Packet-Switched On-Chip FPGA Overlay Networks

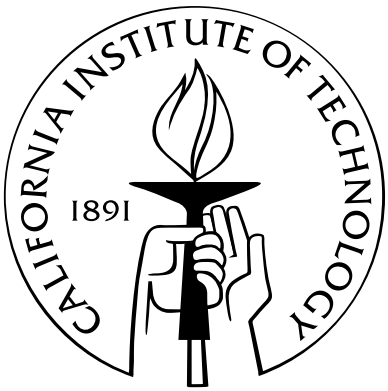
Thesis by

Nachiket Kapre

In Partial Fulfillment of the Requirements

for the Degree of

Master of Science



California Institute of Technology

Pasadena, California

2006

(Submitted May 31, 2006)

Acknowledgements

André DeHon has been my teacher, adviser, mentor and a parent away from home for the last three years of my life. He has been the driving force behind this research and has put up with endless mid-course corrections and somersaults during this work. This thesis is due to him.

Nikil Mehta joined this lab two years ago and I have thoroughly enjoyed working with him. His technical writing and presentation skills have helped make our work accessible and comprehensible to the community at large. I have learned a lot from him in these past two years.

This work emerged out of the lessons learn from the class project of the CS184b Computer Architecture course taught by André in the spring of 2005. Michael deLorimier and Raphael Rubin were the original TAs who provided tool support for the class project. Their insight has benefited this work immensely and this research owes a lot to these two. Henry Barnor and Michael Wilson helped craft the network hardware for this project, on Xilinx chips, and in recognition of their work they are now happily employed with Altera.

Discussions with Eylon Caspi, Kees Vissers, Steve Trimberger, Shepard Siegel, Stephen Neuendorffer, and Ian Eslick at the several Graph Machine retreats in Boston and Santa Barbara helped us frame the presentation of our research so as to appeal to the research community.

Gunjan, Rohit, Sukhada, Abhishek, Helia, Amir and Saket have been members of my extended family here in the United States. No amount of acknowledgements would do justice to the support all of these have provided me.

I would also like to thank a friend from my undergraduate years in India, Sumeet Kulkarni. Endless technical debates and arguments with him have helped shape the engineer inside me during those formative years.

Last, but certainly not the least, I want to thank my wonderful parents Aruna and Ganesh Kapre, my sibling Richa Kapre and my loving grandmothers Sumati Kapre and Sudha Shrikhande who continue to support, encourage and love me from thousands of miles away.

Abstract

As we scale to larger chip capacities, it becomes possible to map large, concurrent applications to programmable fabrics. These applications often have irregular and dynamic communication requirements. Packet-switched networks provide efficient implementations for such applications on these fabrics. In this research, we show how to engineer high-performance packet-switched on-chip networks and provide quantitative comparisons between different kinds of these networks. We analyse different network topologies and justify selection of topologies based on experimental results. We investigate packet-switched and time-multiplexed styles of routing and provide guidance on which style is appropriate for which application.

We can summarize the key contributions of this work as follows:

- We show how to engineer a low-overhead, high-performance, sample packet-switched overlay network with 8 PEs (single-size) that runs at 166 MHz and occupies $\approx$ 10K slices (30%) of a Xilinx Virtex-2 6000 FPGA device.
- We generate workloads for benchmarking our networks from real, communication-rich applications mapped to the GraphStep System Architecture [22] (ConceptNet Spreading-Activation, Sparse Matrix Vector Multiply, Bellman-Ford Shortest-Path) instead of relying on synthetic traffic.
- We evaluate the effects of topology scaling to 100s and 1000s of Processing Elements (PEs) under an FPGA cost model and demonstrate that the Butterfly Fat Tree (BFT) and Mesh topologies achieve comparable performance in all scenarios. Performance difference between the two topologies at any area is within 20% of the other.
- We compare the performance achieved by Directional and Bidirectional Meshes and find that as we scale to larger chip areas, Bidirectional Meshes provide consistently better performance across all applications.
- We further show that Virtual-Channel (VC) based mesh routing algorithms are unsuitable for these on-chip networks under the wiring-rich regime of FPGA substrates. They are as much as 10% worse than the simple mesh routing algorithms (Dimension Ordered and West Side First) at equivalent area.
- We measure area and performance of packet-switched and time-multiplexed overlay networks and show that packet switching outperforms time multiplexing at certain activity factors *i.e.* when the

set of messages drops below 50% of the maximum when mapped to a network with 100 PEs.

- We demonstrate that packet switching can be 5-10× better than time-multiplexing at small FPGA sizes. For most applications, time multiplexing is only about 1.5× better than packet switching. When routing a decomposed ConceptNet workload it can be as much as 3.5× better.

Contents

Acknowledgements **iii**

Abstract **iv**

1 Introduction **1**

1.1 The Problem 1

1.2 Concept of a Network-on-Chip 1

1.3 Communication Patterns 2

1.4 Outline 4

2 Prior Work **6**

2.1 Interconnection Networks 6

2.2 FPGA Packet-Switched NoCs 8

2.3 NoC Topologies 9

3 Background **10**

3.1 Flavors of Packet-Switching 10

3.2 Deadlock 11

3.3 Performance Analysis 11

3.3.1 Serialization 12

3.3.2 Network Bisection 12

3.3.3 Network Latency 13

4 Hardware **14**

4.1 Switch Design 14

4.2 Primitives 16

4.2.1 Split 16

4.2.2 Merge 17

4.3 Other Elements 18

5 Topology **20**
5.1 Limited Bisection Networks . . . 20
5.2 Ring . . . 21
5.3 Mesh . . . 21
5.3.1 Switch Architecture . . . 24
5.3.2 Routing Algorithms . . . 25
5.3.2.1 Deterministic . . . 25
5.3.2.2 Adaptive . . . 27
5.3.2.3 Virtual Channels . . . 27
5.4 BFT . . . 29
5.4.1 Routing Algorithm . . . 30

6 Applications **33**
6.1 ConceptNet Spreading Activation . . . 34
6.2 Sparse Matrix-Vector Multiply . . . 36
6.3 Bellman-Ford Shortest Path . . . 37
6.4 Application Characteristics . . . 38

7 Methodology **40**
7.1 Tool Infrastructure . . . 40
7.2 Cycle Accurate Simulator . . . 41
7.3 Area Model . . . 42
7.4 Latency Model . . . 44

8 Evaluation **48**
8.1 Selection of Buffer Depth . . . 48
8.2 Impact of Topology . . . 49
8.2.1 Effect of PE Scaling . . . 50
8.2.2 Effect of Area Scaling . . . 52
8.2.3 Performance of Virtual-Channel-based Meshes . . . 54
8.2.4 Performance of Directional and Bidirectional Meshes . . . 57
8.2.5 Performance of BFTs and Meshes . . . 57
8.2.6 Explaining Quality of Performance . . . 59
8.2.7 Effect of Application Communication Requirements . . . 59
8.2.8 Universal Topology . . . 63
8.3 Comparison with Time Multiplexing . . . 63
8.3.1 Effect of PE Scaling . . . 66

8.3.2 Effect of Area Scaling 67
8.3.3 Effect of Activity Factor 68
8.3.4 Effect of Multiple Graph Steps 71

9 Future Work **72**

10 Conclusions **74**

Bibliography **76**

List of Tables

1.1 Role of Various Communication Patterns 2

2.1 Summary of Prior Work in FPGA Network-on-a-Chip Designs 8

4.1 Area, Latency and Speed of 32-bit Packet-Switched Switching Primitives on a Xilinx Virtex-2 6000 -4 18

5.1 List of explored Topologies 32

6.1 Application Graphs 38

7.1 Area and Latency of Switchboxes of different topologies with 1-deep buffers with no wire pipelining 44

8.1 Ratio of Achieved Performance to Best Performance (Over All Areas and Applications) 65

List of Figures

2.1 ConceptNet Network I/O per Cycle vs. Network Size 8

3.1 Deadlock in a Packet-Switched Network 11
3.2 Network Bisection 12
3.3 Communication Time vs. PEs with respect to lowerbounds of a BFT $c = 1, p = 0.5$ for `gemat11` (SMVM, Section 6.2) 13

4.1 Conceptual Diagram of an N-input M-output switch 14
4.2 Conceptual Diagram of Split, Merge and TM-Merge Primitives 15
4.3 Conceptual Diagram of an N-input M-output switch composed using Splits and Merges 15
4.4 Internal Hardware Details of a 2-output Split 16
4.5 Internal Hardware Details of a 2-input Merge 17
4.6 Conceptual Diagram of Input and Output Blocks 19

5.1 Routing Function for the Ring 21
5.2 Ring Topology and a Ring Switch 22
5.3 Effect of Multiple Channels on Hardware Requirements of a Ring 22
5.4 Bidirectional Mesh Topology and a Bidirectional Mesh Switch 23
5.5 Directional Mesh Topology and a Directional Mesh Switch 24
5.6 Directional Mesh Island-Style Switch 25
5.7 Mesh Fast-XY Switch 26
5.8 Dimension Ordered Routing 26
5.9 Structural Diagram of an Arity-4 Bidirectional Mesh Switch DOR 27
5.10 West-Side-First Routing 28
5.11 Structural Diagram of an Arity-4 Bidirectional Mesh Switch WSF 28
5.12 Idea of Virtual Channels 28
5.13 Duato's Fully Adaptive Routing Function for the Bidirectional Mesh 30
5.14 Minimal Adaptive Routing Function for the BFT 31
5.15 BFT Topology and BFT Switches 31
5.16 Structural Diagram of BFT T and Π Switch 32

6.1 ConceptNet PE . . . 35
6.2 SMVM PE . . . 36
6.3 Bellman-Ford PE . . . 37

7.1 Logic-Circuit for Simulation . . . 42
7.2 Dual-Phase Simulation Algorithm : Phase-1 . . . 43
7.3 Dual-Phase Simulation Algorithm : Phase-2 . . . 43
7.4 FPGA Characterization Experiment . . . 45
7.5 Wire Lenghts in the BFT . . . 46
7.6 Comparing Worst-Case Latencies on the Mesh and the BFT . . . 47

8.1 Communication Time vs. PEs for `gemat11` (SMVM) with different buffer depths . . . 49
8.2 Zoom of Figure 8.1 . . . 49
8.3 Communication Time vs. Slices for `gemat11` (SMVM) with different buffer depths . . . 50
8.4 Communication Time vs. Area . . . 51
8.5 Communication Time vs. Area . . . 53
8.6 Actual Communication Time vs. PEs for `gemat11` (SMVM) . . . 53
8.7 Communication Time vs. PEs for non-VC and VC Meshes . . . 55
8.8 Ratio of Communication Time of non-VC and VC meshes vs. Area . . . 56
8.9 Ratio of Communication Time of Bidirectional Mesh/Directional Mesh vs. Area . . . 58
8.10 Ratio of Communication Time of Mesh/BFT vs. Area . . . 60
8.11 Lowerbound Communication Time vs. PEs for `gemat11` (SMVM) . . . 61
8.12 Ratio of Actual to Lowerbound Communication Time vs. PEs for `gemat11` (SMVM) . . . 61
8.13 Communication Time vs. Area . . . 62
8.14 Ratio of Actual Communication Time of Bidirectional Mesh W=1 DOR and BFT c=1 p=0.5 to Best Communication Time vs. Area . . . 64
8.15 Ratio of Time-Multiplexed/Packet-Switched Communication Time for Identical Topologies . . . 66
8.16 Communication Cycles vs. Area for Best Topologies . . . 67
8.17 Ratio of Time-Multiplexed/Packet-Switched Communication Time for Identical Area 69
8.18 Communication Cycles vs. Activity for Sample Topology with 256 PEs . . . 70
8.19 Activity Crossover vs. PEs . . . 70
8.20 Performance of Time-Multiplexing vs. Packet-Switching as a function of Graph Steps 71

Chapter 1

Introduction

1.1 The Problem

Communication is beginning to limit the performance of modern digital systems. In conventional ISA processors, communication requirements are handled implicitly through memory. However, memory bandwidth scales poorly and is becoming a major barrier to achieving high performance. In spatial architectures *i.e.* FPGAs, computation can be laid out in raw hardware where communication can be represented explicitly using wires between the program's operators. This allows us to exploit the full capability of the available silicon and potentially achieve higher performance. This increased performance, however, comes at the expense of increased wiring. Wiring accounts for a significant portion of the chip area, dictates its clock cycle period (*i.e.* fastest speed at which the chip can run) and consumes most of the its power. We can reduce wiring area requirement by time sharing these wiring resources. In an extreme form of timesharing, we permit only one pair of compute operators to send and receive messages at a time over a set of shared wires (*i.e.* a bus). This form of interconnect causes communication to be fully sequentialized, leading to poor performance. We need a solution that has modest area requirements while still delivering high performance for these communication dominated applications.

1.2 Concept of a Network-on-Chip

Network-on-Chip (NoC) is an economical and efficient solution for realizing the communication requirements of applications on a single chip. NoCs can be engineered to have less area than a spatially connected design and provide much better performance than a bus. Communication networks have been long been studied for multi-processor systems and supercomputers. With increasing silicon capacities, it has now become possible to map these systems efficiently onto single chips. The network itself is a collection of switches connected to each other using shared wiring channels. The compute operators or PEs are allowed to inject traffic into the network simultaneously at these switches.

Table 1.1: Role of Various Communication Patterns

Characteristics	**Configured**	**TM**	**PS**	**Circuit**
Know communication needs	early (compile time)		late (runtime)	
Communication predictability	high		low	
Communcation throughput compared to physical link throughput	>	<	<	<
PE-to-PE latency compared to packet length	n/a	n/a	>	<
Channel Utilization	high		low	
Switch Logic Area	low	low	high	high
Switch Memory Area	low	high	modest	low
Relative Latency	lowest	low	highest	moderate

The underlying cost model of the substrate influences the area efficiency and performance of these networks. We choose to investigate the mapping of these networks to FPGA substrates. We overlay our networks on top of FPGA logic to implement switching functions and use programmable FPGA routing resources for wiring the switches. This provides a cost model that has been relatively unexplored for implementing NoCs. FPGA overlay NoCs also offer the unique opportunity to quickly and easily reconfigure networks to create application-specific topologies. This also enables runtime flexibility and rapid design space exploration.

1.3 Communication Patterns

Applications have diverse communication requirements. Different styles of programmable interconnect are relevant for different communication requirements. Applications with dynamic communication requirements can be efficiently mapped to a specific style of NoCs called Packet-Switched Networks. We study Packet-Switched On-Chip Networks extensively in this thesis. We capture the nature of communication supported by a few commonly used programmable interconnects in Table 1.1. We discuss the characteristics of these different switching styles to motivate the application space where packet-switching is relevant.

Spatially Configurable Interconnect When the interconnection pattern among PEs is a static characteristic of the application (*i.e.* not data dependent) or highly predictable, and applications require that we move data between PEs at a rate **comparable to or higher than** the throughput of the primitive physical communication links, it is efficient to dedicate a communication channel to performing each PE-to-PE communication tasks by spatially configuring the physical links (*e.g.* traditional, FPGA configured switching). That is, each programmable switch in the network has local configuration state, and we set the state in the switches to provide direct paths between producing

and consuming PEs. Once set, the PE simply places data on its output port and the data propagates over the pre-allocated, configured path from the source to the sink with no additional delays beyond the physical switching delays. Consequently, data communication latency is minimized. Further, switches can be very simple with minimal state, minimizing the area required per switch. When the required PE-to-PE application bandwidth is higher than the bandwidth of a primitive communication channel, multiple channels can be configured in parallel to provide the required application bandwidth (*e.g.* multi-bit datapaths).

Circuit-Switched Interconnect When the interconnection pattern among PEs is **not known in advance or highly data dependent**, and hence unpredictable such that the PE-to-PE transfer rate can be significantly **lower than** the throughput of the primitive physical communication links, and the latency of the network is small compared to the data sent in a unit PE-to-PE transfer, it is efficient to circuit switch logical PE-to-PE communications over the physical communication channels. That is, rather than pre-configuring switches or switching behavior, a destination tag is attached to each data component. The switches dynamically accept these route requests and allocate a path to the destination if one is available. Since the switches must make online switching decisions they are more complicated, and hence larger and slower, than Spatially-Configured Interconnect and Time-Multiplexed Interconnect. Unlike Time-Multiplexed Interconnect no configuration memory is required in the switches; unlike Packet-Switched Interconnect no memory is required to provide data buffering in the network. Further, since switching decisions are made based only on instantaneous, local information, switch utilization is lower than Time-Multiplexed Interconnect whose configuration can be globally coordinated offline; to achieve a target network bandwidth, this means the circuit switched network will typically need more physical links than Time-Multiplexed Interconnect. Since connections are opened between source and sink and held open, data bundles of arbitrary length can be sent at the full data rate of the primitive channel with low additional latency once the connection is established. This means the network resources are dedicated to this connection during the setup of the link and the data transfer; if the network latency is long compared to the length of data transmission, this can result in inefficient utilization of the physical network links.

Packet-Switched Interconnect When the interconnection pattern among PEs is **not known in advance or highly data dependent**, and hence unpredictable such that the PE-to-PE transfer rate **can be significantly lower than** the throughput of the primitive physical communication links, and the latency of the network is large compared to the data sent in a unit PE-to-PE transfer, it is efficient to packet switch logical PE-to-PE communications over the physical communication channels. That is, rather than pre-configuring switches or switching behavior, a destination tag

is attached to each data component to create a packet. The switches dynamically accept packets and switch them along paths towards their destination as the paths become free. Queues with backpressure are used to accommodate resource limitations in the network. Since the switches must make online switching decisions, packet switches are more complicated, and hence larger and slower, than Spatially-Configured Interconnect and Time-Multiplexed Interconnect. Unlike Time-Multiplexed Interconnect no configuration memory is required in the switches; but, some memory is required to provide the data queues. Further, since switching decisions are made based only on instantaneous, local information, switch utilization is lower than Time-Multiplexed Interconnect whose configuration can be globally coordinated offline. However, when routing packets, we only need to support the instantaneous traffic demands; if the instantaneous traffic demands are very low compared to the potential traffic demands, the ability to route only the instantaneous traffic may result in a net reduction in route time despite the fact that physical resource utilization is lower than Time-Multiplexed Interconnect.

Time-Multiplexed Interconnect When the interconnection pattern among PEs is a static characteristic of the application (*i.e.* not data dependent) or highly predictable, and applications have many PE-to-PE connections that transfer data at a rate **significantly lower than** the throughput of the primitive physical communication links, it is efficient to statically schedule logical PE-to-PE communications over the physical communication channels. That is, each switch has an FSM or memory which can be programmed to give it a distinct switch configuration on each primitive switching cycle. This program is executed repeatedly (cyclic schedule) to route data among sources and sinks. Source-to-sink latency is low as data is routed from link-to-link without additional delays, but may be higher than Spatially Configured Interconnect communication since the switch timing must support a state change on every cycle. Switches can be simple, making switching area low. However, if the length of the communication cycle is long, the memory required to hold the switch schedule can become large and dominate the switching area.

For the kinds of small message applications we focus on here, the tens of clock cycles of network latency between PEs guarantees that Circuit-Switched Interconnect is not an appropriate solution. We defer the study of spatially configured networks for these applications for later. In this report, we focus on developing analytic area and time models to select wisely between packet-switched and to time-multiplexed networks. We hope to ground the trends of Table 1.1 into quantitative, empirical trade offs.

1.4 Outline

The thesis is organized into the following chapters

Chapter 2 reviews existing work in the broad area of NoCs, Packet-Switched FPGA Overlay Networks and Topology Selection for NoCs.

Chapter 3 provides an overview of different forms of packet-switched networks available and explains the performance metrics used in subsequent analysis.

Chapter 4 introduces the switching primitives used to build the Packet-Switched Networks and Chapter 5 explains how different topologies are supported using these primitives.

Chapter 7 describes the infrastructure used for our analysis and explains the area and latency models used in the experiments.

Chapter 8 shows data and analysis from our topology exploration studies and comparison with time-multiplexed networks.

Finally, Chapter 9 outlines additional avenues of research suggested in this thesis while Chapter 10 closes with a brief summary of the lessons learned from this study.

Chapter 2

Prior Work

2.1 Interconnection Networks

Interconnection networks have been extensively researched by the networking community since the days of the early telephone switching systems [10]. Networks for multiprocessing systems have also been studied relatively independently by the parallel computing community for decades [5,23,31,37, 56]. Packet-switching was the favored switching style used in these early multiprocessor networks. Consequently, packet-switching has evolved into several styles with different performance characteritics, *e.g.* store-forward, wormhole, virtual cut-through. Different network topologies (physical connectivity between network elements) were also explored to build these networks *i.e.* Hypercubes, Multistage Networks, lower-dimensional Meshes (2D and 3D), Tori, BFTs. The routing functions (that decide how packets reach their destinations) used in these networks also underwent continuous refinements *i.e.* Deterministic Routing, Adaptive Routing, Virtual Channels. Excellent surveys in interconnection networks can be found in [16, 23, 49].

Topology Early networks were primarily built using higher-dimensional topologies (*e.g.* Hypercubes). These networks also required the processors to perform routing of network traffic making them hard to program and greatly limiting performance. As technology evolved, it became possible to isolate routing responsibilities into a separate VLSI chip (*e.g.* Caltech Mesh Router [53, 54] used in the Caltech Mosaic multicomputer, Torus Routing Chip [14]). When realistic 2D VLSI packaging constraints were considered, lower-dimensional topologies (specifically BFTs) were shown to outperform the earlier higher-dimensional cousins in [37]. Subsequent network architectures (*i.e.* MIT J-Machine [48], Cray T3D [12]) were built using these principles. We continue to use these ideas to build on-chip networks.

Routing Algorithms Packet-switched networks were originally built using a store-forward style of switching that required complete packet buffering in the switches. It was possible to route on

these networks without being affected by deadlock given sufficient buffering (theoretically unlimited buffering). With the introduction of the Caltech Mesh Routing Chip [53], wormhole style of switching became very popular. Deadlock was an important issue on these networks and could not be ignored anymore. Hence, it became necessary to study and analyze different deadlock avoidance algorithms. Early worhmole networks used Dimension-Ordered Routing to achieve deadlock-free routing which was very restrictive. This was improved upon with the development of adaptive routing algorithms. Glass and Ni introduced the concept of Turn Model [24] that specified a less restrictive set of routing rules. Duato's concept of extended channel dependency graph [23] facilitated the development of fully adaptive algorithms under a general theoretical framework. These concepts influenced the design of several networks including the Cray T3E [51], and Alpha 21364 [44].

Network-on-Chip As VLSI capacities increased, it became possible to pack multiple functional elements onto a single chip. These functional elements were originally discrete chips on circuit boards that were integrated using board-level bus-based networks. When these systems migrated to single-chip systems, they inherited these buses. Several on-chip bus standards [3, 28, 60] are in popular use even today. But, performance of serial buses does not scale as chip sizes grow. We illustrate the effect of scaling chips sizes on the performance of bus traffic in the following example. Figure 2.1 shows network I/O messages per cycle as a function of PEs on our time-multiplexed network with no bandwidth limitations, given a ConceptNet (Section 6.1) communication load. In Figure 2.1 we see that as the the number of PEs increases, the number of messages which can be injected into the network increases significantly. Since a bus can handle only one network send or receive per cycle, communication will be fully serialized. At small numbers of PEs, the bus would only require 1–2× more cycles to route all messages than an unlimited network since very few messages are being pushed into the network. Most cycles would be dedicated to serialized processing at the PEs. At larger PE counts (> 500) PEs can inject more messages into the network (more of the communication graph is exposed to the network). The bus would require over 100 times as many cycles at these PE counts. Thus, a network capable of processing multiple messages simultaneously would help improve peroformance.

Today's silicon capacities allow networks other than buses to be implemented on single chips. Seitz [52] made the initial case for routing packets instead of wires for efficient on-chip communication. Dally [15] then demonstrated the feasibility of building an on-chip network with less than 10% area overhead. DiMicheli et al. [6] propose building on-chip networks with a layered OSI-like model for standardizing the interface and encapsulating the functionality of different elements of the network appropriately. These initial NoC efforts have laid the foundation for subsequent work in NoCs.

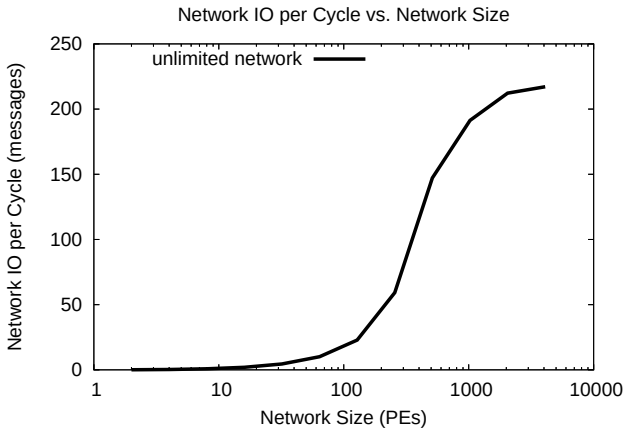


Figure 2.1: ConceptNet Network I/O per Cycle vs. Network Size

Table 2.1: Summary of Prior Work in FPGA Network-on-a-Chip Designs

NoC	**Freq** (MHz)	**Size**	**Chip**	**Switch**		
				Datawidth	**IO**	**Area**
IMEC [40]	40	3×3	Virtex2 Pro 40	16-bit	4	446 slices, 1 BRAM
Hermes [43]	25	2×2	Virtex2 1000 -4	8-bit	5	316 slices
LiPaR [55]	33	3×3	Virtex2 Pro 30 -6	8-bit	5	437 slices
Dimetalk [46]	100	–	Virtex2 -4	32-bit	4	450 slices, 1 BRAM

2.2 FPGA Packet-Switched NoCs

NoCs have been extensively studied under an ASIC cost model. FPGAs have been left relatively unexplored for this mapping. Recent increases in FPGA capacities have made it possibe to map NoCs to FPGAs. Some recent work has begun to examine NoCs under an FPGA overlay cost model [40,43,46,55]. We summarize a representative sample in Table 2.1. While these research efforts are the first to explore the FPGA design space, there are several limitations. Most of the designs reported in Table 2.1 are 2D bidirectional mesh topologies offering no quantitative reasoning behind this topology selection. We need to consider other topologies and evaluate their area and performance to help choose the appropriate network topology. None of these efforts provide scalability analysis to large chip sizes which are useful to help select the correct network configuration for future chips. Most of these studies use synthetic traffic making it hard to offer analysis on the impacts on network design. Marescaux *et al.* [40] reports application performance data only as a proof-of-concept. Finally, most of these networks run at speeds between 25 and 100 MHz which is far below peak FPGA speeds.

2.3 NoC Topologies

Several ASIC NoC research efforts have considered using various topologies for their designs, such as the mesh [32, 42], torus [15], fat-tree [1, 50], octagon [30], and star [34]. Pande *et al.* [50] compare different topologies for networks with 256 PEs. Additional work has attempted to provide a design methodology and automatic synthesis tools for evaluating and generating NoCs [26, 27, 45] . These projects typically evaluate scaling of topologies over 10s of interconnected PEs without evaluating tradeoffs with respect to other topologies. Murali *et al.* [45] provide a tool which explores several different application specific topologies, but they do not examine the effects of scaling of NoC topologies. Most of these existing NoC architectures borrow heavily from previously developed network architectures for parallel computing. On-chip networks have constraints that differ significantly from those of multiprocessors, such as two-dimensional layouts, quadratic unbuffered wire delays, and fewer pin constraints. Therefore, to design efficient NoCs it is essential to reexamine network architectures and topologies under an appropriate cost model.

Chapter 3

Background

In this section, we explain how data is routed over a packet switched network and how this gives rise to different flavors of packet-switched networks. We then consider the issues related to deadlock on these networks when routing data. We then setup metrics that enable us to compare the performance achieved by different networks when routing traffic.

3.1 Flavors of Packet-Switching

The main role of a packet-switched network is to transport packets from source to destination. A packet is the smallest logical unit of data that a PE can inject into the network. A packet consists of multiple flits (packets can also be only one flit). A flit is the smallest physical unit of data that is routed by the network. The first few flits of the packet are called header-flits. They contain the destination address (*i.e.* routing information) of the packet. The actual payload of the packet is contained in the remaining flits.

There are several forms of packet-switched networks that are distinguished by the manner in which they handle flits. In Store-Forward networks, all flits of a packet need to be received completely at the input of the switch before they can be routed to an output. This scheme requires a large amount of buffering when packets are very long and leads to very long network latencies. Given sufficient buffering, this network can be deadlock-free. In wormhole networks, packet can be routed as soon as the header is processed and buffering is avoided. Trailing flits follow the header-flit(s) along the same channels in a pipelined manner. If the header is blocked due to contention in the network, the rest of the flits stop advancing and occupy network resources. These resources stay occupied until the contention is resolved. Switches in a cut-through networks also route packets immediately upon header reception. The key difference is the amount buffering provided at the switches. Cut-through networks provide more buffering in the switches than wormhole networks to reduce the impact of contention on packet flow. This makes the switches more expensive than switches in a wormhole network in terms of area and potentially also increases packet latency.

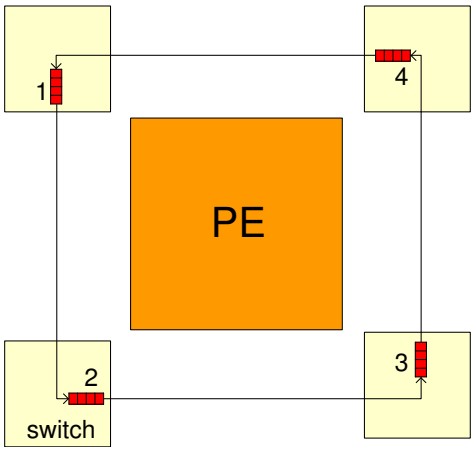


Figure 3.1: Deadlock in a Packet-Switched Network

3.2 Deadlock

Switches and PEs that form a packet-switched network are connected to each other by communication links (channels). The connectivity between elements of the network is defined by topology of the network. This connectivity may introduce physical cycles between the network elements. If we are not careful, packets routing over such a network may deadlock. In deadlock, packets cannot move any further and are consequently unable to reach their destinations. Packets participating in deadlock depend on other deadlocked packets to advance, in a cyclic fashion. We illustrate this in Figure 3.1. Packet 1 is waiting for Packet 2 to proceed, Packet 2 is waiting for Packet 3 to advance, Packet 3 is waiting for Packet 4 to go further while Packet 4 is waiting for Packet 1 to move ahead. All 4 of these packets are deadlocked. Deadlock can also be introduced due to cyclic dependencies between the PEs, but our compute model, described in Chapter 6, avoids this specific case. The key idea used when avoiding deadlock is to break the dependency cycles that may exist in the network. We can deadlock-free routing by either limiting the set of possible turns in each switch or by adding a set of virtual channels to each physical channel and ordering packet flow between these channels. We discuss these issues in greater detail in Section 5.3.2.

3.3 Performance Analysis

To understand the performance of different networks, we identify several quantitative network characteristics which bound the number of cycles required for communication. This allows us to measure the achieved performance of a given topology with respect to an optimal lowerbound.

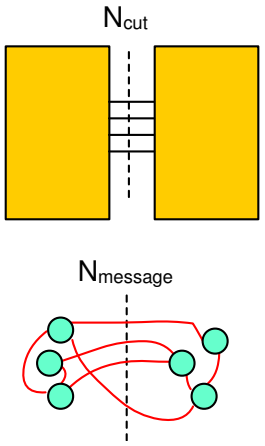


Figure 3.2: Network Bisection

3.3.1 Serialization

We engineer our PEs to handle an external input, output, or self message in one cycle. We can bound on number of cycles spent on incoming and outgoing messages as follows:

$$T_{input} = N_{input} + N_{self} \tag{3.1}$$

$$T_{output} = N_{output} + N_{self} \tag{3.2}$$

Under the compute model described in Chapter 6, our PEs can handle both an input and an output message per cycle.

$$T_{serialization} = \max\left(N_{input}, N_{output}\right) \tag{3.3}$$

$$\tag{3.4}$$

3.3.2 Network Bisection

Bisection of a network is defined as the number of wires crossing from one side of the chip to the other as shown in Figure 3.2. This bisection width limits the maximum number of messages that can travel across the chip in a given cycle. If the number of message bits is greater than the number of physical wires crossing the bisection, then communication must be serialized across the bisection:

$$T_{cut} = \left\lceil \frac{N_{messsage} \times Bits_{message}}{N_{topcut}} \right\rceil \tag{3.5}$$

The top-level bisection may not be the largest serial bottleneck in the network. Hence, we need

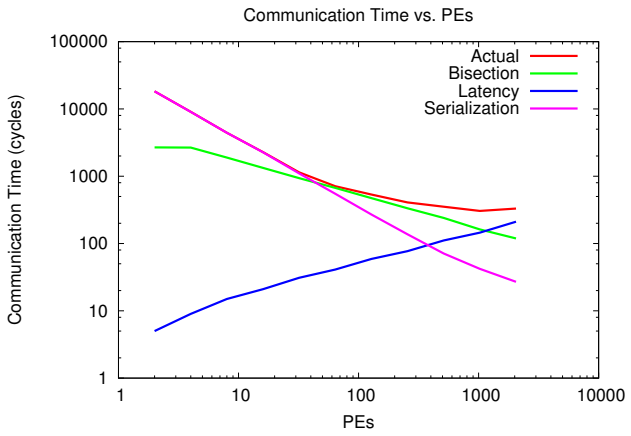


Figure 3.3: Communication Time vs. PEs with respect to lowerbounds of a BFT $c = 1, p = 0.5$ for `gemat11` (SMVM, Section 6.2)

to recursively bisect the network and identify the most limiting of cuts (T_{cut_i}):

$$T_{bw} = \max_{\text{all cuts } i} (T_{cut_i}) \tag{3.6}$$

3.3.3 Network Latency

If the network is sufficiently large, several cycles may be required to traverse the network from one end to the other:

$$T_{latency} = \max_{\text{all routes } i} (route_i) \tag{3.7}$$

Thus, the lower bound on performance of a topology is as follows:

$$T_{cycles} = \max (T_{serialization}, T_{bw}, T_{latency}) \tag{3.8}$$

As an illustrative example, in Figure 3.3 we show how the performance of a topology can be explained in terms of its lowerbounds. Here we plot actual performance (communication cycles required to route a workload) vs. number of PEs for a BFT $c = 1, p = 0.5$, in addition to lowerbounds. Initially the performance of the BFT is dominated by input and output serialization until 64 PEs. At low numbers of PEs most cycles are dedicated towards serialized processing at the PEs. As we increase the number of PEs the number of messages in the network increases (Figure 2.1). Since this is a limited bisection BFT, the performance is subsequently limited by bisection until 1024 PEs, after which it is latency dominated.

Chapter 4

Hardware

4.1 Switch Design

The primary function of a switch in a packet-switched network is to accept packets on inputs and route them to appropriate outputs as shown in Figure 4.1. The routing algorithm used in the switch decides the correct output for an incoming packet. This algorithm may cause multiple incoming packets to request routing to the same output. It may also have to choose between multiple outputs for a given packet, all of which are available. This can happen in the adaptive set of routing algorithms that attempt to provide as many possible routing choices to a packet as possible without getting deadlocked. The general problem of assigning outputs to input packets is a bipartite matching problem locally within the switch. Designing a switch to implement this algorithm with a large number of inputs and outputs at high speed is non-trivial. We want the entire packet-switched network to run at high-speed. Our pipelined PEs run at a speed close to the maximum possible speed of the on-chip FPGA memories (*200 MHz out of 250 MHz*). We expect the network to run at comparable speeds to prevent system performance from being limited by the network. Hence, we choose to design the the network to run at 200 MHz on commodity FPGAs. To achieve this performance target we simplify the logic for output and input selection. We compose the switch using a cascade of two basic primitives, a split and a merge, shown in Figure 4.2. This

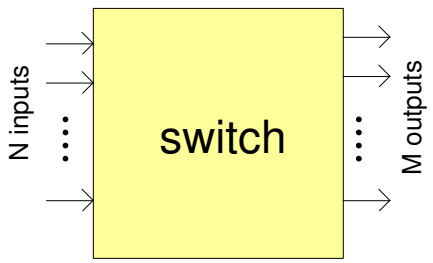


Figure 4.1: Conceptual Diagram of an N-input M-output switch

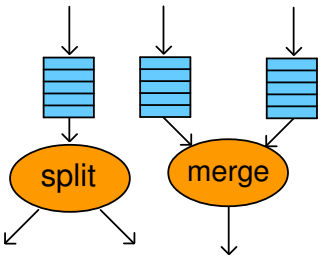


Figure 4.2: Conceptual Diagram of Split, Merge and TM-Merge Primitives

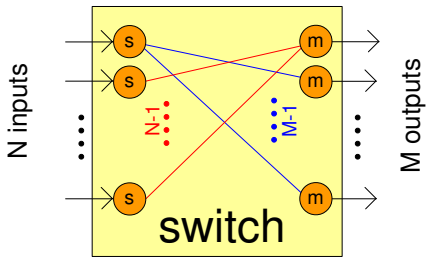


Figure 4.3: Conceptual Diagram of an N-input M-output switch composed using Splits and Merges

allows the individual primitives to be pipelined independently. The switch that is composed out of these primitives then continues to run at the target frequency. For the simple switch shown in Figure 4.1, we show a split-merge based implementation in Figure 4.3. An additional benefit of using these primitives is the simplicity of composing switches for different topologies. Each topology has a unique connectivity pattern that can be easily represented as a network of splits and merges. We then specify this pattern for each topology to implement the switch as required. Next, we look at issues that help motivate design requirements for these primitives.

Synchronization We operate the interfaces between the primitives based on producer-consumer synchronization. More specifically, we use the Tagged Data-Presence and Queues with Back-pressure design patterns [19]. This allows packet transfers between the primitives to be negotiated smoothly. Data-Presence signals are generated by the producer to inform the consumer about availability of valid packet data. The Back-Pressure signals are generated by the consumer to inform the producer whether it is ready to process data.

Buffering A merge may receive packets on multiple inputs simultaneously. A split input may receive a packet wanting to route to an output that is currently unavailable. These blocked packets need to be handled appropriately. We provide buffering at the inputs to temporarily store the packets until the required resources become available again. Depending on how many flits a packet contains, a buffer depth of 1 might make it either a wormhole network (multiple flits per packet) or a

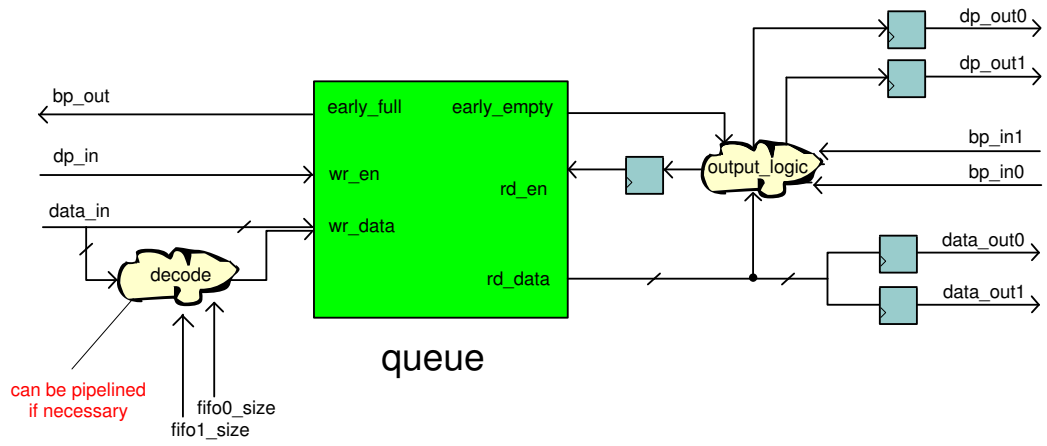


Figure 4.4: Internal Hardware Details of a 2-output Split

cut-through network (one flit per packet). At larger buffer depths, the network becomes an instance of a cut-through network. Buffer size must be chosen judiciously since it directly influences the area required by the switch. Buffering also has its disadvantages. Head-of-line blocking may increase the latency experienced by a packet as it moves through the network. This happens when a packet at the head of the buffer is waiting on a blocked output resource. Rest of the packets in the buffer are then forced to wait for the head packet to get routed first before they can move. Thus, buffer-sizing is an important design criteria that we'll revisit in Section 8.1.

4.2 Primitives

4.2.1 Split

The split primitive has one input, multiple outputs and a routing function which selects one of the outputs for the incoming packet. We show the internal hardware details of a 2-output split primitive in Figure 4.4. A split primitive is composed of input logic to decode the destination address, a FIFO to buffer packets and output logic to determine the correct output port for the packet. We choose to simplify output logic by computing the routing decision as soon as the packet is received at the inputs. For deterministic routing functions, the decision logic is usually simple and straightforward. For adaptive routing functions, we may need to provide additional congestion information about the outputs. For these cases, we can pipeline the routing logic as required. However, we need to ensure that the input backpressure signal is generate equally early. The output interface logic is also simple and can be easily packed into 1–2 levels of 4-LUTs.

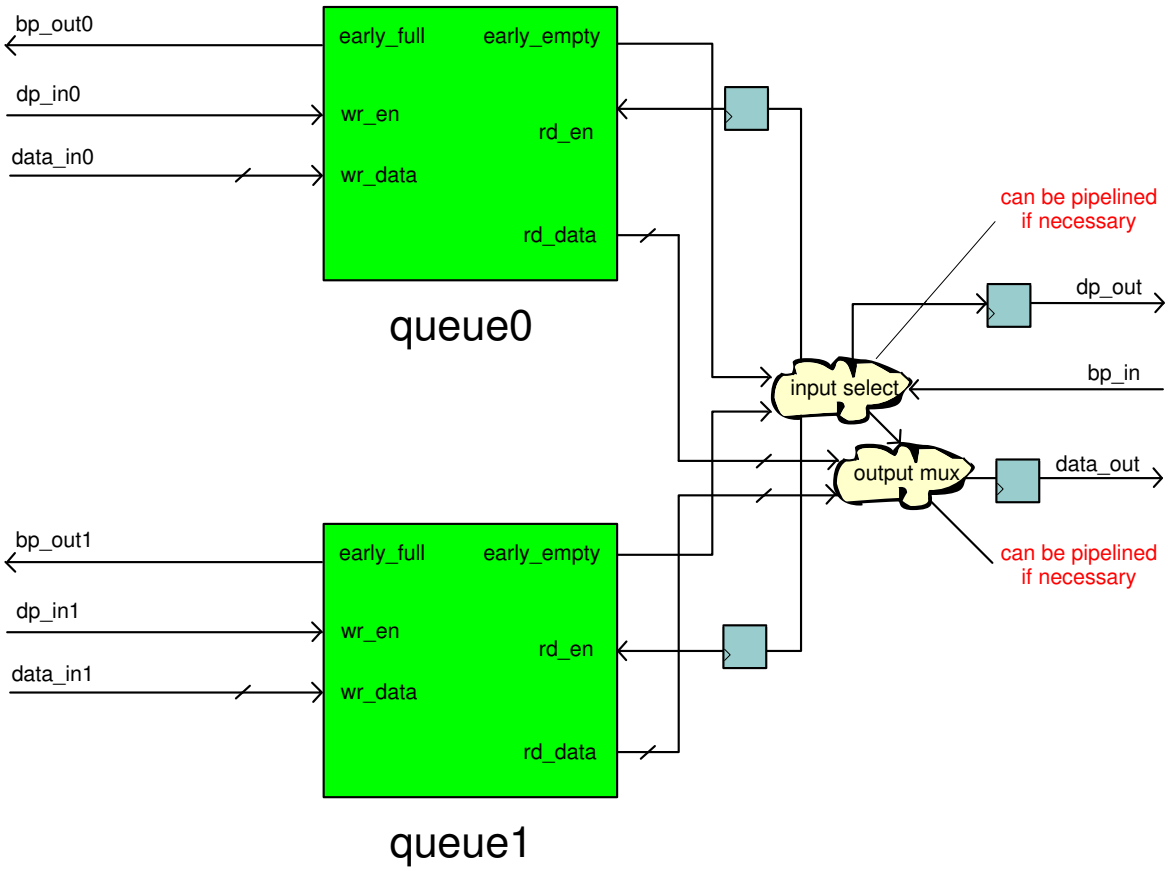


Figure 4.5: Internal Hardware Details of a 2-input Merge

4.2.2 Merge

The merge primitive has multiple inputs and one output. Packets arriving on the inputs must compete for a single output. A simple scheme of arbitration would be to select between the input ports in a round robin fashion. We use a more adaptive scheme that selects a packet based on FIFO occupancies of the input queues and a priority function. We show hardware details of a 2-input merge in Figure 4.5. A merge primitive is composed of an input FIFO for each input followed by selection logic to choose between these inputs. When limited to a few inputs, the input selection hardware can fit comfortably in 2–3 4-LUTs. When selecting from more than 4 inputs, we may need the selection logic. We require a multiplexer to select between the read data ports of the input FIFOs. It may be necessary to pipeline the multiplexer if a large number of inputs are present.

Area and latency figures for packet-switched primitives are shown in Table 4.1, assuming a 32-bit datapath (16-bits data, 16-bits destination address) and a buffer depth of 16. Slices are a unit of measuring area in FPGAs. A single slice contains 2 4-LUTs on a Xilinx Virtex-2 6000 device. Newer FPGAs *i.e.* Xilinx Virtex-5 user 6-LUTs and they have 4 6-LUTs per slice. We measure slices

Primitive	**Area** (Slices)			**Latency** (Cycles)	**Speed** (MHz)
	Queue		Total		
	Ctrl	Buffer			
16-deep Buffer					
Split-2	30	33	80	2	218
Merge-2	60	66	154	2	200
Split-3	30	33	88	2	217
Merge-3	90	99	254	2	200
Split-4	30	33	96	2	212
Merge-4	120	132	340	2	223
1-deep Buffer					
Split-2	0	8	23	2	269
Merge-2	0	16	58	2	262
Split-3	0	8	30	2	269
Merge-3	0	24	94	2	252
Split-4	0	8	32	2	265
Merge-4	0	32	115	2	252

Table 4.1: Area, Latency and Speed of 32-bit Packet-Switched Switching Primitives on a Xilinx Virtex-2 6000 -4

using the Virtex-2 metric. A Xilinx Virtex-2 contains $\approx$ 32K slices. The largest of the newer Xilinx Virtex-4s contain $\approx$ 100K slices.

4.3 Other Elements

The PEs that generate and receive packets can have a different datapath bitwidth than the network. To send a network message, a packet needs to broken into a sequence of flits prior to network injection. We serialize data into flits before inserting them into the network and deserialize them on reception. PEs may also have multiple input and output ports. Consequently, we must distribute packets into specific output ports as required by the routing algorithm. We would also be required to multiplex the incoming packets on the multiple input ports. We encapsulate all this extra functionality into specialized IO blocks as shown in Figure 4.6. We reuse the split and merge primitives described in previous section (Section 4.2) and design additional serializer and deserializer primitives for use in the IO blocks. The input block contains an instance of the deserializer primitive for each input followed by a wide-merge. The deserializer primitive collects multiple flits of a packet before forwarding it to the wide-merge. The output block contains a wide-split followed by an instance of the serializer primitive for each output. The serializer primitive on the other hand accepts a parallel input and injects flits into the network sequentially. While the serializer and deserializer primitives are useful in supporting multi-flit packets, we current do not use them for this study since our applications are all single-flit (see Chapter 6).

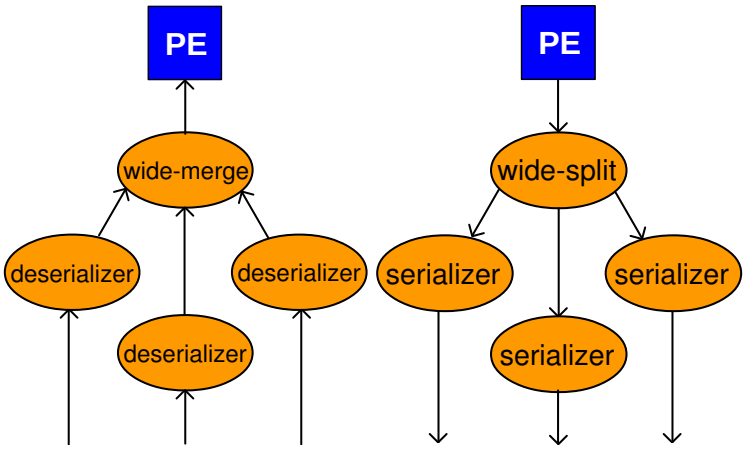


Figure 4.6: Conceptual Diagram of Input and Output Blocks

Chapter 5

Topology

Topology of a network refers to the arrangement of switches and PEs in the network. Choice of topology has a significant impact on the performance and area requirements of on-chip networks. In this chapter, we first motivate the kinds of networks we consider for our analysis. We then describe the networks in detail and explain how we use the hardware primitives described in Chapter 4 to compose switches in these topologies.

5.1 Limited Bisection Networks

The most commonly used topology for NoCs is the bus [3, 28, 60]. With no segementation, a bus can only handle a single network send and receive per cycle. Applications with large numbers of PEs will become quickly bandwidth bottlenecked on the bus. Topologies such as crossbars, multistage networks (*e.g.* Beneš [4], Clos [11]), stars, and hypercubes [35] all represent networks, at the other extreme, that have a large bisection. More specifically, crossbars have enough switching logic to allow routing any permutation while the switching limitations in the Beneš network allows routing any permutation with a possible rearrangement of existing routes. These topologies are prohibitively expensive in terms of area two-dimensional VLSI layout [57], especially when scaled to 1000s of PEs.

[33] observes that typical designs do not require full connectivity between the PEs. We can use Rent's rule to characterize the wiring requirements of our application ($IO = cN^p$, where N = compute nodes in the application, IO = input and output messages sent between the compute nodes). We observe that most designs are characterized by a Rent parameter p, where $0.5 < p < 0.75$. Thus, most designs operate in a limited bisection region with $p < 1$, but do require a bisection more than $p = 0$. We can characterize all topologies by the tunable Rent parameter p (*e.g.* a bus is a $p = 0$ topology, while fully connected topologies are $p = 1$) and attempt to design a network with a p that will match application requirements. We will see that over our range of applications $p = 0$ networks can quickly become bisection limited, while $p = 1$ networks avoid bandwidth bottlenecks at a significant area cost.

```
RING ROUTING FUNCTION
ΔX = destination.X − switch.X
if ΔX == 0
   direction = PE_EXIT
else if ΔX > 0
   direction = EAST
else if ΔX < 0
   direction = WEST
```

Figure 5.1: Routing Function for the Ring

We examine rings, meshes and Butterfly Fat Trees (BFTs) [25,37] over a range of configurations. Rings and meshes can be parameterized by their channel width w to increase bandwidth, while BFTs can be parameterized around base channel width c and wire growth rate p. In particular we examine rings with $p = 0$ and $w = 1, 2$, BFTs with $c = 1, 2$ and $p = 0, 0.5, 0.67, 1$, and two-dimensional meshes (both directional and bidirectional) with $p = 0.5$ and $w = 1, 2$. We also vary the virtual channel count (discussed in greater detail in Section 5.3.2) $VC = 2, 4$ on bidirectional meshes.

5.2 Ring

In a ring topology, the network elements are connected as shown in Figure 5.2. Each switch is connected to two neighbouring switches and one PE. We build bidirectional rings, in which, every pair of neighbouring switches can exchange packets going in either direction on independent physical channels. Thus, in the simplest configuration, each ring switchbox shown in Figure 5.2 has one bidirectional connection to a PE and one bidirectional connection to each of its two neighbouring switchboxes. We leave the connections emerging from the two extreme ends of the ring unconnected. This avoids the possibility of having a physical cycle in the network and ensures deadlock-free routing. The existence of bidirectional communication channels between the switchboxes continues to ensure full routability. We describe the simple routing algorithm used, in Figure 5.1.

We also build multiple parallel rings of width W to achieve larger bisection bandwidth ($N_{topcut} = W$). Rings with a larger bisection can potentially carry more traffic. These rings, however, need extra hardware to select between the different channels. This is shown in Figure 5.3. Thus, exploring multiple channels allows us to evaluate the impact of sacrificing logic area for interconnect area to get better performance.

5.3 Mesh

We build both directional and bidirectional meshes for our analysis. In bidirectional mesh networks, switches are connected in a 2D array structure as shown in Figure 5.4 with each switch connected to a maximum of 4 possible neighbouring switches (switches on the mesh boundary have 3 neighbours

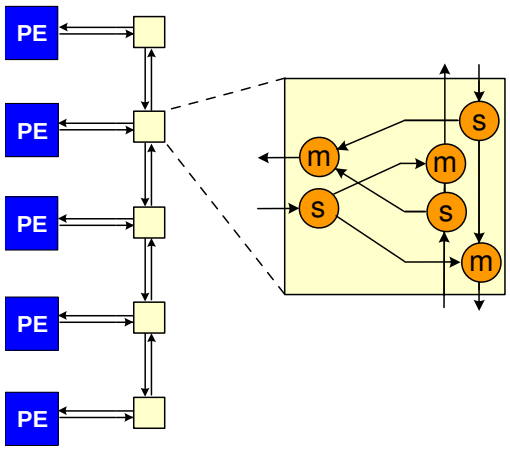


Figure 5.2: Ring Topology and a Ring Switch

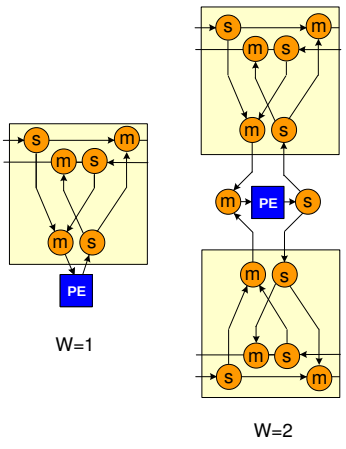


Figure 5.3: Effect of Multiple Channels on Hardware Requirements of a Ring

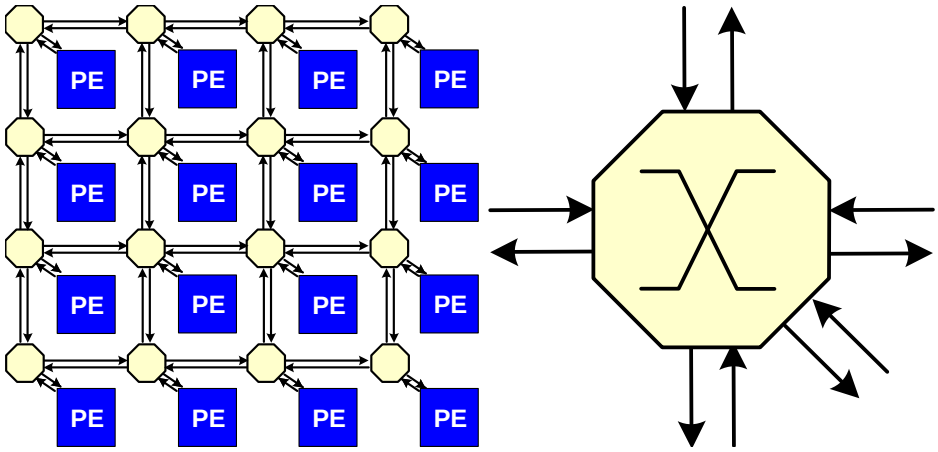


Figure 5.4: Bidirectional Mesh Topology and a Bidirectional Mesh Switch

while corner switches have 2 neighbours). There are as many switches as PEs and each switch is connected to a unique PE. Every pair of connected switches can exchange packets going in either direction on independent physical channels. The bidirectional mesh is composed of identical bidirectional switches as shown in Figure 5.4. We implement, both, deterministic and adaptive routing algorithms on the bidirectional mesh (*i.e.* Dimension Ordered Routing, West-Side First, Duato's algorithm. See Section 5.3.2).

In directional mesh networks, the switches are still connected in a 2D array structure. However, there are additional restrictions on how packets can move between the switches. Every row or column in the directional mesh allows packets to flow in only one direction. To ensure complete connectivity, these directions alternate every other channel (*i.e.* the odd columns have packets that can go only NORTH while the even columns have packets that can go only SOUTH, see Figure 5.5). Additionally, the PEs are required to have two ports into the network for full network reachability. We compose the mesh using a set of 4 directional switchboxes (one for each 2D mesh orientation). We implement only deterministic routing algorithm for the directional mesh (*i.e.* Dimension Ordered). We could implement other adaptive routing algorithms, but existing directional restrictions complicate the design. For this study, we choose not to implement these adaptive algorithms.

For both flavors, we vary channel width W to build meshes with richer bisection bandwidths ($N_{topcut} = O(W \times \sqrt{N})$) at the expense of logic area. Once packets are injected into a physical channel, they are routed to their destinations wholly in that channel. These packets are not allowed to switch channels along the way. Hence, the PEs must be distribute their traffic evenly across the different channels to optimize their use.

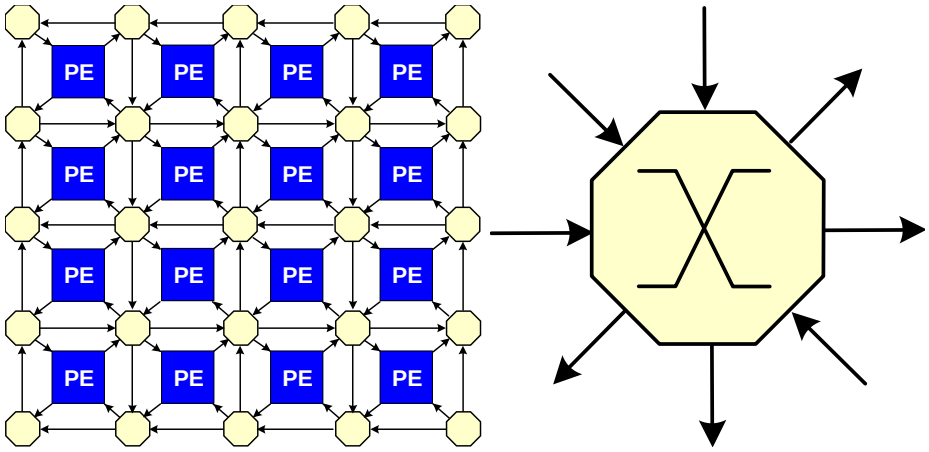


Figure 5.5: Directional Mesh Topology and a Directional Mesh Switch

5.3.1 Switch Architecture

We compose the mesh switchboxes in several different ways using the split and merge primitives. Arity of the primitive (*i.e.* the number of inputs of the merge, number of outputs of the split), the arrangement of the primitives (*i.e.* order in which they are connected) and the implemented routing algorithms give rise to a rich set of possible switchboxes with different areas and traversal latencies. We tabulate the area and latencies of these switchboxes in Table 7.1

Island-Style Switchbox We have two architectures for composing switches for the Directional Mesh. The first architecture is shown in Figure 5.6 which is inspired by the island-style connectivity of programmable FPGA interconnect described in [7, 8]. The key idea here is to minimize the number of splits and merges and hence area at the expense of longer switch latency. This switch architecture is not relevant to the bidirectional mesh.

Fast-XY Switchbox An alternative design that optimized latency at the expense of area is shown in Figure 5.7. The observation here is that packets spend most of their time traversing along the X (from X+ to X-, or X- to X+) or Y (from Y+ to Y-, or Y- to Y+) directions and taking 90^o turns only occasionally. In certain routing algorithms, turns are actually taken only once in the entire journey (Dimension Ordered Routing, Section 5.3.2.1). We can minimize the latency experienced by a packet when taking these through connections. This is illustrated in Figure 5.7 with the blue colored route for a packet going from $X+$ to $X-$. The packet experiences the latency of only one split and one merge as it passes through the switch. When a packet needs to take a turn, it has to go through one additional split and merge as shown by the red route in the same figure. Finally, packets going to and from a PE are processed last and see the most latency represented by the green route. We build Fast-XY switches for both directional and bidirectional mesh. In both cases, the

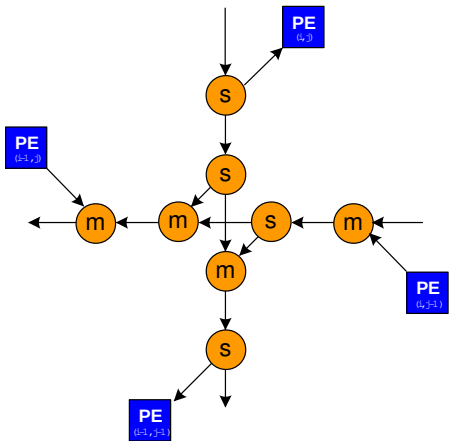


Figure 5.6: Directional Mesh Island-Style Switch

three split cascade is implemented for each input and the three merge cascade is implemented for each output. The internal wiring between the cascades is specific to the routing function used and the nature of the mesh.

High-Arity Switchbox The two architectures discussed earlier are to be preferred when only 2-input merges and 2-output splits are available. With larger arity splits and merges, it is possible to build more compact switches that use fewer splits and merges. Currently, we implement larger arity switches only for the bidirectional mesh (it is possible, with additional effort, to build these for the directional mesh as well). Two implementations of this switchbox implementing two different routing algorithms are shown in Figure 5.9 and in Figure 5.11.

5.3.2 Routing Algorithms

5.3.2.1 Deterministic

In deterministic routing, the path taken by a packet traveling between a given source-destination pair is always fixed. The path is intelligently chosen so as to avoid deadlock. For a 2D Mesh, this is easily achieved by forcing all packets to route in the same, particular dimension first followed by the other dimension once the first dimension is equalized. This form of routing is called Dimension Ordered Routing (DOR). For example, if a packet wants to route from PE $2, 1$ to PE $6, 8$, it first routes along the $+X$ direction until it reaches a switch in column 6 after which is routes in the $+Y$ direction to eventually reach the destination. DOR can be implemented in the switch by disabling a set of turns. Specifically, if we route along the X channel first, then turns going from Y to X are disabled. Thus, in this case, a total of four turns are disaled *i.e.* NE, NW, SE and SW. The resulting switch implementation is shown in Figure 5.9.

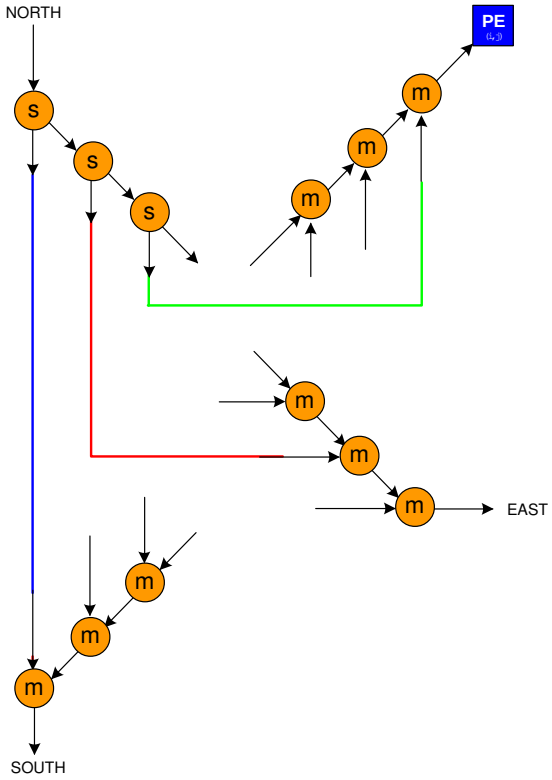


Figure 5.7: Mesh Fast-XY Switch

Dimension Ordered Routing Function
$\Delta X = destination.X - switch.X$
$\Delta Y = destination.Y - switch.Y$
if $\Delta X == 0$ && $\Delta Y == 0$
direction = PE_EXIT
else if $\Delta X > 0$
direction = $EAST$
else if $\Delta X < 0$
direction = $WEST$
else if $\Delta Y > 0$
direction = $NORTH$
else if $\Delta Y < 0$
direction = $SOUTH$

Figure 5.8: Dimension Ordered Routing

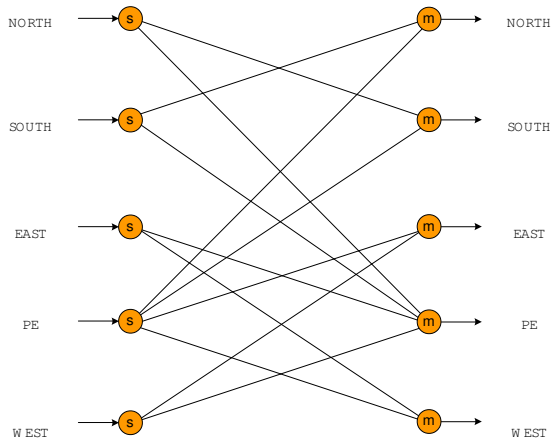


Figure 5.9: Structural Diagram of an Arity-4 Bidirectional Mesh Switch DOR

5.3.2.2 Adaptive

Deterministic routing algorithms are very restrictive and cannot adapt to congestion in the network. Partially adaptive algorithms based on the Turn Model [24] attempt to be less restrictive than deterministic algorithms and more responsive to network conditions at a moderate increase in implementation cost. While Dimension Ordered Routing disables a set of four 90^o turns, algorithms using the Turn Model prevent only two such turns in the network. We choose to implement the West-Side-First (WSF) routing algorithm in our analysis. In WSF, NW and SW turns are disabled and packets are routed in the West direction ($-X$) first if the destination is to the left of the source. Turns in East, North and South directions are taken adaptively based on network conditions. We show a switch implementation of this algorithm in Figure 5.11 which highlights the new connections added in red. The area cost of this switch is higher than the DOR switch (Figure 5.11) due to larger arity splits and merges for the affected ports. We show this in Table 7.1

5.3.2.3 Virtual Channels

Virtual Channels were originally developed to achieve deadlock-free routing in wormhole networks. They can also be used for flow control to help packets route around congested resources. Virtual channels attempt to provide full adaptivity while routing packets at the expense of significant buffer area overhead and switch complexity.

The key idea behind this scheme is the virtualization of the physical wiring resources in the network. The physical channel between two network elements is shared by multiple virtual channels. Usage of the physical channel is managed by virtual channel buffers allocated on either ends of the link. This is shown in Figure 5.12. Only one virtual channel is permitted to use the physical channel in a given cycle. Flits from other channels are stored in these virtual buffers while the

West-Side First Routing Function
$\Delta X = destination.X - switch.X$
$\Delta Y = destination.Y - switch.Y$
if $\Delta X == 0$ && $\Delta Y == 0$
direction = PE_EXIT
else if $\Delta X < 0$
direction = $WEST$
else if $\Delta X > 0$ && $\Delta Y > 0$
direction = Select$(EAST, NORTH)$
else if $\Delta X > 0$ && $\Delta Y < 0$
direction = Select$(EAST, SOUTH)$
else if $\Delta X == 0$ && $\Delta Y > 0$
direction = $NORTH$
else if $\Delta X == 0$ && $\Delta Y < 0$
direction = $SOUTH$

Figure 5.10: West-Side-First Routing

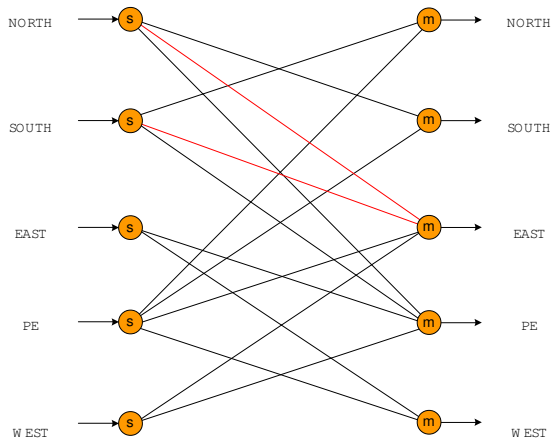


Figure 5.11: Structural Diagram of an Arity-4 Bidirectional Mesh Switch WSF

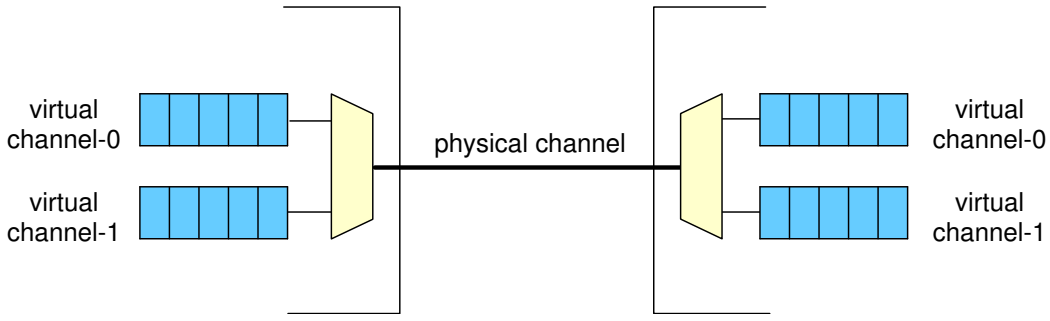


Figure 5.12: Idea of Virtual Channels

channel is being utilized. Free space in the destination buffers is used to determine which virtual channel gets to use the physical channel in a given cycle. Packets traveling in a virtual channel can switch between channels based on the the deadlock-free routing function implemented in the switches. Virtual channels can also be used to improve link utilization. This is achieved by reducing the amount of time a packet stays blocked waiting for an output to become available. Packets are allowed to bypass blocked packets by forwarding along an alternative virtual channel. Thus, the physical channel that was idle due to the blocked packet gets utilized. Virtual channels are very useful when operating in the pin-constrained regime of multi-chip networks. The number pf physical channels possible in an on-chip networks is far greater than that possible in off-chip networks due to chip IO limitations. In such conditions, we have to time-multiplex the different on-chip physical channels over the limited set of IO pins for routing offchip packets. These on-chip physical channels effectively behave as virtual-channels in the off-chip network.

We use Virtual Channels to investigate a better implementation of deadlock-free routing on the on-chip mesh. We use a high quality fully adaptive deadlock-free routing algorithm by Duato [23] for our exploration. For these experiments, we study the behaviour of the network by increasing the number of virtual channels (VC) per port. We need a minimum of 2 virtual channels for correct operation of the routing algorithm. Duato's algorithm splits the set of available virtual channels into two. One set of virtual channels (A) allows packets to turn in all possible directions while the second set (B) implements an adaptive deadlock-free routing algorithm (we use Dimension Ordered Routing in B). Packets in A that are unable to continue routing along A *escape* to a virtual channel in B. Once in B, the packet is not allowed to re-enter A. When the packet is limited to routing within B, it is guaranteed to reach the destination since B implements a proven deadlock-free routing scheme.

5.4 BFT

A Butterfly-Fat-Tree is a network where the switches are organized in a folded-tree-like structure and the PEs are connected to the leaf-level switches. The "fatness" of the tree can be tuned by changing the number of channels or switches at upper levels in the tree. The folding of subtrees gives rise to a resemblance to "butterfly" networks. We can represent the "fatness" mathematically as the value of the Rent parameter p. We compose BFTs of arity-2 using T and Π switches as shown in Figure 5.15 [20, 58]. The Rent paramter p is translated into a sequence of growth rates for each level in the tree (a growth rate of 1 implies T switches at that level and a growth rate of 2 implies Π switches at that level). This translation is described in [18] Increasing the Rent parameter of the BFT increases its bisection bandwidth ($N_{topcut} = c \times n^p$), thereby allowing more bisection traffic per cycle. Contrast this with the mesh ($N_{topcut} = W \times \sqrt{N}$) with a $p = 0.5$ and a ring ($N_{topcut} = W$) with a $p = 0$. BFTs with larger values of p also require more switches, which influences the number

DUATO'S FULLY ADAPTIVE ROUTING FUNCTION

$\Delta X = destination.X - switch.X$
$\Delta Y = destination.Y - switch.Y$
if $\Delta X == 0$ && $\Delta Y == 0$
  direction = PE_EXIT
else if $\Delta X < 0$
  direction = $WEST$
else if $\Delta X > 0$ && $\Delta Y > 0$
  direction = Select($EAST\_A, NORTH\_A, EAST\_B$)
else if $\Delta X > 0$ && $\Delta Y < 0$
  direction = Select($EAST\_A, SOUTH\_A, EAST\_B$)
else if $\Delta X < 0$ && $\Delta Y > 0$
  direction = Select($WEST\_A, NORTH\_A, WEST\_B$)
else if $\Delta X < 0$ && $\Delta Y < 0$
  direction = Select($WEST\_A, SOUTH\_A, WEST\_B$)
else if $\Delta X == 0$ && $\Delta Y > 0$
  direction = $NORTH\_A, NORTH\_B$
else if $\Delta X == 0$ && $\Delta Y < 0$
  direction = $SOUTH\_A, SOUTH\_B$
else if $\Delta Y == 0$ && $\Delta X > 0$
  direction = $EAST\_A, EAST\_B$
else if $\Delta Y == 0$ && $\Delta X < 0$
  direction = $WEST\_A, WEST\_B$

Figure 5.13: Duato's Fully Adaptive Routing Function for the Bidirectional Mesh

of PEs that can be packed on a chip. For example, in a ConceptNet PE based network (195 slices for each PE), given 130K slices, we can either fit a $p = 0$ BFT with 128 PEs or a $p = 0.5$ BFT with only 64 PEs.

Actual composition of the T and π switches using the hardware primitives are shown in Figure 5.16. We hierarchically place the network on the FPGA and pipeline the upper stages of the tree based on layout feedback to retain high performance even when stages cannot be placed adjacent to one another.

5.4.1 Routing Algorithm

We use a minimal adaptive routing algorithm for routing on the BFT [38]. Packets climbing up the tree make adaptive routing decisions based on FIFO occupancies. When the packet reaches the common ancestor of the source and destination PEs, it begins its descent. During descent, the route is completely deterministic for a given destination. Appropriate bits from the destination address guide the packet downhill to the destination. This is called destination-tag routing. We describe this algorithm in Figure 5.14.

We enumerate the list of all topologies explored in Table 5.1.

ADAPTIVE BFT ROUTING FUNCTION

```
if uphill
   if tswitch()
      direction = TOP
   if piswitch()
      direction = Adaptive_Select(TOP_LEFT, TOP_RIGHT)
else if downhill
   if destination.X[height] == 0
      direction = BOTTOM_LEFT
   else
      direction = BOTTOM_RIGHT
```

Figure 5.14: Minimal Adaptive Routing Function for the BFT

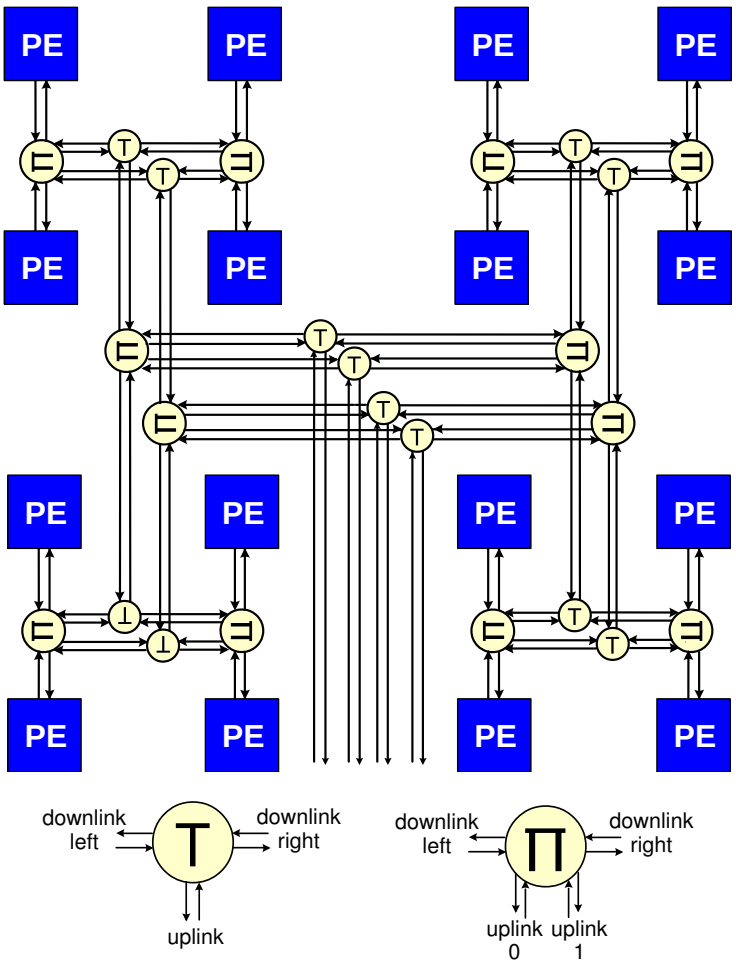


Figure 5.15: BFT Topology and BFT Switches

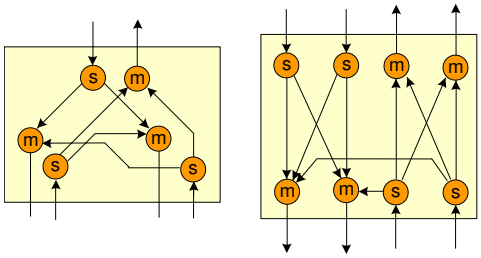


Figure 5.16: Structural Diagram of BFT T and Π Switch

Topology	Channel-Width (W) or Virtual-Channel (VC)	Rent-Parameter (p)	PEs	Total
Ring	1, 2, 4	0	2–2048	33
Directional Mesh Island-Style	1, 2	0.5	2–2025	88
Directional Mesh Fast X-Y Style	1, 2	0.5	2–2025	88
Bidirectional Mesh Arity-4 DOR	1, 2, 4	0.5	2–2025	132
Bidirectional Mesh Arity-4 WSF	1, 2, 4	0.5	2–2025	132
Bidirectional Mesh Fast X-Y Style	1, 2, 4	0.5	2–2025	132
Bidirectional Mesh Duato (VCs)	2, 4	0.5	2–2025	88
BFT	1, 2	0, 0.5, 0.67, 1	2–2048	88
Total		26	-	781

Table 5.1: List of explored Topologies

Chapter 6

Applications

As with most other NoC research efforts, we could have used synthetically generated workloads for our analysis. But we choose to use workloads generated from a range of real applications that are representative of typical communication conditions and hence predictive of real performance. Using several real workloads further allows us to capture the variability of communication characteristics in the real world. Synthetic workloads are useful for initial analysis when actual applications are not available. As we scale the number of PEs for a given application, the amount of network traffic that gets injected into the network from each PE changes. These trends are not captured when using synthetic benchmarks where the injection rate is kept constant.

We examine communication workloads for applications mapped to the GraphStep system architecture [22] which is a specific form of Bulk Synchronous Parallel (BSP) model of computation [59]. The GraphStep system architecture is a high-level model for capturing graph algorithms, abstracted from a detailed hardware implementation. The computation is a graph where each node is an actor that communicates to neighboring nodes through message passing. Computation proceeds in steps, each of which is a three phased Receive-Update-Send sequence. Nodes are barrier synchronized on the end of each Send-Receive phase. We model the communication of the Receive and Send phases on our network, and measure network performance as the number of cycles required to route a workload in a single GraphStep. One benefit of using this compute model for generating our workloads is the separation of the communicate phase from the compute phase. Moreover, the messages generated in the communicate phase are not required to obey any ordering constraints and are independent of any dependencies in the compute graph. This allows us to generate our traffic apriori without actually modeling the compute step and we can inject messages into the network using an order of our choosing. Thus, these workloads are easily reproducible and do not require complex, simultaneous simulation of computation and communication. While this could limit the generality of the analysis, this provides us with initial evidence to help compare different networks. The analysis can then be extended using a more diverse set of workloads as mentioned in Chapter 9.

In our particular FPGA implementation, multiple graph nodes are physically placed on each

PE where they are time-multiplexed. The PEs consist of specialized processing logic coupled with small, local, high bandwidth on-chip memories (*e.g.* Xilinx BlockRAMs [61]). PEs are capable of processing and sending messages in a single cycle. This allows applications to potentially send and receive messages on every cycle from every PE in the network.

GraphStep allows us to examine applications that inject large amounts of traffic into the network. To illustrate the importance of examining applications with high communication requirements mapped to large-scale networks, consider Figure 2.1. Here we plot network I/O messages per cycle as a function of PEs on a network with no bandwidth limitations, given a ConceptNet [39] communication load of 27K messages. Small networks ($<$ 100 PEs) require only 1–10 network sends or receives per cycle, as most cycles are dedicated to serialized processing at PEs. Larger networks ($>$ 100 PEs) require up to 200 network sends or receives per cycle. Consequently, examining network topologies for small networks or for applications with light communication requirements will not load networks enough to distinguish trade-offs in network design. It is essential to examine both large-scale networks and applications which stress those networks in order to fully characterize the performance differences between network topologies.

We describe the set of GraphStep applications chosen for our analysis briefly in this section.

6.1 ConceptNet Spreading Activation

ConceptNet [39] is common-sense reasoning knowledge base represented as a graph, where nodes represent concepts and edges represent semantic relationships. Queries to this knowledge base activate initial nodes in the graph, corresponding to the set of keywords in the query. Activated nodes send messages to neighbors along their edges, activating neighbors in turn. As the computation proceeds, larger portions of the graph become activated. The percentage of active edges over the whole graph (activity factor) depends on the initial query and what step of the spreading activation is being performed. In the case of complex queries or multiple simultaneous activations, the entire graph may become activated after a small number of steps. We route workloads from several query sets that touch different portions of the graph. These queries have to a range of activity factors between 1%–100% which correspond to consecutive steps in selected queries. Details of our FPGA implementation can be found in [22] and are shown in Figure 6.1. We wrote VHDL for the ConceptNet PEs, implementing the 3 phase algorithm of the GraphStep system architecture [22]. We implement the required multipliers using pipelined LUT-level logic instead of the on-chip 18x18 multipliers for speed. This PE requires 200 slices on a Virtex-2 6000.

SpreadingActivation

```
foreach graphstep
    // receive
        foreach message m
            e ← m.sink
            e.data ← m.data
        wait for step synchronization
    // update
        foreach graph node g
            g.sum ← 0
            foreach graph edge e of g
                g.sum ← g.sum + e.data
    // send
        foreach graph node g
            foreach graph edge e of g
            value ← g.sum × e.discount
            if (value>THRESHOLD)
                new message (e.sink , value)
```

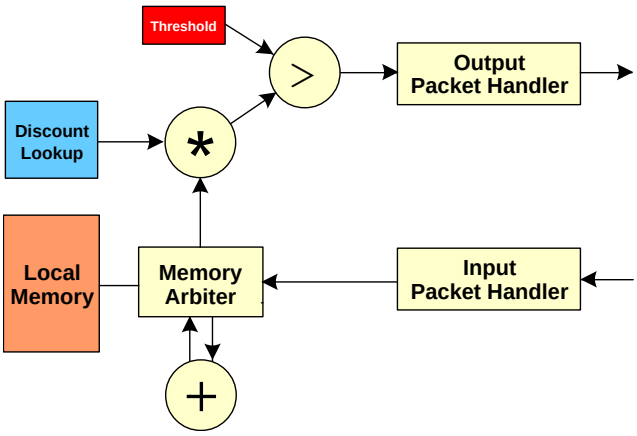


Figure 6.1: ConceptNet PE

MatrixVector Multiply

```
foreach graphstep
  // receive
    foreach message m
      e ← m.sink
      e.vector_value ← m.vector_value
    wait for step synchronization
  // update
    foreach graph node g
      g.vector_value ← 0.0
      foreach graph edge e of g
        g.vector_value ← g.matrix_value × e.vector_value
  // send
    foreach graph node g
      foreach graph edge e of g
        new message (e.sink , g.vector_value)
```

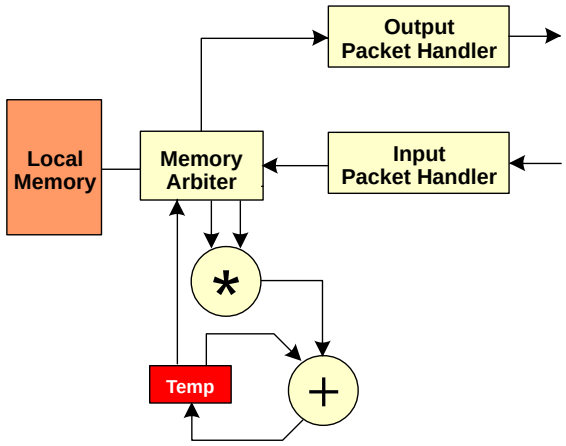


Figure 6.2: SMVM PE

6.2 Sparse Matrix-Vector Multiply

Iterative Sparse Matrix-Vector Multiply (SMVM) is used in several numerical routines (*e.g.* Conjugate Gradient, GMRES). In each iteration a set of dot products between the vector and matrix rows is performed. New values for the vector to be used in the next iteration are calculated and the matrix-vector multiply step is repeated. We can represent this computation as a graph where nodes represent matrix rows and edges represent the communication of the new vector values. In each iteration messages must be sent along all edges. We map representative matrices from the Matrix Market suite [47] to generate workloads for our experiments. We use an FPGA implementation of this algorithm based on deLorimier *et al.* [21] which we illustrate in Figure 6.2. They also use LUT-level logic to implement double-precision floating point computation for high-speed operation. This makes the PE very large and requires around 4000 slices.

BELLMAN FORD - LABELING PHASE

```
foreach graphstep
  // receive
    foreach message m
      e ← m.sink
      e.distance ← m.distance
    wait for step synchronization
  // update
    foreach graph node g
      g.distance = inf
      foreach graph edge e of g
        if e.distance ≤ g.distance
          g.distance ← e.distance
  // send
    foreach graph node g
      foreach graph edge e of g
      new_distance ← g.distance + e.cost
        new message (e.sink , new_distance)
```

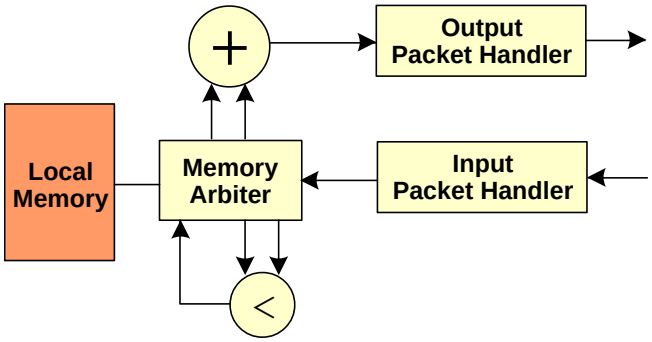


Figure 6.3: Bellman-Ford PE

6.3 Bellman-Ford Shortest Path

The Bellman-Ford algorithm is used in several CAD applications such as single source shortest paths (*e.g.* FPGA Routing [41]), finding negative edge weight cycles (*e.g.* Retiming [36]), and slack propagation (*e.g.* Static Timing Analysis). The algorithm simply relaxes all edges in each step until quiescence. A relaxation consists of computing the minimum over all values received on input edges. We run Bellman-Ford over representative ISPD98 [2] benchmark graphs to produce communication workloads that capture the structure of circuit netlists. We sketch the datapath and the operation of a Bellman-Ford PE in Figure 6.3. VHDL for the Bellman-Ford PE can be derived in a straightforward manner from the ConceptNet PE and is assumed to have the same size of 200 slices (although it could potentially be optimized).

Table 6.1: Application Graphs

Graph	Nodes	Edges	Max		Rent	
			Fanin	Fanout	c	p
ConceptNet						
`small`	15026	27745	63	64	33	0.5
SMVM						
`add20`	2395	17319	124	124	21	0.6
`bcsstk11`	1473	17857	27	30	89	0.2
`rdb32001`	3200	18880	6	6	24	0.5
`gemat11`	4929	33185	27	28	23	0.8
`utm5940`	5940	83842	30	20	300	0.2
`fidap035`	19716	218308	18	18	222	0.0
Bellman-Ford						
`ibm01`	12752	36455	33	93	28	0.3
`ibm05`	29347	97862	9	109	25	0.6

6.4 Application Characteristics

An understanding of application communication characteristics can aid network designers in selecting an appropriate network topology and size. Characteristics of the application graphs used for our exploration are shown in Table 6.1. In Section 3.3 we identified performance limiting characteristics of network topologies; here, we identify three key performance limiting characteristics of application graphs:

Total Messages The number of messages (*i.e.* graph edges) generated by an application dictates how serialized (Eq. 3.4) that application is. When the ratio of workload size to number of PEs is on the order of 100, the application is typically serialization limited. For example, we observe that for small workloads (17K-35K messages), applications are no longer serialized beyond 100s of PEs. Additionally, applications with significantly more messages (500K or more) require many more PEs (4000 or more) before they are affected by bisection or latency.

Node Fanin-Fanout As network size grows and the number of PEs increase, eventually a PE will contain a single graph node. As input and output messages must be serialized (Eq. 3.4), optimal performance will be limited by the maximum fanin or fanout node. Applications with large fanin or fanout nodes are performance limited when increasing PEs beyond a certain limit (*i.e.* when a single large fanin-fanout graph node resides on a PE). We decomposed application graphs that contained high fanin-fanout nodes *i.e.* `small`. After decomposition all graphs have a fanin-fanout limit of <128 and we ensure that computation remains semantically correct.

Rent Parameter To describe network interconnect richness and locality we used the Rent parameter p (Section 5). For applications we can similarly define an intrinsic Rent parameter p_{graph} to describe communication locality that is independent of actual placement. We can compute this parameter by counting the number of messages that cross the bisection recursively at each level of a recursive partition. Applications with higher p have non-local communication and require networks with sufficient interconnect capable of supporting this communication.

Chapter 7

Methodology

In this section, we present details on the tools we developed for our analysis. We developed a cycle-accurate simulator for routing communication workloads on different networks. We discuss the dual-phase simulation scheme used in this simulator. We model latencies in our networks for an accurate evaluation of performance. We motivate and explain this latency model for different topologies.

7.1 Tool Infrastructure

We use a Java-based infrastructure to construct and evaluate our networks. We generate parameterized configurations of all topologies and evaluate their performance on our workloads. Our tools write out a VHDL netlist representation of each topology along with UCF constraints. We run these through the Synplicity Compiler (v8.0) and the Xilinx ISE (v8.1i) to obtain area and performance numbers. We demonstrate 166 MHz performance for a sample topology (8 PE $p = 0.5$ BFT) on a V2-6000-4. This performance is limited by critical paths in the PE datapath and not the network which is potentially capable of running faster (200 MHz). We also pipeline long wires in the interconnect and model wire delays accurately for all topologies. We use a cycle-accurate simulator to route communication traffic on our packet-switched networks and to obtain cycle counts. We also use an implementation of a greedy router to compute routing schedules for the time-multiplexed network described in [29].

We map applications to our networks using a partitioner and placer based on MLPart v5.2.14 which is part of the UMPack [9] package. While we ensure that single chip logic and interconnect resources are sufficient to map our applications, we assume that application graphs can be mapped to the available on-chip BRAMs. We also assume that the required instruction memory for the time-multiplexed PEs can fit in BRAMs. This allows us to select a message order which optimizes communication time for static scheduling. ConceptNet and Bellman-Ford use a 32-bit network for passing messages while SMVM requires an 80-bit network for single precision floating point

communication. All applications use single-flit packets.

7.2 Cycle Accurate Simulator

We have a very large space for exploring the tradeoffs between the different networks. We generate networks with upto 44 sizes (2 PEs to 2048 PEs) for each one of the 26 topological configurations mentioned in Table 5.1 (Rings, BFTs and Meshes with different parameters). We route 6 benchmarks for each of these networks with as many as 100K messages in each benchmark. We also model the effect of wire latency and switch delay in these networks.

An RTL-level VHDL simulation of our network would provide an unnecessary level of detail that is irrelvant for our analysis. The RTL logic simulators are known to be very slow compared to higher-level functional simulations and runnning all these test cases would take a large amount of simulation time. Instead, we choose to perform a cycle-accurate functional simulation of the network. We compose the network in a hierarchical, bottom-up manner by creating functional models for the different network elements. We develop these functional models in a higher-level programming language, not VHDL. We synthesize, place and route the VHDL descriptions to get area and latency figures for these network elements. The functional models are then programmed with these figures for an accurate simulation. This allows us to not only run our simulations fast, but also permits a rapid design space exploration through quick modifications to the high-level functional models.

When considering sequential circuits without feedback (*i.e.* logic pipelines), we can order the circuit elements (*i.e.* combination logic, registers) by traversing the circuit graph from outputs to inputs in topologically sorted manner. This ordered graph can then be easily simulated in each cycle by visiting the elements in this topological order. For logic circuits with feedback (*i.e.* general RTL netlists), no such order can be guaranteed due to the cyclic dependencies between the circuit elements. We can resolve this ordering problem by using a dual-phase simulation scheme. In this scheme, the logic circuit is grouped into two types of elements, combinational logic blocks and registers as shown in Figure 7.1. Each register in the circuit is represented using two simulation-specific registers, an original register and an extra shadow register. In the first phase, only a write operation is permitted on the the original register while only a read operation is allowed on the shadow register. In this phase, the combinational logic block between the registers is simulated by reading the input values from the shadow input registers and writing the computed value into the original output register. This is represented in Figure 7.2 where the red edges represent the active edges. In the first figure, the inputs values (orange tokens) are read by the combination logic blocks and in the second figure, the outputs register (blue tokens) is updated. All combinational logic blocks in the circuit are simulated in this manner and in any order. In the second phase, the values from the original registers are transferred to their respective shadows. This is illustrated in

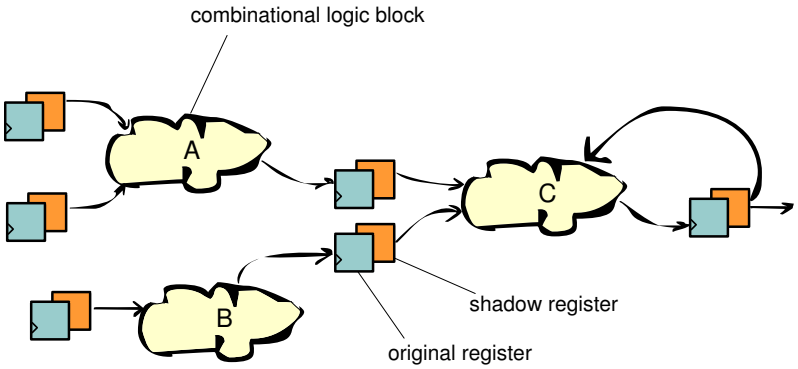


Figure 7.1: Logic-Circuit for Simulation

Figure 7.3 where the blue tokens replace the orange tokens. This completes one cycle of the circuit simulation. The splitting of registers allows the circuit elements to be simulated independently. We simulate our network using this dual phase algorithm.

Using a hierarchical simulation framework also enables us to iteratively increase the simulation detail. During the initial exploration phase, we wrote switch-level functional models that tested the different routing algorithms at the switch-level. This allowed us to verify the correctness of these functions without needing to accurately model the effect of contention at the switch outputs. Once the routing functions were verified, we replaced the switch with an arrangement of interconnected splits and merges. This enabled a proper modeling of blocking behavior between the splits and merges and also contention between resources.

We wrote functional models for the PEs to model the three-phase GraphStep algorithm described in Section 6. A list of packets that need to be sent from a PE is computed before running the simulation. When simulating a ConceptNet workload, this list is generated from an external LISP implementation. For all other benchmarks, we can assume messages on all graph edges. We distribute this list to the respective PEs to generate packets. Some of the generated packets could be self-messages (messages intended for the generating PE itself). Self-messages are handled separately and are not injected into the network. We assume that our PEs have dual-port memories (Xilinx BRAMs are dual-ported). The packet-handling logic in our PE allows for single-cycle packet reception and transmission.

7.3 Area Model

We tabulate the area required by the individual switchboxes in Table 7.1. We assume a buffer-depth of 1 and no interconnect pipelining for these calculations. The 32-bit switches in the table are for ConceptNet and Bellman-Ford implementations while the 80-bit network is for Matrix Multiply. As

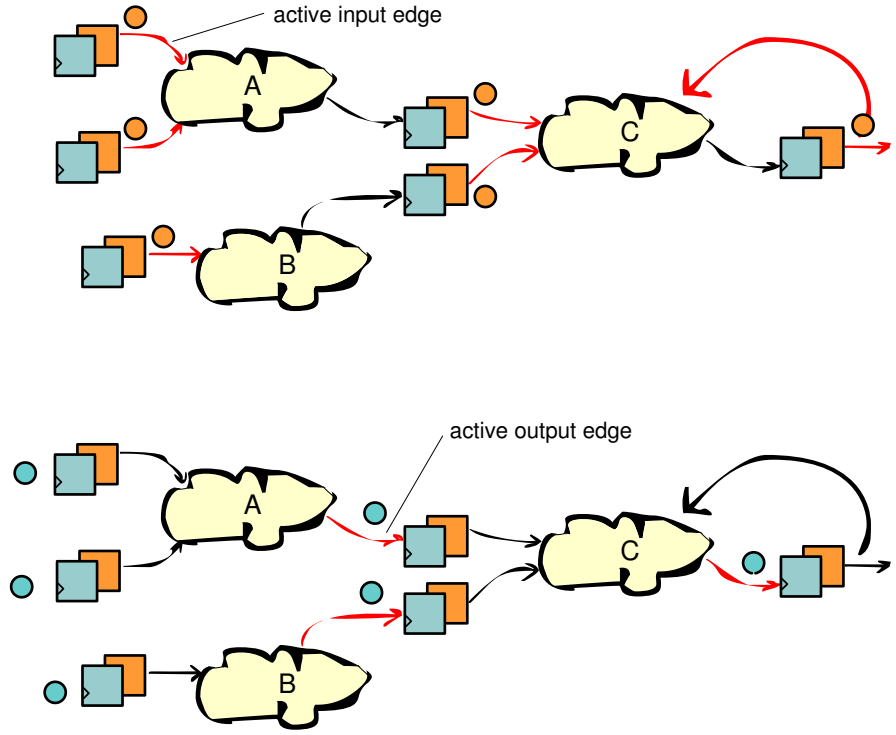


Figure 7.2: Dual-Phase Simulation Algorithm : Phase-1

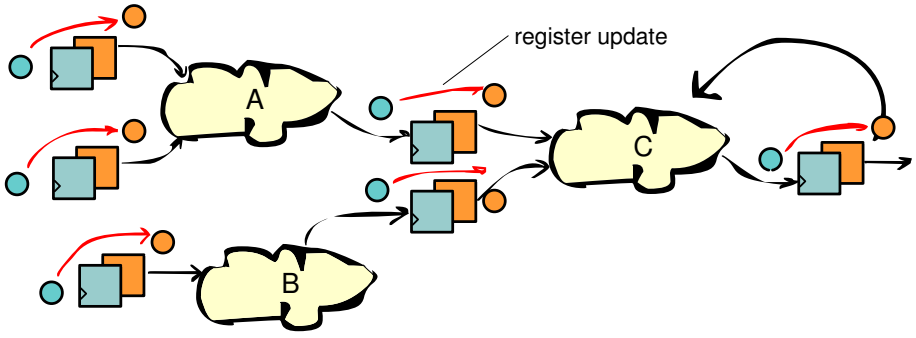


Figure 7.3: Dual-Phase Simulation Algorithm : Phase-2

SwitchBox	IO Ports	Area (Slices)		Latency (Cycles)
		32-bit	80-bit	
Ring	3	243	531	4
Directional Mesh Island-Style	4	324	708	8
Directional Mesh Fast X-Y Style	4	972	2124	4 (fast), 8(slow)
Bidirectional Mesh Arity-4 DOR	5	603	1299	4
Bidirectional Mesh Arity-4 WSF	5	660	1428	4
Bidirectional Mesh Fast X-Y Style	5	1215	2124	4 (fast), 8(slow)
Bidirectional Mesh Virtual-Channels with Duato	5	-	-	8
BFT T-Switch	3	243	531	4
BFT Π-Switch	3	410	886	4

Table 7.1: Area and Latency of Switchboxes of different topologies with 1-deep buffers with no wire pipelining

described in Chapter 5, we compose the switches using splits and merges. For the Virtual Channel switchbox, we currently do not have such an implementation. We use a conservative approximation of area for these switches (area is consistent relative to other switches). For switches in the corners or edges of the meshes, corners of rings, or at the top-levels in the BFT we need less connectivity than usual. We model this by pruning the unnecessary logic and counting area for only relevant primitives.

7.4 Latency Model

The total time required for a packet to traverse the network is a function of both the switch delay and the wire delay along the path. We already model logic delay in our Java primitives. To get an accurate estimation of performance, we choose to model wire delay as well. This is particularly important when trying to compare topologies with different switching requirements.

FPGA Characterization We first characterize the wiring latency of the FPGA to calculate the amount pipelining required for a given wire on the chip. We place two primitives at the extreme ends of a chip side as shown in Figure 7.4. We vary the amount of pipelining provided on the wires between these primitives to achieve 200 MHz performance. We observe that we need 3 levels of registers on the wires between the primitives placed in this configuration. This roughly translates into a requirement of pipelining the wire every ≈60 slices. The largest PE we consider is the SMVM PE (Section 6.2). It is assumed to be placed in a square region of dimensions 65 slices × 65 slices. An unpipelined wire crossing the length of this PE would need ≈ 1.1 cycles to each the other side. Placement congestion in a fully populated FPGA may cause the wire to take a non-minimal, slower route. Thus, if we want to pipeline this wire for 200 MHz performance, we will need to pipeline it twice *i.e.* 2 registers distributed over 65 slices of wire distance.

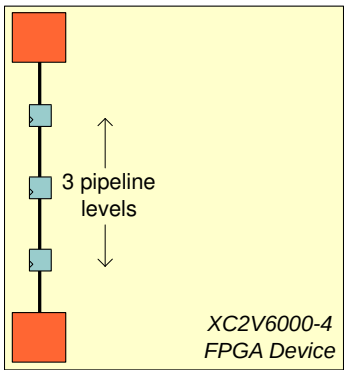


Figure 7.4: FPGA Characterization Experiment

Mesh Wire Delay Model Here, we compute the worst case delay of a placed and routed mesh topology by modeling both switch delay and wire delay. We place a mesh with N PEs in a grid of size $\sqrt{N} \times \sqrt{N}$. The mesh switches are placed near their corresponding PEs. If the network is a bidirectional mesh, there will be $\sqrt{N}$ switches along the horizontal or vertical dimensions. The area of a bidirectional mesh switch is smaller than an SMVM PE. Thus, the amount of pipelining required between neighbouring switches will be dictated by the size of the PE. The wire between these switches will need to be registered twice as described earlier. We can express the worst case latency of a route in a bidirectional mesh as follows:

$$\begin{aligned} Mesh_{latency} &== 2 \times (\sqrt{N} - 1) \times (t_{logic} + t_{wire}) && (7.1) \\ &== 2 \times (\sqrt{N} - 1) \times (2 + 2) && (7.2) \\ &== 2 \times \sqrt{N} + 4 \times \sqrt{N} - 8 && (7.3) \end{aligned}$$

In the equation, t_{logic} is the port-to-port latency of a bidirectional mesh switch from Table 7.1.

BFT Wire Delay Model We place the BFT using the H-tree [19] pattern. Using this pattern, as we climb up the tree, the wires get progressively slower since they interconnect increasingly larger subtrees. We pipeline these wires to get 200 MHz operation. We observe from Figure 7.5 that the length of the wires doubles after every two segments (two levels of switches) as we climb up the tree. Hence, to keep pace with this increasing wire length, we double the amount of interconnect pipelining provided after every two levels. The worst case latency of a route in a BFT is shown here:

$$\begin{aligned} BFT_{latency} &== 2 \times \log N \times t_{logic} + 2 \times \sum_{l=0}^{l=\log(N)-1} t_{wire_l} && (7.4) \\ &== 4 \times \log N + 2 \times (1 + 1 + 2 + 2 + 4 + 4 + \ldots + 2 \times \frac{\sqrt{N}}{4} + 2 \times \frac{\sqrt{N}}{4}) && (7.5) \end{aligned}$$

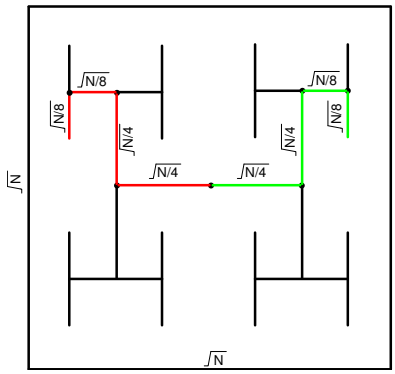


Figure 7.5: Wire Lenghts in the BFT

$$== \quad 4 \times \log N + 4 \times (1 + 2 + 4 + \ldots + 2 \times \frac{\sqrt{N}}{4}) \tag{7.6}$$

$$== \quad 4 \times \log N + 4 \times (1 + 2 + 4 + \ldots + \frac{\sqrt{N}}{2}) \tag{7.7}$$

$$== \quad 4 \times \log N + 4 \times \sqrt{N} \tag{7.8}$$

Note that the wire latencies in both Equation 7.3 and Equation 7.8 are the same ($4 \times \sqrt{N}$). This corresponds to the minimal Manhattan distance between the extreme PEs which is the same same irrespective of topology . The latency in the BFT switchboxes is programmed for each level independently. Within a level, it is programmed separately for wires from the lower level and wires from the level above. The sequence of latencies we use is (1 1 2 2 4 4 ...). Each switch use two values from this array depending on its height in the tree to program its wire delays.

We plot latency obtained from Equation 7.3 and Equation 7.8 in Figure 7.6. We observe that below $\approx$ 300 PEs, the worst case latency on a Mesh is marginally better than the BFT. Above 300 PEs, the logarithmic switch delays help the BFT achieve better worst case latency. But, as we will see in Section 8.2 the Mesh does not always need to route the worst case path.

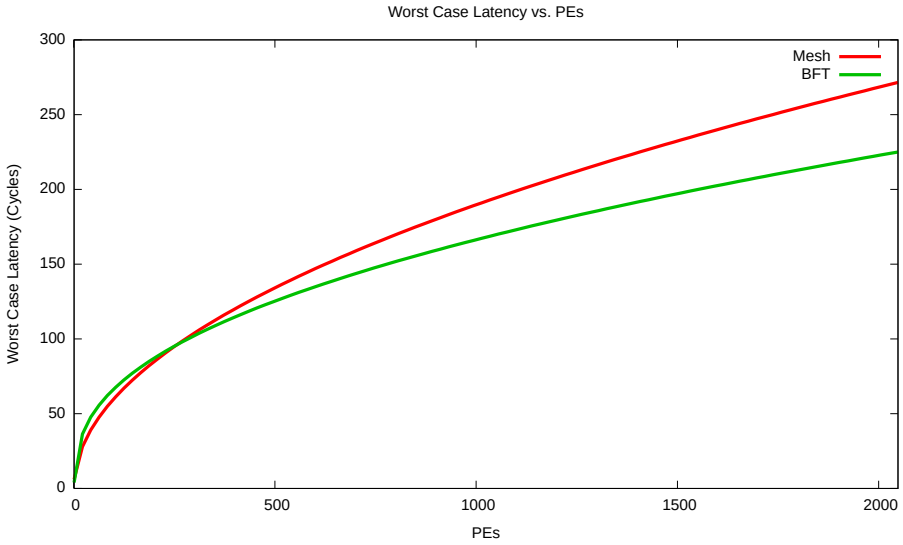


Figure 7.6: Comparing Worst-Case Latencies on the Mesh and the BFT

Chapter 8

Evaluation

We start this section by providing quantitative justification for selection of buffer depth in the hardware primitives. We use these primitives in our subsequent exploration. We then present and analyze results from several experiments. First, we provide quantitative reasoning behind topology selection when designing large-scale networks having a large number of PEs. Next, we compare packet-switching and time-multiplexing to understand the application scenarios under which either one of them is to be preferred.

8.1 Selection of Buffer Depth

The split and merge primitives are parameterized around three variables, datawidth, packet-length and buffer depth. We choose datawidth and packet-length based on application requirements. For the applications considered, the number of bits per message is not very large ($\leq$ 80 bits). Hence, for all our experiments, we set packet-length to 1 and appropriately adjusted datawidth to equal messages size. This allowed us to avoid the area penalty of having specialized IO blocks (Section 4.3) to handle PE IO. It also enabled us to achieve better performance than with messages serialization. That means, we only need to tune buffer depth. As noted in Section 4.2, buffer depth has a large impact on area as well as performance of the network. To quantify this impact and to help choose the right buffer depth, we built a few sample topologies using a $c = 1, p = 0.67$ BFT and routed traffic from the `gemat11` Matrix Multiply (Section 6.2) benchmark. In Figure 8.1, we plot the performance achieved by networks with buffer depths with 1, 2, 4, 8, 16, 32, 64, 128 and 256 locations as a function of the number of PEs in the network. We see that at low PE counts ($<$ 50 PEs), there is no difference between the different networks. At larger PE counts (50-300 PEs), the network with a buffer depth of 1 gets the worst performance. From the zoomed in version of the same plot shown in Figure 8.2, we can see that the best performance is achieved by the network with a buffer depth of 4. Thus, at these PE counts, having too few buffer slots causes network congestion while having too many buffer slots increases network latency leading to poor performance. But, once we factor in area, we see that

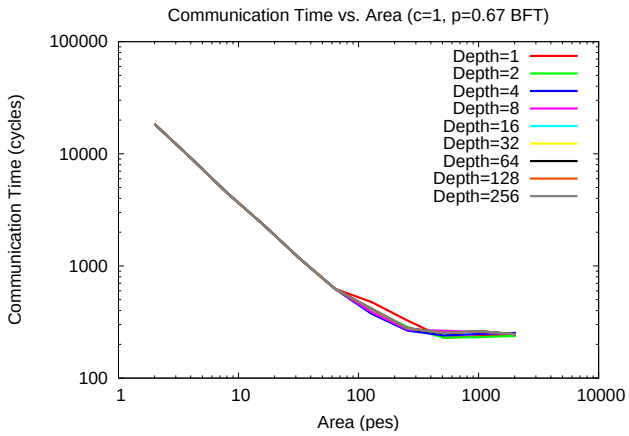


Figure 8.1: Communication Time vs. PEs for `gemat11` (SMVM) with different buffer depths

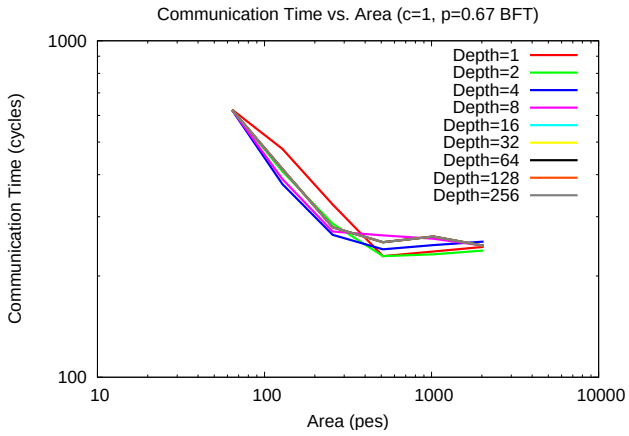


Figure 8.2: Zoom of Figure 8.1

networks with a buffer depth of 1 have the best performance for a given area over most of the area range as seen in Figure 8.3. We attribute this to the smaller area requirement of a 1-deep buffer. At larger areas, once the performance lowerbounds are achieved, there is no significant difference between the different networks.

8.2 Impact of Topology

We present four quantitative comparisons to explore the issues in designing large-scale network topologies. First, we examine how communication cycles scale with number of PEs for different topologies. In this exploration, we compare the performance achieved by a Virtual-Channel-based routing algorithms with deterministic and adaptive algorithms. We quantify the performance difference between BFTs and Meshes at different areas. We also show a comparison between Directional

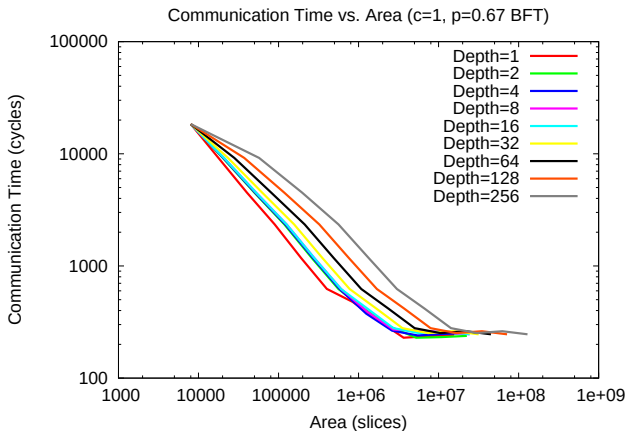


Figure 8.3: Communication Time vs. Slices for `gemat11` (SMVM) with different buffer depths

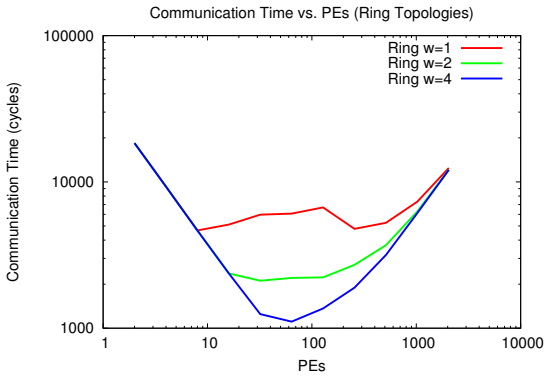
and Bidirectional Meshes. Second, we consider the impact of area and re-evaluate performance scaling of topologies. Third, we examine how topology performance corresponds to both area and application requirements. Fourth, we attempt to choose a single topology that is robust over all areas and for all applications, providing the best worst-case performance. Before presenting these comparisons, we briefly discuss the characteristics of our applications graphs to aid in understanding our results.

8.2.1 Effect of PE Scaling

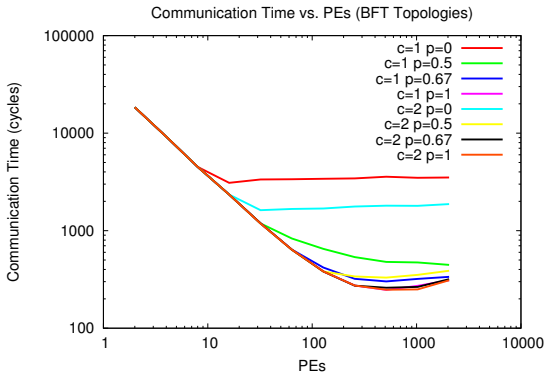
We first explore the effects of increasing the number of PEs on the performance of different topologies, using the `gemat11` graph as an example of the scaling trends of large-scale networks.

In Figure 8.4 we plot communication cycles as a function of PEs for all topologies routing the `gemat11` workload. `gemat11` contains 33K messages; therefore, we expect to it to be serialization limited until 100s of PEs. We see that at low PE counts (< 10 PEs) there is no significant performance difference between different topologies. Here, most of the traffic is local and the network is lightly loaded (see Figure 2.1, Section 3.3.1). This motivates us to further increase PE counts to understand when topologies start to differ.

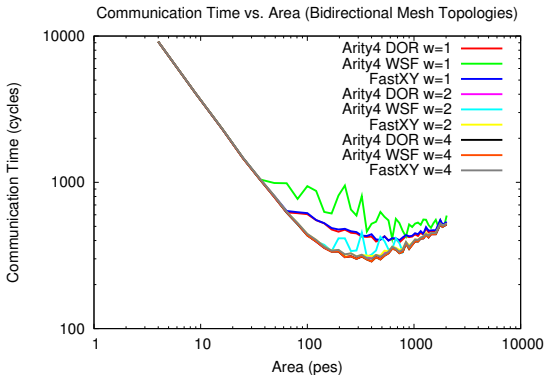
Ring As 10–100 PEs, rings start to bisection bottleneck . These networks have a constant number of channels crossing the bisection and are consequently bisection limited (Eq. 3.6). Rings with a larger channel-width ($W \geq 4$) show similar trends but achieve superior absolute performance than the lower channel-width rings. At very large PE counts, 100–1000 PEs, the rings are severely affected by latency and the number of cycles required to route all messages increases significantly.



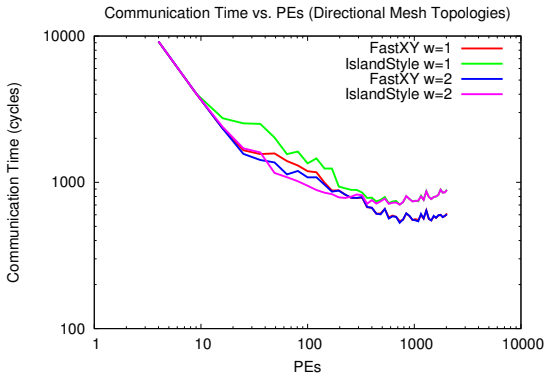
(a) `gemat11` Ring



(b) `gemat11` BFT



(c) `gemat11` Bidirectional Mesh



(d) `gemat11` Directional Mesh

Figure 8.4: Communication Time vs. Area

Mesh Directional Meshes begin to get bandwidth limited starting at 10 PEs. Directional meshes have a very restrictive set of routes that limit achievable performance. Bidirectional Meshes continue to scale well until 100 PEs after which they also start to bisection bottleneck. At very high PE counts ($>$ 1000 PEs), Bidirectional Meshes are eventually limited by latency. Meshes with large channel width achieve better absolute performance than the lower channel width networks. We also observe that DOR-based Mesh switches provide better performance than the WSF-based switches. We are currently trying to understand this counter-intuitive behavior. We expect WSF-based networks to provide performance that is at least as good as DOR, not worse.

BFT BFTs with different values of the Rent parameter p show different scaling trends. BFTs with a $p = 0$ behave similar to rings and are bisection limited at 10s of PEs. At larger PE counts ($>$ 1000 PEs) they begin to get latency limited. BFTs with a $p \geq 0.5$ scale well into the 100s of PEs range and are affected by latency only after 1000s of PEs.

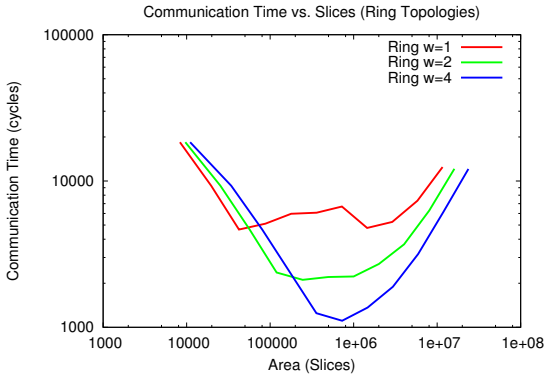
8.2.2 Effect of Area Scaling

In the previous example, we did not consider area requirements for our topologies. Topologies with equal PE counts may have very different area requirements as the number of switches and their sizes vary across topologies (Table 7.1). Additionally, application PEs have varying sizes. As PEs and interconnect use the same area resources on an FPGA, we can trade off area between PEs and interconnect.

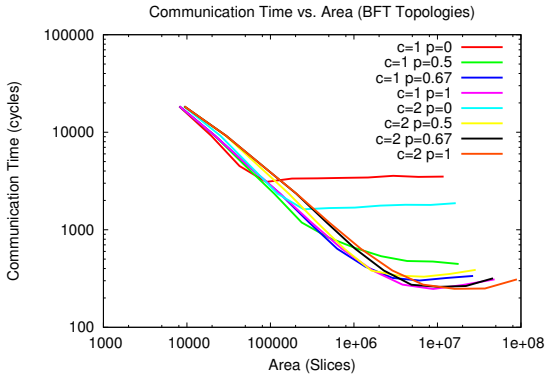
Ring We observe that rings with a larger channel width achieve better performance at larger chip areas. This is observed in the form of shifting of the curves previously seen in Figure 8.4. This is expected, since a ring with a channel width of W requires approximately W times more area than a ring with a channel width of 1. We see similar behavior when comparing other topologies as well.

Mesh In Directional Meshes, we find that the Island Style networks provide better performance only at small areas. The FastXY style switches deliver better performance at larger areas due to the latency optimized paths within the switches. Bidirectional Meshes with DOR routing completely dominate performance at all areas. DOR switches are smaller and were seen to achieve better performance even when scaling PEs, in Figure 8.4.

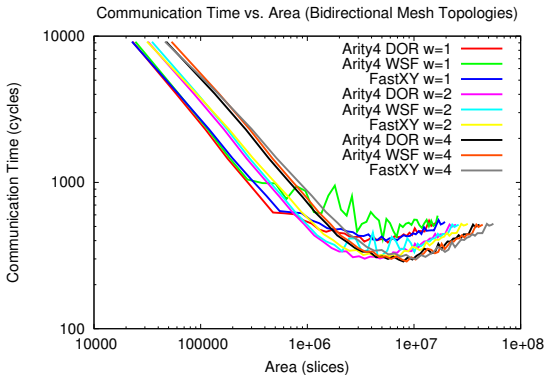
BFTs BFTs with a smaller p require less interconnect area. Hence, we observe that the best BFT topology is different in different area ranges. A $p = 0$ BFT is best below 50K slices, a $p = 0.5$ BFT is better until 200K slices, a $p = 0.67$ BFT achieves superior performance in the 200K-1M slices range and finally the $p = 1$ BFT dominates at areas above that.



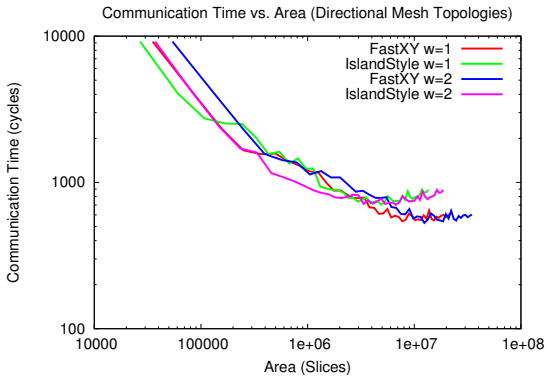
(a) `gemat11` Ring



(b) `gemat11` BFT



(c) `gemat11` Bidirectional Mesh



(d) `gemat11` Directional Mesh

Figure 8.5: Communication Time vs. Area

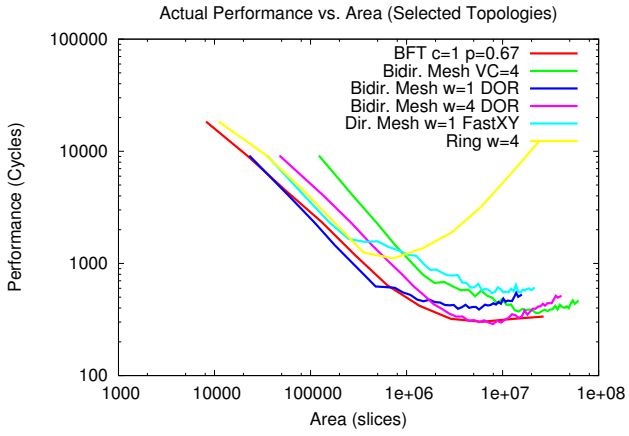


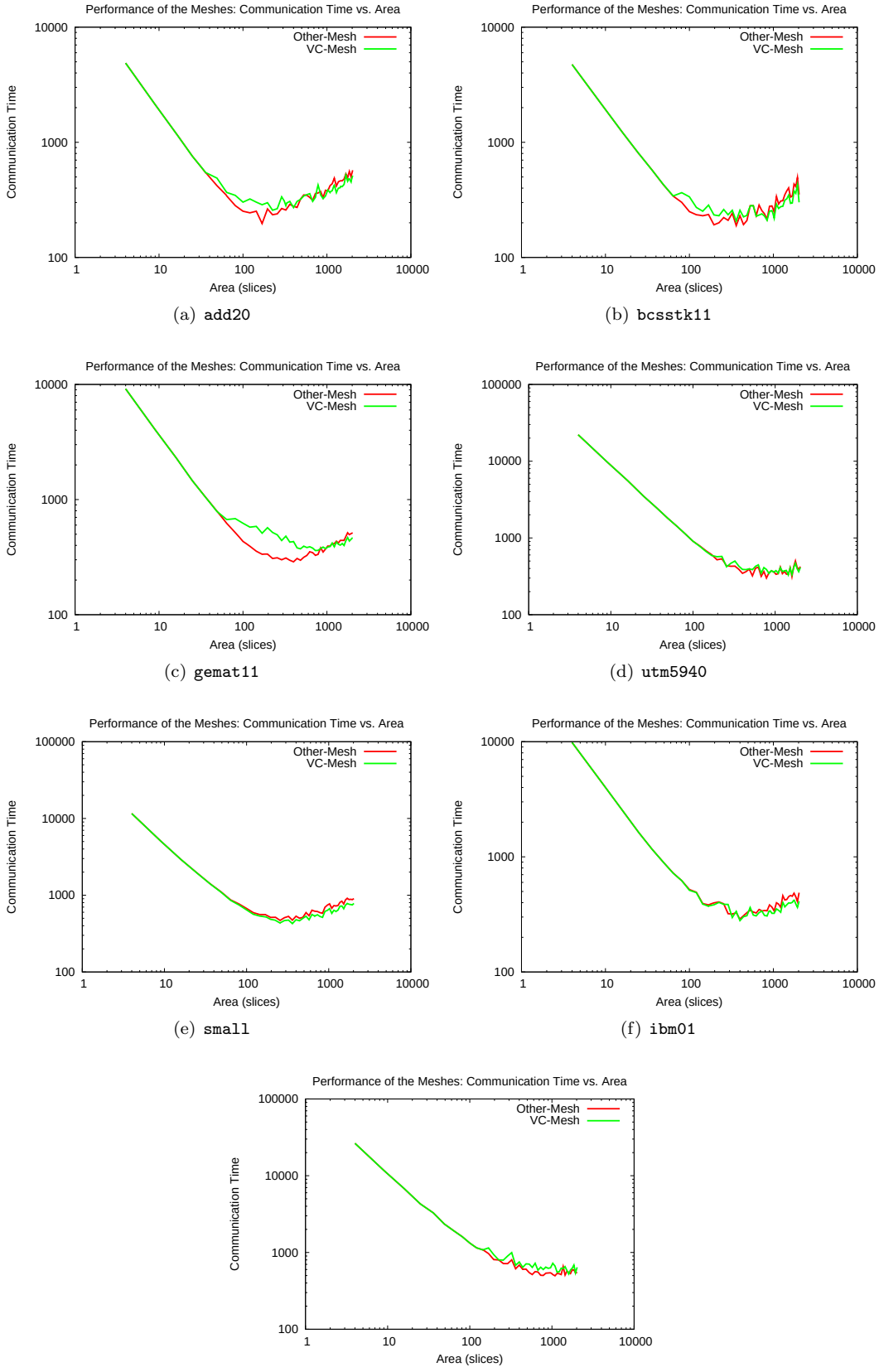
Figure 8.6: Actual Communication Time vs. PEs for `gemat11` (SMVM)

We plot a select few topologies from Figure 8.2.2 together in Figure 8.6. We can clearly see that Ring with $W = 4$ is outperformed by all other topologies and gets affected by latency sooner than the rest. We also find that Virtual-Chanel-based meshes require a much larger area to provide comparable performance as other topologies (see Section 8.2.3). We observe that Bidirectional Meshes outperform Directional Meshes are larger areas (see Section 8.2.4). When comparing the performance achieved by the BFTs and the Meshes, we see that they are very similar and neither one dominates the other (see Section 8.2.5).

8.2.3 Performance of Virtual-Channel-based Meshes

In the previous section, we showed how to compare the performance of different topologies by increasing PE counts and area. In this section, we study the impact of routing algorithms on the performance of the Virtual-Channel-based meshes when increasing PE counts and area. We implement meshes with different routing algorithms and would like to select the best performing algorithm. We can classify the meshes into two types, Virtual-Channel (VC) based and non-Virtual-Channel (non-VC) based meshes. As outlined in Section 5.3.2, VC based meshes can potentially improve the utilization of a given physical link by time-sharing the physical channel between multiple virtual channels. We also noted that this improved performance comes at the expense of additional hardware *i.e.* VC-based switches are larger that regular mesh switches. In both networks, we try to distribute load evenly between the different channels, virtual or physical.

In Figure 8.7, we plot the performance of the best VC based mesh and the best non-VC based mesh topologies. We observe that at low PE counts, there is no performance difference between these two types of meshes. At larger PE counts, the best performance achieved by the VC-based mesh is unable to beat the best performance of the non-VC based mesh (except `small` and `ibm01`). We attribute this performance difference to the larger bisection bandwidth possible in the non-VC based meshes at same PE counts. When considering area the VC based meshes achieve their peak performance at areas larger than non-VC meshes due to its larger area requirement. In Figure 8.8, we plot the ratio of non-VC-based Mesh communication time to VC-based Mesh communication time. If the ratio is < 1, then non-VC meshes provide better performance. The vertical lines represent the areas at which the respective topologies achieved their best performance (lowest cycle count for routing). We see that the non-VC-based meshes achieve their best performance much earlier than the VC-based meshes. We also observe that the performance achieved by the VC meshes is 10% less than the non-VC meshes and hence conclude that NoC designers should avoid using Virtual-Channels for pure on-chip networks. Virtual-Channel networks are to be preferred when there is a severe bandwidth constraint in the network which is not the case when designing networks inside a chip.



(a) `add20`

(b) `bcsstk11`

(c) `gemat11`

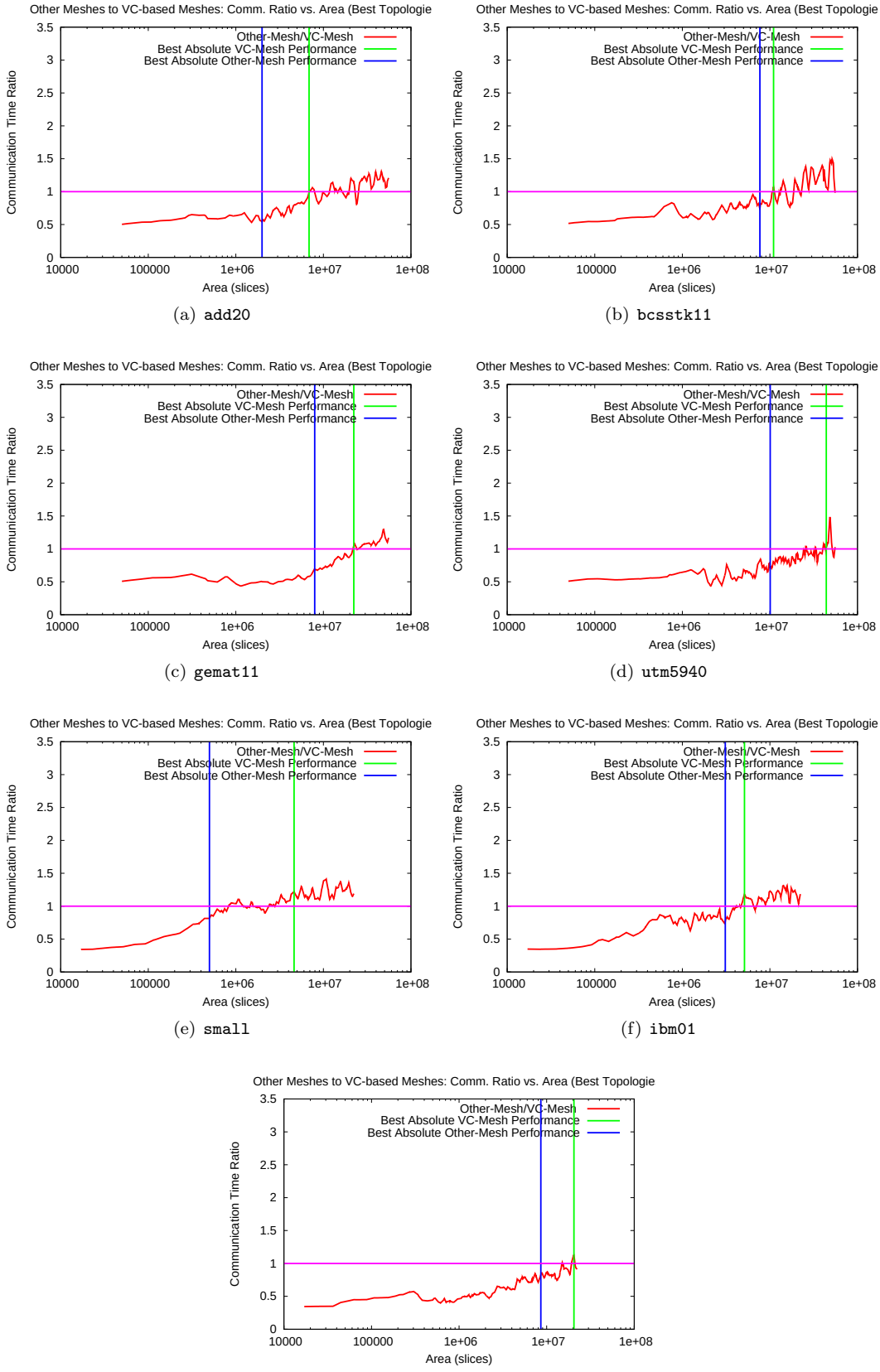
(d) `utm5940`

(e) `small`

(f) `ibm01`

(g) `ibm05`

Figure 8.7: Communication Time vs. PEs for non-VC and VC Meshes



(a) `add20` (b) `bcsstk11`

(c) `gemat11` (d) `utm5940`

(e) `small` (f) `ibm01`

(g) `ibm05`

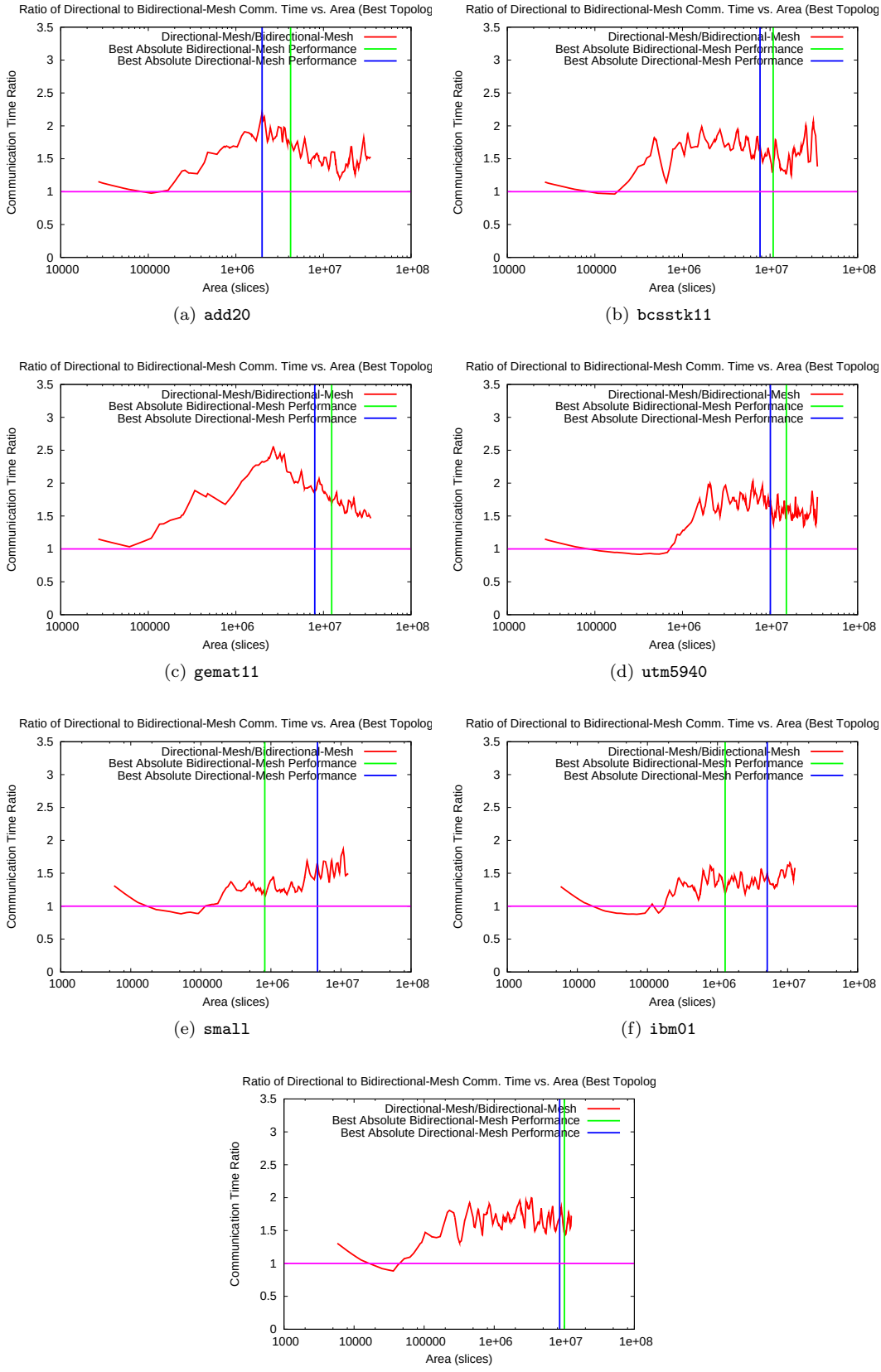
Figure 8.8: Ratio of Communication Time of non-VC and VC meshes vs. Area

8.2.4 Performance of Directional and Bidirectional Meshes

We build two kinds of mesh topologies based on the kind of connectivity provided between the switches, Bidirectional and Directional Meshes. Directional Meshes can be built with smaller switchboxes and can provide compact implementations at small PE counts. Bidirectional Meshes have slightly larger switches but provide more connectivity between the switches. We quantify the performance difference between these two kinds of topologies in Figure 8.9. We plot the ratio of communication cycles required by the Directional Mesh and Bidirectional Mesh vs. chip area. When the ratio is greater than 1, Bidirectional Meshes offer better performance. The vertical lines represent the areas at which the respective topologies achieved their best performance. We observe that Directional Meshes are superior to Bidirectional Meshes at small chip sizes when the performance is mostly serialization dominated. In these operating conditions, it is beneficial to sacrifice network richness for more PEs. After 100K slices, Bidirectional Meshes always outperform Directional Meshes. This factor can be asl large as 2.5 in some cases. In most cases (except `small`), the Bidirectional Mesh achieves the best performance at a smaller area than the Directional Mesh.

8.2.5 Performance of BFTs and Meshes

In Section 8.2.2, we noticed that there was no significant performance difference between the BFT and the Mesh. In Figure 8.10, we plot the ratio of communication cycles of a Mesh to a BFT vs. area. If the ratio is greater than 1, Mesh requires more communication cycles than the BFT. The vertical lines represent the areas at which the respective topologies achieved their best performance. We find that in BFT achieves the best performance as smaller areas in half the applications while the Mesh achieves the best performance earlier in the other half. We further observe that at low areas (below 50K slices), the Mesh requires 50% more cycles than a BFT at equivalent area. At larger areas (50K-10M slices), the Mesh achieves marginally better performance than the BFT. Mesh performance is better than the BFT by less than 20% in the most optimistic case. This difference in performance is due to the different worst case delay experienced by a packet on these topologies for a given placement. In Section 7.4, we had observed that the physical worst case latency (between extreme PEs) on a Mesh was larger than the BFT. The actual worst case latency observed on the Mesh, for the placements we used, was much smaller. This was because the worst case path for the given placement was between PEs that were much closer to each other than the physical worst case. On the BFT, however, the placement required the packet to cross the top-level switchbox to get to the other chip partition. This required the packet to traverse the physical worst case path on the network. Although the BFT shows better theoretical worst-case latency, when considering a real workload and a good placement, we find the exact opposite behavior. Thus, given a good placement, a BFT can never beat the Mesh in the current form. The BFT is, however, only within



(a) `add20`

(b) `bcsstk11`

(c) `gemat11`

(d) `utm5940`

(e) `small`

(f) `ibm01`

(g) `ibm05`

Figure 8.9: Ratio of Communication Time of Bidirectional Mesh/Directional Mesh vs. Area

20% of the best mesh performance for considered workloads and within 10% on average. We revisit this briefly in the Chapter 9 on Future Work.

8.2.6 Explaining Quality of Performance

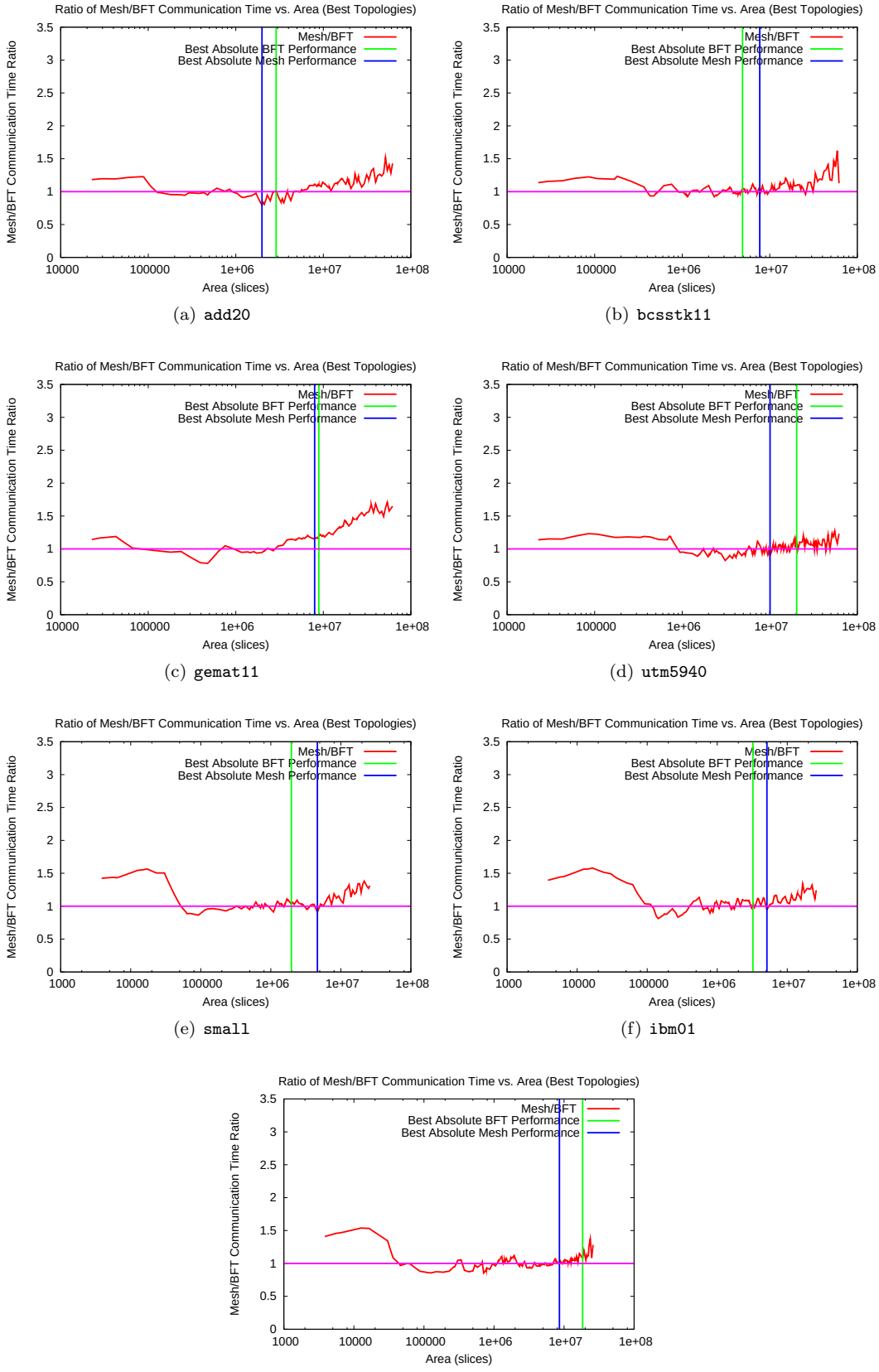
To help characterize why topologies perform differently at larger PE counts relative to application size, we plot the lowerbound performance (Section 3.3) of `gemat` in Figure 8.11 and the ratio of achieved performance to lowerbound performance in Figure 8.12. This helps us understand how close packet-switching can come to achieving lowerbound performance for individual topologies. We observe, in Figure 8.12, that rings are within 2× of their lowerbound. This is due to larger bandwidth provided by the multiple channels ($W = 4$ ring) considered for this comparison. We find that best performing BFTs are within 2.4×, Bidirectional Meshes are within 2.5× and Directional Meshes are within 5.1× of their respective lowerbounds. Directional Meshes have fewer possible routes available which make packet-switched routing on these networks harder than on richer Bidirectional Meshes. We also observe that the VC Mesh is 3× worse than the lowerbound at larger area. This is due to the insufficient routing freedom (bandwidth) in the network and the constraints imposed by the routing algorithm.

8.2.7 Effect of Application Communication Requirements

We plot communication cycles as a function of area in Figure 8.13 for a range of benchmarks: SMVM `rdb32001`, `gemat11`, `utm5940`, `fidap035` and ConceptNet `small`.

We can classify the performance of these applications into characteristic regions. As expected, we see that all topologies for all applications are initially serialization bottlenecked. As area (and consequently PEs) increases, topologies become bisection limited and eventually latency limited. We will see that the area ranges of these characteristic regions are directly correlated to application size.

Initially, performance of all topologies is serialization limited between 1K–500K slices for `gemat11` and `rdb32001` and 1K–1M slices for `utm5940` and `fidap035`. The serialization region for `utm5940` and `fidap035` extends beyond that of `gemat11` and `rdb32001` since the former graphs are larger and remained serialized at larger areas (Section 6.4). When only considering number of PEs, in the serialization limited region all topologies yield the same performance. When considering area, topologies with larger interconnect requirements require larger areas to achieve the same performance as topologies with less interconnect. We find that BFTs with a $p = 0$ provide the best performance in this area range. At large slice counts, 500K–10M slices for `gemat11` and `rdb32001`, 1M–50M slices for `utm5940` and `fidap035`), the network is more heavily loaded. We find that Bidirectional Meshes with DOR routing ($W = 1, W = 2$) and a BFT with $c = 1, p = 0.67$ achieve the best performance in this region. For large communication graphs (*i.e.* `fidap035`), we find that performance continues



(a) `add20` (b) `bcsstk11`

(c) `gemat11` (d) `utm5940`

(e) `small` (f) `ibm01`

(g) `ibm05`

Figure 8.10: Ratio of Communication Time of Mesh/BFT vs. Area

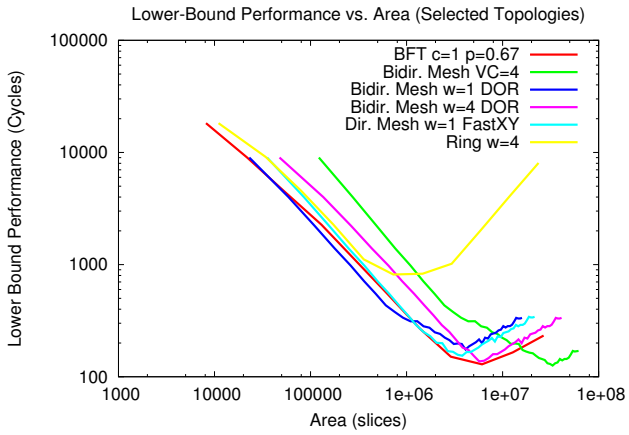


Figure 8.11: Lowerbound Communication Time vs. PEs for `gemat11` (SMVM)

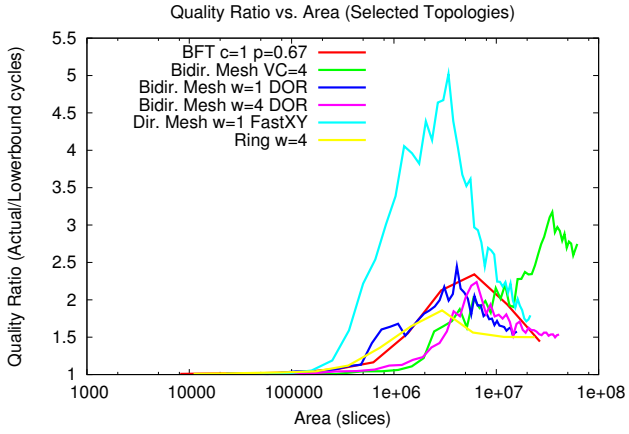
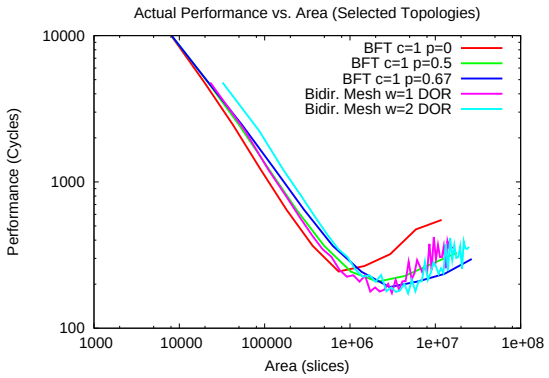
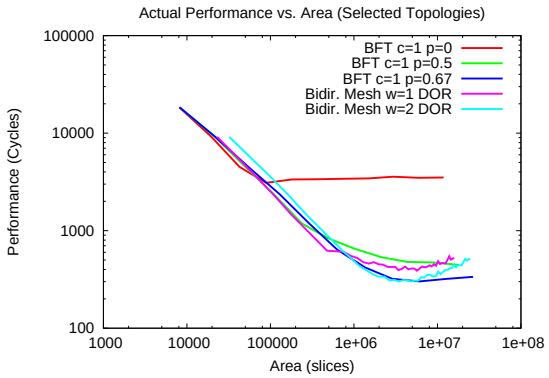


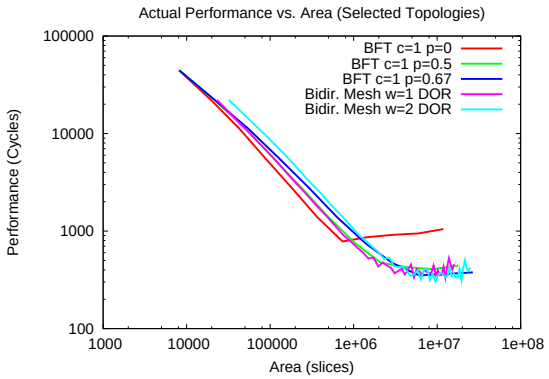
Figure 8.12: Ratio of Actual to Lowerbound Communication Time vs. PEs for `gemat11` (SMVM)



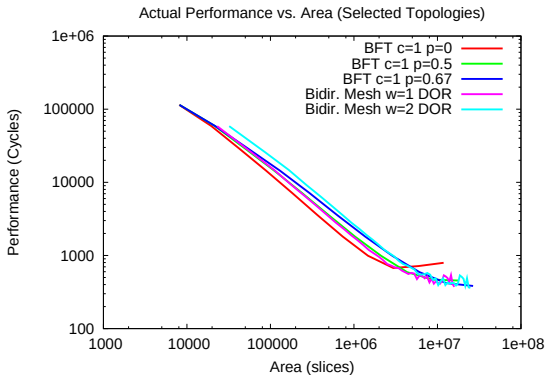
(a) `rdb32001` (18K messages)



(b) `gemat11` (33K messages)



(c) `utm5940` (83K messages)



(d) `fidap035` (218K messages)

Figure 8.13: Communication Time vs. Area

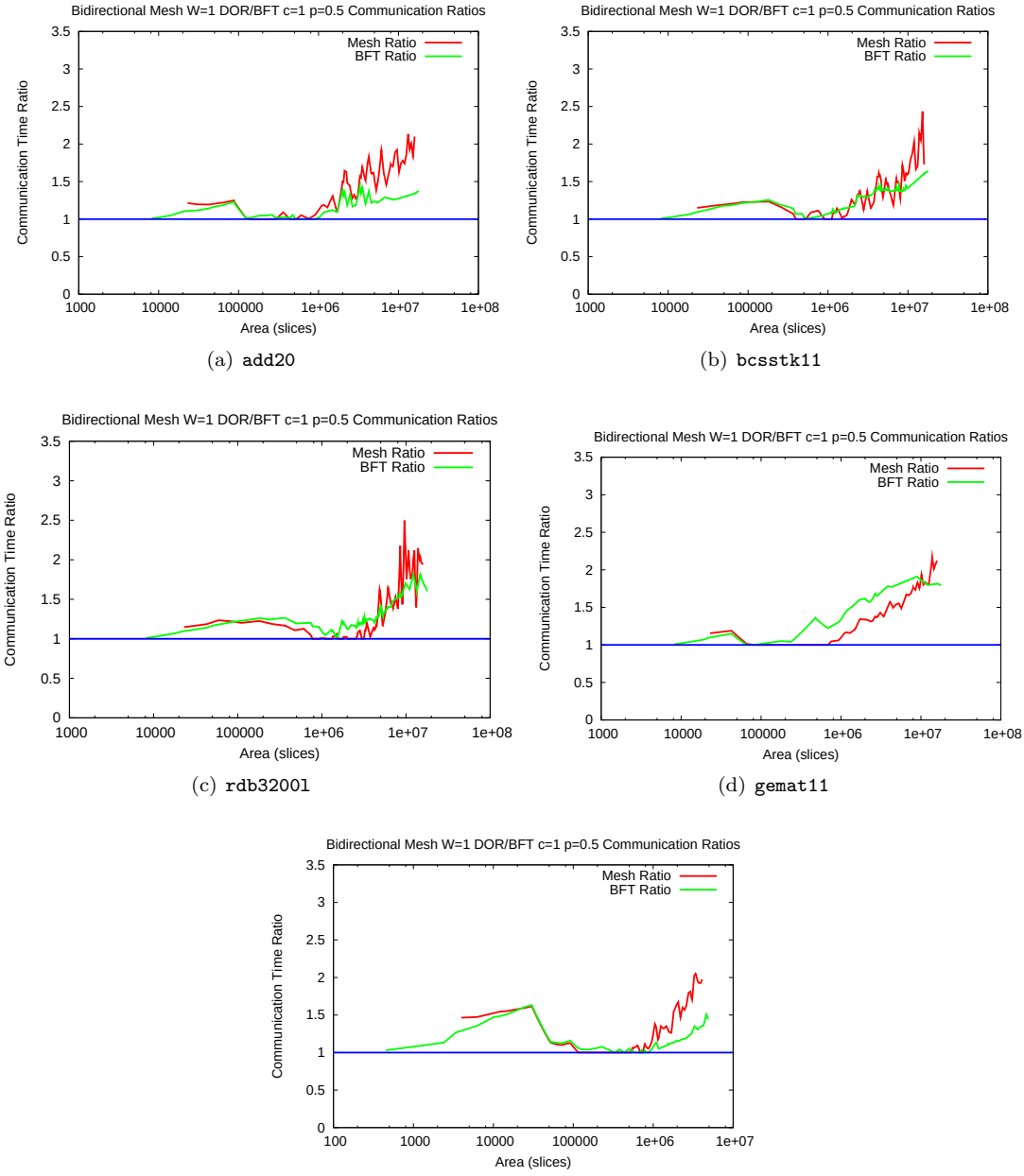
to remain serialized, even for the largest areas considered. We generally observe that, as we increase area, topologies that provide more interconnect achieve better performance. However, at very high chip sizes, performance begins to degrade due to latency and it is preferable to build smaller networks with less area to get better performance.

8.2.8 Universal Topology

In choosing a network topology, designers may not have the opportunity to know the communication requirements of all applications apriori. Additionally, these networks may need to scale to larger areas as design requirements change. Ideally, an NoC designer would like to be able to choose a single topology that performs robustly over all applications and areas. But, we observe that different topologies perform optimally for different areas and applications. This indicates that designing a network with a fixed topology could be inefficient over a range of applications and network sizes [37], [17]. To help quantify the cost of a "one topology fits all" methodology, we compute the ratio of actual performance achieved by each topology to the performance achieved the best topology at a given area. This ratio is always ≥ 1 and lower the value of this ratio, the more robust the topology. We compute this ratio over all areas and over all applications. We tabulate the worst case ratios for for each topology in Table 8.1. We see that the most robust topology, a BFT with $c = 1, p = 0.5$, requires as many as 1.9× the number of cycles required by the best topology. We also observe that the best performing mesh has a worst case ratio of 2.5 which is similar to BFTs. We plot these ratios as a function of area in Figure 8.14. It shows that the worst case is observed only at very large areas and in the average case this ratio is around 1.5. This still indicates that there is a non-trivial cost in selecting a single topology for all areas and applications, motivating application specific topology design.

8.3 Comparison with Time Multiplexing

Selection of an appropriate communication pattern should be based on application communication requirements and characteristics. Time-multiplexing uses static scheduling which requires communication to be predictable and known ahead of time. Packet-switching has no such needs. It is possible to dynamically inject traffic into the network at any time. Hardware requirements are particularly important in comparing packet-switching and time-multiplexing. Packet-switching typically requires larger switchboxes due to buffering and logic required for dynamic decision making (Section 7.3). Time multiplexed switchboxes are typically smaller in terms of raw logic, but if the total number of communication cycles is large, switch context memory may require significant area. To illustrate these area and performance trade-offs, consider the following example. Packet-switched switchboxes may allow us to fit an 8 PE network on a chip. With comparatively smaller switches we could fit



(a) `add20`

(b) `bcsstk11`

(c) `rdb32001`

(d) `gemat11`

(e) `small`

Figure 8.14: Ratio of Actual Communication Time of Bidirectional Mesh W=1 *DOR* and BFT c=1 p=0.5 to Best Communication Time vs. Area

Topology	Worst Case Performance Ratio
BFT $c = 1, p = 0$	14.1
BFT $c = 1, p = 0.5$	**1.9**
BFT $c = 1, p = 0.67$	2.1
BFT $c = 1, p = 1$	2.1
BFT $c = 2, p = 0$	7.5
BFT $c = 2, p = 0.5$	2.8
BFT $c = 2, p = 0.67$	3.4
BFT $c = 2, p = 1$	6.4
Directional Mesh - Island Style $w = 1$	4.0
Directional Mesh - Island Style $w = 2$	6.4
Directional Mesh - FastXY $w = 1$	3.9
Directional Mesh - FastXY $w = 2$	4.9
Bidirectional Mesh - Arity 4 DOR $w = 1$	2.5
Bidirectional Mesh - Arity 4 DOR $w = 2$	2.9
Bidirectional Mesh - Arity 4 DOR $w = 4$	4.7
Bidirectional Mesh - Arity 4 WSF $w = 1$	4.8
Bidirectional Mesh - Arity 4 WSF $w = 2$	8.5
Bidirectional Mesh - Arity 4 WSF $w = 4$	17.0
Bidirectional Mesh - FastXY $w = 1$	2.5
Bidirectional Mesh - FastXY $w = 2$	3.1
Bidirectional Mesh - FastXY $w = 4$	5.0
Bidirectional Mesh - Duato $VC = 2$	4.6
Bidirectional Mesh - Duato $VC = 4$	11.0
Ring $w = 1$	48.8
Ring $w = 2$	48.8
Ring $w = 4$	48.5

Table 8.1: Ratio of Achieved Performance to Best Performance (Over All Areas and Applications)

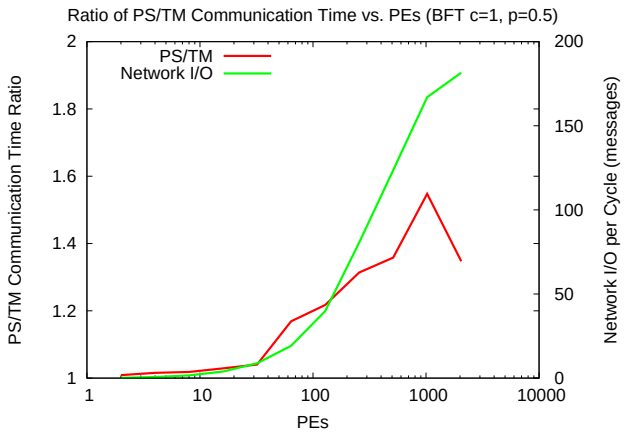


Figure 8.15: Ratio of Time-Multiplexed/Packet-Switched Communication Time for Identical Topologies

a 16 PE time-multiplexed network as long as routing could be completed in 30K cycles. If routing takes more than 30K cycles we can fit only 8 PEs, and at more than 100K cycles we can only fit 4 PEs. Consequently, there are operating ranges where either network my outperform the other.

To setup the analysis, we first provide a quantitative comparison between the performance achievable by offline and online scheduling. For this, we route for identical 100% activity communication loads on identical topologies with the same number of PEs. Next, we consider the impact of area and compare the performance difference between packet-switching and time-multiplexing for identical areas. We also examine the balance between compute and interconnect in order to determine the correct topologies to compare across area. Finally, for given area capacities we examine performance while varying the activity factor of our communication loads. This helps establish the space in which packet-switching outperforms time-multiplexing as a function of routed messages.

8.3.1 Effect of PE Scaling

To characterize the inherent performance difference between offline and online routing, we routed 100% activity communication loads on equivalent topologies, measuring the total number of communication cycles required to route. We collected data for BFTs over a range of channel widths ($c = 1, 2$) and Rent parameters ($p = 0, 0.5, 0.67, 1$). The ratio of packet-switched to time-multiplexed communication time as a function of PEs for a representative BFT with $c = 1$ and $p = 0.5$ is shown in Figure 8.15. Network I/O per cycle (as in Figure 2.1) is also shown on the same graph. At low numbers of PEs (<100) we see that offline and online routing produce equivalent cycle counts. Small numbers of PEs induce a light communication load (1-20 messages per cycle) and little offline/online differentiation. As the number of PEs increase (>100) the communication load increases (100s of messages per cycle), and we begin to see offline routing outperform online routing. Offline routing is

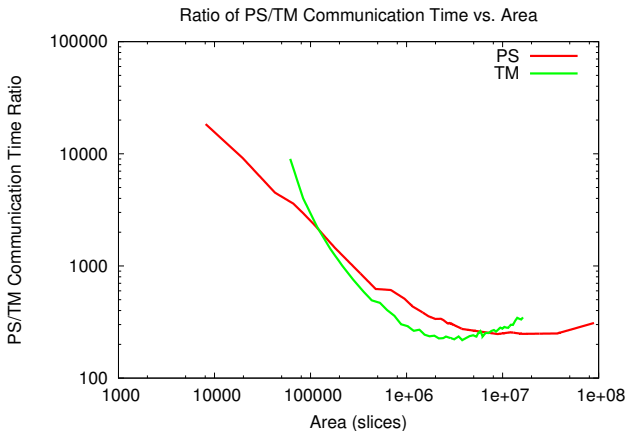


Figure 8.16: Communication Cycles vs. Area for Best Topologies

able to take advantage of global information to make optimal routing decisions on larger networks, while online routing is limited to making local decisions which affect the overall quality of route. As a result, online routing requires up to 50% more cycles than offline routing to route larger networks.

8.3.2 Effect of Area Scaling

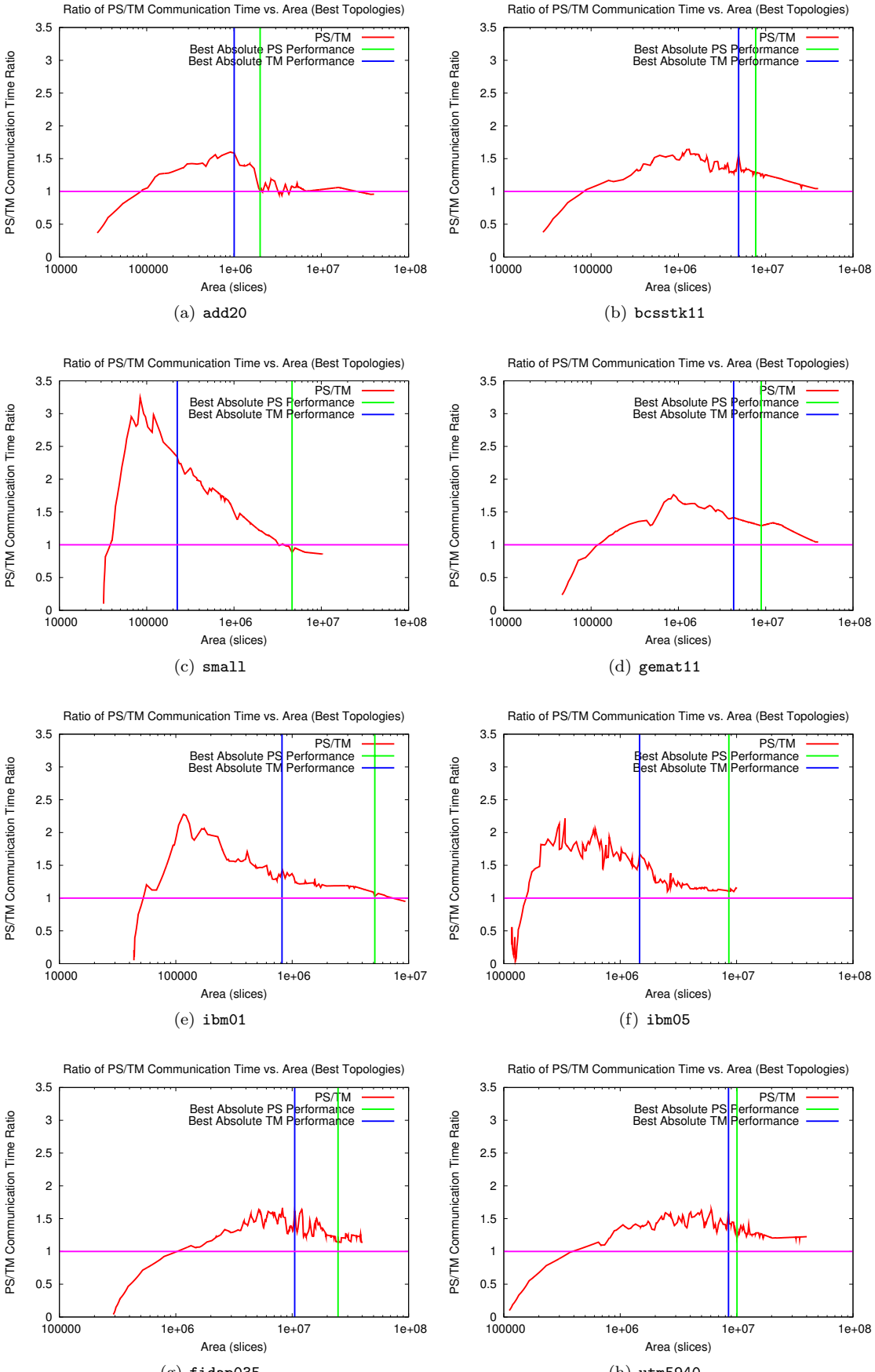
To fully characterize the performance difference between our packet-switched and time-multiplexed networks we must also consider the area they consume. For equivalent topologies at the same PE count, packet-switched and time-multiplexed networks may use significantly different amounts of area due to differences in switch sizes and the amount time-multiplexed context memory needed. To fairly compare this area-time trade off between time-multiplexing and packet-switching, we examine the best packet-switched and time-multiplexed topologies over all area points in our comparison. The composite area-time curves for packet-switched and time-multiplexed networks are shown in Figure 8.16. At smaller areas (<12K slices) time-multiplexing requires more cycles to route than packet-switching. Smaller areas fit fewer PEs, resulting in higher cycle counts. As area increases, however, time-multiplexing can fit more PEs, decreasing serialization and reducing cycle counts. Finally, at large areas, time-multiplexed networks start requiring more area due to increasing latency.

To quantify this trade off, we show the ratio of packet-switched to time-multiplexed communication time as a function of area over these optimal topology points in Figure 8.17. Below 30K slices, the context required to hold the instruction memory in the PEs is very large. This excessive context memory requirement makes time-multiplexed networks infeasible below 30K slices. At small areas above this threshold, 30K–50K slices for `add20`, `bcsstk11`, `small` and `ibm01`, 30K–100K slices for `ibm05`, `utm5940` and `fidap035`, time multiplexing is inefficient and requires between 5-10× as many cycles to route as packet switching. At larger areas (50K–2M slices for `add20`, `bcsstk11`, `small` and

ibm01, and 1M-20M slices for utm5940 and fidap035) packet-switching requires 1-3.5× as many cycles to route as time-multiplexing, in the worst case. On an average packet switching is only 1.5× worse than time multiplexing. The worst case performance ratio of 3.5× is achieved when routing a decomposed ConceptNet workload. Time-multiplexed router uses a tighter routing strategy when routing these decomposed edges than packet switching. Also, at the largest area values (above 1M slices for add20, bcsstk11, small and ibm01, and above 20M slices for utm5940 and fidap035), time multiplexing's advantage begins to shrink down to around 1-1.2× (occasionally also dropping below packet switching). This is due to time-multiplexed performance being limited by edge fanout or fanin limit and latency starting to dominate the context requirement. During this interval the packet-switched network is able to close the performance gap.

8.3.3 Effect of Activity Factor

So far we have compared packet-switching and time-multiplexing assuming 100% communication loads. When routing ConceptNet queries (Section 6.1) we may be required to communicate only a fraction of the total workload. We call this fraction activity factor. Different queries touch different portions of the graph. Hence, the magnitude of activity factor varies with the query. The time-multiplexed router has no prior knowledge of which exact subset of messages will need to routed for the different queries. Hence, it is required to route the full 100% workload. On the other hand, packet-switching only needs to route those edges that are active. At some activity factor less than 100% packet-switching should be able to outperform time-multiplexing when both are given the same amount of area. In Figure 8.18, we plot communication cycles vs. activity for the a sample topology with 256 PEs. We observe that below 10% activity, packet switching requires fewer cycles to route the partial workload than time multiplexing. The exact fraction at which this crossover occurs depends on the size of the workload and the network. In Figure 8.19 we plot activity crossover as a function of PEs by choosing only the best performing topologies at those PE counts. If the crossover is high (50–100%), it is best to use packet switching for the network. If the crossover is less (1–50%), it is best to use time multiplexing in the network. At small PE counts, packet switching can match the performance of time multiplexing even at 100% activity. This is mainly because performance is fully serialized. At larger PE counts, once the network is no longer serialization bottlenecked, packet switching can match time multiplexing only at smaller activity factors. After about 100-200 PEs, packet switching is able to get better performance than time multiplexing only below 1% activity.



(a) `add20`

(b) `bcsstk11`

(c) `small`

(d) `gemat11`

(e) `ibm01`

(f) `ibm05`

(g) `fidap035`

(h) `utm5940`

Figure 8.17: Ratio of Time-Multiplexed/Packet-Switched Communication Time for Identical Area

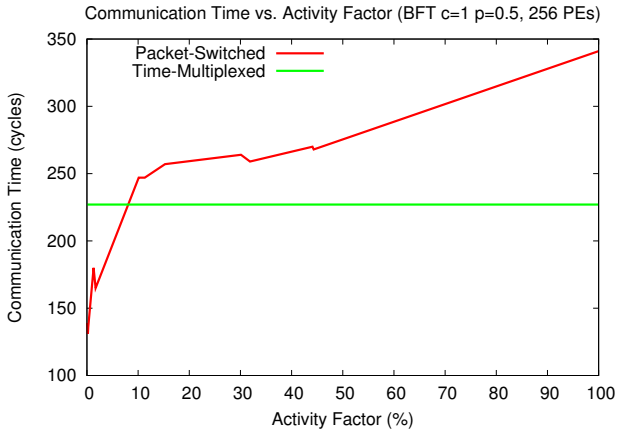


Figure 8.18: Communication Cycles vs. Activity for Sample Topology with 256 PEs

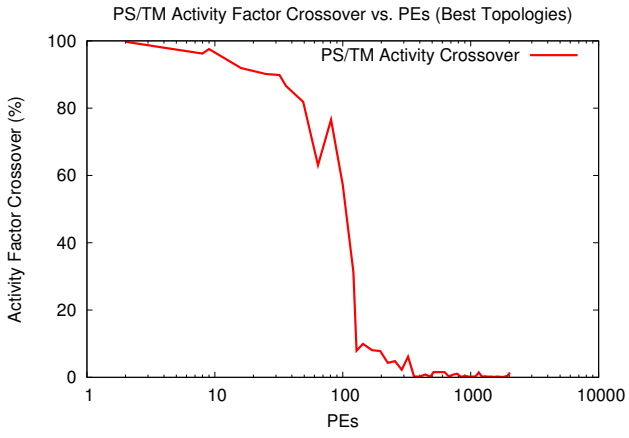


Figure 8.19: Activity Crossover vs. PEs

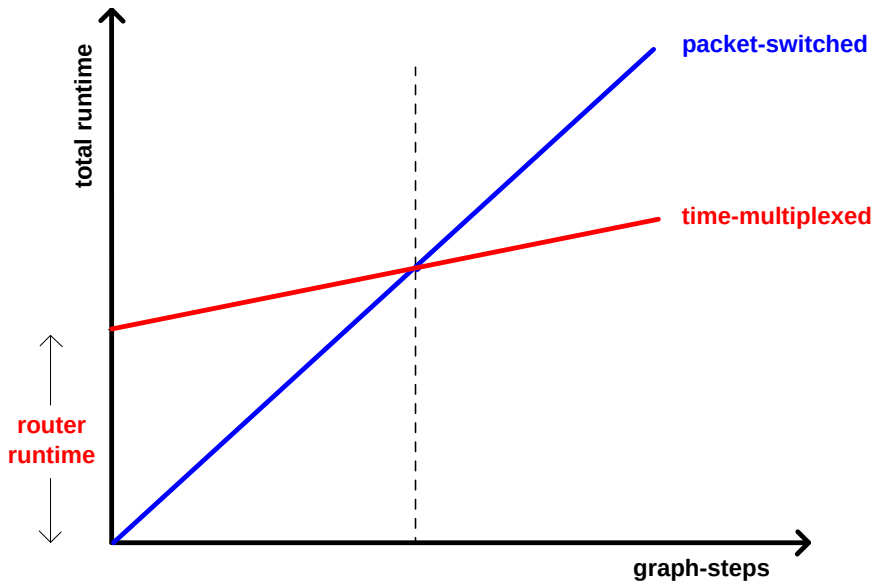


Figure 8.20: Performance of Time-Multiplexing vs. Packet-Switching as a function of Graph Steps

8.3.4 Effect of Multiple Graph Steps

We have considered the performance from a single graph step (Section 6) in our analysis so far. A typical application run could consist of several graph steps ranging from a few (< 8) for ConceptNet queries to thousands for SMVM. We have also not accounted for the runtime of the offline router. We've already shown that at sufficiently large chip areas, the performance achieved by the statically-scheduled time-multiplexed network is superior to the dynamically-routed packet-switched network. When we consider complete execution time that includes the runtime of the router we see a further separation of application scenarios between the two styles. For some applications with a few graph steps, the offline router runtime may be a significant overhead. If we graph the total runtime of the these two networks vs the total number of graph steps in Figure 8.20 (not real data), we will notice two distinct regions. In the first region, the performance of packet-switching is better than time-multiplexing due to the high overhead of offline scheduling. In the second region, after the total number of graph steps (number of times the routed schedule is reused) exceeds a certain threshold, the time-multiplexing outperforms packet-switching. In this region, the initial cost of computing the time-multiplexed schedule has been amortized across the multiple graph steps and we can take advantage the better route quality. Thus, in applications where the statically routed schedule need not be used several times (below the crossover threshold), packet-switching is the preferred switching style. The exact crossover depends of the size of the application and the number of graph steps required to be routed.

Chapter 9

Future Work

Communication patterns and densities can vary greatly between different applications. Exploration of more applications would help us better characterize the trade-offs between different topologies and between packet switching and time multiplexing. We are particularly interested in mapping larger communication graphs with fanin and fanout decomposition of the graph nodes to more evenly distribute the communication load.

We've observed that no single topology performs best over different applications. Ideally, one would want to synthesize a topology customized for every application so as to maximize performance. Hence, we are motivated to automate this process of topology selection based on a given application workload.

Previously, higher dimensional meshes such as as k-ary n-cubes were shown to be less efficient than two-dimensional meshes due to pin constraints [13]. As NoCs are not I/O limited like multiprocessor networks, it may be worth revisiting higher dimensional mesh topologies in the future under an appropriately revised cost model.

We currently implement only Dimension Ordered Routing on the Directional Mesh. We would like to extend that to include other adaptive schemes. We would also like to fully understand the differences in performance due to DOR and WSF routing algorithms on Bidirectional Meshes.

In Section 8.2.5, we had concluded that it was not possible for the BFT to achieve better performance than the Mesh, given a good placement. We can, however, attempt to reduce the worst-case latency on a BFT by adding shortcut connections between the switches as in [18]. Packets routing on these shortcut connections do not experience the worst-case latency in the network by avoiding the switches in the upper levels of the tree. This optimization is required only if performance on the topology is latency limited.

For this study, we restricted our analysis to reconfigurable FPGA substrates. However, we would still like to revisit this analysis under the ASIC cost model to see how the conclusions differ because of a revised area model for the switches and interconnect. We would also like to evaluate the sensitivity of changing technology parameters (switch delay, wire delay, gate area, power and

others) on performance of the network. A custom network implementation can be later integrated into the FPGA fabric itself to avoid the large area cost of overlaying the network.

Our network design space exploration has so far been limited to single chip networks. We hope to extend this work to multiple-chip networks, examining similar area-time trade-offs. We would want to build special switchboxes for handling inter-chip traffic and separate the on-chip and off-chip networks for simplicity of routing.

Chapter 10

Conclusions

We demonstrate scalable, high performance implementations of packet-switched FPGA overlay networks operating at 166MHz. We compose our switches using fundamental split and merge primitives that can be pipelined independently and composed to form switches. We pipeline the PE datapath and long wires in the interconnect to achieve this high-speed operation. The final system performance is limited by the critical path in the PE datapath, not the switches. The switches in our network are capable of running at speeds in excess of 200 MHz.

When designing our switching primitives, we observed that networks with a buffer-depth of 1 provide the best performance at a given area than networks with larger buffers. This makes all our networks cut-through and significantly lowers the area requried to implement the primitives.

We show that when the number of PEs is small with respect to application size, choice of network topology does not impact performance as applications are mostly serialized. At larger PE counts (100-1000s of PEs), topologies begin to show characteristic scaling trends. Performance of rings is limited by latency between 10–100s of PEs. BFT and Mesh performances can be distinguished only between 100–1000s of PEs.

We compared different kinds of rings, BFTs and Meshes for quantifying topology selection. As expected, we found that rings scale poorly and become latency bottlenecked at high PE counts. We observed that BFTs and Meshes provide similar performance and occasionally when the Mesh has better performance, it is within 20% of the BFT performance. We created two kinds of Meshes, Directional and Bidirectional. When we compared the performance of these two Meshes, we found the Bidirectional Mesh to provide much better performace at large areas and consistently across all applications. Directional Meshes provided better performance at small chip sizes due to their smaller switches. We further compare Virtual-Channel-based Mesh topologies with the Meshes that used adaptive or deterministic routing algorithms. We found that VC based Meshes were unable to provide better absolute performance even at equivalent PE counts (except for 2 benchmarks). Non-VC meshes were able to outperform VC based Meshes at equivalent area by as much as 10% because they can accomodate more channels *i.e.* provide more bisection. Since VC-based Meshes

use signficantly larger switches, their best performance is achieved at much larger areas.

We observe that different applications have distinct communication requirements depending on size and communication locality. When measuring the performance of a given topology we found that it varies significantly over $1K - 10M$ slices and across all of our benchmarks. Therefore, there is no single topology that routes all applications well for all areas. This is illustrated by the BFT $c = 1, p = 0.5$ topology which requires as many as 1.9× the number of cycles as the best topology to route all applications for all areas.

To aid designers in choosing the appropriate communication pattern between time-multiplexing and packet-switching, we characterize the tradeoffs associated with these networks and quantify the application conditions under which each is preferred. For our set of applications, offline scheduling offers up to a 50% performance increase over online scheduling for equivalent topologies. Time multiplexing provides better overall performance because, when computing an offline schedule, it can take advantage of the global state of the network for routing the workload. Packet switching is dynamic and routing decisions are computed locally in the switches. At equivalent areas, packet-switching is 5-10× faster for small areas while time-multiplexing is up to 1.5× faster for larger areas (up to 3× faster in the extreme case). At smaller PE counts, time-multiplexed networks have excessive PE instruction context requirements and need a large area for the mapping. At larger PE counts, the area requirement of the packet-switched network is dominated by the larger switches and time-multiplexed networks are able to provide better performance. At sufficiently large areas, packet-switched network are again able to match the performance of time-multiplexed networks since those networks begin to get latency bottlenecked. Below 100 PEs, packet-switching offers better performance than time-multiplexing for activity factors between 50–100%. Above 100 PEs, packet-switching can provide better performance only when activity factors are between 1–50%.

Bibliography

[1] A. Adriahantenaina, H. Charlery, A. Greiner, L. Mortiez, and C. A. Zeferino. SPIN: a Scalable, Packet Switched, On-chip Micro-network. In *Proceedings of the Conference and Exhibition on Design, Automation and Test in Europe*, pages 1128–1129, 2003.

[2] C. J. Alpert. The ISPD98 Circuit Benchmark Suite. In *Proceedings of the International Symposium on Physical Design*, pages 80–85, 1998.

[3] ARM. *AMBA Bus specification*, 1999.

[4] V. Beneš. Permutation Groups, Complexes, and Rearrangeable Multistage Connecting Networks. *Bell System Technical Journal*, 43:1619–1640, July 1964.

[5] V. Beneš. *Mathematical Theory of Connecting Networks and Telephone Traffic.* Academic Press, New York, NY, 1965.

[6] L. Benini and G. D. Micheli. Networks on Chips: A New SOC Paradigm. *IEEE Computer*, 35(1):70–78, 2002.

[7] V. Betz and J. Rose. Cluster-Based Logic Blocks for FPGAs: Area-Efficiency vs. Input Sharing and Size. In *Proceedings of the IEEE Custom Integrated Circuits Conference*, pages 551–554, May 1997.

[8] V. Betz, J. Rose, and A. Marquardt. *Architecture and CAD for Deep-Submicron FPGAs.* Kluwer Academic Publishers, 101 Philip Drive, Assinippi Park, Norwell, Massachusetts, 02061 USA, 1999.

[9] A. Caldwell, A. Kahng, and I. Markov. Improved Algorithms for Hypergraph Bipartitioning. In *Proceedings of the Asia and South Pacific Design Automation Conference*, pages 661–666, January 2000.

[10] R. J. Chapuis. *100 Years of Telephone Switching: Electronics, Computers and Telephone Switching, 1960-1985.* IOS Press, 2003.

[11] C. Clos. A study of non-blocking switching networks. *Bell Systems Techical Journal*, 32(2):406–424, March 1953.

[12] Cray Research, Inc. *CRAY T3D System Architecture Overview Manual*, 1995. URL `http://www.cray.com/PUBLIC/product-info/mpp/T3D_Architecture_over/T3D.overview.html`.

[13] W. Dally. Performance Analysis of k-ary n-cube Interconnection Networks. *IEEE Transactions on Computers*, 39(6):775–785, June 1990.

[14] W. J. Dally and C. L. Sietz. *The Torus Routing Chip*, volume 1 of *Distributed Computing*, pages 187–196. Springer-Verlag, 1986.

[15] W. J. Dally and B. Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Design Automation Conference*, pages 684–689, 2001.

[16] W. J. Dally and B. Towles. *Principles and Practices of Interconnection Networks*. Morgan Kauffman, 2004.

[17] A. DeHon. Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization). In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 69–78, February 1999.

[18] A. DeHon. Unifying Mesh- and Tree-Based Programmable Interconnect. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 12(10):1051–1065, October 2004.

[19] A. DeHon, J. Adams, M. deLorimier, N. Kapre, Y. Matsuda, H. Naeimi, M. Vanier, and M. Wrighton. Design Patterns for Reconfigurable Computing. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 13–23, April 2004.

[20] A. DeHon, R. Huang, and J. Wawrzynek. Hardware-Assisted Fast Routing. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 205–215, April 2002.

[21] M. deLorimier and A. DeHon. Floating-Point Sparse Matrix-Vector Multiply for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 75–85, February 2005.

[22] M. deLorimier, N. Kapre, N. Mehta, D. Rizzo, I. Eslick, R. Rubin, T. E. Uribe, T. F. Knight, Jr., and A. DeHon. GraphStep: A System Architecture for Sparse-Graph Algorithms. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2006.

[23] J. Duato, S. Yalamanchili, and L. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann Publishers, San Francisco, 2003.

[24] C. J. Glass and L. M. Ni. The Turn Model for Adaptive Routing. Technical report, Department of Computer Science, Michigan State University, 1991.

[25] R. I. Greenberg and C. E. Leiserson. *Randomness in Computation*, volume 5 of *Advances in Computing Research*, chapter Randomized Routing on Fat-Trees. JAI Press, 1988. Earlier version MIT/LCS/TM-307.

[26] A. K. Gupta and W. J. Dally. Topology Optimization of Interconnection Networks. *Computer Architecture Letters*, 4, July 2005.

[27] W. H. Ho and T. M. Pinkston. A Design Methodology for Efficient Application-Specific On-Chip Interconnects. *IEEE Transactions On Parallel and Distributed Systems*, 17(2):174–190, February 2006.

[28] IBM. *CoreConnect Specification*, 1999.

[29] N. Kapre, N. Mehta, M. deLorimier, R. Rubin, H. Barnor, M. J. Wilson, M. Wrighton, and A. DeHon. Packet-Switched vs. Time-Multiplexed FPGA Overlay Networks. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE, 2006.

[30] F. Karim, A. Nguyen, and S. Dey. An Interconnect Architecture for Networking System on Chips. *IEEE Micro*, 22(5):36–45, September/October 2002.

[31] C. P. Kruskal and M. Snir. The Performance of Multistage Interconnection Networks for Multiprocessors. *IEEE Transactions on Computers*, C-32(12):1091–1098, Dec. 1983.

[32] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. berg, K. Tiensyrj, and A. Hemani. A Network on Chip Architecture and Design Methodology. In *International Symposium on VLSI*, pages 117–124, 2002.

[33] B. S. Landman and R. L. Russo. On Pin Versus Block Relationship for Partitions of Logic Circuits. *IEEE Transactions on Computers*, 20:1469–1479, 1971.

[34] S.-J. Lee, K. Lee, S.-J. Song, and H.-J. Yoo. Packet-Switched On-Chip Interconnection Network for System-On-Chip Applications. *IEEE Transactions on Circuits & Systems*, 52(6):308–312, June 2005.

[35] F. T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes*. Morgan Kaufmann Publishers, Inc., 1992.

[36] C. Leiserson, F. Rose, and J. Saxe. Optimizing Synchronous Circuitry by Retiming. In *Third Caltech Conference On VLSI*, March 1983.

[37] C. E. Leiserson. Fat-Trees: Universal Networks for Hardware Efficient Supercomputing. *IEEE Transactions on Computers*, C-34(10):892–901, Oct. 1985.

[38] C. E. Leiserson et al. The Network Architecture of the Connection Machine CM-5. In *Symposium on Parallel Architectures and Algorithms*, pages 272–285, San Diego, California, June 1992. ACM.

[39] H. Liu and P. Singh. ConceptNet – A Practical Commonsense Reasoning Tool-Kit. *BT Technical Journal*, 22(4):211, October 2004.

[40] T. Marescaux, V. Nollet, J.-Y. Mignolet, A. B. W. Moffat, P. Avasare, P. Coene, D. Verkest, S. Vernalde, and R. Lauwereins. Run-Time Support for Heterogeneous Multitasking on Reconfigurable SoCs. *INTEGRATION, The VLSI Journal*, 38(1):107–130, 2004.

[41] L. McMurchie and C. Ebeling. PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 111–117. ACM, February 1995.

[42] M. Millberg, E. Nilsson, R. Thid, S. Kumar, and A. Jantsch. The Nostrum Backbone - a Communication Protocol Stack for Networks on Chip. In *International Conference on VLSI Design*, 2002.

[43] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. HERMES: an Infrastructure for Low Area Overhead Packet-switching Networks on Chip. *INTEGRATION, The VLSI Journal*, 38(1):69–93, 2004.

[44] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb. The Alpha 21364 Network Architecture. In *IEEE MICRO 2002*, 2002.

[45] S. Murali and G. D. Micheli. SUNMAP: A Tool for Automatic Topology Selection and Generation for NoCs. In *Proceedings of the ACM/IEEE Design Automation Conference*, 2004.

[46] Nallatech. Simplifying Communication Across DSP Networks. Programmable World, 2003. <`http://www.mactivity.com/xilinx/pw2003/workshops/presos/wsa3_nallatech.pdf`> .

[47] NIST. Matrix Market. <`http://math.nist.gov/MatrixMarket/`> , June 2004. Maintained by: National Institute of Standards and Technology (NIST).

[48] M. Noakes and W. J. Dally. System Design of the J-Machine. In W. J. Dally, editor, *Advanced Research in VLSI: Proceedings of the Sixth MIT Conference*, pages 179–194, Cambridge, Massachusetts, 1990. MIT Press.

[49] P. P. Pande, C. Grecu, A. Ivanov, R. Saleh, and G. D. Micheli. Design, Synthesis and Test of Networks on Chips. *IEEE Design and Test of Computers*, 22(5):404–413, 2005.

[50] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh. Performance Evaluation and Design Trade-offs for Network-on-chip Interconnect Architectures. *IEEE Transactions on Computers*, 54(8):1025–1040, August 2005.

[51] S. L. Scott and G. M. Thorson. The Cray T3E network: adaptive routing in a high performance 3D torus. In *IEEE Hot Interconnects Symposium IV*, 1996.

[52] C. Seitz. Let's Route Packets Instead of Wires. In *Proceedings of the 6th MIT Conference on Advanced Research in VLSI*, pages 133–137, 1990.

[53] C. Seitz and W.-K. Su. A Family of Routing and Communication Chips Based on the Mosaic. In *Proc. of 1993 Symposium on Research on Integrated Systems*, 1993.

[54] C. L. Seitz. Mosaic C: An Experimental Fine-Grain Multicomputer. In A. Bensoussan and J.-P. Verjus, editors, *Future Tendencies in Computer Science, Control and Applied Mathematics: Internantional Conference on the Occasion of the 25th Anniversary of INRIA*, pages 69–85. Sprinter-Verlag, December 1992.

[55] B. Sethuraman, P. Bhattacharya, J. Khan, and R. Vemuri. LiPaR: A LightWeight Parallel Router for FPGA based Networks on Chip. In *Proceedings of the Great Lakes Symposium on VLSI*, 2005.

[56] H. Siegel. *Interconnection Networks for Large-Scale Parallel Processing.* Lexington Books, Lexington, MA, 1985.

[57] C. D. Thompson. A Complexity Theory of VLSI. Technical Report CMU-CS-80-140, Department of Computer Science, Carnegie-Mellon University, 1980.

[58] W. Tsu, K. Macy, A. Joshi, R. Huang, N. Walker, T. Tung, O. Rowhani, V. George, J. Wawrzynek, and A. DeHon. HSRA: High-Speed, Hierarchical Synchronous Reconfigurable Array. In *Proceedings of the International Symposium on Field-Programmable Gate Arrays*, pages 125–134, February 1999.

[59] L. G. Valliant. A Bridging Model for Parallel Computation. *Communications of the ACM*, 33(8):103, August 1990.

[60] D. Wingard. MicroNetwork-Based Integration for SOCs. In *Proceedings of the ACM/IEEE Design Automation Conference*, pages 673–677, June 2001.

[61] Xilinx, Inc., 2100 Logic Drive, San Jose, CA 95124. *The Programmable Logic Data Book-CD*, 2005.