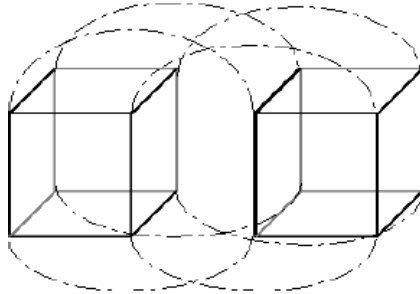


Figure 4.31: 3-Dimensional Torus network

Based on PCB layout model, we map the Torus network in 2D and calculate the additional wire lengths that are connecting the 3D cubes in Torus network with 3, 4 and 5 dimensions. Wire links connecting one pair of 3D cubes to another incorporate additional delay which can be calculated based on the wire delay cost model. This model states that the delay on wire increases logarithmically with the wire length while the channel width size is held constant [12].

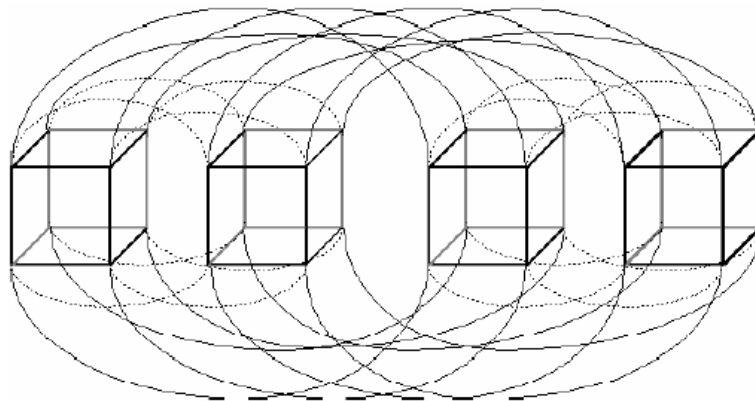


Figure 4.32: 4-Dimensional Torus network

The logarithmic nature of the propagation delay is based on the wire capacitance. As the degree of Torus network dimension increases, the propagation delay (T_c) on the wires connecting remote cubes rises.

$$T_c \propto 1 + \log l = 1 + \left(\frac{d}{2} - 1\right) \log k \quad (4.38)$$

where l represents the units of wire length and $\log l$ signifies the supplementary delay per additional unit length [11]. Consequently, the additional wire length required to connect 3D cubes in Torus network increases by 1 unit length for each additional dimension, when implemented on the printed circuit board [10]. Due to space constraints on the line card, we do not consider more than 16 cubes. We compare the Torus network with our interconnect assuming there is 1 cm difference between all nodes in each cube. This is not including the long lines connecting cubes in multiple dimensions in Torus network. There are three wire lengths which entail longer routing than 1 cm in Torus network as we increase its dimensions. These wires have the notations T' , T'' and T''' . Increasing l by 1 unit length provides the following further delays: $T' = 63.7$ ps, $T'' = 64$ ps, and $T''' = 64.31$ ps (with respect to 100 psec propagation delay per 1 cm of wire).

Performance comparison between a folded Torus network to our 3D-mesh interconnect shows that for lower number of faces (up to 15) 3D-mesh has lower latency. As the number of faces surpasses 15 faces, Torus network has slightly lower latency than our interconnect (shown in Fig. 4.33). The difference in latencies results from the dominance of D (average distance) for the 3D-mesh over the T' , T'' and T''' terms as the number of faces increases. Above 15 faces (in the 3D-mesh case) D is much larger in magnitude than the effect of T' , T'' and T''' on the Torus latency.

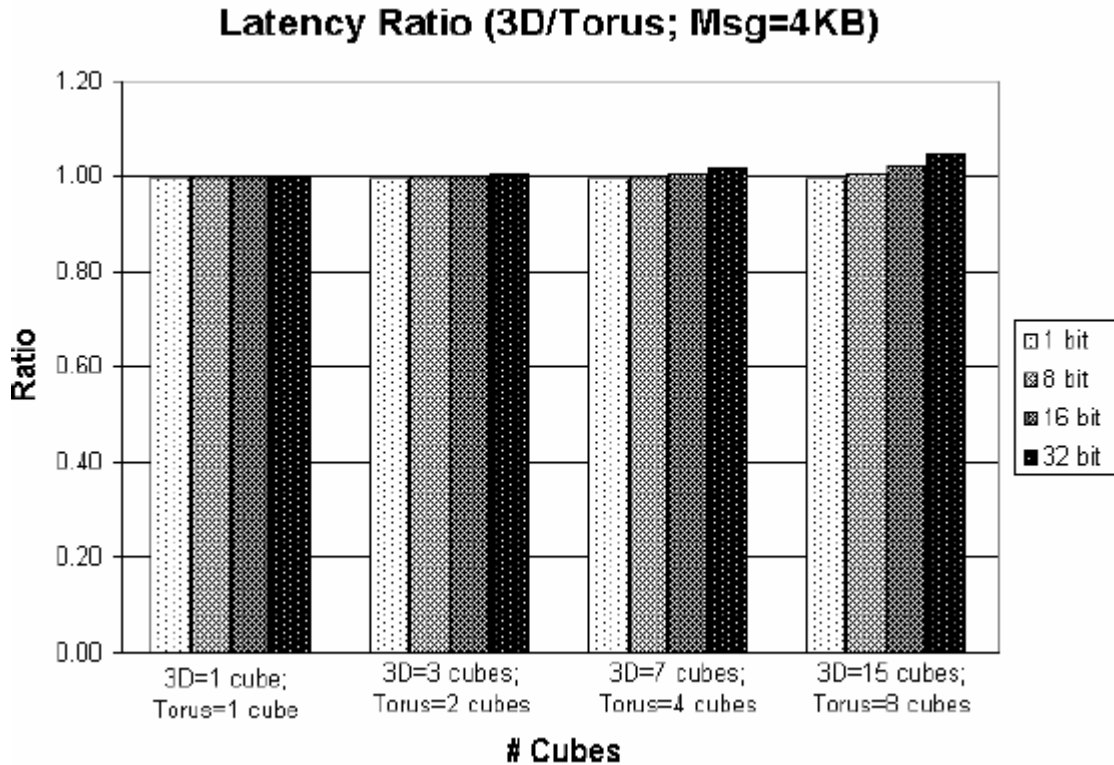


Figure 4.33: Latency Ratio 3D-mesh vs. folded Torus

The tradeoff between number of faces and the latency supports our model over the Torus network. Our 3D-mesh interconnect is used as a chip-to-chip communication network embedded onto a line card. Since there is limited space on the PCB board, the interconnect which consumes less space will be preferred given minimal performance differences. Our 3D-interconnect, if viewed in 2D, uses N straight bus segments in sequence which resemble a parallel bus with multiple switches. On the other hand, folded Torus network requires double the bus width to connect source device to destination which increases the cost and wire density of the interconnect layout. 3D-interconnect is more feasible choice and provides high performance for its application under strict physical constraints.

4.9 3D-bus performance analysis

We adopt Dally's basic equation for k -ary n -cube interconnects [12] and modified it on our design. The resulting latency (L) equation is

$$L_{3D} = \left(\frac{M}{w} - 1 + D\right) * T + D * T_n \quad (4.39)$$

where M is the message size, w is the channel width, D is the Manhattan distance, T represents the propagation delay of one bit in one unit length which is equal to 62.5ps (per 1 cm), and T_n is the node processing (switching) time. For the best case, $D=c$ and for the worst case, $D=4*c$. The header latency is larger than the rest of the message ($\frac{M}{w} - 1$) since it sets the nodes direction in time T_n . The rest of the message just requires to propagate through the channels in T time per unit length.

Throughput is defined as the rate in which the packets are exiting the bus for a certain message size per second. The resulting throughput (T_p) equation is

$$Tp_{3D} = \frac{M}{\left[T * \left(\frac{M}{w} - 1\right) + D\right] + D * T_n} \quad (4.40)$$

4.9.1 3D-bus vs. shared-bus

We use equation 4.39 to calculate the latency. In addition, we are interested in determining the optimal number of cubes that are needed in order to outperform the shared-bus.

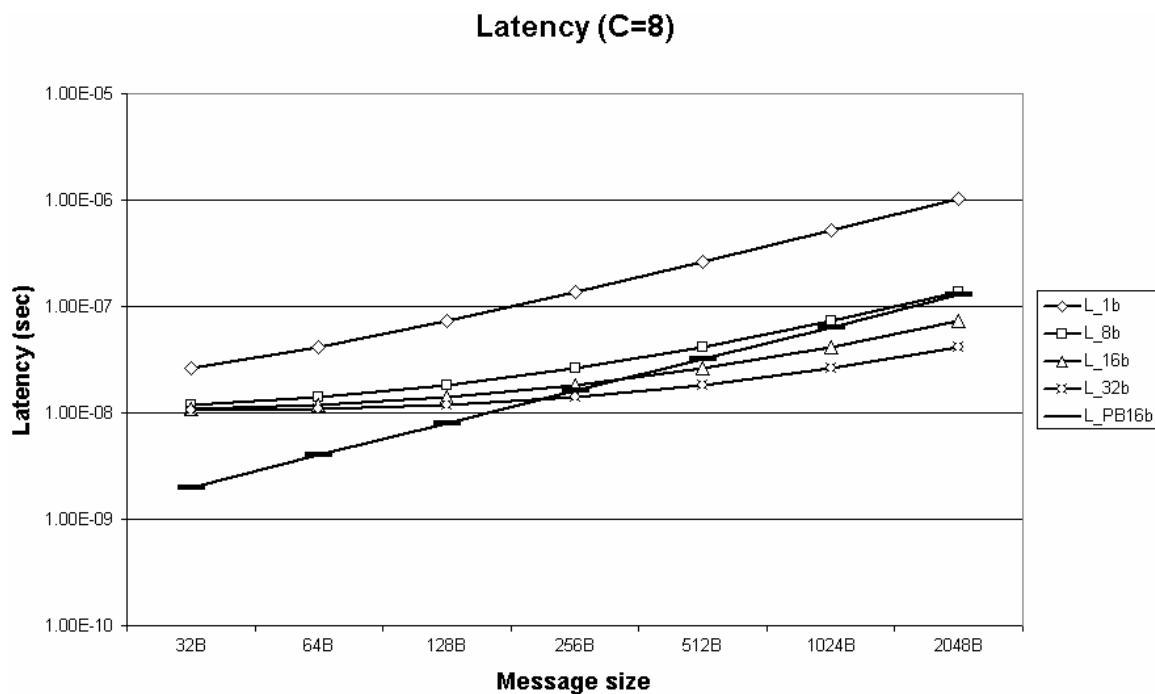


Figure 4.34: 3D-bus latency vs. shared-bus

Fig. 4.34 depicts the latency for different message lengths. Both the shared bus and the 3D-bus have the same length ($C=8$). Fig. 4.34 portrays that latency increases with message size. For messages larger than 256 B, 3D-bus has lower latency compared to shared bus with the same channel width. Moreover, for large message size the latency of 3D-bus with channel width of 16 and 32 bits is lower than shared bus.

4.10 Memory bandwidth

Current DRAM speeds are not enough to handle the requirements associated with equivalent access frequencies of 5 GHz per node (this rate is based on maximum bit transfer per line of 100 psec plus one switching delay 100 psec). SRAM arrays are expensive and have high power consumption but their current latencies make it a good candidate if a buffering scheme is used. SRAM double-buffer, for example, could receive

data at full speed when writing to memory. This double buffer could then be interfaced with a large DRAM array for mass storage. As DRAM speeds increase, this further reduces the stress and demands on the buffering system [33].

Memory modules employed in our interconnect need to be compatible with the interconnect throughput rate. Memory devices such as QDR SRAM that can reach 10.22 Gbps [34], DDR2 SDRAM, which taps 4.15 Gbps [26] (1.0375 Gbps per bank), or RLDRAM-II with data rate of 28.8 Gbps [65] (3.6 Gbps per bank) are good candidates for our interconnect. There are new memory technologies that exceed fast memories and could be utilized as well if multiple memories and buffering scheme are used. Providing that cost is not the major factor in a cutting-edge, high-speed design, it seems very likely that these technologies could be employed to create SRAM capable of operating at 5 GHz. SRAM technologies such as pure silicon SRAM was demonstrated at speeds exceeding 4 GHz. Other technologies, such as MRAM, GaAs, SiGe, and InP could provide speeds far exceeding those of current Silicon SRAM [68].

4.11 Area analysis

Area constraint is a crucial factor in selecting an interconnect design that can provide high-performance and at the same time fits into the physical dimensions available on the line card printed circuit board (PCB). The interconnect must compromise its channel width size, number of nodes and additional hardware elements (buffers, switches etc.) in order to meet space limitations. Current multi-layer PCB technology allows the

implementation of higher dimensional networks. The area consumed by the layout of a k -ary n -cube network can be expressed [96]:

$$\frac{16N^2}{(Y^2 - 1)k^2} + O\left(\frac{N^2}{Y^2k^2}\right) \quad (4.41)$$

where, N represents the number of nodes forming the network and Y signifies the number of layers. Assuming that the implementation is done using dual layer, for the example 8-ary 2-cube, 4-ary 3-cube and 3D-mesh, each having 64 nodes we get:

$$Area_{4\text{-ary},3\text{-cube}} \cong \frac{16 * 64^2}{(2^2 - 1) * 4^2} \cong 1365$$

$$Area_{8\text{-ary},2\text{-cube}} \cong \frac{16 * 64^2}{(2^2 - 1) * 8^2} \cong 341$$

$$Area_{3D\text{-mesh}} \cong \frac{16 * 8^2}{(2^2 - 1) * 2^2} * 8 + \sqrt[3]{8} * 7 \cong 697$$

Note that these values do not show exact occupied areas in real units. They are used to compare one topology to another one. The 3D-mesh area equation uses 2-ary 3-cube network parameters, then it is multiplied by 8, since there are 8 cubes in 16 faces (2 faces comprise 1 cube) that are connected to each other through physical channels. Since 4 channels connecting one cube to another, it requires 7 groups of 4 channels each to connect those cubes. The term $\sqrt[3]{N}$ represents the additional wire length connecting one cube to another with respect to the number of nodes in the network [97].

The calculations above show that if implemented on the PCB 3D-mesh will consume more area than 8-ary 2-cube but less than the 4-ary 3-cube. Although area is a primary concern but it is not the only design parameter. In fact, as long as the

performance figures are comparable, the 3D-mesh might be chosen because of its easy scalable structure. That is, inserting more nodes in the example 3D-mesh is only a multiple of 4 nodes to any side of the structure, while the other interconnects require symmetry in the number of PE or memories that are added. Second, the 3-dimensions of the interconnect provides higher routing flexibility, a characteristic of high-dimensional networks, while consumes less space (as shown with respect to the 4-ary 3-cube). Thus, it makes sense to explore variations of k -ary n -cube architectures along with the standard ones.

CHAPTER 5: EXPERIMENTAL RESULTS

5.1 K-ary n-cube simulation framework

We developed an event-driven, custom-designed interconnect simulation environment to evaluate the performance of packet-based off-chip k -ary n -cube interconnect architectures for line cards. The simulator uses the state-of-the-art software design techniques to provide the user with a flexible yet robust tool, that can emulate multiple interconnect architectures under non-uniform traffic patterns. Moreover, the simulator offers the user with full control over network parameters, performance enhancing features and simulation time frames that make the platform as identical as possible to the real line card physical and functional properties. The objective of this simulator is to compare among different off-chip interconnect architectures for network line cards and determine which interconnect can significantly increase the memory bandwidth and the overall system throughput.

Our simulation model includes statistical data such as IP length distribution and physical measures of PCB placement and spacing, as well as network properties such as IP packet size, in order to increase the accuracy of our calculations. In addition, we apply true IP network properties such as switching, propagation and routing latencies. The simulator provides real time performance analysis with detailed metrics on packets processed at each simulation cycle and overall detailed results at the end of each simulation. The user input parameters using an easy to use GUI or command line interface.

5.1.1 K-ary n-cube interconnect simulator architecture

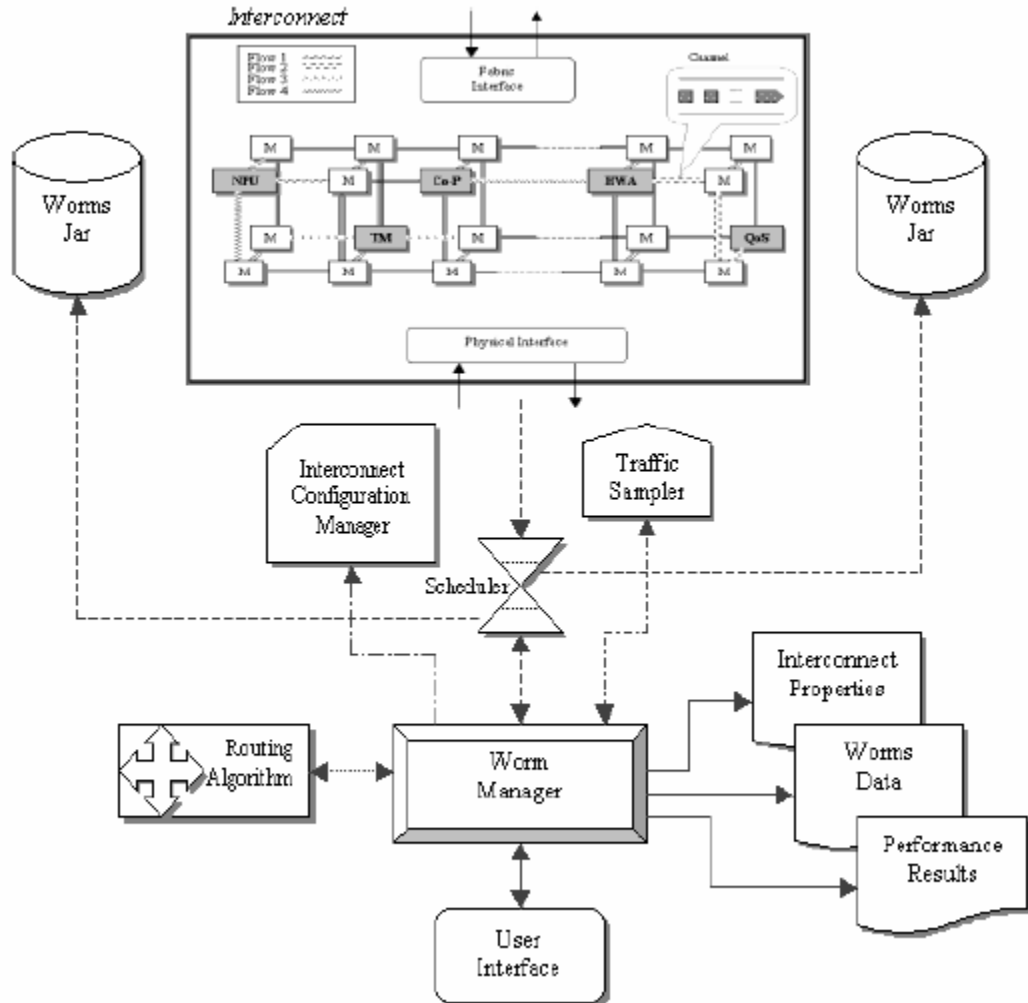


Figure 5.1: Simulation control modules

The simulator architecture, shown in Fig. 5.1, depicts the interconnect interaction with the control modules which adjust, collect and modify the interconnect settings, dataflow, and performance metrics. The simulator configuration manager sets the interconnect type, its properties (wire propagation delay, switching delay or routing delay) and enable/disable enhanced features such as channel width, VC on/off, bidirectional channel. The interconnect properties are set by the user interface and

recorded to allow the configuration manager to be updated via the worm manager. The worm manager utilizes interconnect properties and configuration parameters in order to set accordingly, other modules in the system that participate in the simulation. The traffic sampler continuously records performance data such as throughput, latency, routing accuracy, interconnect bandwidth utilization and interconnect resource utilization. This information is feedback to the worm manager which accordingly adjusts worm generation rate and load balances the traffic. The routing algorithm receives each individual worm location and its destination node from the worm manager. Then, it determines for each worm the shortest route possible by avoiding spots of heavy traffic.

The worm jar is a storage module that contains all the generated worms waiting to enter the interconnect. The total number of worms during simulation are initially determined by the user. The simulator generates worms randomly, but since not all worms can immediately enter the interconnect (when the interconnect reaches its full capacity) they are being stored in the worm jar. On the other side, there is another worm jar which contains all the worms that reached their destination. The scheduler is responsible to inject worms into the interconnect taking into account the total network capacity and traffic load. Since the worm manager knows the total number of worms that are modeled throughout the simulation, it must inform the scheduler the end of the simulation (no more worms to model).

The simulator accounts for all practical parameters characterizing off-chip interconnect architectures such as, switching delays (T_s), routing delays (T_r) and propagation delays (T_w) as well as the complete functionality of each system components (nodes, links, PE/Memory, interfaces, virtual channels, and channel partitioning). The

user has the option to change each one of these parameters in case new technology introduces higher standards. Simulation time is based on a unit cycle which is equal to one clock cycle (T_w+T_r). All other delays are calculated as multiples of it. That provides the advantage of having single uniform simulation clock.

Message size in bytes and message generation-time are obtained using pseudo-random number generator, which is utilized to resemble the randomness of packet transmission by both processors and memories. Each worm is linked to performance-bookkeeping function, which records its latency, throughput, simulation cycles, failures, and route-taken from the moment the worm enters the interconnect until it completely reaches its destination. A comprehensive performance results are provided at the end of each simulation.

5.1.2 Simulator modeling approach

There are three important system components which are closely coupled, interact and affect each other (Fig. 5.2). It includes, the user interaction with the system through the user interface and its relations with the worm manager, the interconnect and the worm classes. The user creates the interconnect type and worms, determines the worms' routes and sizes, and instructs the worm manager when to model the simulation. The interconnect class is the main physical structure that the worm is using in order to propagate from source node to its destination. The interconnect properties may change, which affect the worm routing flexibility and resources it can use (such as virtual channels) to avoid deadlock and thus, a retransmission.

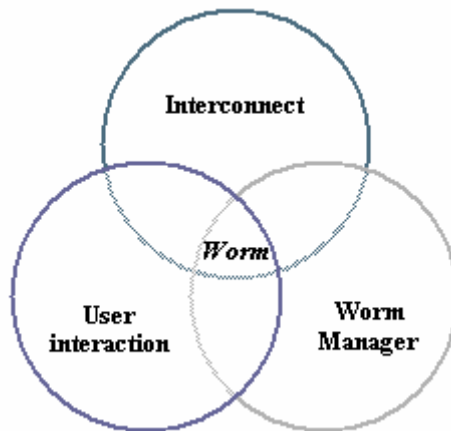


Figure 5.2: Class relationship diagram

The worm manager class records worm data, determines arrival and departure of worms using the worm generation algorithm to load balance the number of worms processed simultaneously within the interconnect. In the center, the worm class routes itself through the interconnect and flags the worm manager when it is completely modeled or if has entered deadlock/livelock and needs to be retransmitted. The routing algorithm, used by the worm, adaptively enables the worm to find its best path possible to its destination as network traffic increases and the interconnect resources become occupied.

The cyclic diagram (Fig. 5.3) depicts the relationship between the interconnect architecture, the simulator and the physical properties of resources available. The interconnect architecture represents the physical structure and includes all the hardware required to implement it. The properties represent two types of parameters: physical parameters of electrical components comprising the interconnect (such as wire delays, switching delays, routing delays), and parameters of additional features, which enhance the interconnect performance (for example, channel partitioning, virtual channels,

interconnect configuration). The simulator emulates the interconnect functionality in order to evaluate and compare different configurations and settings.

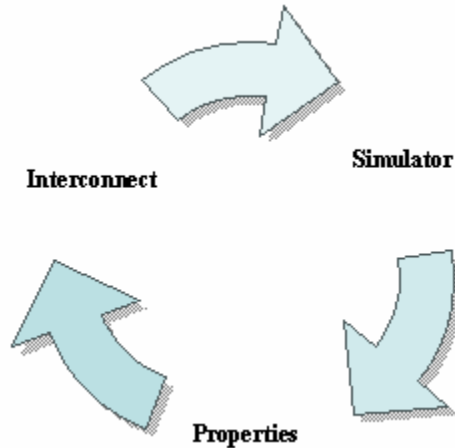


Figure 5.3: Cyclical relationship

The simulation setup (Fig. 5.4) is an abstract view of high level system components and their interaction in order to initialize and execute simulation. First, the simulation properties are set for the simulation.

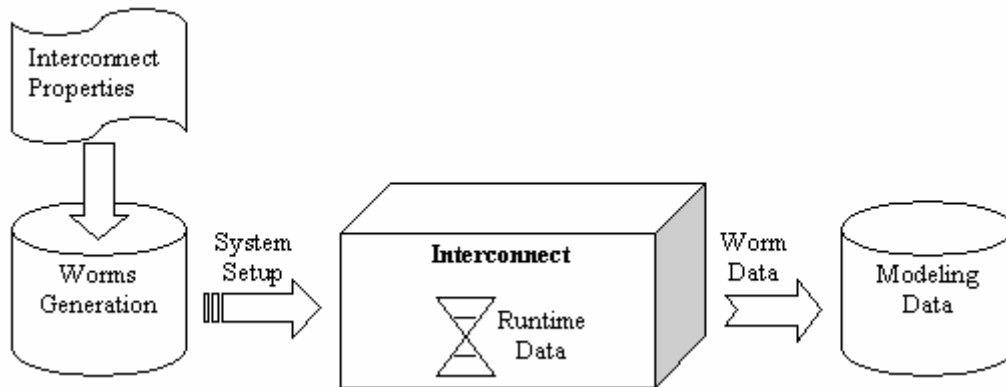


Figure 5.4: Simulation setup

The properties are set by the user and are crucial for worm generation, timing delays, and other simulation aspects. Then, the messages (worms) are created and are placed in a data structure (jar). Since the interconnect can change configurations, PEs and

memories change their location accordingly, and therefore, source/destination addresses must be correctly set before the worms can be generated. When the user chooses to run the simulation, the properties and the data of the worms in the jar are recorded in separate files. The interconnect receives worms from the jar of generated worms according to a probability called worm generation rate (GR) which is controlled by the user. In addition, the user can determine how many worms can occupy the interconnect at all simulation cycles by changing the value of the max worms in interconnect variable (MWII). If no value is set for this variable, the default value is unlimited number of worms. The worms that enter the interconnect are modeled until they reach their destination. All runtime worm data is collected in a separate output file which provides individual details about each worm. After the complete simulation is modeled, a file is generated recording the performance of the simulation.

Fig. 5.5 shows a UML class diagram of the interconnect architecture. A single type of interconnect is a set of faces which each comprise of multiple nodes. Within each node there are multiple ports. A node can be modeled as either a memory or a PE. Hence, the node still possess the same structure and functionality as any node, but it reserves one port as an I/O port to the device. Interconnect properties affect worm routing flexibility and resources it can use, while propagating through the interconnect, such as virtual channels (VC) and/or sub-channeling (SC). The port class contains VCs and SCs which are modeled as logical topologies on top of the physical network architecture. VCs as well as SCs have a great effect on the worms transmission success/failure rates and deadlock/livelock avoidance. Although VCs improve routing accuracy and reduce worm

transmission failure rate, it also increases worm latency and interconnect implementation costs.

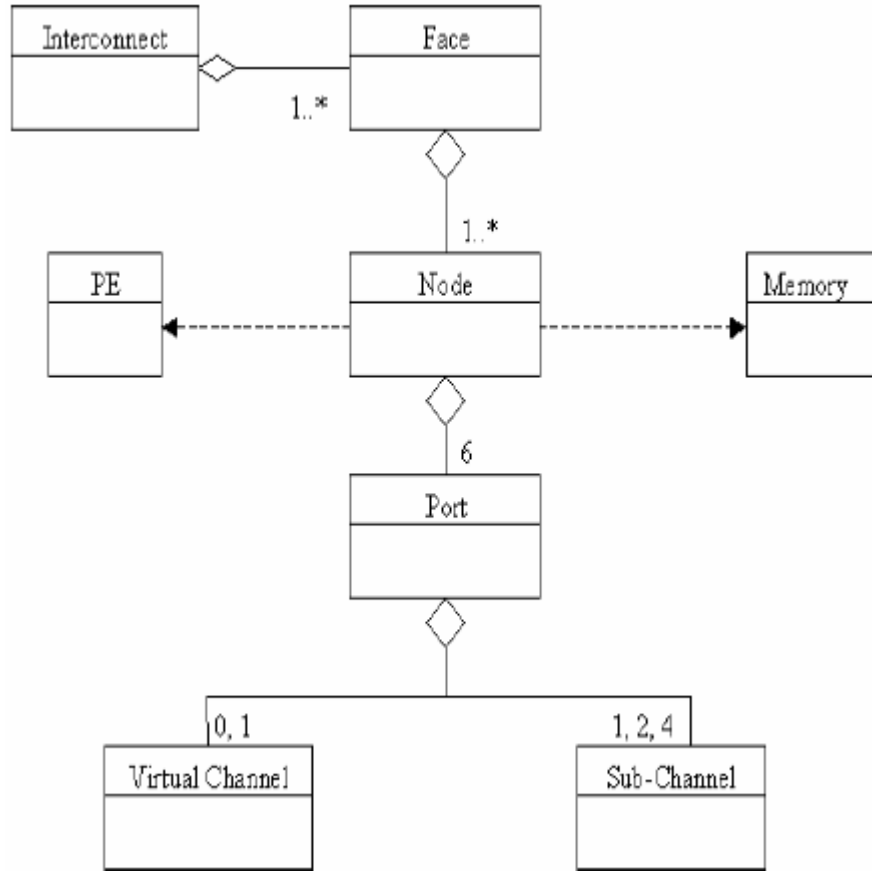


Figure 5.5: UML class diagram of interconnect architecture

The worm manager class records worm data, arrival and departure time stamps of worms, and controls the worm generation rate in order to load balance the number of worms processed simultaneously within the interconnect. The worm class encapsulates the properties of a worm such as a header with source/destination fields and shortest path coordinates. The worm class shares data and collaborates with all other classes. It routes itself through the interconnect, while continuously being monitored by the worm manager. The adaptive routing algorithm is used by the worm to determine the best

available path that it can take to reach destination. The worm updates its shortest path coordinates with each movement to ensure its optimal path even when it is required to take a detour as a result of hot-spot node.

5.1.3 Software components

Fig. 5.6 portrays a dynamic model (action oriented) of the routing algorithm class and its subclasses with interconnect system components and the worm manager class. This model determines the actions performed by the routing algorithm in order to maneuver each worm within the interconnect with respect to its current position, its destination and traffic conditions [56]. The routing algorithm is closely coupled with the worm manager since the worm manager controls worms entering and leaving the interconnect while the routing algorithm controls the worms within the interconnect. First, the routing algorithm analyzes the source node type (where the worm is generated) and the enabled interconnect features such as virtual channels, bidirectional channels and PE-M configuration. Then, it checks the preferred (shortest path) direction in which the worm needs to move. The routing algorithm scans each node's port and dictates the movement of the worm giving priority to ports which are pointing in direction towards its destination. If none of the ports are available, the routing algorithm will check the availability of virtual channels. If enabled, the worm is queued into one of the virtual channels until one of the ports clears. If virtual channels are not available then the routing algorithm notifies the worm manager of a worm failure. This will result in a

retransmission of the same worm to to/from the same source/destination in order to lower any data lose.

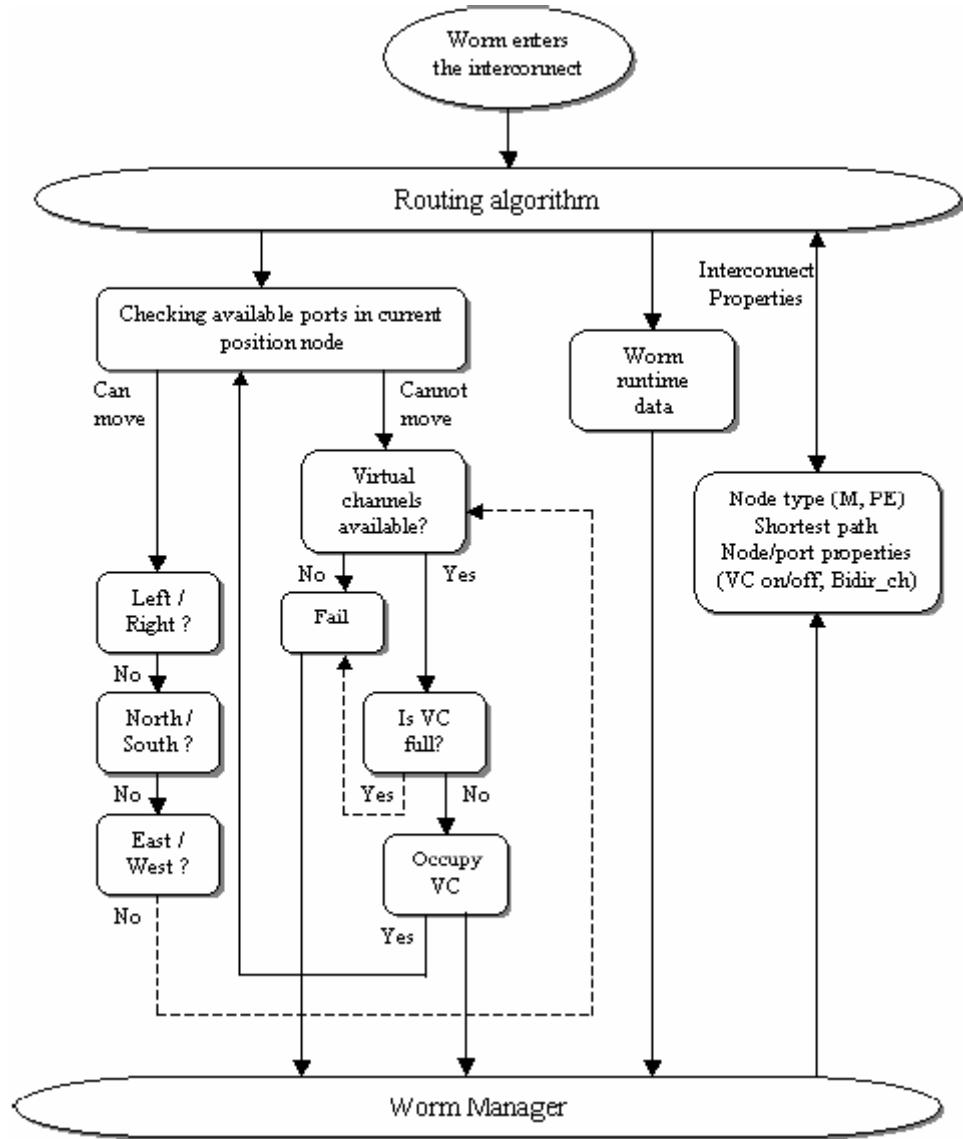


Figure 5.6: Dynamic model of routing algorithm

Fig. 5.7 depicts a sequence diagram of the user interaction with the simulator classes/objects. The sequence chart is used to determine which data to automate in the simulator and which must be input exclusively by the user [56][58]. User input data is used/shared by two other classes: the file class and the worm manager class. The user has

two choices, using default settings or change settings/properties in order to simulate the interconnect with different configuration.

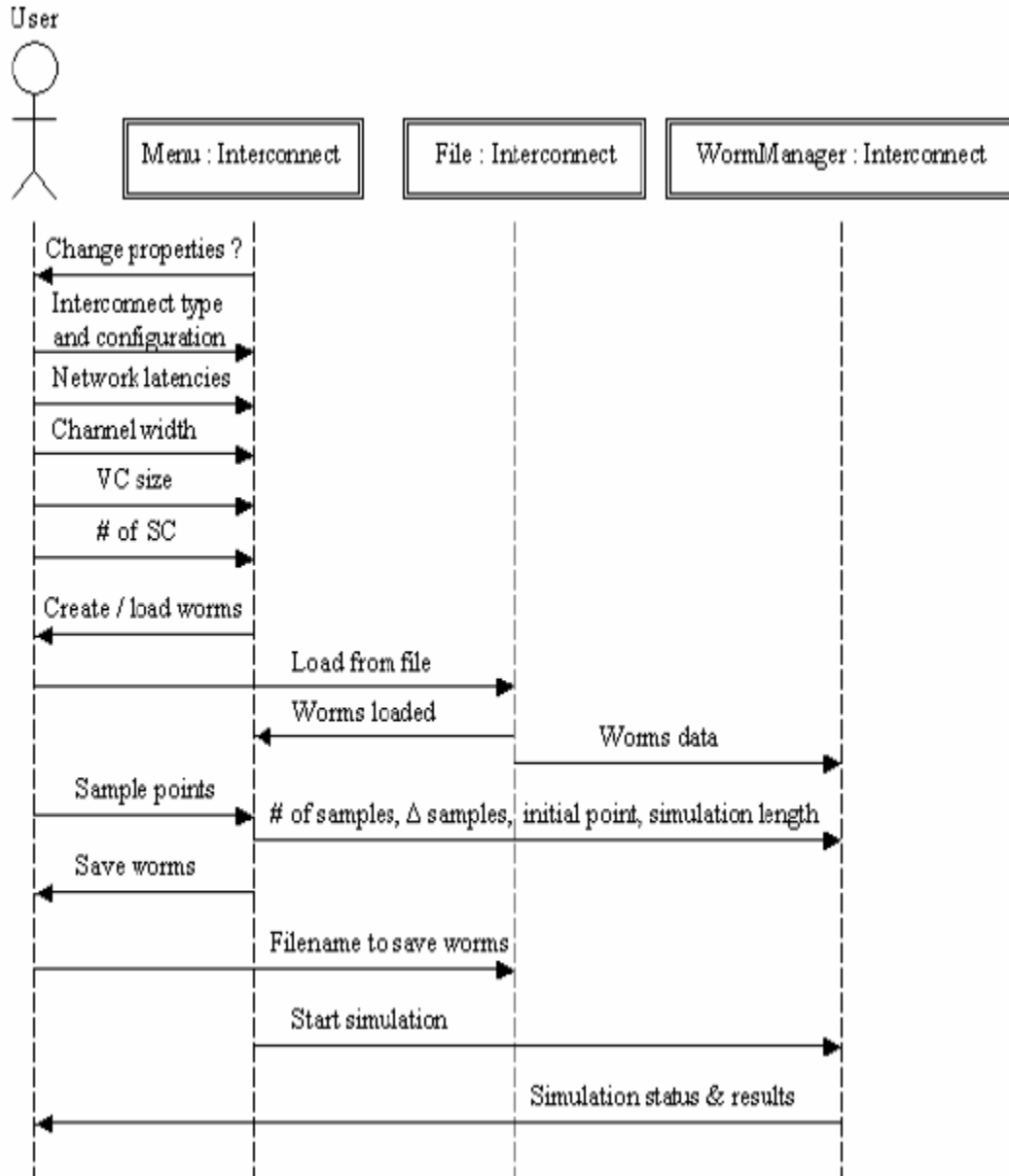


Figure 5.7: Sequence diagram of user-simulator interaction

Once the interconnect type and configuration are defined the user must complete the following steps before simulation executes: select if new worms will be generated or worms should be restored from an existing file, determine the number of worms to

simulate, decide if worms are generated randomly or manually, input the number of sampled throughput points (include the initial sampling point and the number of simulation cycles between samples) and select if the newly generated worms will be saved or not.

5.1.4 Optimization strategies

5.1.4.1 The singleton class

The singleton classes [60], such as *WormManager* and *Interconnect*, guarantee that only one class instantiation is created (Figs. 5.8-5.9). The single instance is held as a static variable as a private member of the class. These singleton classes are not automatically initialized. Instead, initialization occurs the first time that singleton class' *create()* method is called by the client. The *create()* method also allows the callers to access methods of that singleton class. In the same manner it can destroy the object. The *interconnect* is a singleton class: only one *interconnect* is created per simulation. The *WormManager* creates a new *interconnect* for each simulation and destroys it when done. The reason for this is that there might be different configurations which require construction of the object in different ways. The following figures show all the objects and functions (public and private) included in each of these singleton classes.

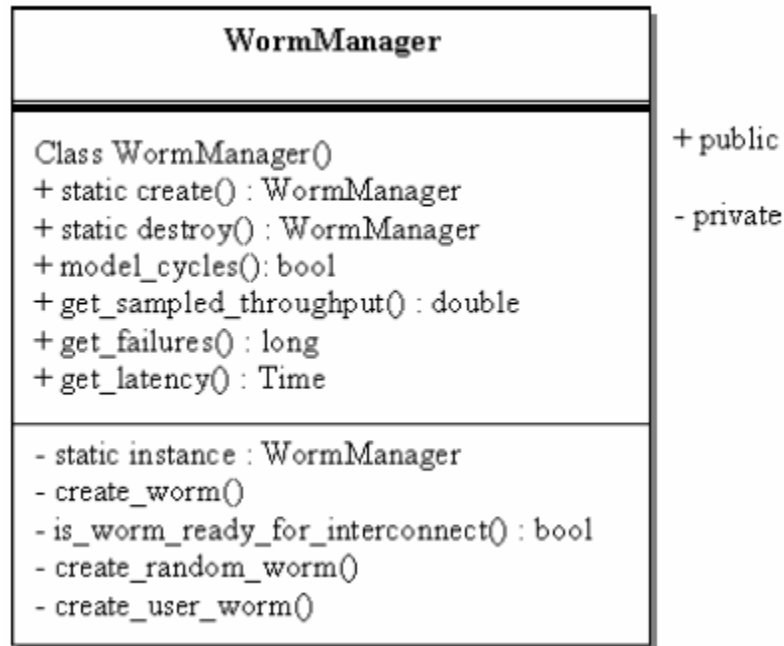


Figure 5.8: *WormManager* singleton

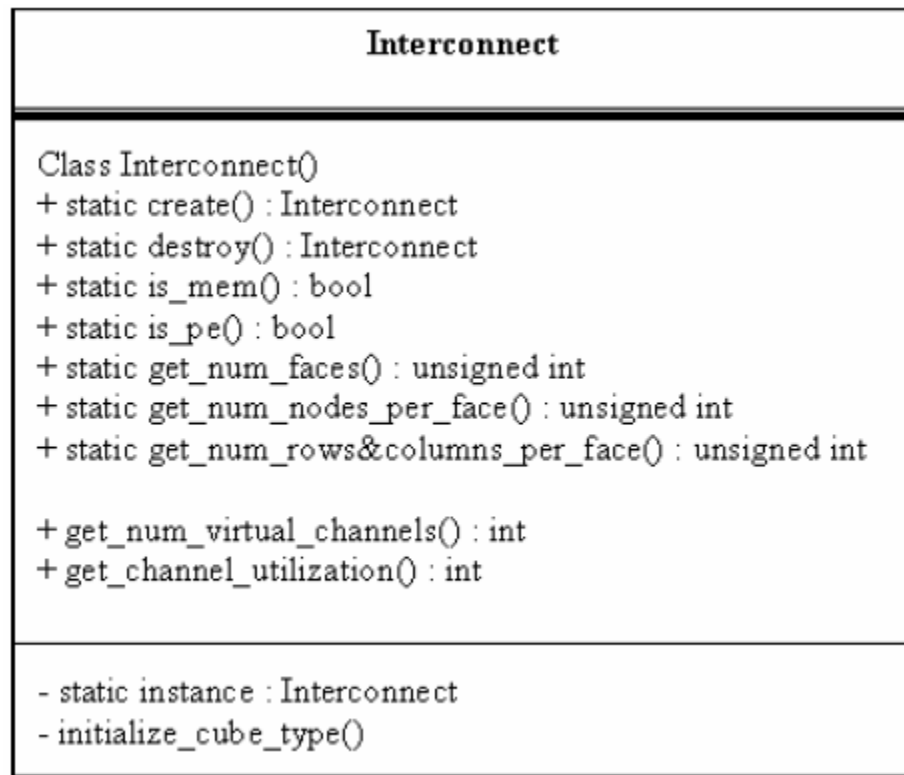


Figure 5.9: *Interconnect* singleton

5.1.4.2 Pure virtual functions

The *SaveRestoreInterface* class provides save and restore functions which are pure virtual functions which forces derived classes to override [61]. The following is an example of pure virtual function signatures:

```
class SaveRestoreInterface{  
  
    public:  
  
    virtual File &save(File&file)=0;  
  
    virtual File &restore(File &file)=0;
```

In the save/restore functionality, we utilize a sentinel as a safeguard to assure that the correct version of code is used. The sentinel is recorded in the save file. Upon restore, it is verified that the saved file matches the current software version.

```
const int save_restore_sentinel = 3;  
  
File &WormManager::save(File &file){  
  
    file.save(save_restore_sentinel);  
  
    Properties::save(file);  
  
    return file.save(worms_to_process);}  
  
File &WormManager::restore(File &file){  
  
    int tmp_save_restore_sentinel;  
  
    file.restore (tmp_save_restore_sentinel);  
  
    if (tmp_save_restore_sentinel !=save_restore_sentinel){  
  
        std::cerr<<"cannot restore file<<  
  
        tmp_save_restore_sentinel<<std::endl;
```

```
restored = false;  
else{  
  
Properties::restore(file);  
  
Interconnect::create();  
  
file.restore(worms_to_process);  
  
restored = true;  
  
return file;}
```

5.1.4.3 System design with Standard Template Library (STL) functions

The STL is a general purpose library of algorithms and data structures. The STL enables generic programming where reusable functions, data structures and algorithms are available for the programmer [59]. The ability to achieve composition was attained by using the Standard Template Library (STL) map class, as shown in the following code [59]. The interconnect is basically constructed of three main components: a face, node, and port. The interconnect has a map of faces, which has a map of nodes, etc. By creating this type of relationship, the location of worms in the interconnect is accessed as if the interconnect and its composed parts are arrays. The same effect could have been traded off with the STL vector class; the map class offers safer memory management though degrades the performance.

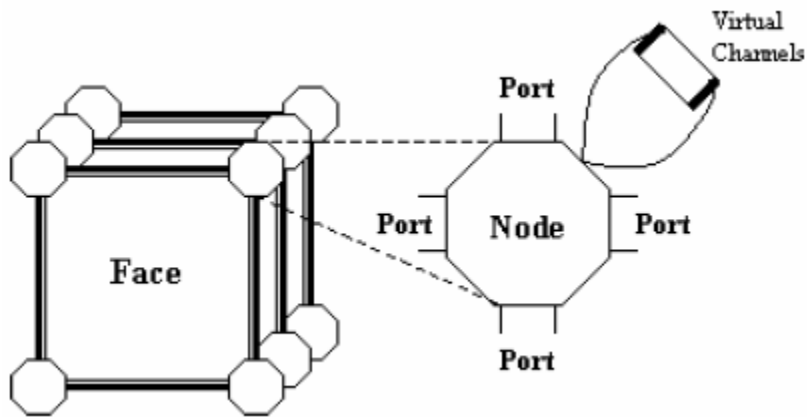


Figure 5.10: Structural hierarchy of classes: face, node, port

For the 3D-mesh interconnect, each face has four nodes at the corners. Each node has 4 ports. Therefore, a map is created for each component to organize the connectivity and construct the interconnect structure (shown in Fig. 5.10). Each component has an ID and an integer variable which represents the number of instances available [57].

Face #ID to face map:

```
typedef std::map <int, Face> FaceMap;
```

Node #ID to node map:

```
typedef std::map <int, Node> NodeMap;
```

Port #ID to port map:

```
typedef std::map <int, Port> PortMap;
```

VC #ID to VC map:

```
typedef std::map <int, VirtualChannel> MemoryManager;
```

5.1.5 Simulator parameters

The simulator generates worms at a certain pace and monitor their performance from the moment they enter the interconnect. Worm generation is depended on a random generator, which randomly generates worms with variable size and source/destination addresses, and worm manager, which monitors the number of worms entering the interconnect and the number of worms currently in the interconnect. First the random generator determines the maximum number of worms that can be generated by checking the interconnect type. Each interconnect type has a different number of PE/memory modules.

The maximum number is the total number of message generating elements within the interconnect, thus PEs + memories. The random generator generates worms without considering the status of worm currently propagating through the interconnect. It places all the worms it generates, arbitrarily within each cycle, in a Jar. Jar is a class type (C++) which cooperates with the worm manager to synchronize the placement of worms from the Jar into the interconnect. The rate in which worms are placed in the interconnect is determined by the worm manager and not the random generator since the worm manager closely monitors the status of all processed worms. The random generator can be adjusted so that the number of worms generated will increase or decrease. This variable is called Generation Rate (GR) and it is represented as a percentage rate. For example, 90% generation rate, means that there is a 90% chance that some worms will enter the interconnect within the next cycle. The worm manager, on the other hand, contains a variable called Worms In the Interconnect (WII) which controls the maximum number of

worms that can be transferred in the interconnect at any simulation cycle. For example, if $WII = 50\%$ then only 50% of the maximum number of worms possible (PEs+memories) will be occupying the interconnect at any simulation cycle. Both parameters, GR and WII, have a major effect on simulation performance. High value of GR and WII can overflow the interconnect with worms, thus significantly reducing throughput/goodput, increasing latency and creating many deadlocks. On the other hand, if GR and WII have a very low value the utilization rate of the interconnect bandwidth and resources will be low, since only few worms will be generated and since the interconnect occupancy is low, then they will quickly get to their destination. The disadvantage of low GR and WII is that it reduced performance since only few worms are being generated and transferred. As a result, there must be an equilibrium point between GR and WII that will result in peak performance without over-utilizing or under-utilizing the interconnect.

Another perspective is the worm failure rate. We want to keep failure rate as low as possible, since retransmission of worms requires more bookkeeping and consumes more resources. High failure rate is bad and indicates that worms are able to reach their destination with a very low success rate. One of the performance measures during simulation, using variable GR and WII, is to find the best value for both GR and WII to keep high throughput, low latency and reduce failure rate to almost 0. The following simulation results and figures depict this relationship.

Performance comparison with variable Generation Rates

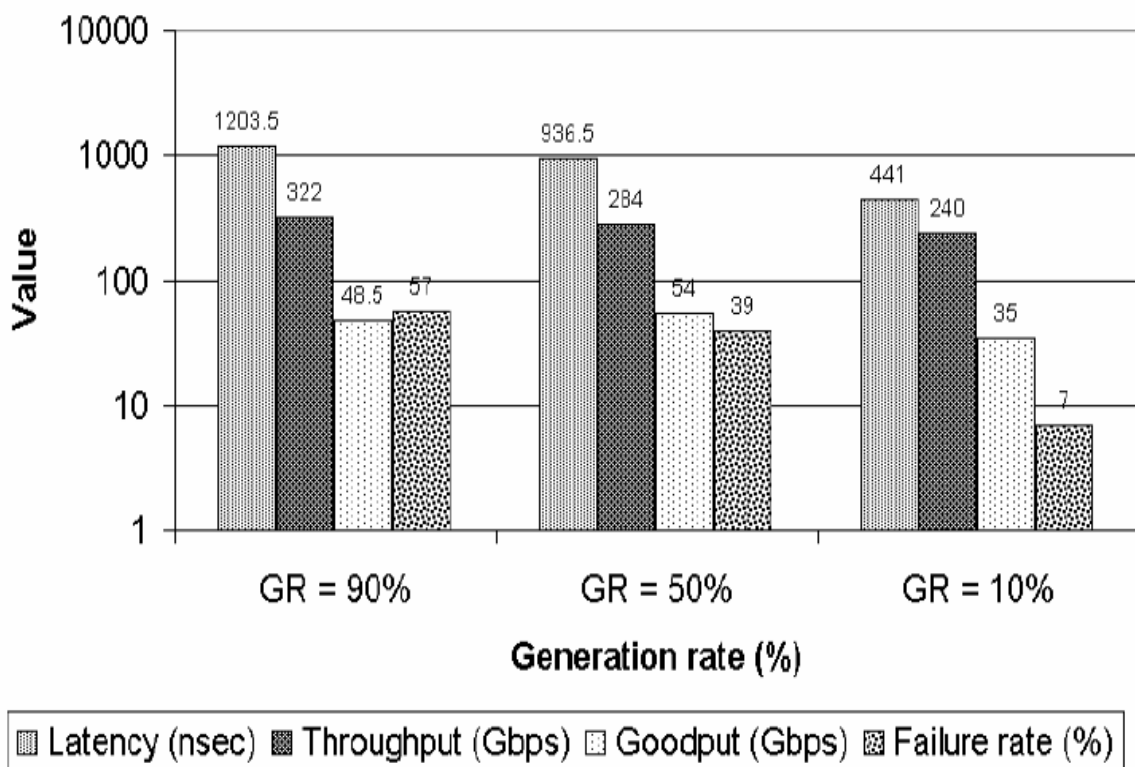


Figure 5.11: Generation rate

Figure 5.11 shows performance of the 3D-mesh interconnect (configuration 3) as generation rate decreases. Intuitively as GR decreases, both throughput and latency will decrease as well. The goodput is kept steady since the goodput is an average measure of bits processed throughout the simulation. GR has a major effect on failure rate.

As GR decreases, the failure rate decreases significantly since less worms occupy the interconnect and now it is easier for worms to route their way, in the shortest path, to their destination. The variable worms in the interconnect has a lesser influence on performance results.

Performance comparison with limit on number of worms in the interconnect

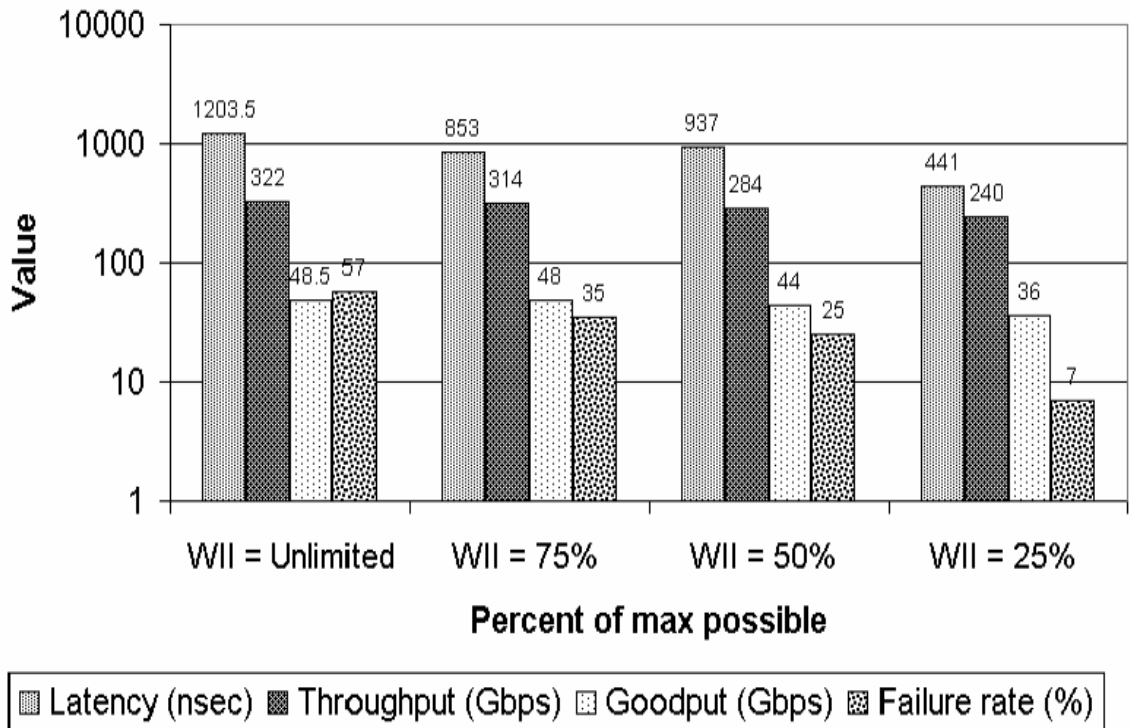


Figure 5.12: Maximum worms in the interconnect (MWII)

Fig. 5.12 portrays that although throughput and latency were reduced as the maximum WII was decreasing, the effect was not that significant as GR had. Only failure rate was reduced significantly from the same reason above.

Fig. 5.13 depicts the tradeoff between GR and max WII. In general, as GR and WII rate was reduced, the failure rate was reduced to 0. We see that the advantage of adjusting both the GR and WII is that we can reduce failure rate to almost 0, while keeping high throughput and low latency in tact.

Performance comparison with variable GR and WII

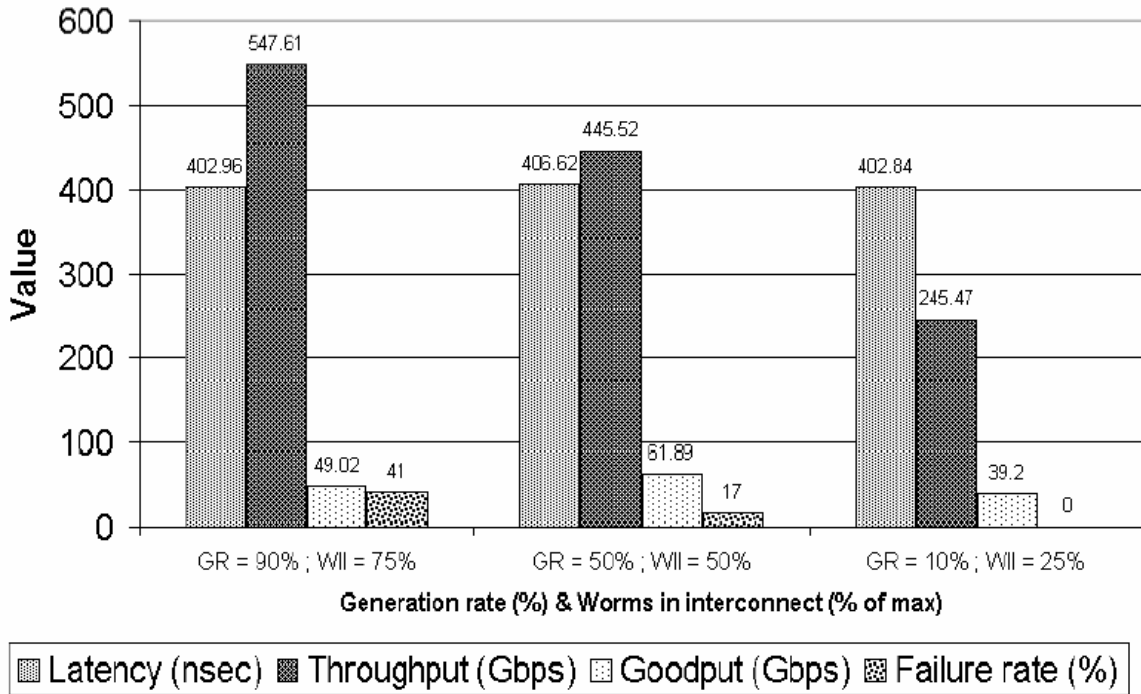


Figure 5.13: Generation rate and maximum worms in the interconnect

5.2 3D-mesh, 4-ary 3-cube & 8-ary 2-cube simulation results

5.2.1 Latency and throughput analysis

Latency represents the time it takes for a worm to reach destination. Depending on the worm movement, latency sums wire transfer, switching and routing delays at each cycle. The resulting latency is an average of latencies collected from all worms generated, at the end of the simulation. We chose three representative k -ary n -cube interconnects for our simulations: 8-ary 2-cube, 4-ary 3-cube and 3D-mesh (all three interconnects have 64 nodes).

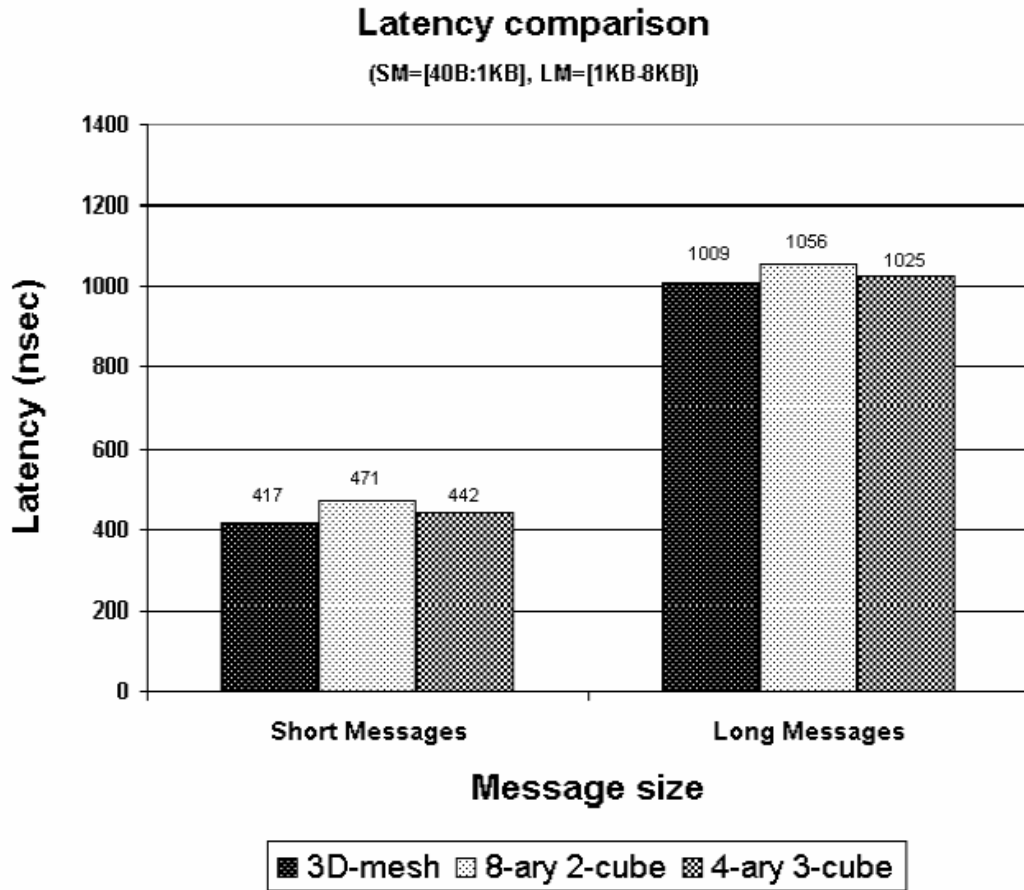


Figure 5.14: Latency comparison

Fig. 5.14 shows a comparison among all three interconnects with VC and channel partitioning enabled. The results shown are an average of 10 different simulations with both short (128B-1KB) and long (1KB-8KB) worms and identical interconnect settings. The lowest latency was recorded for the 3D-mesh, while the 4-ary 3-cube network has slightly higher latency than the 3D-mesh.

Fig. 5.15 portrays the latency of each interconnect with respect to the offered load. Offered load determines the probability that each node comprising the interconnect will generate a message within each simulation cycle.

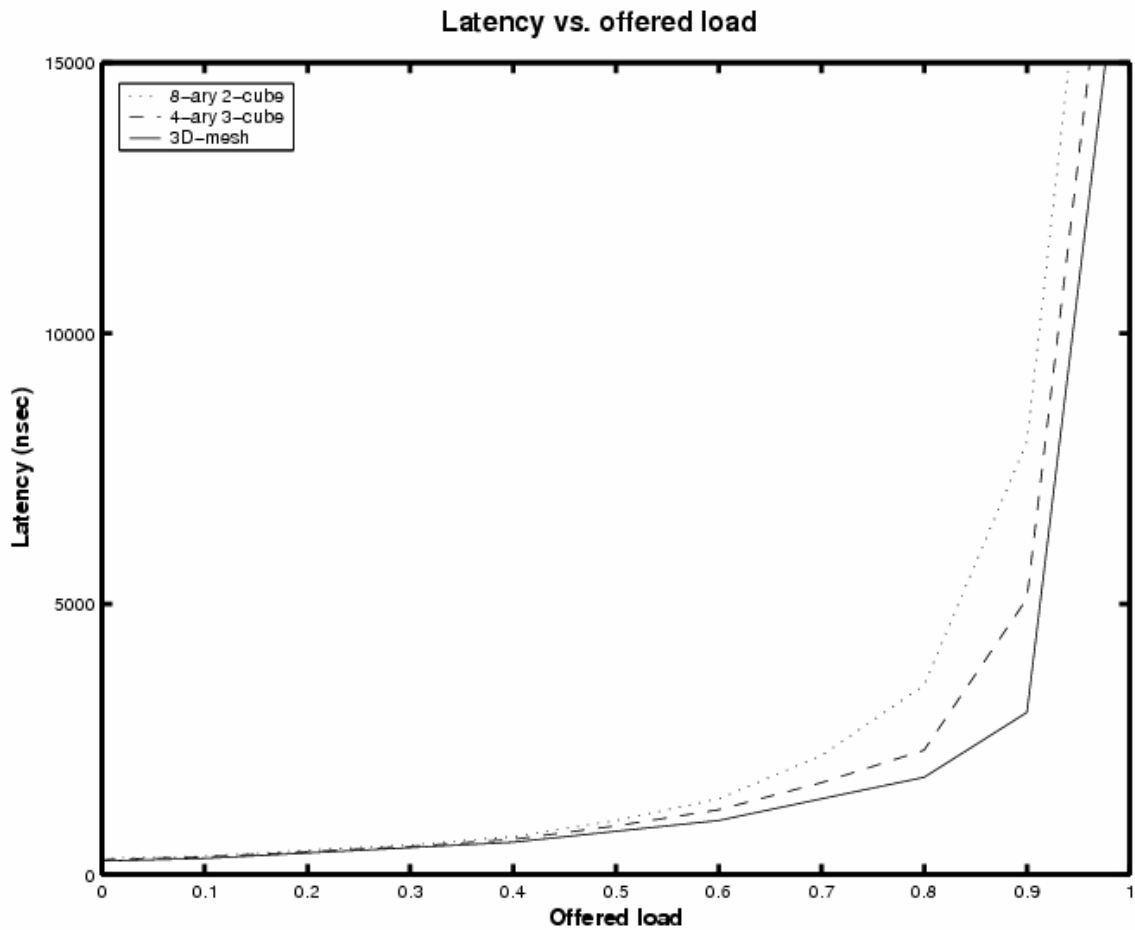


Figure 5.15: Latency vs. offered load

For example, if the offered load is set to 0.1 there is a chance that 10% of the total nodes in the interconnect will generate a message at each simulation cycle. Fig. 5.15 shows that as offered load increases the latency increases exponentially for all interconnects. 3D-mesh interconnect is able to sustain the highest offered load out of the three interconnects.

Throughput is measured by taking samples of the total bits processed within the interconnect at each cycle. Throughput significantly increases when VC are enabled since VC allow more worms to occupy the interconnect without transmission failures. Fig. 5.16

shows that the highest throughput was reached by the 3D-mesh interconnect for both short and long messages.

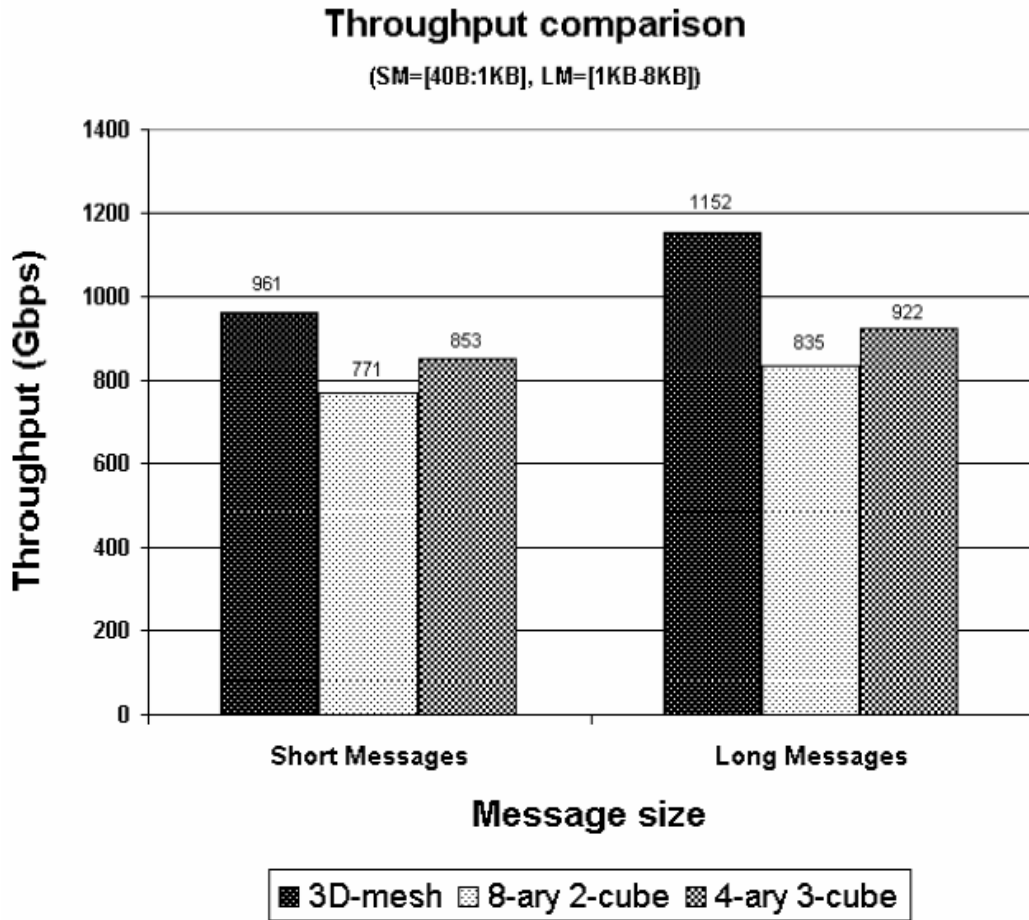


Figure 5.16: Throughput comparison

5.2.2 Worm allocation and distribution

Worm allocation and distribution, depict in Fig. 5.17, shows three groups of worms: worms currently propagating in the interconnect, worms waiting in jar to be modeled and worms that finished and reached destination.

Distribution of worms modeled during simulation

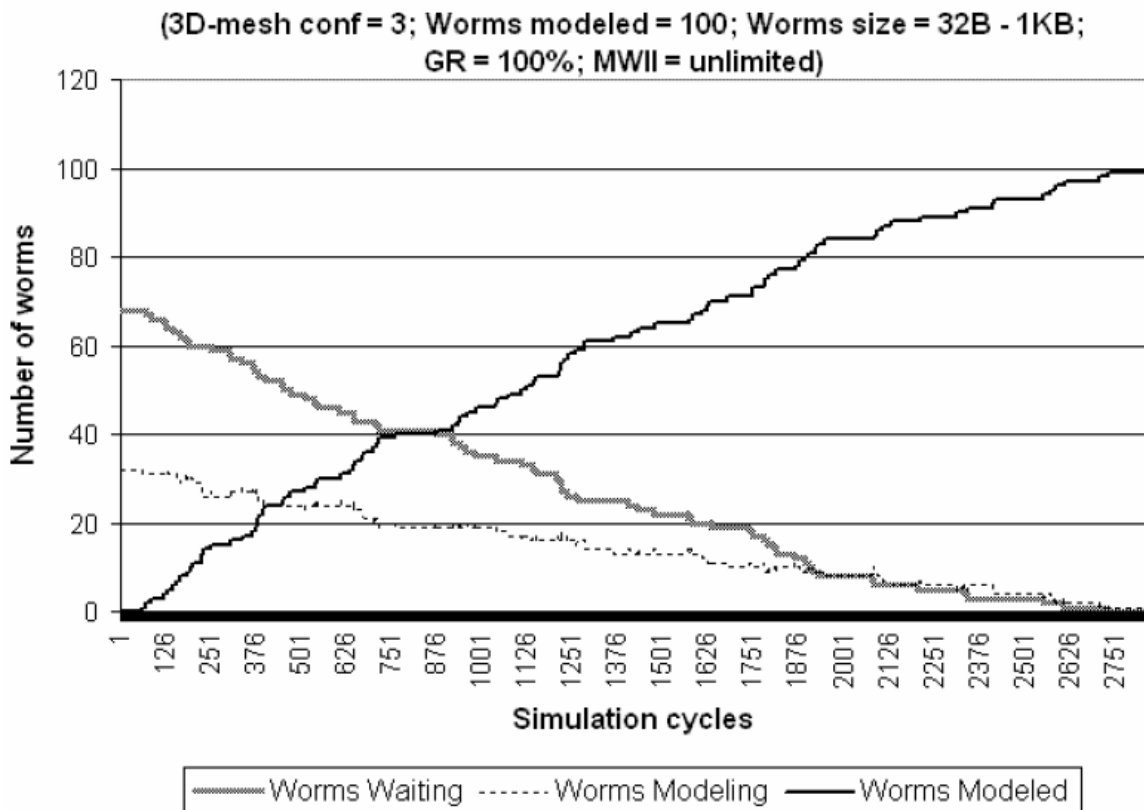


Figure 5.17: Worm allocation & distribution

Fig. 5.17 provides a good indication of the worm manager functionality. The figure shows that the number of currently modeled worms (worms in the interconnect) increases as the number of worms waiting in the jar and the number of already modeled worms (finished) decreases. When VC are enabled more worms occupy the interconnect and with faster rate than presented in Fig. 5.17. Fig. 5.18 depicts that since more worms are modeled, the number of worms waiting to be modeled diminished. It is also noticeable that when VC are enabled more simulation cycles are required. Fig. 5.19 portrays worm allocation and distribution when SC=4. The result show that worms being

modeled are reaching a flat saturation point in which the same number of worms occupies the interconnect as long as there are more worms to model in the jar.

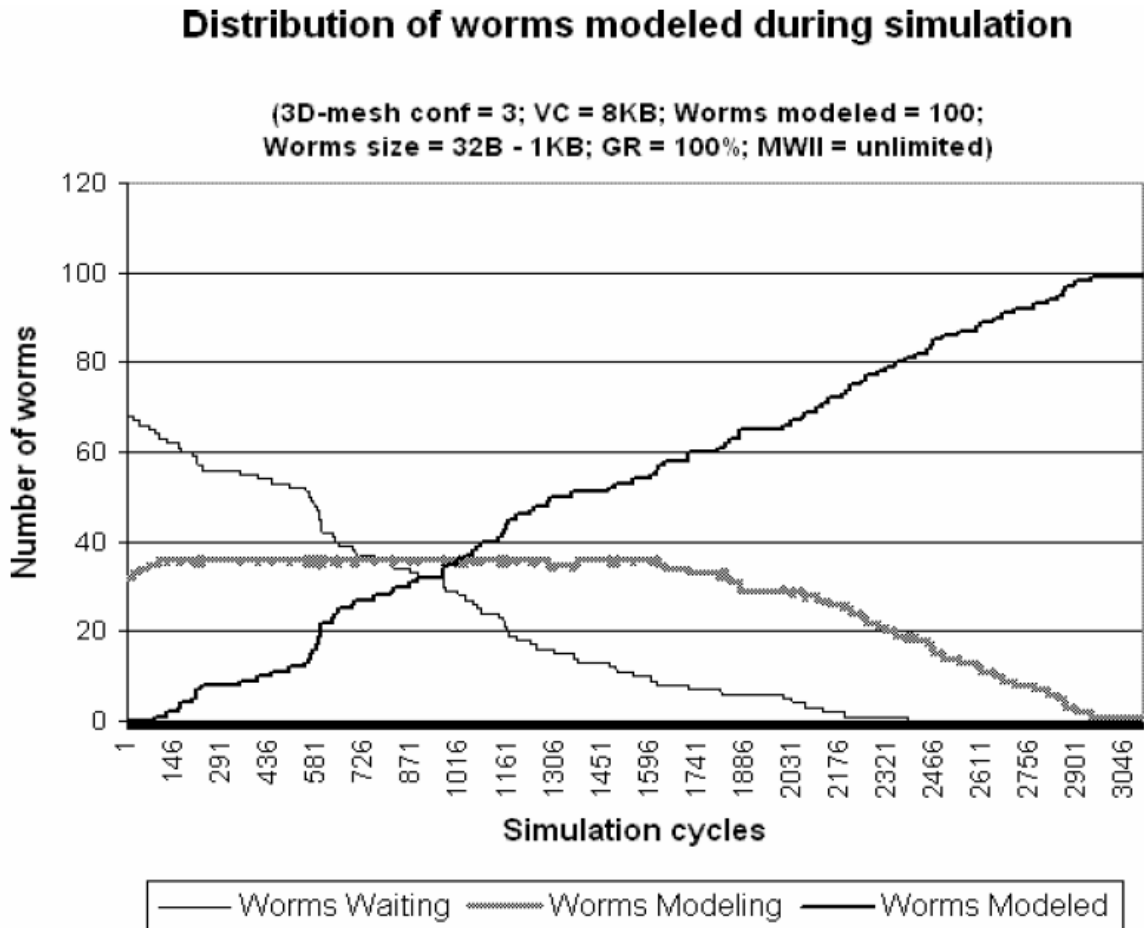


Figure 5.18: Worm allocation & distribution with VC=8KB

Worms are utilizing every channel available until the interconnect reaches saturation. However, simulation cycles are less than those taken when VC were enabled.

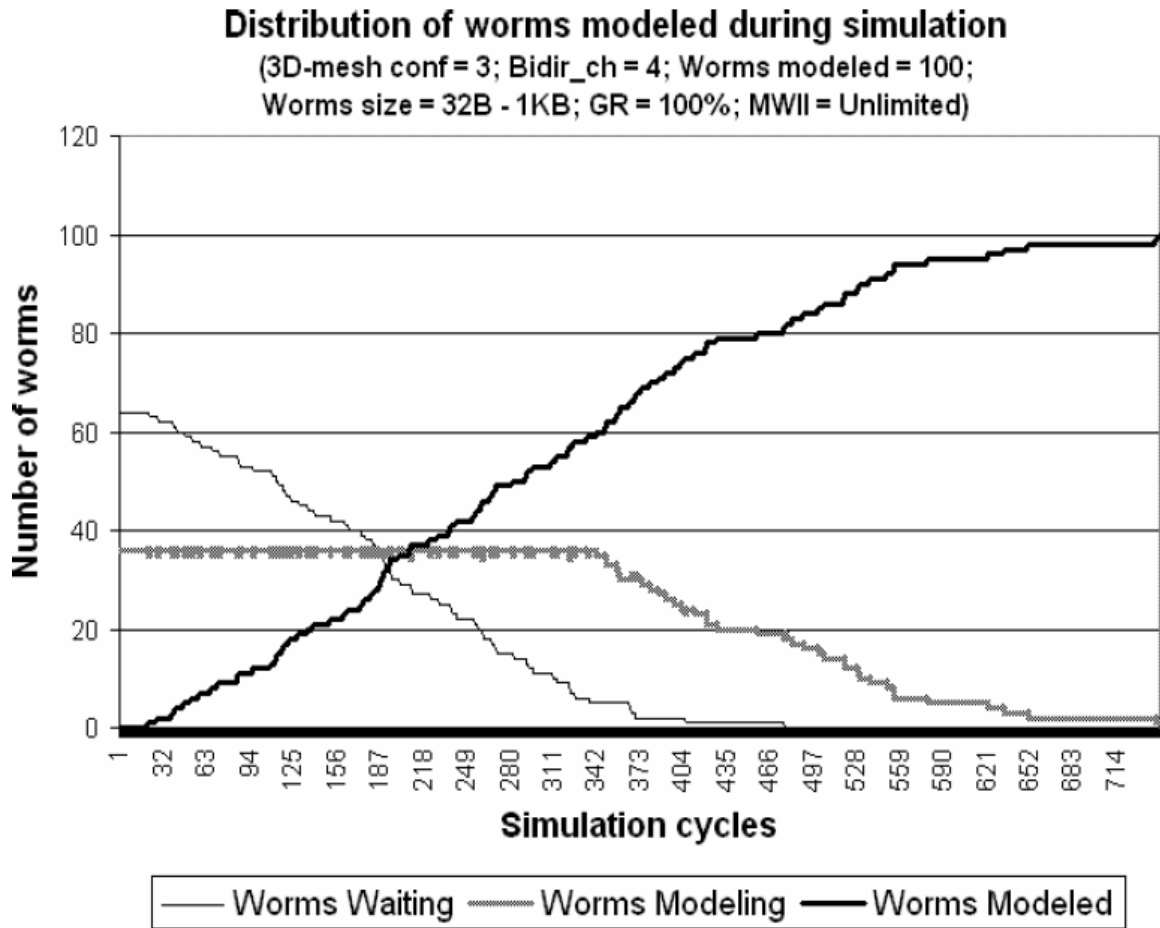


Figure 5.19: Worm allocation & distribution with SC=4

5.2.3 Routing accuracy

Routing accuracy measures how close the actual path of each worm is to its shortest path. Routing accuracy is calculated by taking the ratio between the shortest path possible to the actual path taken, which signifies the worm's deviation from its shortest path. Fig. 5.20 shows simulation of 100 worms using 3D-mesh interconnect with VC disabled and no sub-channeling. At the top of the figure the line portrays the number of additional links passed by each worm until it reached its destination. On the bottom part

we see the deviation of each worm (top line) from its shortest path (bottom line). If both lines overlap, then the worm took its shortest path. The path a worm takes depends on the traffic load at certain nodes of the interconnect. As the load increases, most worms deviate from their shortest path and adaptively propagate to their destination avoiding areas of hot-spots.

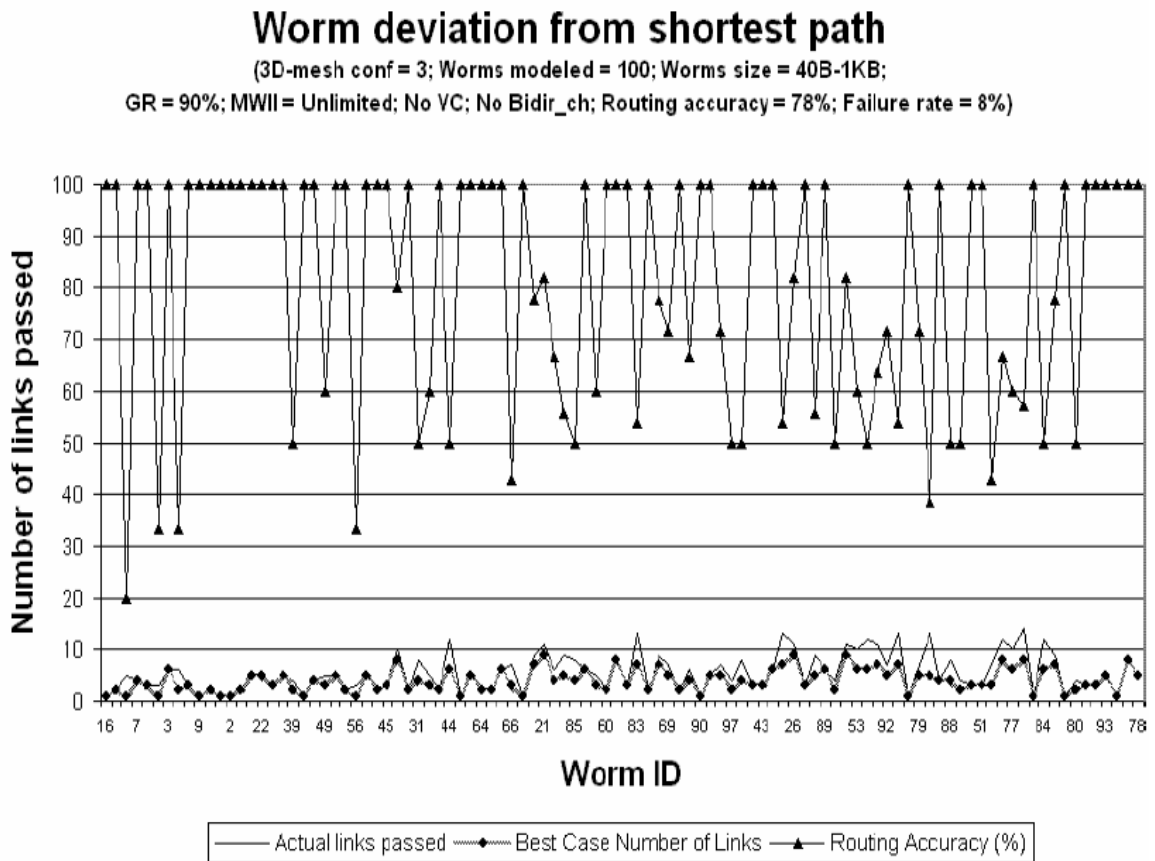


Figure 5.20: 3D-mesh worm deviation from shortest path

Fig. 5.20 shows the number of channels passed for each worm modeled using 3D-mesh interconnect. The top line shows the percentage of deviation from the shortest path. For example, if the top line points for a certain worm #ID to 100, that means the worm took the shortest path possible. If the value of the line is equal to 20, the worm deviated from its shortest path by 80% (more channel links). At the bottom of the figure there are

two overlapping lines. The thin line represents the actual path taken (measured in channel links). The thick line represents the shortest path. Therefore, when both line completely overlapping each other for a certain worm, that worm took the shortest path. As the number of channel links passed increases with respect to the shortest path possible, the thin line becomes further apart from the thick line.

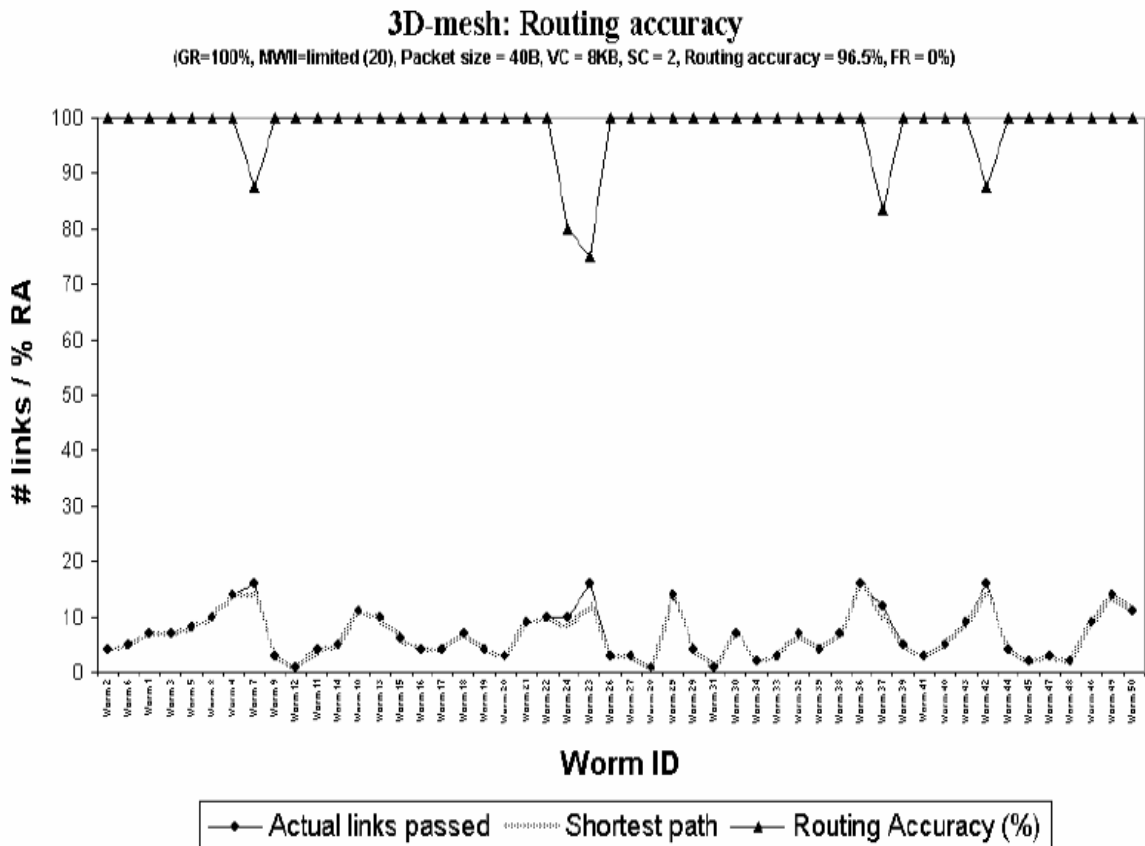


Figure 5.21: 3D-mesh routing accuracy with VC=8KB and SC=2

In Fig. 5.20 neither VC nor SC were enabled. Therefore, more worms deviated from their shortest path due to high traffic load at certain interconnect nodes. Fig. 5.21 shows the routing accuracy of all worm modeled using 3D-mesh. In this simulation 100 short worms (40B) were generated and both VC/SC were enabled. That resulted in a very high routing accuracy (96.5%) and no failures.

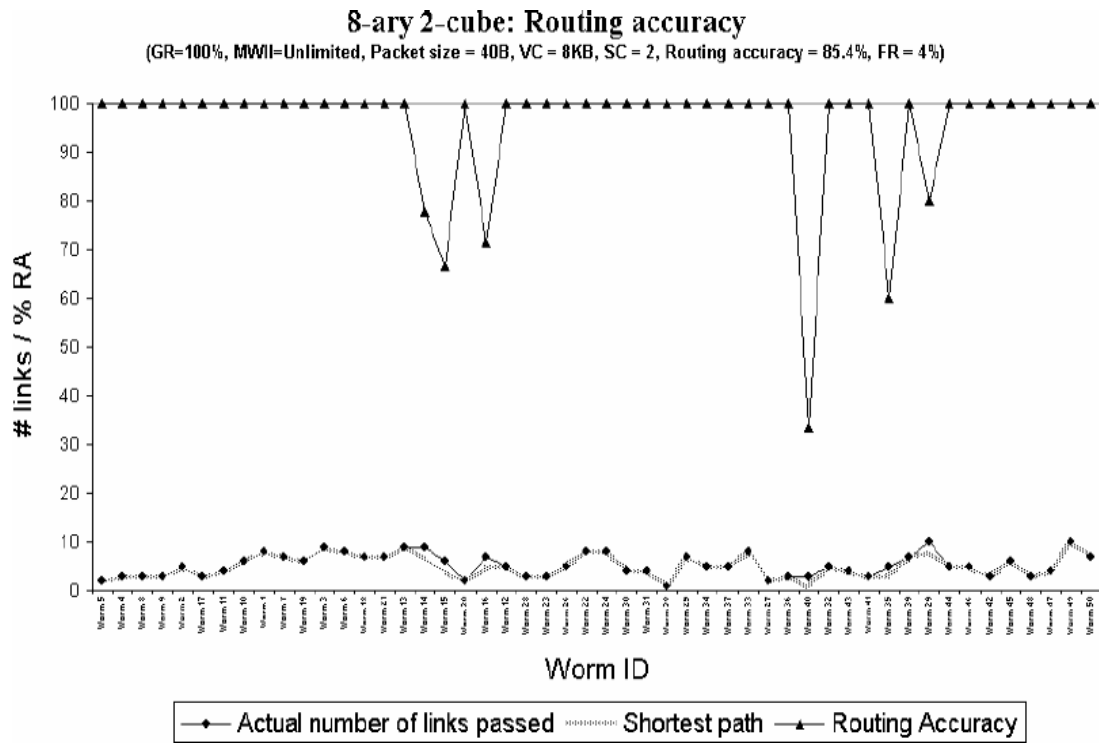


Figure 5.22: 8-ary 2-cube routing accuracy with VC=8KB and SC=2

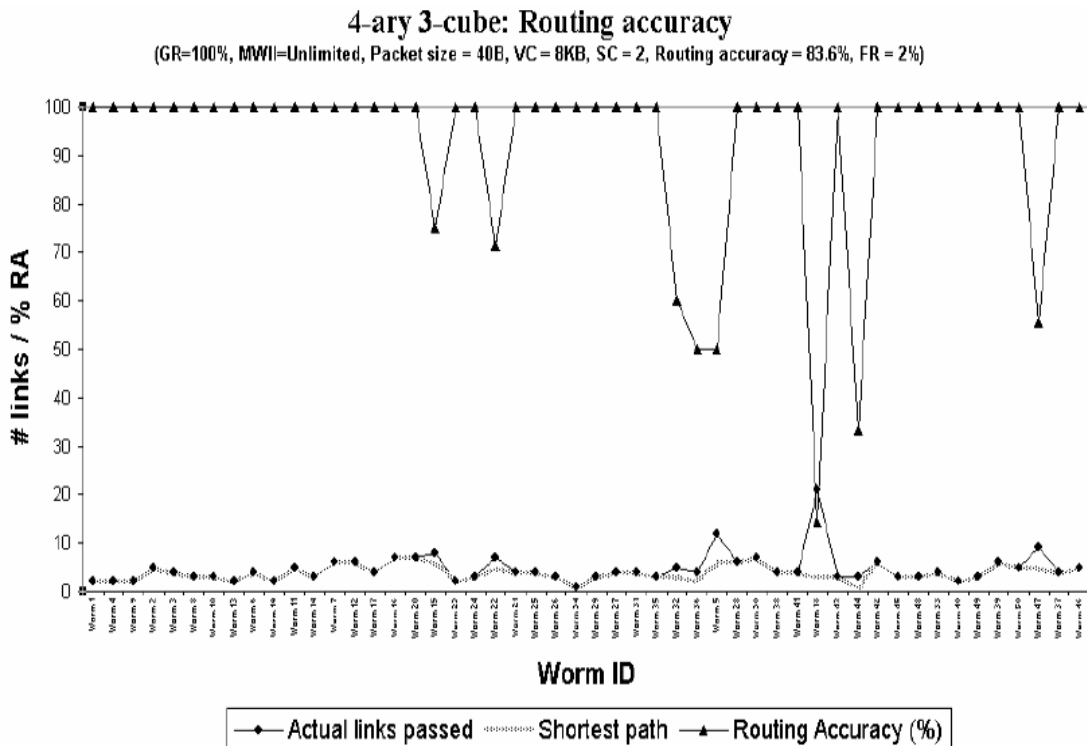


Figure 5.23: 4-ary 3-cube routing accuracy with VC=8KB and SC=2

The same simulation setup was used to evaluate routing accuracy of 8-ary 2-cube (Fig. 5.22) and 4-ary 3-cube (Fig. 5.23). All routing accuracy figures prove that the routing algorithm, with additional features such as VC and SC, is highly effective and on average provides above 80% routing accuracy with very low failure rate.

5.2.4 Interconnect and bandwidth utilization

Interconnect bandwidth utilization measures the number of occupied channels (or sub-channels) with respect to the total number of channels available in the interconnect.

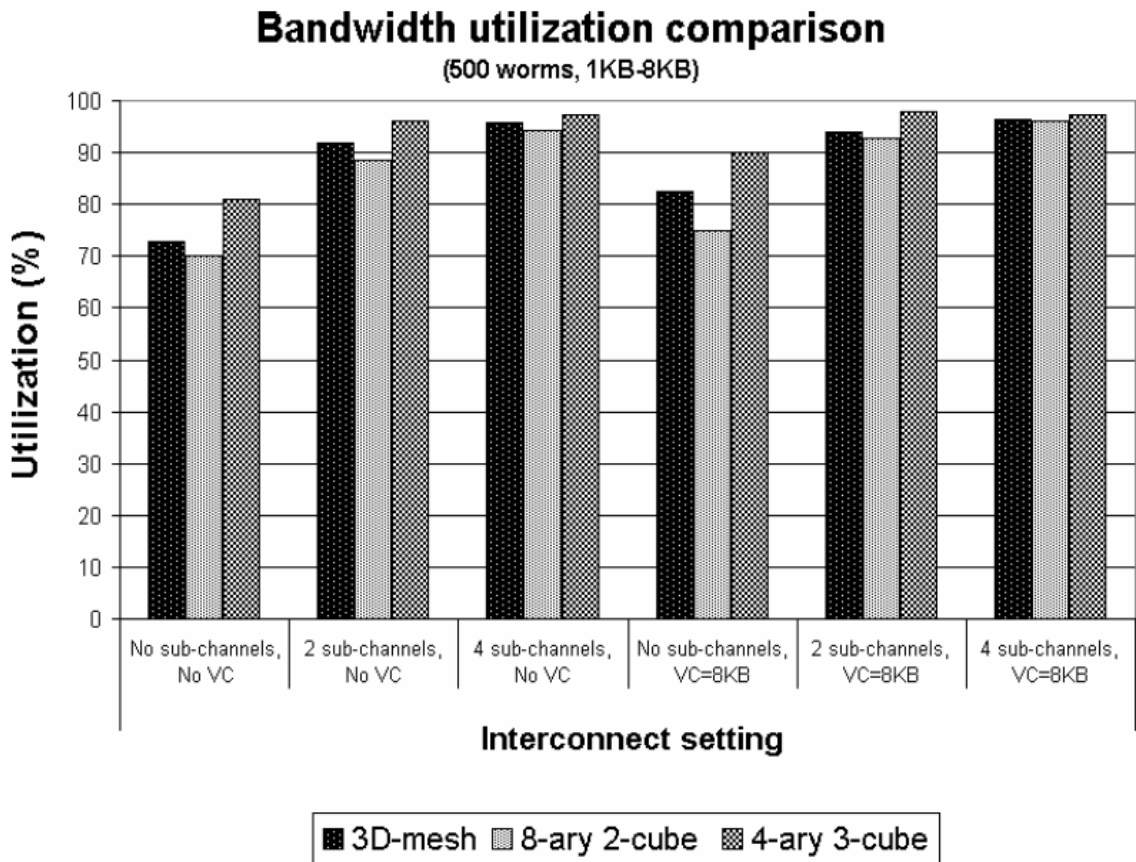


Figure 5.24: Bandwidth utilization rate

Fig. 5.24 portrays that the highest bandwidth utilization is achieved by using the 4-ary 3-cube network, while the 8-ary 2-cube has the lowest utilization rate. Sub-channeling improves bandwidth utilization as channel are partitioned into more sub-channels. The combination of VC and SC bring all interconnects close to their full capacity.

Interconnect utilization counts the number of busy ports within each traffic controller per simulation cycle. At the end of simulation it provides the average of ports that were set to busy status out of the total number of ports available in the interconnect throughout simulation. The results of interconnect utilization show very close relationship to bandwidth utilization (Fig. 5.25).

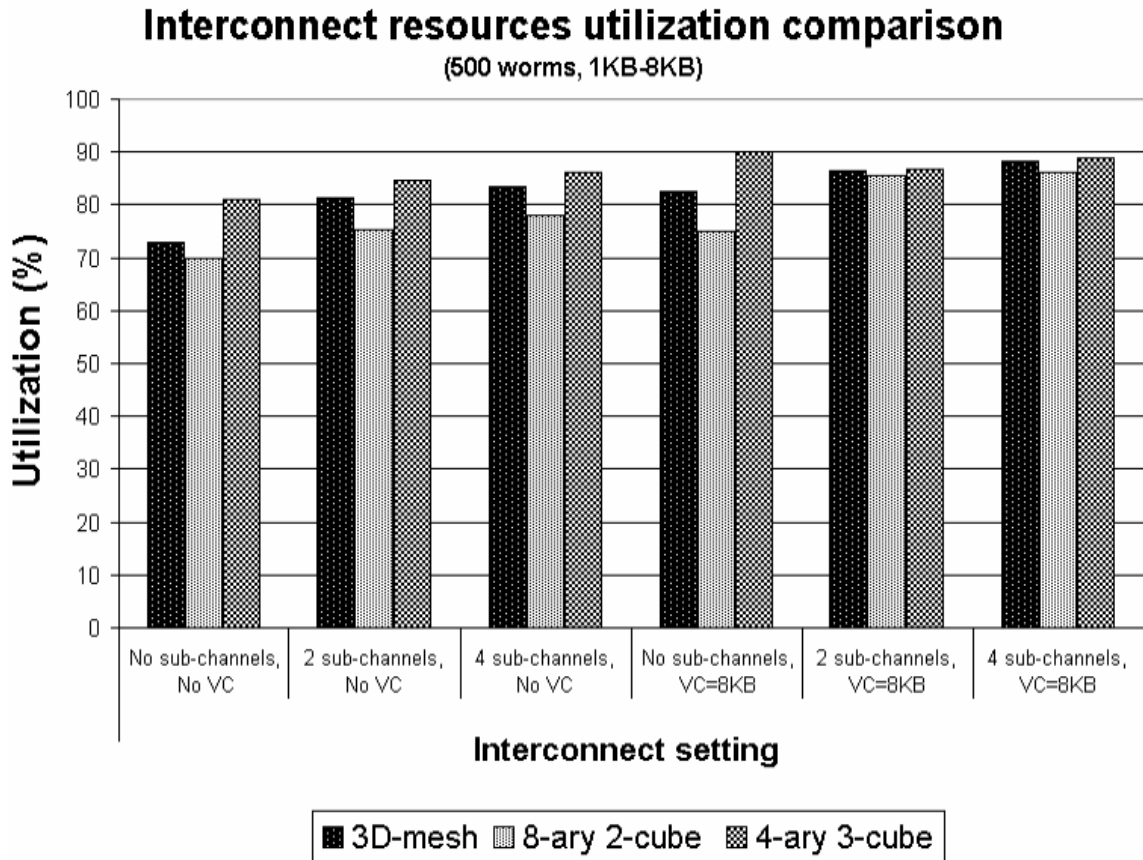


Figure 5.25: Interconnect utilization rate

Again, 4-ary 3-cube ports are set to busy status more often than the 3D-mesh or 8-ary 2-cube. Although interconnect utilization seems an equivalent measure to bandwidth utilization, it is a little different since ports status is not directly related to the channel usage.

An output ports can stay in not-busy state if a worm that intend to use it is buffered into virtual channels. Since each traffic controller has four ports the channel connected to the non-busy port can be utilized by a worm entering from a different direction.

5.2.5 Failure rate

Failure rate is a measure of the number of worms, out of the total worms generated, that were retransmitted during simulation. Retransmission takes place when a worm is blocked and cannot obtain the resources it required to maintain an active status within the interconnect. For example, when VCs are disabled, then a worm will require retransmission if it cannot be routed to any output port within a certain node for more than one simulation cycle.

Fig. 5.26 depicts a failure rate comparison for all interconnect types with VC switched to enabled/disabled. This figure shows that VC significantly reduces failure rate. Moreover, the size of VC has a major effect on failure rate as well (Fig. 5.29). As the size of VC increases more worms can be buffered for longer periods of time within each node instead of failing and being retransmitted.

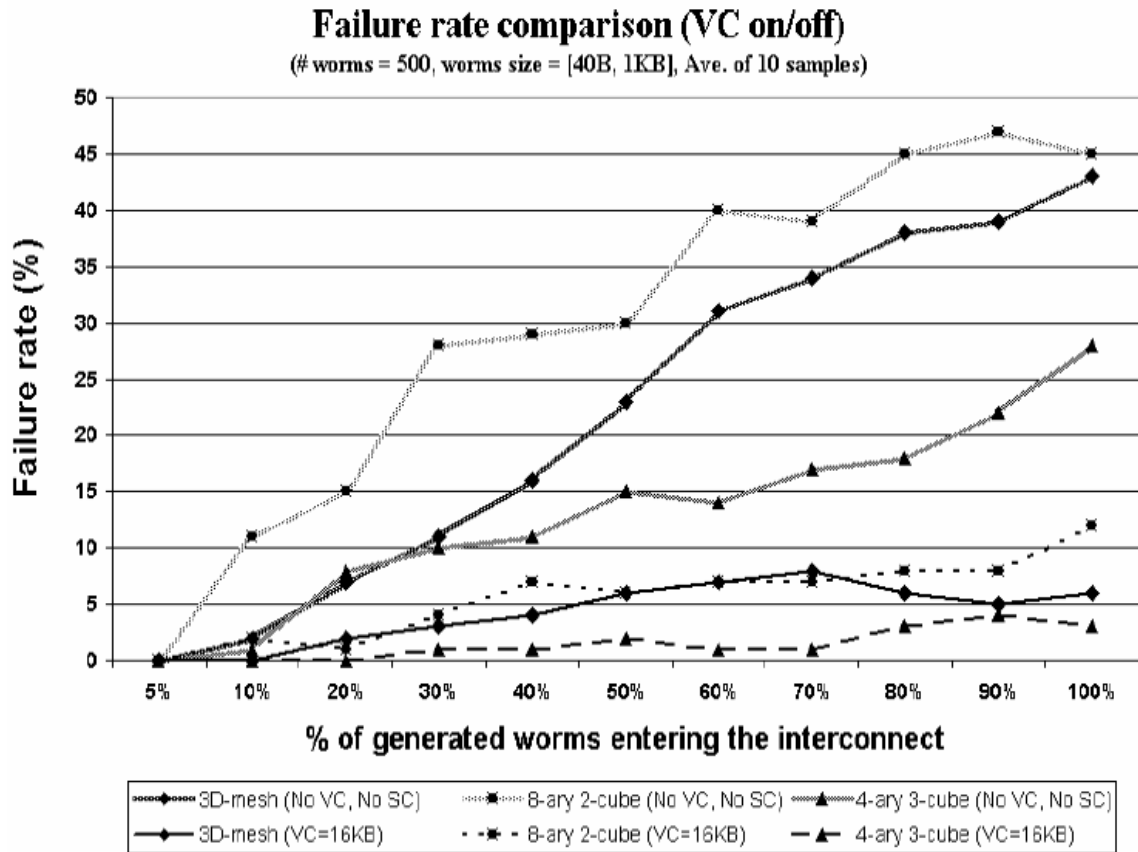


Figure 5.26: Failure rate comparison with VC switched on/off

Failure rate significantly decreases when sub-channeling is enabled. Figs. 5.27-5.28 show that failure rate was reduced to single digit rates when SC=2 or SC=4. Since SC divides the physical channel into two (or four) sub channels, it provides worms with additional paths that can be taken while propagating to their destination. With SC, worms share the same channel and are able to propagate in different directions within each channel. Therefore, routing flexibility increases dramatically, contention and hot-spots diminish and failure rate is reduced.

At each cycle there is a random number of worms generated in which only a fraction of those worms enter the interconnect. The software interface provides the user with a variable called generation rate (in units of %) which determines what percent of

worms, which are generated at each cycle, will enter the interconnect. The rest of the worms will be discarded.

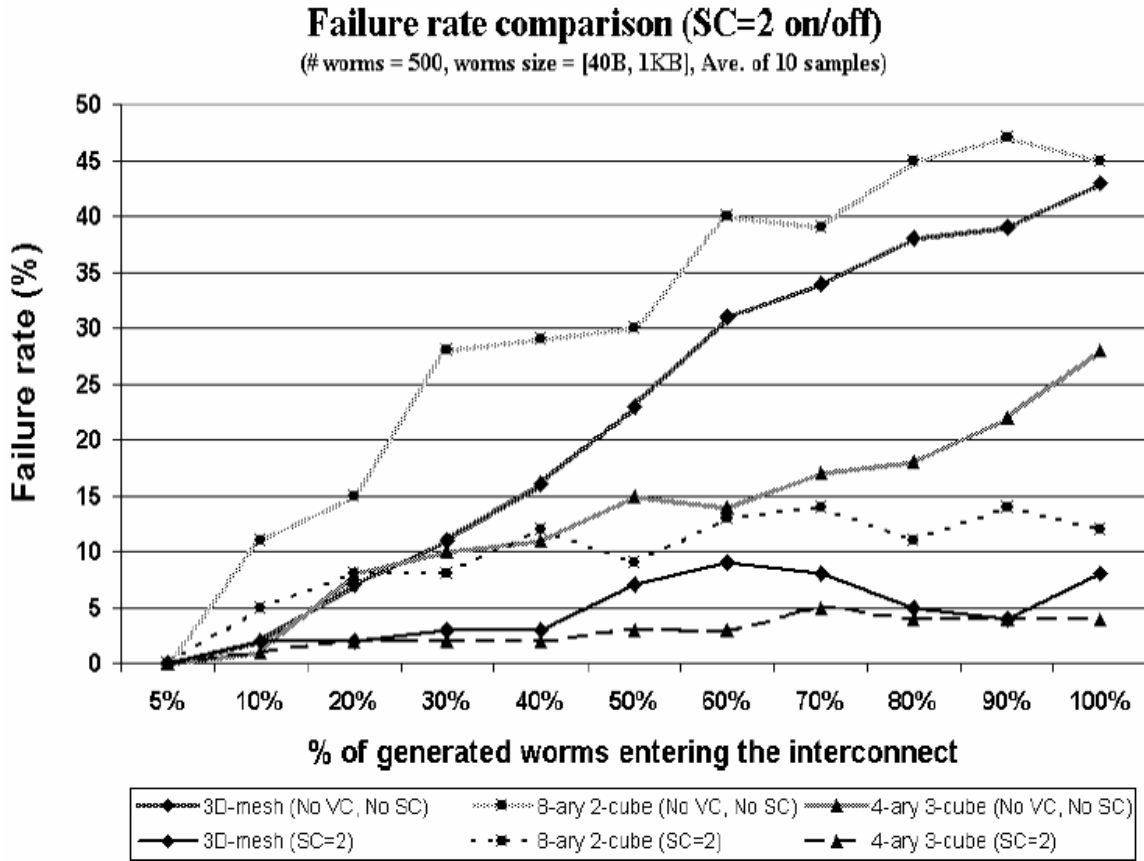


Figure 5.27: Worm failure rate with SC=2 switched on/off

For example, if at a certain simulation cycle 10 worms were generated and the parameter g is set to 70% then only 7 of them will enter the interconnect. The value of g is shown in Figs. 5.26-5.28 as the x -axis. The combination of VCs and SCs reduce failure rate dramatically, on average, to less than 10% for all interconnects. This is a 75% reduction since VCs allow worms to be buffered within a node until one of the ports becomes available. SCs provide worms with additional alternative paths to surpass areas of traffic congestion.

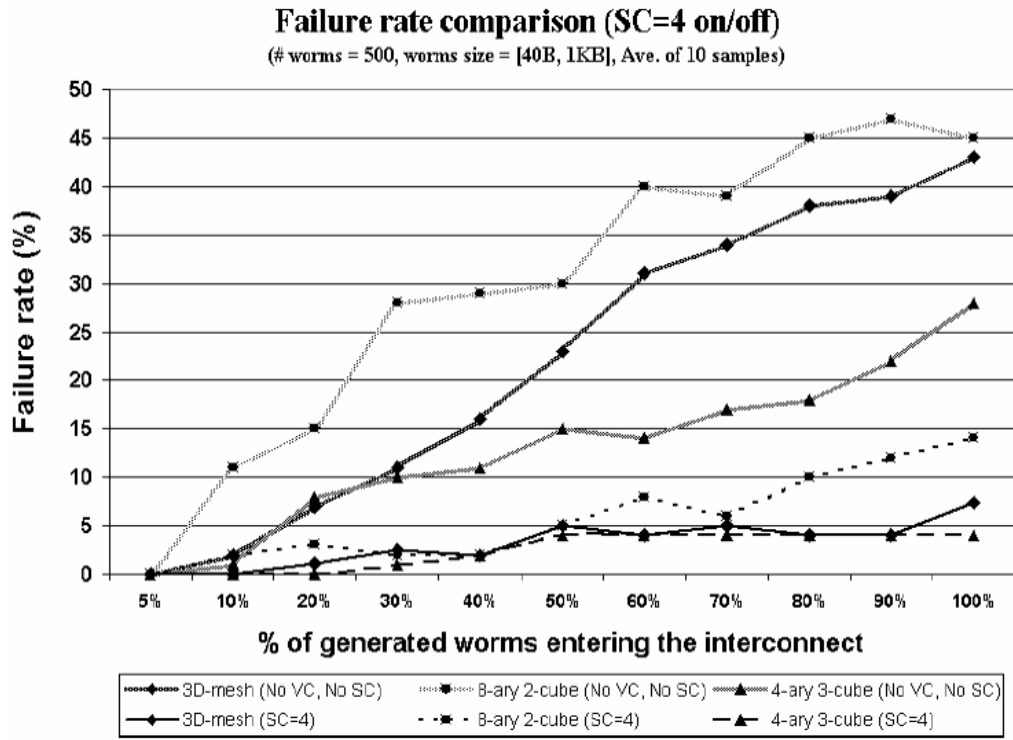


Figure 5.28: Worm failure rate with SC=4 switched on/off

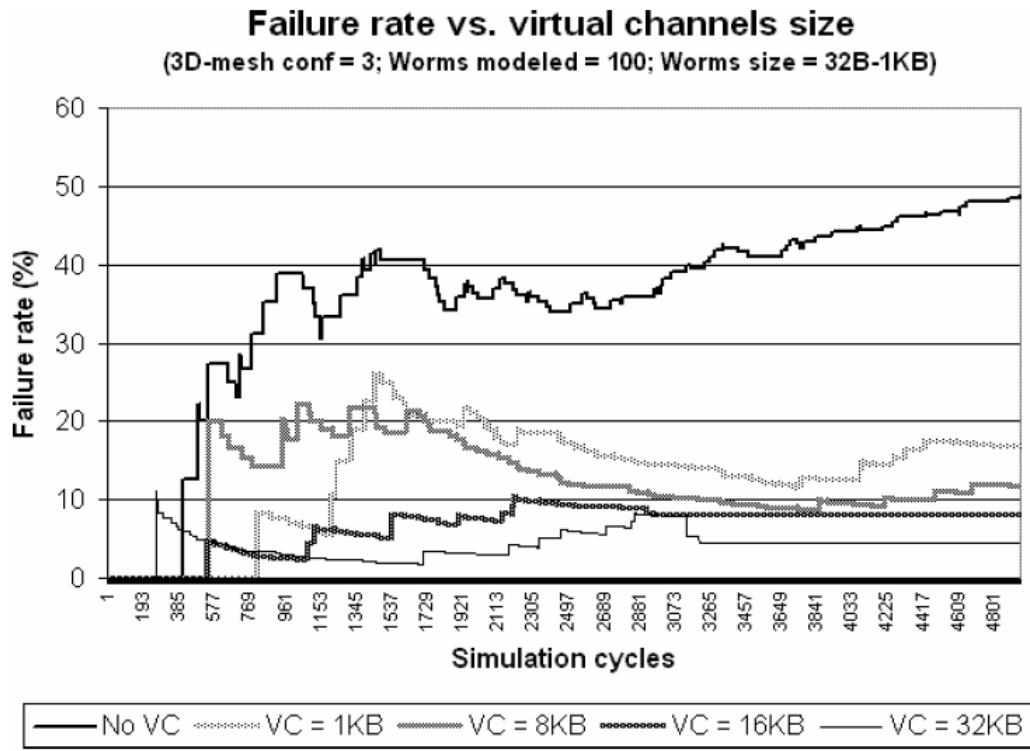


Figure 5.29: Failure rate vs. VC size

Fig. 5.29 shows that the size of VCs (in KB) influences failure rate as well. Small size VCs will become fully occupied quicker than a large size VC. If a worm is buffered more than few simulation cycles and still cannot find an available output port then it will result in a transmission failure.

5.2.6 Routing accuracy vs. hot-spot nodes

In this simulation the paths taken by all worms, using 3D-mesh, 8-ary 2-cube and 4-ary 3-cube interconnects, were recorded. Then, the paths were analyzed to collect the nodes which were most frequently used and as a result, caused other worms to deviate from their shortest path to avoid transmission failure.

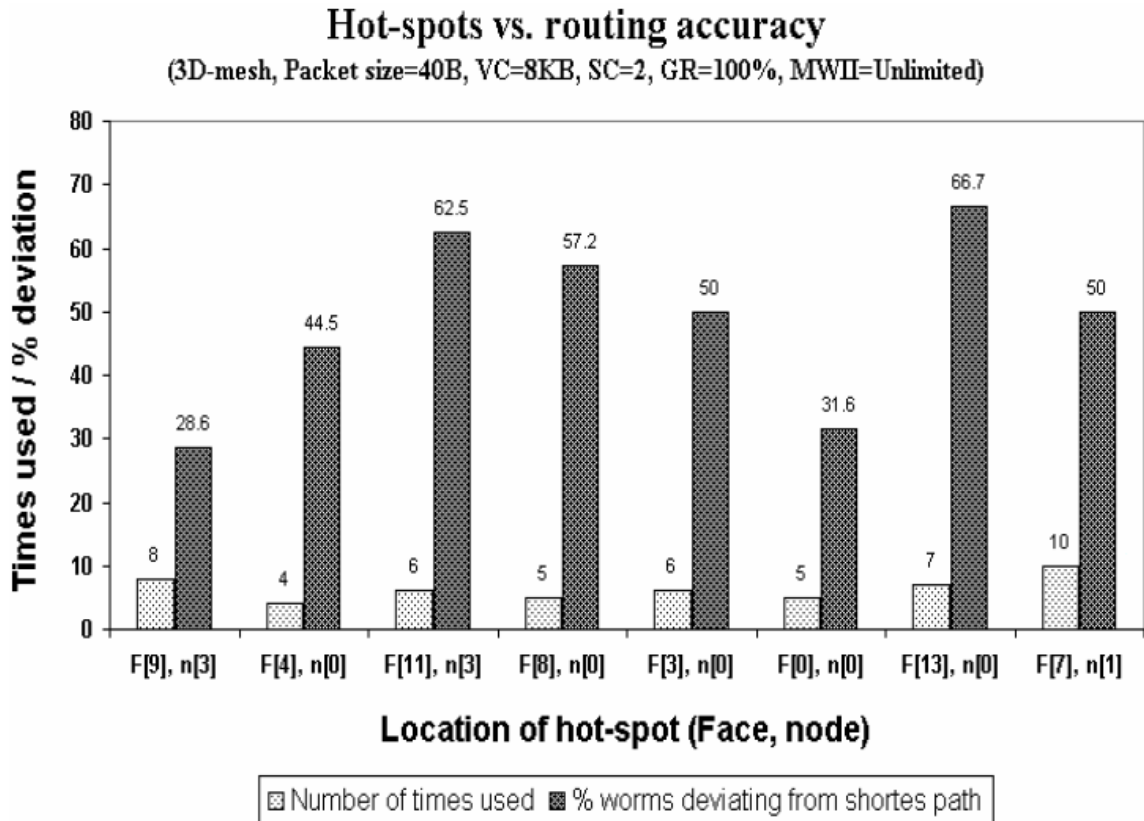


Figure 5.30: 3D-mesh routing accuracy vs. hot-spot nodes

Results for all interconnects show (Figs. 5.30-5.32) that some hot-spot nodes caused approaching worms to deviate from their shortest path by 50%-60% more channel links. For example, Fig. 5.30 depicts that hot-spot in face 11 node 3 (F[11], n[3]) caused 6 approaching worms to deviate from their shortest path by 62.5%. Hot-spots patterns are not repeated in the same locations. Traffic is randomly generated with random message lengths and from random nodes. Moreover, since an adaptive routing algorithm changes the path worms take in each simulation every simulation creates hot-spots in different location and in different frequency. Fig. 5.31 shows a hot-spot which occurred in face 3 node 6 (F[3], n[6]), that caused approaching worms to deviate from their shortest path by an average of 85%. In Fig. 5.32 one of the hot-spots, located on face 0 node 43 (F[0], n[43]) caused approaching worms to deviate from their shortest path by as much as 66%.

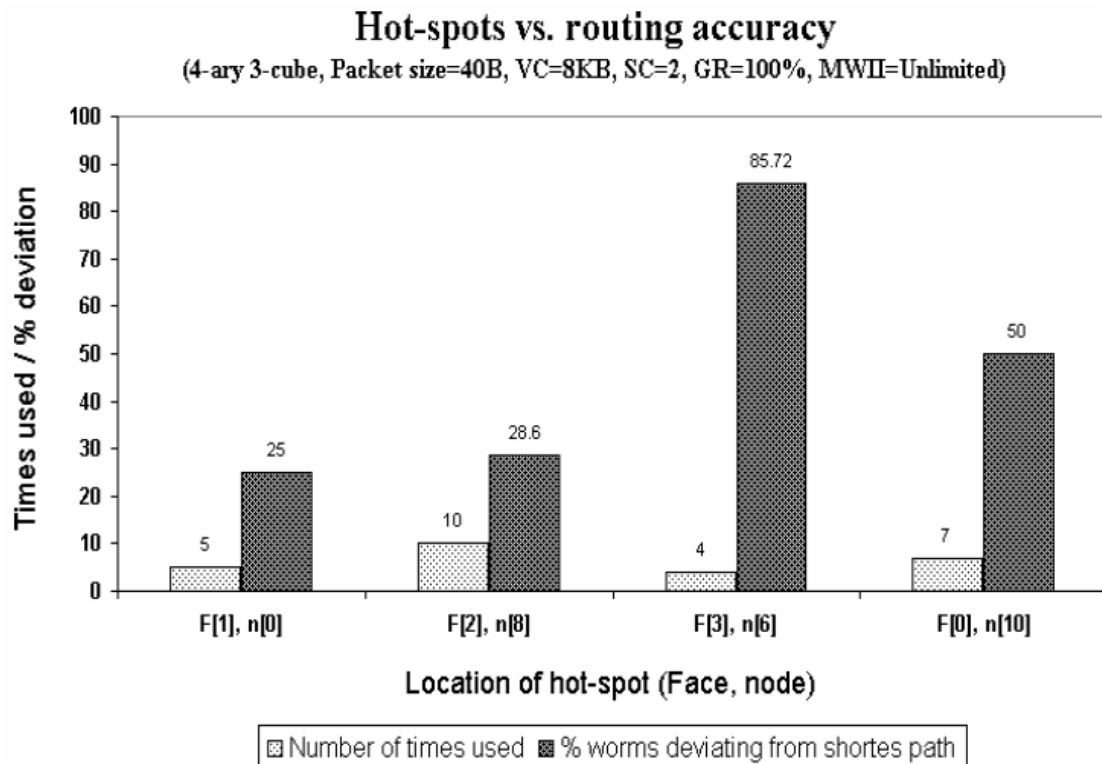


Figure 5.31: 4-ary 3-cube routing accuracy vs. hot-spot nodes

Although only few hot-spots occur per simulation, their effect on performance was significant. As the rate of hot-spot increases (a function of traffic load), worms tend to deviate from their shortest path more frequently and, as a result, the overall interconnect latency increases.

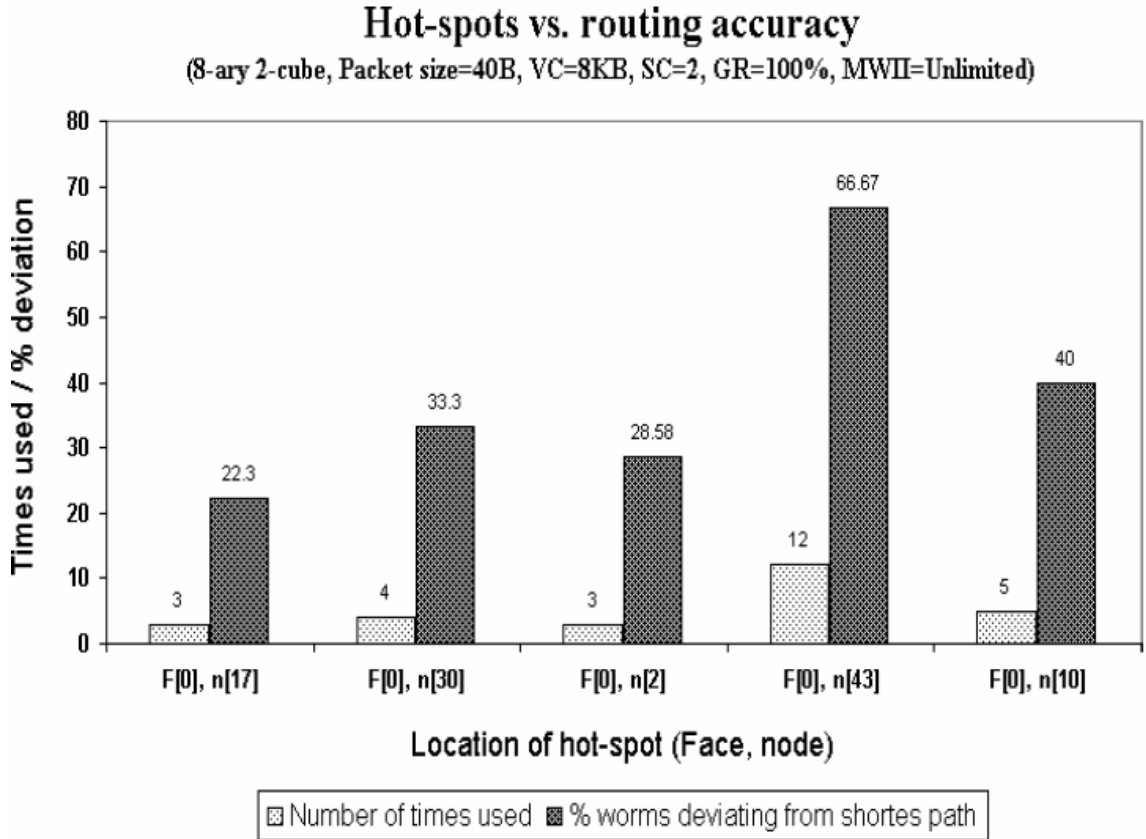


Figure 5.32: 8-ary 2-cube routing accuracy vs. hot-spot nodes

5.2.7 K-ary n-cube interconnects performance comparison with common interconnects

In this simulation we evaluate our 3D-mesh, 8-ary 2-cube, and 4-ary 3-cube interconnects with other currently used high-performance interconnect technologies such as Infiniband [70], Hypertransport [72] and PCI-Express [73]. We used reported results

provided by each individual vendor to compare with our results. In addition, the performance properties of these technologies takes into account a constant channel size of 32-bits and a single communication link.

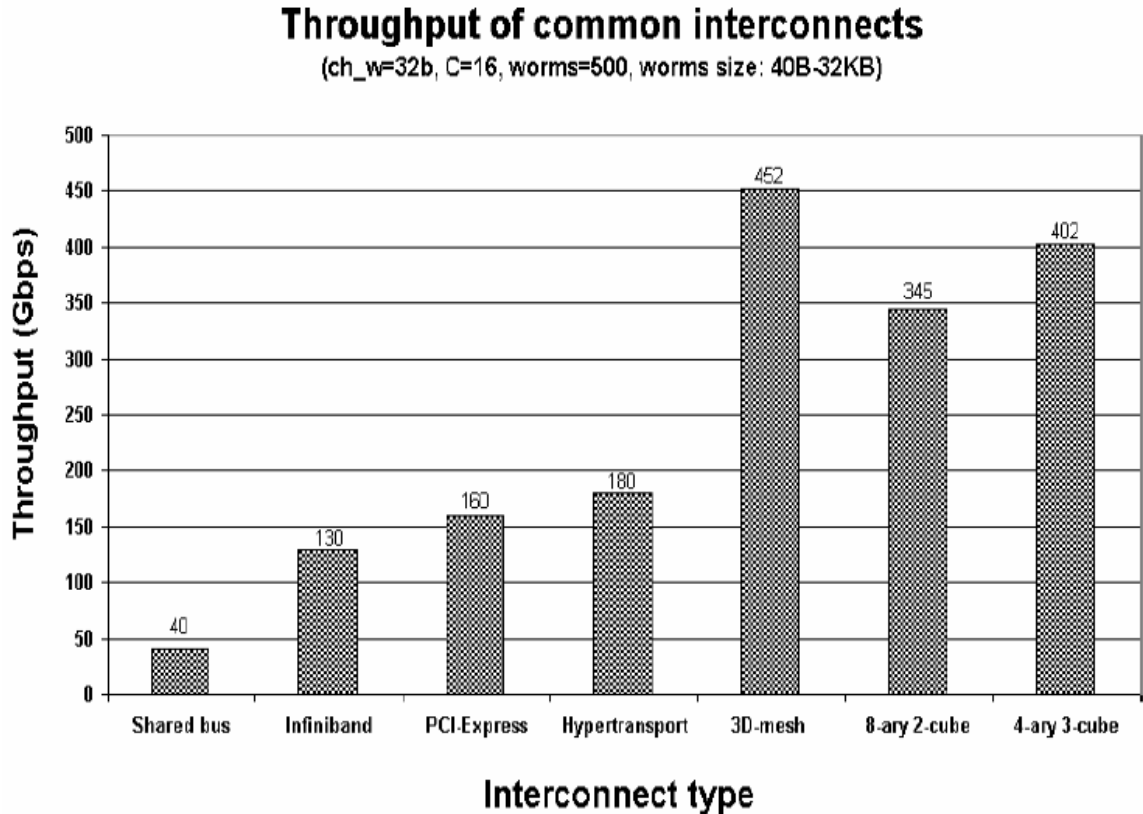


Figure 5.33: Throughput comparison: k -ary n -cube interconnects vs. common interconnect technologies

For the 3D-mesh interconnect the settings are: channel width is 32 bits, interconnect size is 16 cubes, number of worms generated is 10, each worm is 1KB in size. Virtual channels as well as channel partitions were enabled. The throughput comparison results are shown in Fig. 5.33. The throughput values of the 3D-mesh, 8-ary 2-cube and 4-ary 3-cube interconnects represent the average throughput of each interconnect. 3D-mesh shows superior results compared to all of its competitors reaching

a peak throughput of 452 Gbps (about twice the throughput of the best interconnect available not including the other types of k -ary n -cubes tested).

5.3 3D-bus Simulation Results

5.3.1 Latency of 3D-bus vs. shared-bus

Shared-bus is commonly used as a communication link between network processor and multiple memories. Our goal is to explore the limitations of shared bus and its capability to scale for higher line rates. In this simulation we compared shared-bus with 3D-bus (both with channel width=16b). The shared-bus length is equal to the 3D-bus length.

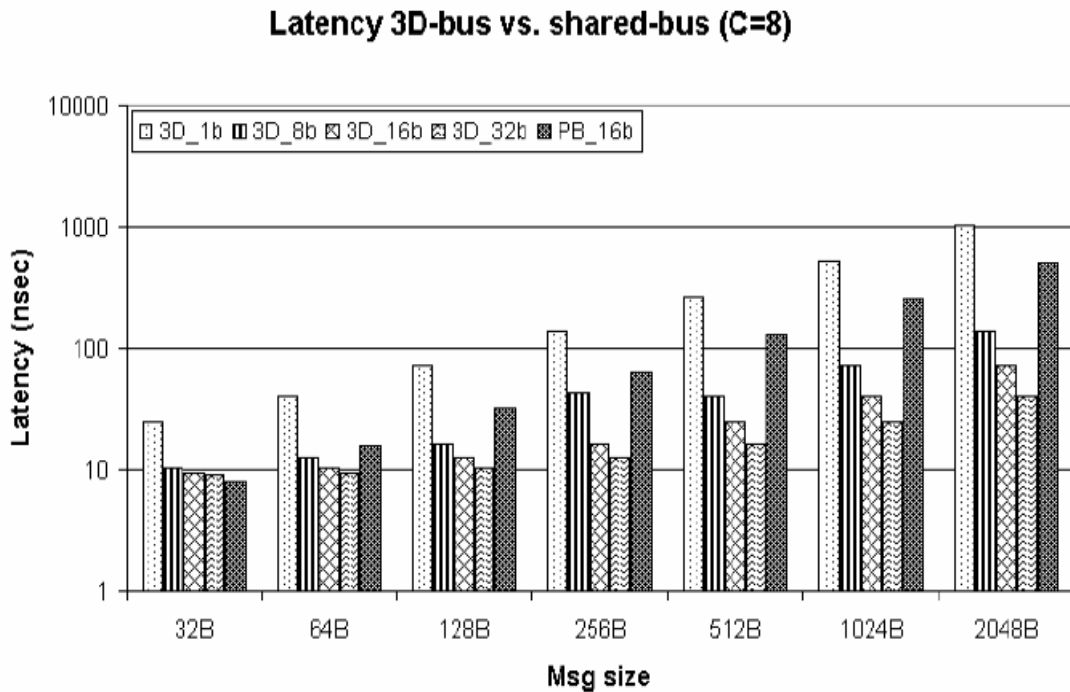


Figure 5.34: Latency of 3D-bus vs. shared-bus

Comparing the shared-bus with the 3D-bus (Fig. 5.34) shows that while message size is small, shared-bus has lower or equal latencies compared to its counterpart. The results portray that switching latency becomes dominant in the 3D-bus when short messages are generated, while there is no switching incorporated in the shared-bus. As messages become longer, only the header of each message introduces switching delay while the rest of the data (larger size) propagates through the interconnect following the header. Shared-bus, on the other hand, copes with queuing of message generating modules and fair bus access grants, each module “holds” the bus longer and therefore, queuing becomes a bottleneck.

5.3.2 Throughput of 3D-bus vs. shared-bus

Throughput measures the number of bits transmitted per unit of time. The interconnect settings of this simulation resembles the one for latency. Message size increases in multiples of two and the channel width varies from 1b to 32b. The number of cubes composing the interconnect is 8. The shared bus settings are 16b channel width and the same length as the 8 cubes measured in series.

Fig. 5.35 portrays that shared bus throughput outperformed 3D-bus throughput for small size messages, since switching delays become dominant for short messages. For long messages, only the header of each message introduces switching delay while the rest of the packets propagate through the interconnect (following the header) in one propagation delay per channel.

Thus, once the header reaches destination the rest of the message is transmitted with lower latency and higher throughput. Long messages generated by modules connected to shared bus cause large arbitration latencies and therefore, performance degradation. 3D-bus peak throughput reached 405 Gbps. Another observation shows that throughput increases with higher intervals as the channel width doubles in size.

Throughput 3D-bus vs. shared-bus (C=8)

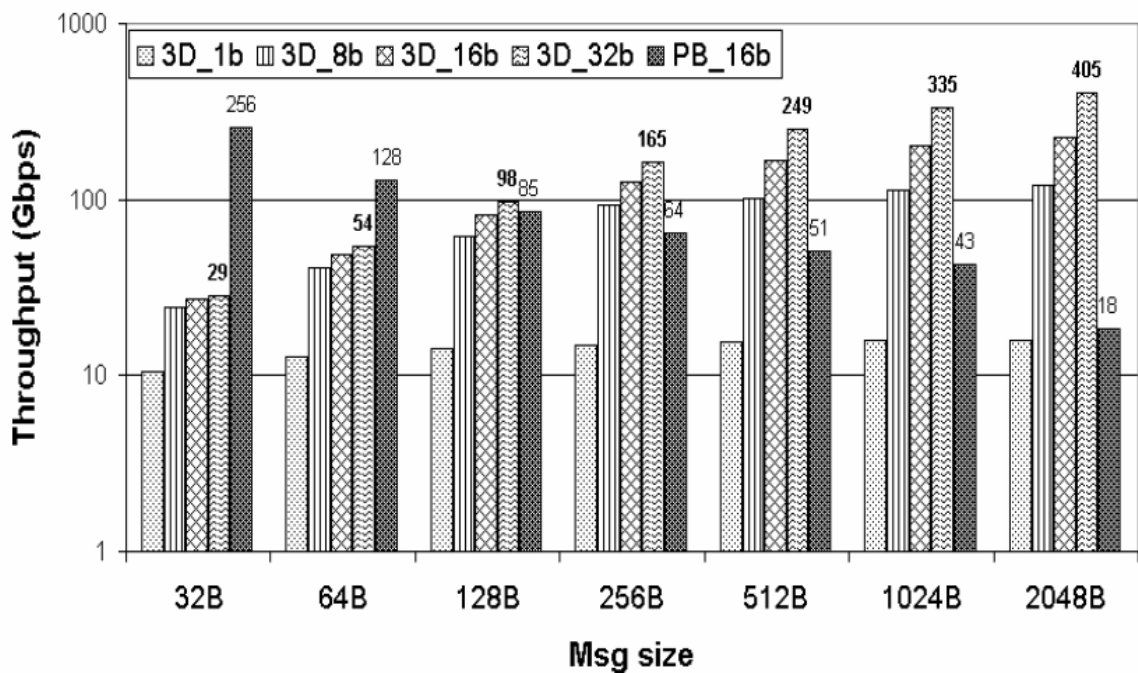


Figure 5.35: Throughput of 3D-bus vs. shared-bus

5.3.3 3D-bus routing accuracy

Routing accuracy measures how close the actual path taken by all worms generated within one simulation to the ideal (shortest) path. If the actual path taken is equal to the shortest path then routing accuracy equals 100%. In this simulation our goal is to investigate the effect of VC and SC on routing accuracy.

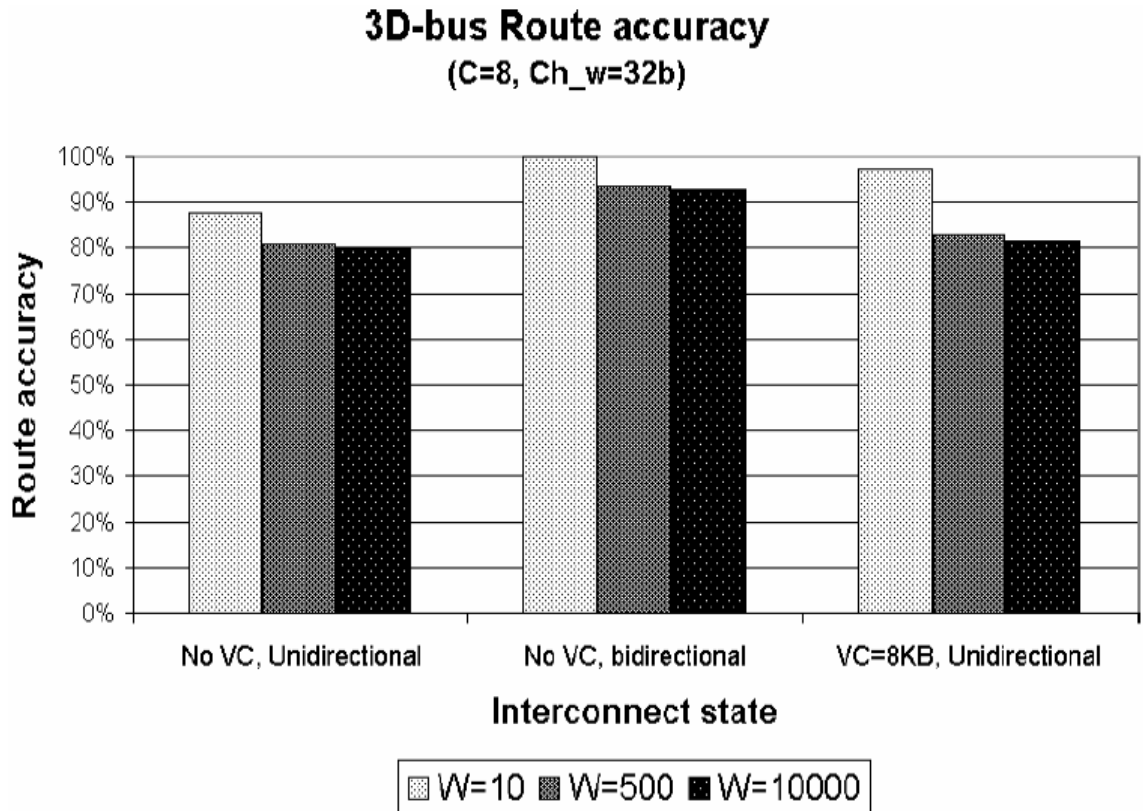


Figure 5.36: 3D-bus average routing accuracy

Fig. 5.36 portrays that both VC and SC significantly increase the average routing accuracy. When SC were enabled routing accuracy increased by 16% while VC contribute a 10% increase.

5.3.4 3D-bus failure rate

Failure rate measures the number of worms that were retransmitted as a result of their inability to reach destination. This scenario can occur in multiple situations. First, if a worm at a current node cannot be forwarded to an output port and virtual channels are not enabled then it causes a failure and it will result in retransmission of the worm.

3D-bus failure rate (C=8, Ch_w=32b, W=50)

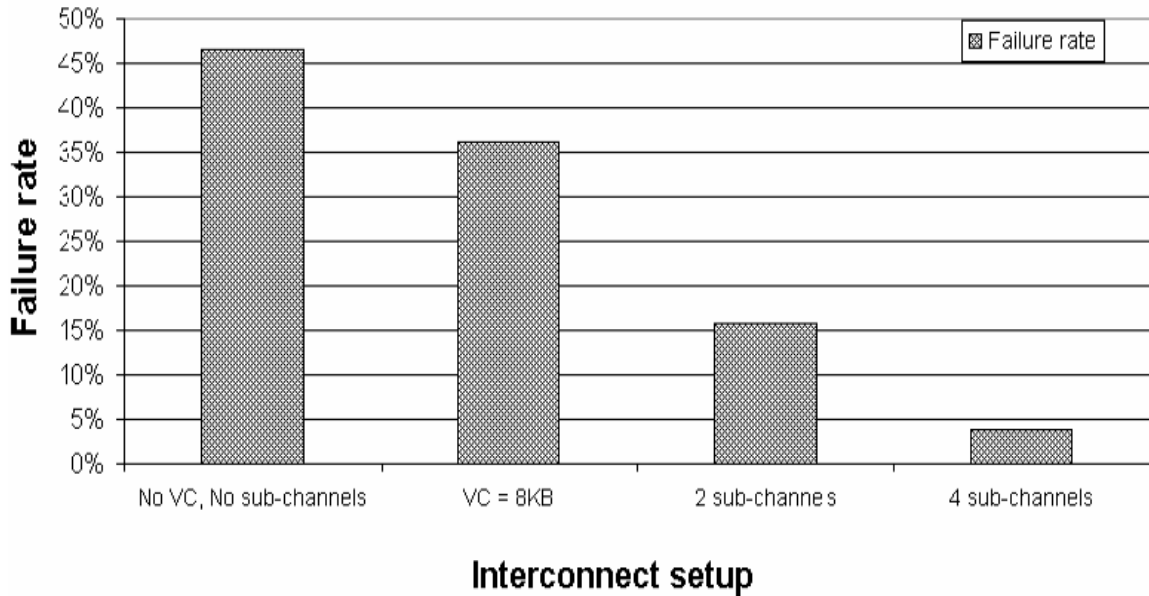


Figure 5.37: 3D-bus worm failure rate

Second, when a virtual channel is enabled but reaches its max capacity (and all ports are still busy) then it requires a retransmission of the worm. Simultaneously, the virtual channel is flushed so that a new worm can occupy it. Failure rate is measured as the number of retransmitted worms vs. the total number of worms generated (Fig. 5.37).

Simulation results portray that failure rate diminished when virtual channels as well as sub-channeling were used. The highest failure rate was reached when there were no virtual channels and no channel partitioning. Since many worms collide due to heavy congestion and deadlocks, many worms need to be retransmitted. Virtual channels lessen the number of retransmissions by allowing worms to be queued until the path is cleared. The size of the virtual channels determines the number of packets that can be queued. If a virtual channel becomes full as a result of a worm overflowing it, then the same worm is

retransmitted. Sub-channeling, on the other hand, significantly reduced retransmissions due to the fact that worms have more flexibility in choosing a route.

5.3.5 3D-bus latency with memory and PE interfaces

3D-bus interconnect can connect more than 4 PEs and 4 memories on each of its sides by adding two interfaces: memory interface and PE interface. Modules connected to each interface cannot send messages simultaneously since there are only four input nodes on each of the interconnect sides. Therefore, at each simulation only eight modules can transmit data into the interconnect while the other modules have to wait in a round robin process scheduling. In addition, if the interfaces are enabled, any module being selected to transmit data at a certain simulation cycle will incorporate additional switching delay in order to count for the delay the interface introduces when connected to the 3D-bus interconnect.

Fig. 5.38 shows that, as the number of modules (PEs and/or memories) on each side of the interconnect increases latency increases, almost exponentially, as well. As more PEs/memories are connected the time that each element is required to wait increases rapidly and as a result latency reaches almost 1msec (about 10 times higher than the latency recorded with peak throughput of 308 Gbps). Latency at this level reduces the throughput dramatically (from PE=16/M=16 and above).

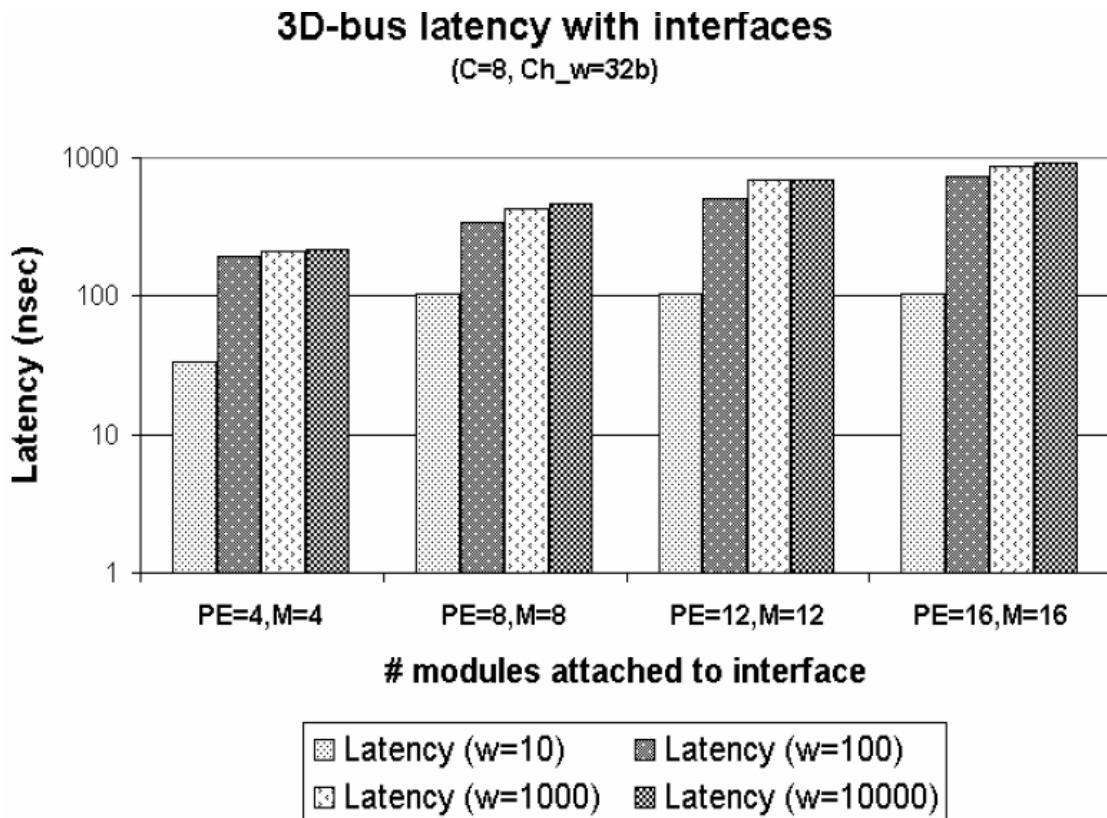


Figure 5.38: 3D-bus latency with interfaces attached

The figure emphasizes the effect of worm size on latency as well. Long worms means that messages are longer which requires extended transmission time. Hence, other modules are required to wait significantly longer for their turn to transmit data.

5.3.6 3D-bus performance comparison with common interconnects

In this simulation we evaluate our 3D-bus with other currently used high-performance interconnect technologies such as Infiniband [70], Hypertransport [72] and PCI-express [73].

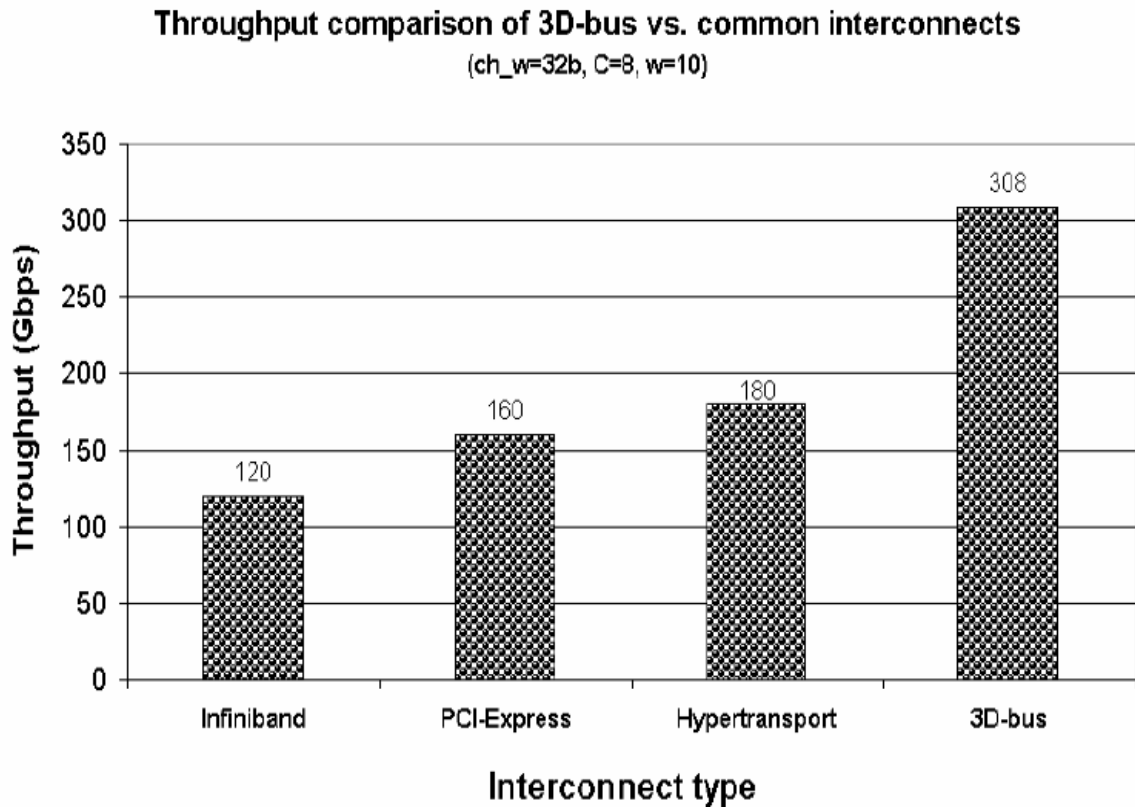


Figure 5.39: 3D-bus throughput comparison with common interconnects

To be accurate, 3D-bus is set to resemble all same parameters used for each of the compared interconnects. The following values are used: channel width is 32b, interconnect size is 8 cubes, Number of worms generated is 10, each worm is 1KB in size. Virtual channels and channel partitions are not used. The results are shown in Fig. 5.39. From Fig. 5.39 we see that 3D-bus shows superior results compared to all of its competitors although none of the enhanced features were used (VC, sub-channeling) which can enhance its performance even further.

CHAPTER 6: RESEARCH CONTRIBUTIONS & FUTURE WORK

In this work, we explored the characteristics of different types of interconnect architectures to increase the off-chip memory bandwidth on line cards. In addition, we provided a thorough performance analysis for multiple interconnect architectures using real incoming traffic characteristics such as randomness of packets generation with variable IP packet lengths. We developed a state-of-the-art simulation framework which can simulate multiple k -ary n -cube interconnects with different processor-memory configurations to adopt non-uniform traffic. We developed two variations of k -ary n -cube interconnects called 3D-mesh and 3D-bus. The 3D-mesh interconnect provides a multi-path off-chip linkage between processing elements and memory modules and allows packets to be shared and transferred by different processor and memory modules on the network line card simultaneously. 3D-mesh outperforms all of its competitors, whether they are k -ary n -cube based interconnects or other commercial interconnect technologies. Moreover, it can physically and functionally fit into the stringent line card area limitations and thus, provides the optimal interconnect architecture that is feasible to be manufactured on the line card's PCB.

The result of this work has the potential of changing interconnect architectures permanently not only on line cards but also for the bus mechanisms used in PC architectures and on-chip communications mechanism used within the network processors. Network processing hardware design is the fastest growing field in the networking industry. This research provides additional depth into off-chip communications mechanisms in multi-processing environments (such as network

processors), especially, for implementing it on line cards. Our contributions are not only centered in the research of interconnects architectures and their applications, but also in developing new alternative designs that can replace current interconnects, such as the shared bus which is currently experiencing a memory access bottleneck. Moreover, the optimal interconnect - the 3D-mesh is a scalable, cost-effective, high-throughput low-latency interconnect, that, based on our results, provides significantly better performance than all of its counterparts.

The interconnect architecture is an integrated part of a line card. It needs to perform under constantly increasing line rates and variable packet sizes. Thus, we enhanced it with cutting-edge packet flow control mechanisms, such as virtual channels, sub-channeling and traffic controllers to reduce the effect of hot-spots and heavy traffic congestion. Moreover, our wormhole message passing mechanism combines multiple routing techniques to avoid message deadlocks and livelocks. The routing algorithm transfers messages through the interconnect, adaptively, with the least number of channels crossed (shortest path possible) and message failure rate. Our simulation results proved that those techniques are highly effective and will perform well with higher incoming line rates.

Our visual, custom-designed, event-driven, network-simulator emulates off-chip, k -ary n -cube interconnects and is the only simulator, to the best of our knowledge, that can evaluate k -ary n -cube interconnects with different processor-memory configurations and with real network physical characteristics (such as propagation delays, switching, enhancements etc.). The simulator is using state-of-the-art software optimization techniques such as STLs, singleton classes, pure virtual functions and more, to deliver the

most accurate, modular, and user friendly product. The simulator is available in two versions, a visual version and user interface version. Thus, the user can view the functionality and the movement of different worms (messages) through the interconnect and collect information about areas of heavy traffic loads, hot-spots and more.

Future work will include the implementation (layout and fabrication) of the interconnect on the line card's PCB. We would also like to expand our network simulator to include different Internet traffic workloads and to modify the simulator to enable it to simulate all types of k -ary n -cube interconnects, not only the ones with small dimensions.

APPENDIX: NETWORK SIMULATOR MANUAL

The network simulator provides the user with an easy to use visual representation of k -ary n -cube interconnects. The main features of this simulator include:

- Full control over interconnect performance enhancing elements and deadlock/livelock or hotspots avoidance techniques (MWII, VC, SC).
- Complete control over message generation and message length (GR, min, max, source and destination data in face-node-port format).
- Visual representation of the interconnect, its settings and message flow during simulation. The visual model dynamically changes its characteristics, such as size of channels and color depth, to adapt to changes in interconnect structure, traffic load and nodal occupancy. In addition, the user can change 3D-view of the interconnect and zoom in/out.
- Supports multiple configurations per interconnect type. For each configuration the user can set up the channel width, interconnect latencies, and scalability.
- Total user control during simulation to increase/decrease/pause simulation and to change pacing and sampling rate.
- Complete runtime data of simulation performance and detailed, per worm, performance metrics output files.
- Worms import/export to enable multiple simulations with same workload. Moreover, default values for all parameters can be modified by utilizing two configuration files.

Simulator menus

In order to run the simulator, the user has to run the executable file: “3DInterconnectSimulation.exe”. The initial window (Fig. A1a) shows a 3D-mesh (default type) interconnect. For each interconnect, a sphere represents a memory module and a 3D-cube represents a PE. All memories and PEs are connected via channels (thin lines). The user can change the interconnect orientation by pointing the mouse on the object and then click-and-hold the mouse while pulling it to any direction. As a results, the interconnect will change its 3D-view (angle). In order to move the interconnect up/down or right/left use the keyboard arrows to do so. In addition, the interconnect view can be zoom in/out. This can be achieved by using the “Page Up”, “Page Down” keyboard buttons.

At the top of the window is the simulator menu (Fig. A1a). The user can choose among the following tabs: File, View (Fig. A1b), Properties, Simulation and Help. The file menu allows the user to load worms from a saved file or quit simulation.

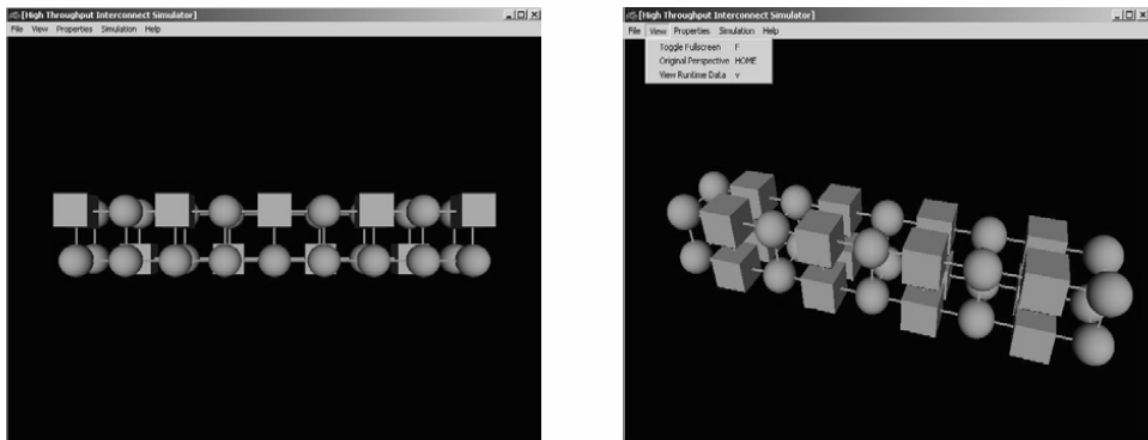


Figure A1: a) Main simulator window b) The view menu

The view menu has three options: “Toggle full screen” will adjust the simulator window to full monitor size view, “Original perspective” sets the interconnect to its original orientation and “View runtime data” will turn on a smaller window which includes simulation run time data.

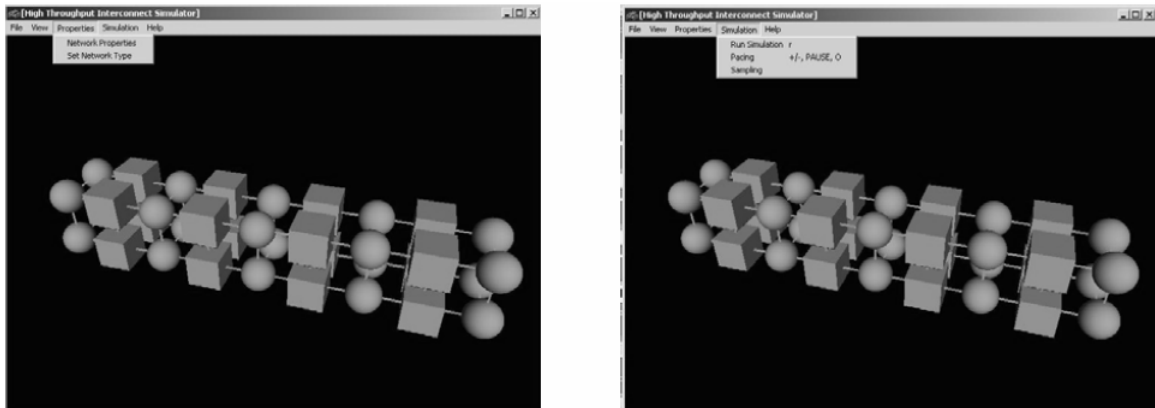
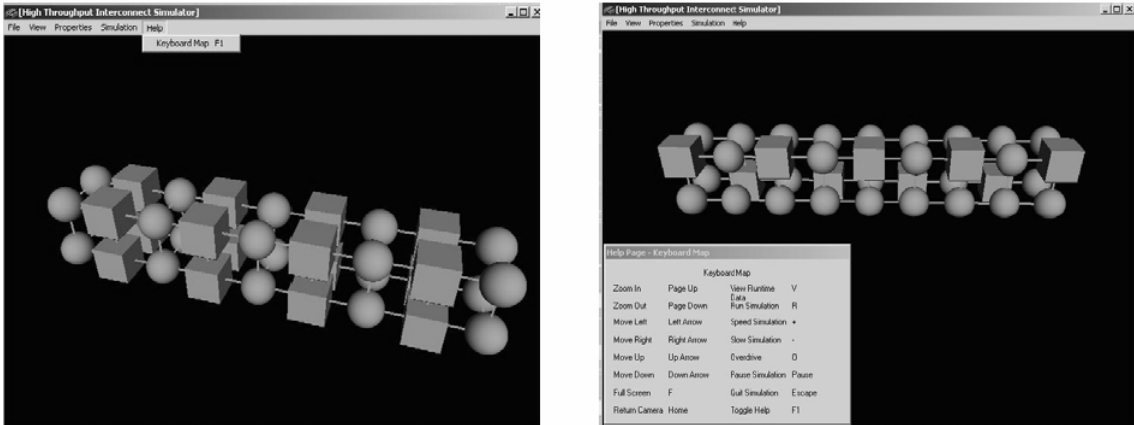


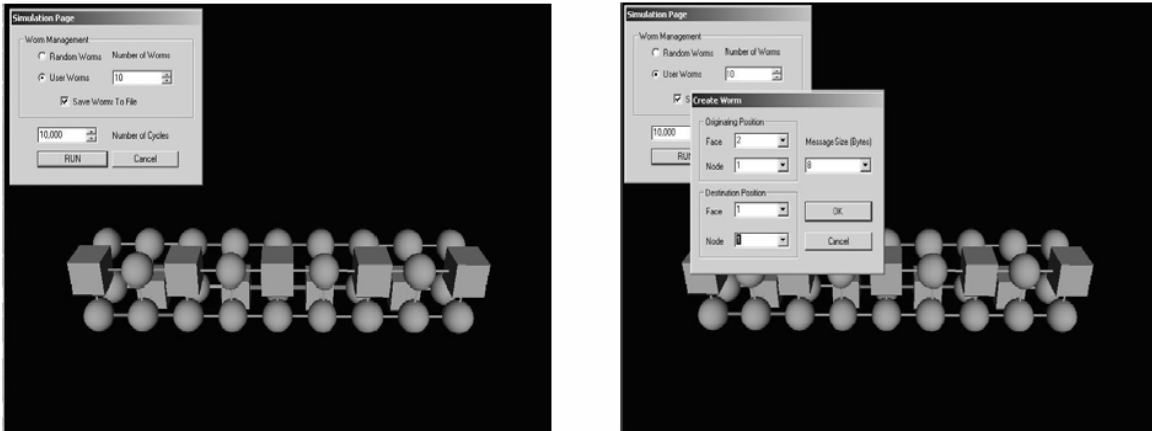
Figure A2: a) Selecting the properties menu b) Selecting the simulation menu

The “Properties” menu, shown in (Fig. A2a), sets the interconnect type (type and configuration) as well as interconnect properties, which include different system delays, worms data, and to enable/disable enhancement features. The “Simulation” menu, shown in (Fig. A2b), includes run simulation button, pacing (simulation speed) and sampling (performance sampling). The last menu is the “Help” menu to assist the user with different buttons and their functionalities (Fig. A3).

The user can determine if he/she wishes to generate random worms (worms characteristics are determined randomly by the system) or user worms which require the user to input the number of worms to simulate and, in addition, for each worm to determine its source/destination locations and message size. Worm generation is set via the simulation menu. Figs. A4a and A4b show the worm generation menus.



First the user must select if the worms are randomly generated or manually configured by the user. If worms are manually set (user worms) then the user will have to input the worms data. Whether worms are randomly generated or user worms, there is an option to save them into a file for future use. If the user wishes to load saved worms in order to run simulation with different settings, he/she must do so via the File menu.



Choosing interconnect type

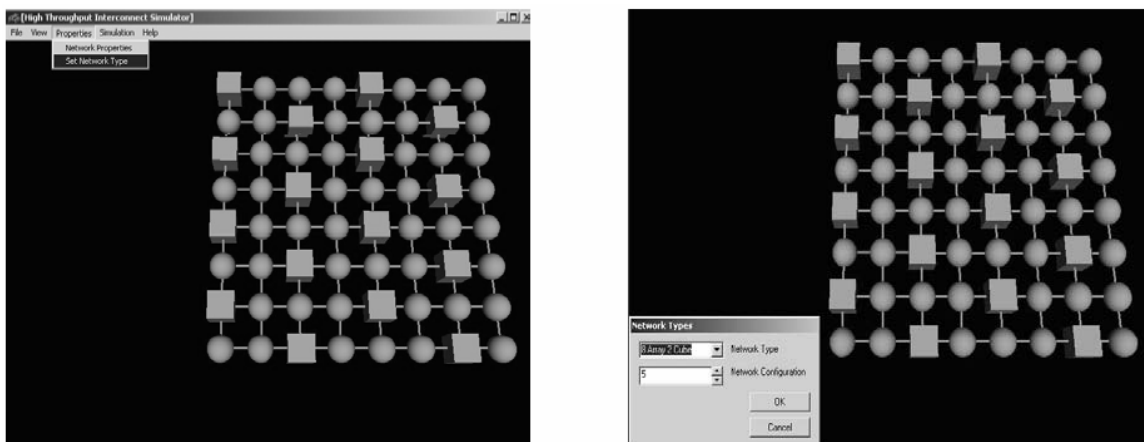


Figure A5: a) Network type selection b) Network type and configuration

The user can choose among three different types of interconnects: 3D-mesh, 4-ary 3-cube and 8-ary 2-cube. In order to select the interconnect type follow the next steps: in the “Properties” menu select “Set Network Type” (Fig. A5a). The “Network Types” will open and allow the user to change the interconnect type and configuration (Fig. A5b). The following figures (A6a-A6c) show a single configuration of each interconnect type. Interconnect configuration will be updated only after the OK button is clicked.

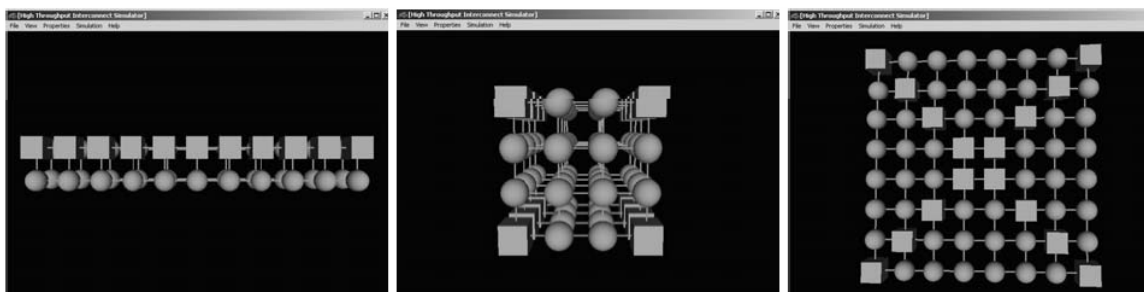


Figure A6: a) 3D-mesh b) 4-ary 3-cube c) 8-ary 2-cube

Setting network properties

The interconnect properties menu contains a vast number of system parameters, latencies and performance enhancement settings. Changing interconnect properties is done by selecting the “Properties” tab, then “Network properties” (Fig. A7a). The interconnect properties window is shown in Fig. A7b. Explanation of the interconnect properties inputs:

- Number of faces (applicable only to 3D-mesh): determines the number of faces comprising the interconnect. This input parameter is useful to evaluate performance vs. interconnect scalability.
- Channel width: sets the channels size (in bits).
- Max number of Worms: this parameter limits the number of worms being modeled during simulation. This feature can provide indication on interconnect saturation point.
- Generation rate: determines the percentage of worms entering the interconnect, at a certain simulation cycle, out of the total number of worms generated.
- Virtual channels (on/off and size): sets virtual channels to on/off and their size in KB. The default size is 1KB.
- Number of sub-channels: sets the number of sub-channels. Default value is 1 which means that channels are not partitioned.
- Delays: setting system delays which include propagation, switching and routing delays.

- Message size limits: the high and low limits of message size. This feature is useful if the user wishes to use random worm generation but wants to keep worm size within a certain range.

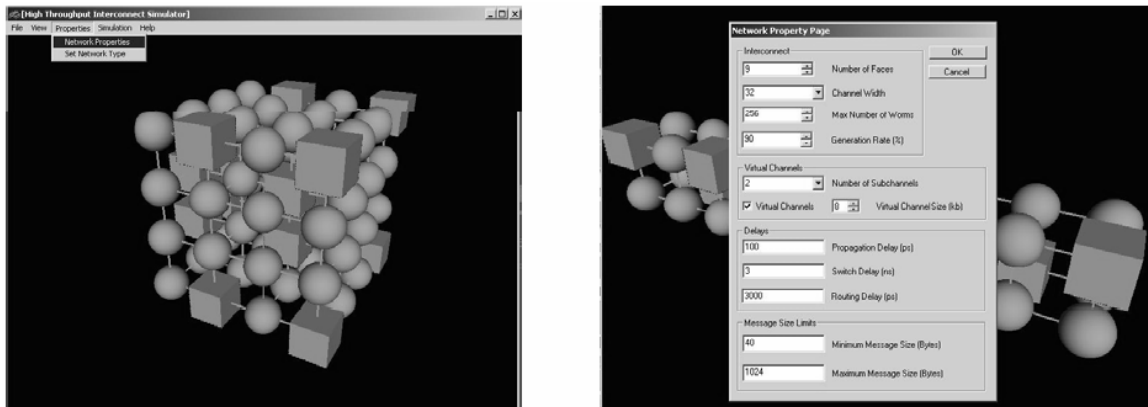


Figure A7: a) Network properties selection b) Network properties menu

Running simulation

Before running simulation there are two more features to get familiar with: pacing and sampling. The pacing window allows the user to change simulation speed (+ increase / - decrease). The overdrive button will run simulation in the fastest speed and will increase the simulation cycle intervals. The pause button will pause simulation. A second press on the pause button will resume simulation.

Throughput can be measured in two methods. One is without sampling and therefore, results in an average throughput calculated from all simulation cycles and presented to the user at the end of the simulation. Second, throughput can be sampled within certain intervals during simulation and then the average of those samples is

calculated. The sampling window provides this functionality. If enabled, the user must specify the number of samples and the intervals between samples. The default settings are 1,000,000 samples with interval of 1 cycle. Thus, if the user enables sampling but does not change the default values it will result in the average throughput.

When all interconnect settings and parameters are updated simulation can start. To perform simulation select the “Simulation” tab of the main menu. Then, to allow sampling click on the sampling tab (Figs. A8a-A8b), otherwise choose the “Run Simulation” function.

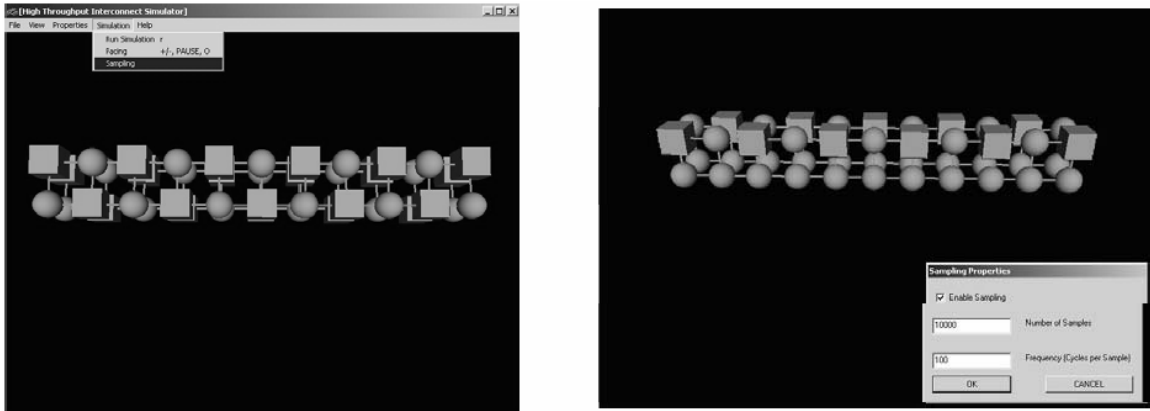


Figure A8: a) Selecting the sampling menu b) Sampling menu content

While simulation is running the user can change its pace by selecting the pacing tab in the simulation menu (Fig. A9a).

The pacing window pops-up as shown in Fig. A9b and the user can slide the bar to increase or decrease simulation speed. Fig. A10a portrays the pacing window with the overdrive option enabled. Fig. A10b depicts simulation pause.

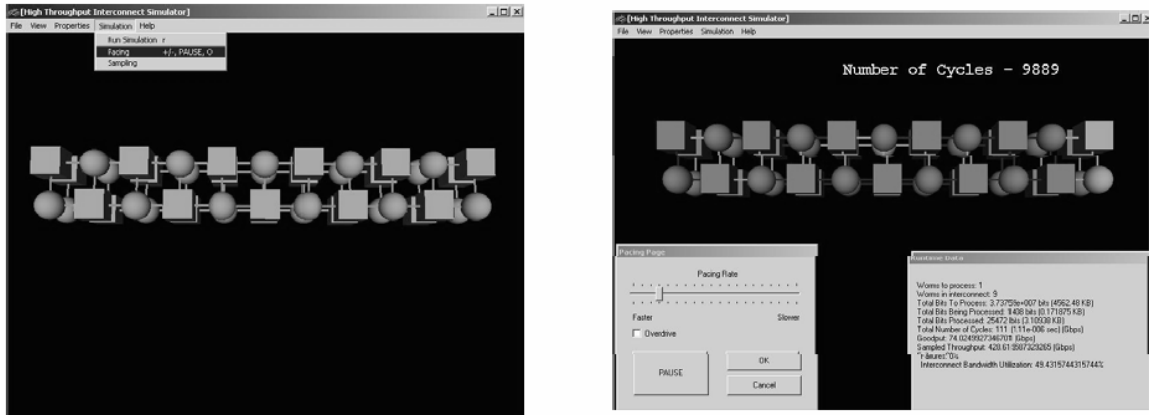


Figure A9: a) Simulation pacing menu b) Simulation pacing setup

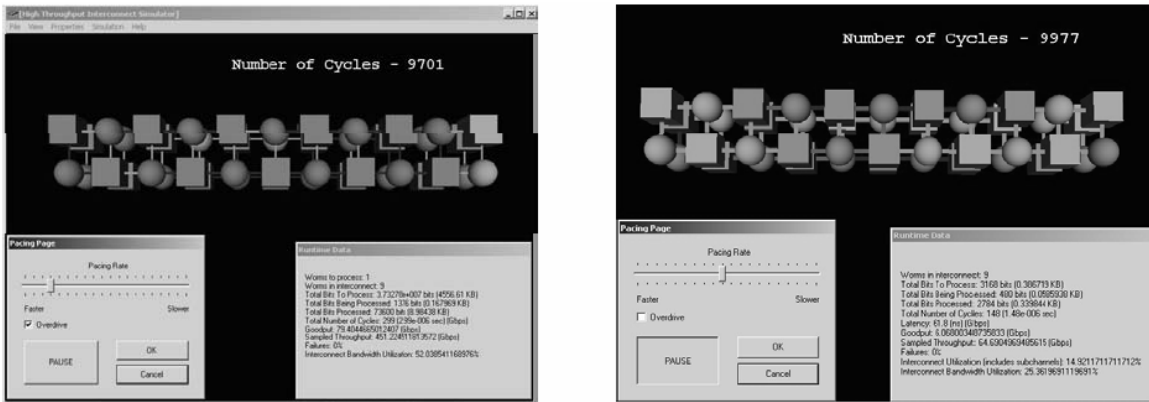


Figure A10: a) Simulation pacing overdrive b) Simulation pause on/off

Managing input/output files

In each simulation the user has the option to save/restore worms that are generated (user worms or random generated worms). If the user wishes to save the worms to allow future reuse of these worms, he/she can save it by checking the “Save Worms To File” box in the simulation page. In this case the user will be asked to input the file name

and the file will be saved into the simulator directory (Fig. A11b). The option to restore saves worms is provided in the “File” menu under “Load worms” (Fig. A11a).

The simulator generates multiple output files to provide a deeper insight into performance data and to facilitate simulation verification. The simulator generates four Excel-based files, stored in “Data Output” folder, at the end of each simulation.

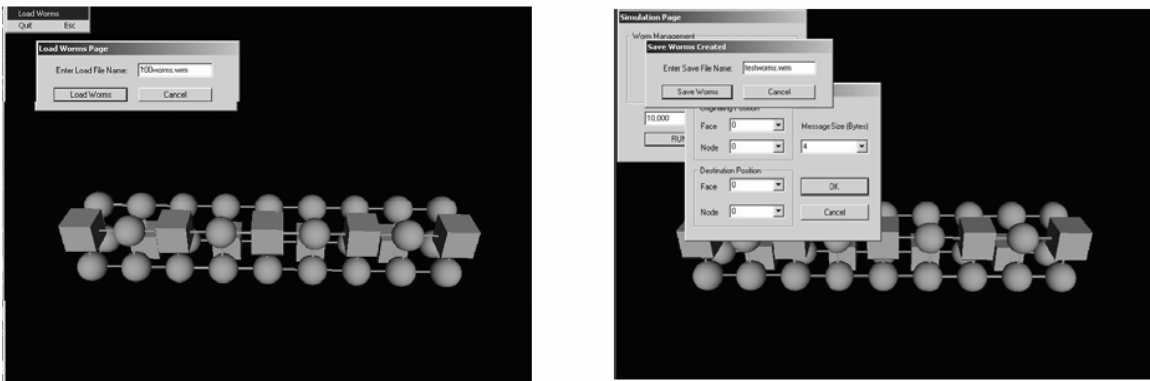


Figure A11: a) Import worms from file b) Export worms to a file

These files contain simulation records of: worm data, routing, hot-spots, routing accuracy, failure rate, distribution of worms waiting to be modeled, currently modeled worms and worms done, as well as general interconnect statistics, which includes bandwidth and resource utilization, interconnect user settings and comprehensive performance analysis. The files names are: modeling_data.xls, properties.xls, runtime_data.xls and worm_data.xls.

The “modeling_data.xls” file (Fig. A12a) provides the final simulation results. It includes statistical data such as the number of worms modeled, total number of bytes processed and the number of simulation cycles and performance metrics, for example, throughput, latency, utilization, routing accuracy and failure rate.

The file “properties.xls” (Fig. A12b) presents the interconnect physical properties and simulation settings. Interconnect physical settings include the interconnect type, channel width, virtual channels, sub-channels, scalability, latencies, and configuration. Simulation settings contain worm generation rate, maximum number of worms in the interconnect, and the range of worm's length.

| | A | B | C |
|----|--|---|-------------------|
| 1 | Data Name | | Data Value |
| 2 | | | |
| 3 | Worms Modeled | | 128 |
| 4 | Bytes Processed | | 70868 |
| 5 | Total Number of Cycles | | 613 |
| 6 | Sampled Throughput (Gbps) | | 412 |
| 7 | Latency (ns) | | 453.624 |
| 8 | Analytical Throughput (Gbps) | | 9.52368 |
| 9 | Goodput (Gbps) | | 298.344 |
| 10 | Number of Failures | | 248 |
| 11 | Failure Rate (%) | | 21.875 |
| 12 | Routing Accuracy (%) | | 81.3793 |
| 13 | Interconnect Resources Utilization (%) | | 57.1385 |
| 14 | Total Bandwidth Utilization (%) | | 31.0286 |

| | A | B | C |
|----|---------------------------|---|----------------|
| 1 | Property | | Setting |
| 2 | | | |
| 3 | Number of Cubes | | 3 |
| 4 | Channel Width | | 128 bits |
| 5 | Number of Subchannels | | 4 |
| 6 | Sub-channel width | | 32 bits |
| 7 | Virtual Channels | | NO |
| 8 | Virtual Channel Size | | 1024 bytes |
| 9 | Minimum Message Size | | 40 bytes |
| 10 | Maximum Message Size | | 1024 bytes |
| 11 | Propagation Delay | | 100 (ps) |
| 12 | Switching Delay | | 3 (ns) |
| 13 | Routing Delay | | 3000 (ps) |
| 14 | Max Worms in Interconnect | | 1.00E+06 |
| 15 | Worm Generation Rate | | 90 |
| 16 | Cube Type | | 4 Array 3 Cube |
| 17 | Cube Configuration | | 5 |

Figure A12: a) Worm modeling data b) Simulation properties

The file “Runtime_data.xls” (Figs. A13-A15) gives per-cycle performance parameters. This file shows how performance metrics are changing values from one simulation cycle to another. For example, it can indicate the distribution of worms (controlled by the worm manager) within the interconnect. The user can view at which simulation cycle the interconnect was saturated and the min/max values of each performance parameter. In Figs. A14 and A15 the following fields: sample throughput, latency, failure rate and route accuracy are set to 0 since the worm manager does not collect those performance metrics while the interconnect is being populated with worms. This is typically happening in the first 20-30 cycles of simulation. Sampled throughput is

showing, N/A values, since it has not sampled the interconnect throughput yet. It is depended upon the sampling rate, set by the user, before simulation has started.

| | A | B | C | D | E | F | G | H |
|----|---------------------|---|----------------------|-----------------------|----------------------|---------------------------|----------------------------|------------------------|
| 1 | Cycle Number | | Worms Waiting | Worms Modeling | Worms Modeled | Worms Waiting Size | Worms Modeling Size | Bytes Processed |
| 2 | | | | | | | | |
| 3 | 1 | | 254 | 2 | 0 | 140356 | 8 | 0 |
| 4 | 2 | | 252 | 4 | 0 | 140340 | 24 | 0 |
| 5 | 3 | | 238 | 18 | 0 | 140268 | 96 | 0 |
| 6 | 4 | | 227 | 29 | 0 | 140152 | 208 | 4 |
| 7 | 5 | | 218 | 38 | 0 | 140000 | 344 | 20 |
| 8 | 6 | | 193 | 63 | 0 | 139748 | 556 | 60 |
| 9 | 7 | | 193 | 63 | 0 | 139504 | 724 | 136 |
| 10 | 8 | | 183 | 73 | 0 | 139240 | 856 | 268 |
| 11 | 9 | | 182 | 74 | 0 | 138964 | 952 | 448 |
| 12 | 10 | | 174 | 82 | 0 | 138640 | 1076 | 648 |
| 13 | 11 | | 174 | 82 | 0 | 138424 | 1076 | 864 |
| 14 | 12 | | 174 | 82 | 0 | 138152 | 1128 | 1084 |
| 15 | 13 | | 172 | 84 | 0 | 137844 | 1212 | 1308 |

Figure A13: Worm run-time data

| | I | J | K | L | M | N | O |
|----|----------------------------------|---------------------|-------------------------------------|-----------------------|----------------------|---------------------------|-------------------------|
| 1 | Sampled Throughput (Gbps) | Latency (ns) | Analytical Throughput (Gbps) | Goodput (Gbps) | Badput (Gbps) | Number of Failures | Failure Rate (%) |
| 2 | | | | | | | |
| 3 | N/A | 0 | 0 | 0 | 20.6452 | 0 | 0 |
| 4 | N/A | 0 | 0 | 0 | 61.9355 | 0 | 0 |
| 5 | N/A | 0 | 0 | 0 | 247.742 | 0 | 0 |
| 6 | N/A | 0 | 0 | 2.58065 | 536.774 | 0 | 0 |
| 7 | N/A | 0 | 0 | 10.3226 | 887.742 | 0 | 0 |
| 8 | N/A | 0 | 0 | 25.8065 | 1434.84 | 0 | 0 |
| 9 | N/A | 0 | 0 | 50.1382 | 1868.39 | 0 | 0 |
| 10 | N/A | 0 | 0 | 86.4516 | 2209.03 | 0 | 0 |
| 11 | N/A | 0 | 0 | 128.459 | 2456.77 | 0 | 0 |
| 12 | N/A | 0 | 0 | 167.226 | 2776.77 | 0 | 0 |
| 13 | N/A | 0 | 0 | 202.698 | 2776.77 | 0 | 0 |
| 14 | N/A | 0 | 0 | 233.118 | 2910.97 | 0 | 0 |
| 15 | N/A | 0 | 0 | 259.653 | 3127.74 | 0 | 0 |

Figure A14: Worm run time data – cont.

| | P | Q | R |
|----|-----------------------------|---|--|
| 1 | Routing Accuracy (%) | Sampled Interconnect Resources Utilization (%) | Sampled Total Bandwidth Utilization (%) |
| 2 | | | |
| 3 | -1.#ND | 1.78571 | 0.446429 |
| 4 | -1.#ND | 4.46429 | 1.33929 |
| 5 | -1.#ND | 17.8571 | 5.35714 |
| 6 | -1.#ND | 31.25 | 11.6071 |
| 7 | -1.#ND | 44.6429 | 19.1964 |
| 8 | -1.#ND | 57.1429 | 31.0268 |
| 9 | -1.#ND | 66.0714 | 40.4018 |
| 10 | -1.#ND | 76.7857 | 47.7679 |
| 11 | -1.#ND | 81.25 | 53.125 |
| 12 | -1.#ND | 83.9286 | 60.0446 |
| 13 | -1.#ND | 84.8214 | 60.0446 |
| 14 | -1.#ND | 87.5 | 62.9464 |
| 15 | -1.#ND | 89.2857 | 67.6339 |

Figure A15: Worm run time data – cont.

The file “Worm_data.xls” presents detailed information about each modeled worm (Figs. A16-A18). Each worm has an ID (worm number), length (in bits), source (initial position) and destination (final position). Based on its source and destination addresses the simulator calculates the shortest path(s) that the worm can take (column: Best Case Number of Links).

The column: Number of Links on Successful Run, is the actual number of links (channels) the worm took in order to reach destination. This number is affected by network traffic load. Routing accuracy is calculated by taking the ratio between these two columns. Throughput and latency are given in two forms: best case and actual recorded values. The best case values are based on the assumption that the worm is taking the shortest path. The actual values of throughput and latency are based on the sampled values recorded at different simulation intervals.

| | A | B | C | D | E | F |
|----|-----------|---|-------------------|---------------------|----------------------------|----------------------------|
| 1 | Worm Name | | Number of Packets | Message Size (bits) | Initial Position | Final Position |
| 2 | | | | | | |
| 3 | Worm 6 | | 18 | 576 | f[0] n[7] p[MESSAGE_PORT] | f[2] n[14] p[MESSAGE_PORT] |
| 4 | Worm 23 | | 16 | 512 | f[3] n[13] p[MESSAGE_PORT] | f[2] n[4] p[MESSAGE_PORT] |
| 5 | Worm 37 | | 17 | 544 | f[0] n[1] p[MESSAGE_PORT] | f[2] n[6] p[MESSAGE_PORT] |
| 6 | Worm 56 | | 31 | 992 | f[2] n[2] p[MESSAGE_PORT] | f[2] n[14] p[MESSAGE_PORT] |
| 7 | Worm 3 | | 43 | 1376 | f[2] n[7] p[MESSAGE_PORT] | f[0] n[10] p[MESSAGE_PORT] |
| 8 | Worm 88 | | 18 | 576 | f[1] n[4] p[MESSAGE_PORT] | f[2] n[2] p[MESSAGE_PORT] |
| 9 | Worm 8 | | 47 | 1504 | f[3] n[2] p[MESSAGE_PORT] | f[0] n[4] p[MESSAGE_PORT] |
| 10 | Worm 42 | | 29 | 928 | f[0] n[8] p[MESSAGE_PORT] | f[2] n[11] p[MESSAGE_PORT] |
| 11 | Worm 15 | | 54 | 1728 | f[2] n[4] p[MESSAGE_PORT] | f[3] n[15] p[MESSAGE_PORT] |
| 12 | Worm 2 | | 59 | 1888 | f[2] n[7] p[MESSAGE_PORT] | f[3] n[5] p[MESSAGE_PORT] |
| 13 | Worm 72 | | 49 | 1568 | f[2] n[0] p[MESSAGE_PORT] | f[0] n[1] p[MESSAGE_PORT] |
| 14 | Worm 4 | | 67 | 2144 | f[2] n[13] p[MESSAGE_PORT] | f[2] n[14] p[MESSAGE_PORT] |
| 15 | Worm 70 | | 50 | 1600 | f[0] n[10] p[MESSAGE_PORT] | f[0] n[7] p[MESSAGE_PORT] |

Figure A16: Modeled worms data

| | G | H | I | J |
|----|------------------------------|--|---|----------------------------------|
| 1 | Total Modeling Cycles | Number of Links on Successful Run | Links on Successful Run Excluding Virtual Channels | Best Case Number of Links |
| 2 | | | | |
| 3 | 23 | 5 | 5 | 5 |
| 4 | 20 | 4 | 4 | 4 |
| 5 | 22 | 5 | 5 | 3 |
| 6 | 35 | 4 | 4 | 4 |
| 7 | 48 | 5 | 5 | 5 |
| 8 | 21 | 3 | 3 | 3 |
| 9 | 52 | 5 | 5 | 5 |
| 10 | 51 | 7 | 7 | 5 |
| 11 | 60 | 6 | 6 | 6 |
| 12 | 62 | 3 | 3 | 3 |
| 13 | 52 | 3 | 3 | 3 |
| 14 | 68 | 1 | 1 | 1 |
| 15 | 53 | 3 | 3 | 3 |

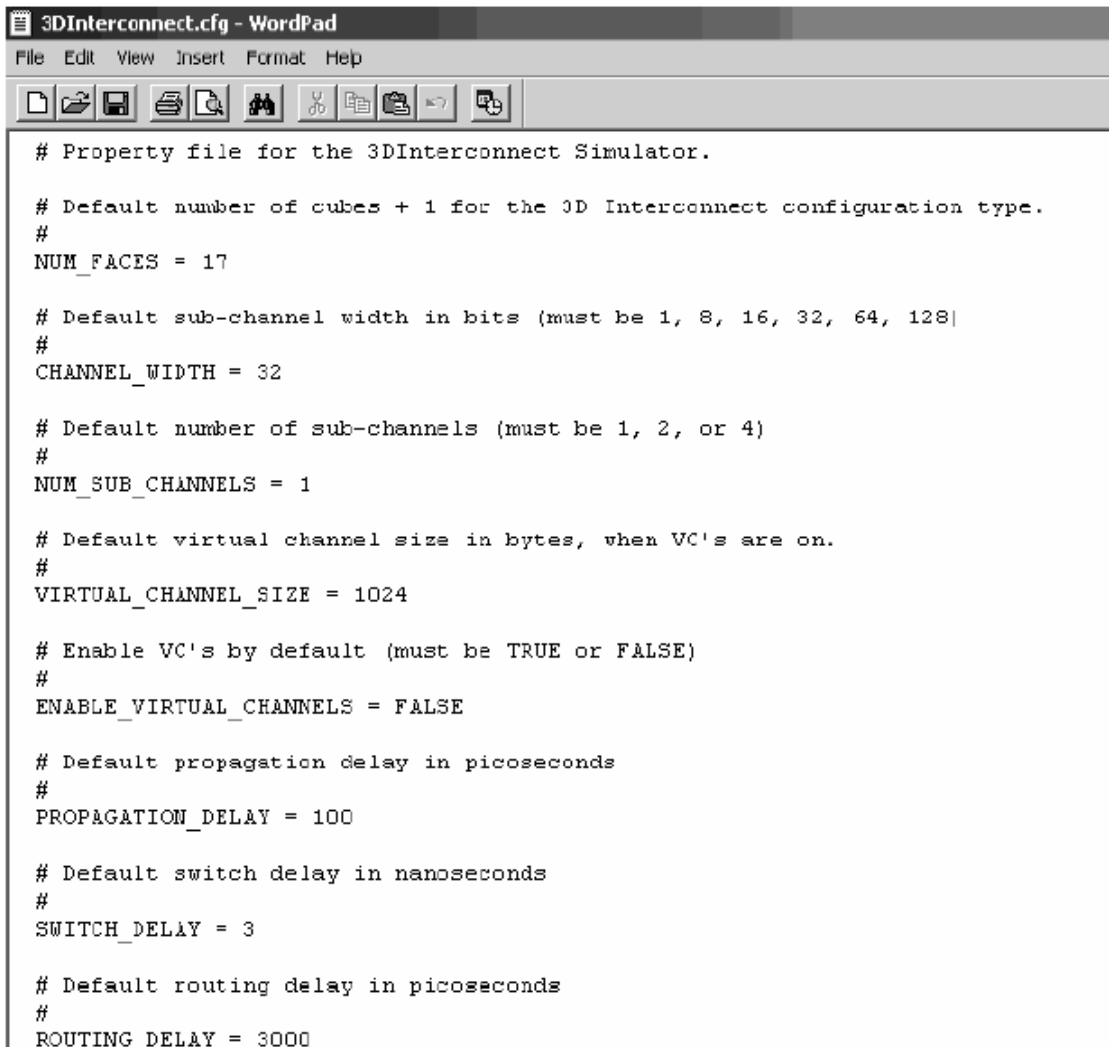
Figure A17: Modeled worms data - cont.

| | K | L | M | N | O | P |
|----|-----------------------------|---------------------------|-------------------------------|------------------------------------|---------------------|--------------------------|
| 1 | Routing Accuracy (%) | Number of Failures | Best Case Latency (ns) | Best Case Throughput (Gbps) | Latency (ns) | Throughput (Gbps) |
| 2 | | | | | | |
| 3 | 100 | 0 | 83.2 | 6.92308 | 83.2 | 6.92308 |
| 4 | 100 | 0 | 70.9 | 7.22144 | 70.9 | 7.22144 |
| 5 | 60 | 0 | 67.9 | 8.01178 | 80.1 | 6.79151 |
| 6 | 100 | 0 | 117.4 | 8.44974 | 117.4 | 8.44974 |
| 7 | 100 | 0 | 160.7 | 8.56254 | 160.7 | 8.56254 |
| 8 | 100 | 0 | 71 | 8.11268 | 71 | 8.11268 |
| 9 | 100 | 0 | 173.1 | 8.68862 | 173.1 | 8.68862 |
| 10 | 71.4286 | 5 | 117.3 | 7.91134 | 129.5 | 7.16602 |
| 11 | 100 | 0 | 200.9 | 8.60129 | 200.9 | 8.60129 |
| 12 | 100 | 0 | 198.1 | 9.53054 | 198.1 | 9.53054 |
| 13 | 100 | 0 | 167.1 | 9.3836 | 167.1 | 9.3836 |
| 14 | 100 | 0 | 210.7 | 10.1756 | 210.7 | 10.1756 |
| 15 | 100 | 0 | 170.2 | 9.40071 | 170.2 | 9.40071 |

Figure A18: Modeled worms data - cont.

The simulator imports two configuration files (“3DInterconnect.cfg” and “3DInterconnectSimulation.cfg”) which contain the default values of system parameters required to perform simulation (their default values). Before starting a new simulation, the user can change those parameters by changing the corresponding values in the configuration file. After the configuration change is saved, the simulator will update its default values with the new ones. The interconnect configuration file (Figs. A19-A20) contains the interconnect default values of physical properties, such as channel width, number of sub-channels, delays, generation rate, etc. The simulator configuration file

(Fig. A21) contains the values of simulation parameters. For example, pacing rate (in cycles) and initial number of simulation cycles.



```
# Property file for the 3DInterconnect Simulator.

# Default number of cubes + 1 for the 3D Interconnect configuration type.
#
NUM_FACES = 17

# Default sub-channel width in bits (must be 1, 8, 16, 32, 64, 128)
#
CHANNEL_WIDTH = 32

# Default number of sub-channels (must be 1, 2, or 4)
#
NUM_SUB_CHANNELS = 1

# Default virtual channel size in bytes, when VC's are on.
#
VIRTUAL_CHANNEL_SIZE = 1024

# Enable VC's by default (must be TRUE or FALSE)
#
ENABLE_VIRTUAL_CHANNELS = FALSE

# Default propagation delay in picoseconds
#
PROPAGATION_DELAY = 100

# Default switch delay in nanoseconds
#
SWITCH_DELAY = 3

# Default routing delay in picoseconds
#
ROUTING_DELAY = 3000
```

Figure A19: Interconnects configuration file

```

# Default number of samples to take
#
NUMBER_OF_REAL_TIME_SAMPLE_POINTS = 1000000

# Default number of cycles between sample points
#
CYCLES_BETWEEN_SAMPLE_POINTS = 1

# Default cycle to start sampling
#
FIRST_REAL_TIME_SAMPLE_CYCLE = 1

# Default minimum message size in bytes
#
MINIMUM_MESSAGE_SIZE = 4

# Default maximum message size in bytes
#
MAXIMUM_MESSAGE_SIZE = 1048576 # 1 MB

# Rate at which the scheduler prefers to send a worm into the interconnect.
# Must be 1-100.
#
WORM_GENERATION_RATE = 90

# Default directory to place excel spreadsheets
#
EXCEL_DIRECTORY = Data Output

```

Figure A20: Interconnects configuration file - cont.

```

# Properties for the 3D Interconnect Simulation

# The GUI will draw a cycle update in the time PACING_RATE / PACING_FACTOR
# Should be a number between 100 and 2000 with a step size of 100
# (i.e. 100, 200, 300, ...). The smaller the number, the faster the run.
#
PACING_RATE = 1000

# The GUI will draw a cycle update in the time PACING_RATE / PACING_FACTOR
#
PACING_FACTOR = 1

# Default value that appears for the "Run" ("r") option for number of
# cycles to model. Number between 0 and 10000
#
DEFAULT_NUMBER_OF_CYCLES_TO_RUN = 10000

```

Figure A21: Simulator configuration file - cont.

LIST OF REFERENCES

1. W. J. Dally, A. Singh, B. Towels, and A. K. Gupta, "Globally Adaptive Load-Balanced Routing on Tori", *Computer Architecture Letters*, vol.3, Mar. 2004.
2. J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks", *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320-1331, December 1993.
3. A. Agarwal, "Limits on Interconnection Network Performance", *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398-412, 1991.
4. J. H. Kim, "Planar Adaptive Routing (PAR): Low-cost Adaptive Networks for Multiprocessors", *MS Thesis*, University of Maryland College Park, 1990.
5. P. Berman, L. Gravano, G. Pifarre, and J. Sanz, "Adaptive Deadlock and Livelock Free Routing with All Minimal Paths in Torus Networks", *IEEE Transactions on Parallel and Distributed Systems*, vol. 5, issue 12, pp. 1233-1251, Dec. 1994.
6. W. J. Dally, "Fine-grain Message-Passing Concurrent Computers", *Proceedings of the Third Conference on Hypercube Computers*, pp. 2-12. ACM press, January 1988.
7. J. Y. Ngai and C. L. Seitz. "A Framework for Adaptive Routing in Multicomputer Networks", *Proceedings of the 1st Annual ACM Symp. on Parallel Algorithms and Architectures*, pp. 1-9. ACM Press, June 1989.
8. L. M. Ni and C. J. Glass, "The Turn Model for Adaptive Routing", *Proceedings of the 19th annual international symposium on Computer architecture*, pp. 278-287, 1992.

9. W. J. Dally, "Express Cubes: Improving the Performance of k-ary n-cube Interconnection Networks", *IEEE Transactions on Computers*, vol. 40, issue 9, pp. 1016-1023, 1991.
10. Grübl Andreas, "Distributed HAGEN The Hardware side", Kirchhoff Institute of Physics.(web:www.kip.uni-heidelberg.de/vision/public/techreport_gruebl.pdf)
11. Jantsch Axel, "Communication Performance in Network on Chips", Royal Institute of Technology, Stockholm, November 2003. (web: <http://www.imit.kth.se/courses/2B1447/NetworkOnChips.pdf>)
12. W. J. Dally, "Performance analysis of k-ary n-cube interconnection networks", *IEEE Transactions on Computers*, vol. 39, no. 6, pp. 775-785, 1990.
13. K. Hwang, *Advanced Computer Architecture*, McGraw-Hill, 1993.
14. J. Zalewski, *Advanced Multimicroprocessor Bus Architectures*, IEEE Computer Society Press, 1995.
15. W. J. Dally and H. Aoki, "Deadlock-free Adaptive Routing in Multicomputers Networks using Virtual Channels", *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, Issue 4, April 1993, pp. 466-475.
16. A. Folkestad and C. Roche, "Deadlock probability in unrestricted wormhole routing networks", *Proceedings of the International Conference on Communications (ICC)*, pp. 1401-1405, 1997.
17. W. J. Dally, "Virtual Channel Flow Control", *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-205, 1992.
18. Dally, W. J. and Towles, B., "Route packets, not wires: on-chip interconnection networks", *Proceedings of the Design Automation Conference*, pp. 684 - 689, 2001.

19. S. Scott and G. Thorson, "The Cray T3E Network: Adaptive Routing in a High Performance 3D Torus", *Proc. of Hot Interconnects IV*, Stanford University, August 1996.
20. Cisco corp., "Caltech and Partners Build a High-speed Network for Global High-energy Physics Collaboration", customer success story, 2005.
21. C. L. Seitz, "The cosmic cube", *ACM Trans. on Communications*, pp. 23-33, vol. 1, 1985.
22. G. Ming Chiu, "The Odd-Even Turn Model for Adaptive Routing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp. 729-738, 2000.
23. O. Lysne, "Deadlock Avoidance for Switches based on Wormhole Networks", *Proceedings of the Annual International Conference of Parallel Processing*, pp. 68 - 74, 1999.
24. C. Mick, R. Johnson, and P. Kumar, "Common switch interface for fabric independence and scalable switching", IEEE 802 Plenary Tutorials, 1998.
25. O. Lysne, "Deadlock avoidance for switches based on wormhole networks", *Proceedings of the International Conference on Parallel Processing*, September 1999.
26. B. Davis and B. Jacob, "DDR2 and low latency variants", *Proc. of the Memory Wall Workshop*, at the International Symposium on Computer Architecture, 2000.
27. W. J. Dally and C. Seitz, "The Torus Routing Chip", *Journal of Distributed Computing*, vol. 1, no. 3, pp. 187-196, 1986.
28. W. J. Dally, *Message-Passing Concurrent Computers*, Chapter 7. Addison-Wesley.

29. Cypress Semi. Corp., “Quad data rate (QDR) SRAM clocking scheme”, White paper, 2000.
30. P. Kermani and L. Kleinrock, “Virtual Cut-through: A New Computer Communications Switching Technique”, *Computer Networks*, vol. 3, no. 4, pp. 267-286, 1979.
31. S. Konstantinidou and L. Snyder, “Chaos Router: Architecture and Performance”, *Proceedings of the International Symposium on Computer Architecture*, pp. 212-221, 1991.
32. D. Linder and J. Harden, “An Adaptive and Fault Tolerant Wormhole Routing Strategy for k-ary n-cubes”, *IEEE Transactions on Computers*, vol. 40, no. 1, pp. 2-12, January 1991.
33. Rambus Inc., “Rambus technologies for the network communications market”, White paper, Jan 2001.
34. Cypress Semi. Corp., “Interfacing the QDR to the Delta39K”, White paper, July 10, 2001.
35. Y. Zhang, “Microstrip-multilayer delay line on printed-circuit board”, Technical Report, University of Nebraska, Lincoln, April, 2003.
36. P. Sassone, “Commercial trends in off-chip communication”, Technical Report, Georgia Institute of Technology, May 2003.
37. D. Halliday, “The evolution of mezzanine modules for next-generation telecom architectures”, *CompactPCI-Systems*, June 2003.

38. P. Gupta, S. Lin, and N. McKeown, "Routing lookups in hardware at memory access speeds", *Proceedings of IEEE INFOCOM*, pp. 1240-1247, San Francisco, CA, April 1998.
39. N. Shah, "Understanding Network Processors", *M.S. thesis*, UC. Berkeley, 2001.
40. T. Kocak and J. Engel, "A survey on network processors", *Technical Report*, School of EECS, Univ. of Central Florida, Sept., 2002. (web: <http://www.cs.ucf.edu/~tkocak/TR/NPsurvey.ps>)
41. G. M. Chiu, "The odd-even turn model for adaptive routing", *IEEE Transactions on Parallel and Distributed Systems*, vol. 11, no. 7, pp.729-738, July 2000.
42. S. Konstantinidou and L. Snyder, The Chaos Router, *IEEE Transactions on Computers*, vol. 43, No. 12, December 1994.
43. A. Bar-Noy, P. Raghavan, B. Schieber and H. Tamaki, "Fast deflection routing for packets and worms", *Proceedings of the twelfth annual ACM symposium on Principles of distributed computing*, pp. 75-86, 1993.
44. A. Rădulescu and K. Goossens, "Communication Services for Networks on Chip", Philips Research Laboratories, 2002.
45. "Interconnects: Classification of Parallel Architectures", Lecture slides, Computer Architecture-Univ. of Erlangen-Nürnberg, Germany. (web: www3.informatik.uni-erlangen.de/Lehre/RA/SS2001/Skript/05a-interconn1.pdf)
46. A. Lebeck, "Interconnection Networks", Lecture slides, CPS220 - Advanced Computer Architecture - Duke University. (web: <http://www.cs.duke.edu/courses/fall01/cps220/lectures/icn1-6up.pdf>)

47. W. L. Bain and S. R. Ahula, "Performance Analysis of High-speed Digital Buses for Multiprocessing Systems", *Proceedings of the 8th annual symposium on Computer Architecture*, pp. 107-133, 1981.
48. M. O. Khaoua, "A Performance Model for Duato's Fully Adaptive Routing Algorithm in k-Ary n-Cubes", *IEEE Transactions on Computers*, vol. 48, no. 12, pp. 1297-1304, 1999.
49. W. J. Dally, "Virtual-Channel Flow Control", *IEEE Transactions on Parallel and Distributed Systems*, vol. 3, no. 2, pp. 194-199, 1992.
50. J. Kim and C. R. Das, "Hypercube Communication Delay with Wormhole Routing", *IEEE Transactions on Computers*, vol. 43, no. 7, pp. 806-813, 1994.
51. A. Agarwal, "Limits on Interconnection Network Performance", *IEEE Transactions on Parallel and Distributed Systems*, vol. 2, no. 4, pp. 398-412, 1991.
52. W. A. Najjar, A. Lagman, S. Sur and P. K. Srimani, "Analytical Models of Adaptive Routing Strategies", Department of Computer Science, Colorado State University, August 10, 1994.
53. K. Hwang, *Advanced Computer Architecture*, McGraw-Hill, 1993.
54. W. L. Bain and S. R. Ahuja, "Performance Analysis of High-Speed Digital Buses for Multiprocessing Systems", *Proceedings of the 8th annual symposium on Computer Architecture*, pp. 107-133, 1981.
55. Y. Zhang, "Microstrip-multilayer delay line on printed-circuit board", Technical Report, University of Nebraska, Lincoln, April, 2003.
56. S. R. Schach, *Classical and Object-Oriented Software Engineering*, 3rd edition, Irwin group, 1996.

57. S. Meyers, *Effective STL: 50 Specific Ways to Improve Your Use of the Standard Template Library*, Addison-Wesley, Boston, Mass., 2001.
58. D. G. Fritz, R. G. Sargent, "An overview of hierarchical control flow graph models", *Proceedings of the IEEE Simulation conference*, pp. 1347 - 1355, 1995.
59. "Introduction to the Standard Template Library", SGI, white paper, 2003 (web: http://www.sgi.com/tech/stl/stl_introduction.html)
60. M. Townsend, "The Singleton Design Pattern", Microsoft corp., MSDN library, Feb. 2002. (web:<http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbda/html/singletondespatt.asp>)
61. "Pure Virtual Functions and Abstract Classes", Microsoft corp., MSDN library, 2005. (web:http://msdn.microsoft.com/library/default.asp?url=/library/en-s/vccore98/html/_langref_pure_virtual_functions_and_abstract_classes.asp)
62. H. S. Azad and M. O. Khaoua, "A Simple Mathematical Model of Adaptive Routing in Wormhole k-ary n-cubes", *Proceedings of the 2002 ACM symposium on Applied computing*, pp. 835-839, 2002.
63. Y. M. Boura and C. R. Das, "Modeling Virtual Channel Flow in Hypercubes", *Proceedings of the First IEEE Symposium on High Performance Computer Architecture*, pp. 166-175, 1995.
64. D. Mayhew and V. Krishnan, "PCI express and advanced switching: Evolutionary path to building next generation interconnects", *Proceedings of the 11th Symposium on Hot Interconnects*, Stanford, CA, 2003.
65. Micron, 288MB CIO Reduced Latency (RDLRAM-II), white paper, (web: <http://www.RLDRAM.com>)

66. Test Procedures, March. 5, 2001. (web: <http://www.lightreading.com>)
67. T. Dinkelman, "Going to the max on bus efficiency", *EE Times UK*, Dec. 16, 2003. (web: <http://www.eetuk.com/story/OEG20031216S0015>)
68. "Experimental Design Runs at Five Times the Speed of Current Fastest Chips; Could Cut Power Consumption in Half", *IBM Research News*, Feb. 7, 2000. (web: www.research.ibm.com/resources/news/20000207_fact_circuits.shtml)
69. PCI Special Interest Group, "PCI local bus specification, revision 2.2", Dec. 1998.
70. Infiniband Trade Association, "Infiniband architecture specification, rev. 1.0", Oct. 2000 (web: <http://www.infinibandta.org>)
71. K. Marquardt, "Hitting the 10-Gbit mark with SPI-4.2", *CommsDesign*, Sept. 10, 2002. (web: www.commsdesign.com/design_corner/OEG20020910S0010).
72. HyperTransport Consortium, "HyperTransport technology: Simplifying system design", Oct. 2002 (web: <http://www.hypertransport.org>).
73. PCI Special Interest Group, "PCI express base specification rev. 1.0a", Apr. 2003. HyperTransport Consortium, "HyperTransport Technology Specifications", 2005. (web: <http://www.hypertransport.org/tech/index.cfm>)
74. RapidIO Trade Association, "RapidIO rev. 3", (web: <http://www.rapidio.org>)
75. AD8152 X-stream digital crosspoint switch data sheet, Analog Devices, Inc. (web: www.analog.com/Analog_Root/productPage/productHome/0,,AD8152,00.html)
76. Clearspeed Technology Inc., (web: <http://www.clearspeed.com>)
77. IBM PowerNP Network Processors, (web: http://www-3.ibm.com/chips/products/wired/products/network_processors.html)

78. Vitesse Semiconductor Corp., (web: <http://www.vitesse.com>)
79. Cisco Systems, Product Briefs and Press Releases, (web: <http://www.cisco.com>)
80. Mindspeed Technologies, (web: <http://www.mindspeed.com>)
81. Xelerated Packet Devices, (web: <http://www.xelerated.com>)
82. Agere Systems, (web: http://www.agere.com/enterprise_metro_access/network_processors.html)
83. Motorola C-5 Network Processors, (web: http://e-www.motorola.com/webapp/sps/site/prod_summary.jsp?code=C-5)
84. GlobespanVirata, Inc., (web: <http://www.virata.com/comprocessors.html>)
85. Broadcom Corp., (web: <http://www.broadcom.com>)
86. EZchip Technologies, (web: <http://www.ezchip.com>)
87. Juniper Networks, Inc., (web: <http://www.juniper.net>)
88. Paion Corp. (web: <http://www.paion.com>)
89. EZCHIP, “7-Layer packet processing : A Performance Analysis”, White paper, (web: http://www.ezchip.com/images/pdfs/ezchip_7layers.pdf)
90. Applied Micro Circuits Corp., (web: <http://www.amcc.com>)
91. J. W. Jung, R. Mudumbai, D. Montgomery, and H. K. Kahng, “Performance evaluation of two layered mobility management using mobile IP and session initiation protocol”, *Proceedings of the Global Telecommunications Conference*, Vol. 3, pp. 1190 - 1194, 2003.
92. R. Kornblit and E. Schwartzmann, “Multicast Protocols Evaluation in Wireless Domains”, Project report, Technion, Israel.

93. J. Hsu, S. Bhatia, M. Takai, R. Bagrodia, M. J. Acriche, "Performance of Mobile Ad Hoc Networking Routing Protocols in Realistic Scenarios", *Proceedings of the Military Communications Conference*, Vol. 2, pp. 1268 - 1273, 2003.
94. H. Wu, R. M. Fujimoto and G. Riley, "Experiences Parallelizing A Commercial Network Simulator", *Proceedings of the 2001 Winter Simulation Conference*, pp. 1353-1360, 2001.
95. X. Chang, "Network Simulations with OPNET", *Proceedings of the 1999 Winter Simulation Conference*, pp. 307-314, 1999.
96. C. H. Yeh, B. Parhami, and E. A. Varvarigos, "Multilayer VLSI Layout for Interconnection Networks", *Proceedings of the 2000 Int'l Conf. on Parallel Processing*, pp. 33-40, 2000.
97. A. DeHon, "Robust, high-speed network design for large scale multiprocessing", *Technical Report*, No. 1445, MIT, Sep. 1993.