CHIP MULTIPROCESSOR COHERENCE AND INTERCONNECT SYSTEM DESIGN

by

Natalie D. Enright Jerger

A dissertation submitted in partial fulfillment of

the requirements for the degree of

Doctor of Philosophy

(Electrical Engineering)

at the

UNIVERSITY OF WISCONSIN-MADISON

2008

Abstract

This dissertation examines the interactions between the on-chip interconnection network and the cache coherence protocol. By exploring the cache-coherent communication behavior of applications, superior performance can be achieved through the co-design of the on-chip interconnect and the cache coherence protocol. The shift toward many-core architectures allows researchers and designers to leverage solutions from the traditional multiprocessor design space. However, these solutions are not always adequate for the differing needs and opportunities presented by many-core architectures. In this dissertation, I seek to challenge some of the conventional wisdom that has been taken from that design space and has emerged in many-core designs and provide alternative solutions to the communication challenges that our industry faces. Communication will be a central component of these large many-core systems; without efficient communication it will be difficult to scale these systems due to both power envelope limits and performance requirements.

Two co-designed interconnect and protocol solutions are presented. System performance is sensitive to on-chip communication latency; to address this, we propose hybrid circuit switching. This network design removes circuit setup time by intermingling packet-switched flits with circuit-switched flits. This design is further optimized through a prediction-based coherence protocol that leverages the existence of circuits to optimize pair-wise sharing between cores.

The second solution addresses the poor throughput of state-of-the-art on-chip networks in the presence of multicast communication. We present the Virtual Circuit Tree Multicasting router design that removes redundant messages from the network through the construction of multicast trees. Then, we further extend this router architecture to provide network ordering to implement Virtual Tree Coherence. Virtual Tree Coherence is a multicast coherence protocol that leverages both the network ordering and low overhead of our Virtual Circuit Tree Multicasting router to provide low latency cache to cache transfers.

By considering on-chip communication from both the perspective of the network and the coherence protocol, superior systems can be designed. The interconnection networks and protocols presented in this dissertation are two such examples.

Dedication

In loving memory of my grandfather, Ulysses Lupien and my high school English teacher, Dr. Jane Coil Cole: for inspiring me to do something meaningful and believing that I could.

Acknowledgements

Numerous people have contributed to my success throughout graduate school, culminating in the production of this dissertation. First and foremost, I would like to thank my husband, Chris for his unwavering support and understanding over the past six+ years; and for the personal and professional sacrifices he has made so that I might achieve this degree. Among his many wonderful qualities, his sense of humor has helped me through the stressful times. I don't think I could have gotten here without you. I love you.

Many thanks to my parents, Vincent and Judith Enright for their encouragement and unconditional belief in my ability to succeed at my dreams. You're incredible parents. Additionally, I thank my brother Chris and his wife Misty as well as my wonderful extended family for listening dutifully as I explain my research over Christmas dinner and for being proud of my accomplishments.

I have been extremely fortunate to work under the guidance of two amazing advisors. First, I would like to thank Mikko Lipasti for recognizing the promise in a first year graduate student. I have learned so much as a result of his technical expertise and I appreciate his patience and support throughout my graduate career. His guidance and wisdom have helped prepare me for the next steps in my career.

My co-advisor, Li-Shiuan Peh has been a tremendous source of support. From teaching me about interconnection networks to providing me with the tools to forge my own research career, I am truly indebted to her. In additional to her technical expertise, she has been a constant cheerleader providing me with the necessary encouragement and support to push through deadlines. Her energy and passion inspires me. I would also like to thank her for inviting me to come work with her at Princeton. Visiting Princeton gave me an opportunity to interact with another excellent group of graduate students.

In addition to the guidance of my advisors, this dissertation has been shaped through the suggestions of the members of my committee: Kati Compton, Mark Hill and David Wood. I am very grateful for their technical insights as well as the mentoring they have provided in regard to my professional development.

I've made many wonderful friend throughout my time at the University of Wisconsin. First, I owe a tremendous debt to Emily Blem: for the copious amounts of caffeine consumed and countless conversations that have helped me through the last few years. I thank Mike Marty for providing me with feedback on my research and acting as a sounding board. I thank Elizabeth O'Callaghan for collaborating on research outside of computer engineering and giving my brain space to think about issues in higher education and for becoming a good friend in the process. Watching her balance graduate school and three children gives me hope.

I have enjoyed the opportunity for working with the various members of both the PHARM group and Li-Shiuan's research group. Specifically, I thank Kevin Lepak and Trey Cain for their substantial effort in developing PHARMsim and for their patience in imparting their wisdom to a young graduate student. I appreciate the feedback of the other members of the team, specifically, Gordie Bell, Erica Gunadi, Atif Hashmi, Eric Hill, Lixin Su and Dana Vantrease. I'm grateful for the feedback Amit Kumar has provided on this work; he has been a source of positive encouragement over the last several years. Other members of Li-Shiuan's research group have contributed to this work, particularly Noel Eisley and Niket Agarwal who provided traces used for evaluation.

I owe thanks to my graduate school computer architecture cohort including Dan

Gibson, Luke Yen, Yasuko Watanabe, Jarayam Bobba, Philip Wells and Derek Hower; our weekly lunches have provided a nice diversion and I have learned a lot from my interactions with all of you.

My oldest friend, Andrea Jensenius and my cousin, Stephen Cena both helped shape my interactions with computers in my formative years. Without their influence, I do not believe I would have traveled down this path. Thank you.

I am grateful to IBM for supporting this research through two IBM PhD Fellowships. This research was also supported in part by NSF grants and by the MARCO Gigascale Systems Research Center. Additionally I would like to thank the Wisconsin Architecture Affiliates for their insight on my work. Specifically, I thank Steve Kunkel, Steve Vanderwiel and Steve Stephens for taking great interest in my technical development and providing me with a wonderful internship opportunity.

Contents

Chapter

1 Introduction 1

1.1 Many-core Era 2

1.1.1 Technology Projections 2

1.1.2 Challenges of Many-Core 3

1.2 Communication Challenges in the Many-core Era 6

1.2.1 Interconnect Architectures 6

1.3 Cache Coherence for Many-Core architectures 8

1.4 Thesis Contributions 10

1.4.1 Relationship to published works 12

1.5 Dissertation Organization 13

2 Background 14

2.1 Interconnection Network Overview 14

2.1.1 On-Chip vs. Off-Chip interconnects 16

2.1.2 Flit Level Flow Control 17

2.1.3 Topology 18

2.1.4 Routing Algorithm 19

2.1.5 Router Micro-architecture and Pipeline 20

2.1.6 Key Challenges Addressed in this Dissertation 23

2.2 Coherence Overview 25
2.2.1 Memory Consistency Overview 26
2.2.2 Cache Coherence Ordering Invariants and Permissions 27
2.2.3 Coarse Grain Coherence Tracking 33
2.2.4 Desirable Properties 34
2.3 Emerging Workloads 35
2.4 Conclusion 37

3 Hybrid Circuit Switching 39
3.1 Communication Characterization 39
3.1.1 Pair-wise Sharing 40
3.2 Hybrid Circuit Switching Overview 42
3.2.1 Hybrid Circuit Switched Network 44
3.2.2 Setup Network 44
3.2.3 Circuit Switched Pipeline 47
3.2.4 Packet Switched Pipeline 49
3.2.5 Narrow Data Network 49
3.3 Power and Area Overhead 50
3.4 Discussion 51
3.4.1 Circuit thrashing 51
3.4.2 Circuit Fragmentation 51
3.4.3 Setting up Circuits 52
3.4.4 Scalability 53
3.5 Evaluation 53
3.5.1 Methodology 54
3.5.2 Network Performance Results 56
3.5.3 Full system results 61

3.5.4 Synthetic Traffic . . . 63
3.6 Related Work . . . 66
3.6.1 Hybrid Networks . . . 66
3.6.2 Router Delay . . . 67
3.6.3 Multiple Networks . . . 68
3.7 Conclusion . . . 68

4 Virtual Circuit Tree Multicasting 69
4.1 Communication Characterization . . . 69
4.2 Broadcast and Multicast Architectures . . . 71
4.3 Baseline Inefficiencies . . . 73
4.4 Virtual Circuit Tree Multicasting Overview . . . 74
4.4.1 Router Micro-architecture . . . 78
4.4.2 Network Interface Controller . . . 81
4.4.3 Router Pipeline . . . 81
4.4.4 Destination Encoding . . . 83
4.4.5 Example . . . 83
4.5 Discussion . . . 85
4.6 Optimizations . . . 87
4.6.1 Tree Replacement Policy . . . 87
4.6.2 Optimal Tree Selection . . . 88
4.6.3 Tree Collapsing . . . 89
4.6.4 Dynamic Table Partitioning . . . 91
4.7 Power and Area Analysis . . . 92
4.8 Extension to other topologies and routing algorithms . . . 93
4.9 Network Ordering . . . 95
4.10 Evaluation . . . 96

4.10.1 Performance of Various Multicast Scenarios 97
4.10.2 Injection Port Contention 108
4.10.3 Synthetic Traffic Performance 108
4.10.4 Impact of VCTM Optimizations and Modifications 110
4.10.5 Activity/Power 115
4.11 Related Work 117
4.11.1 Multicast Routing 118
4.11.2 Multicast Destination Encoding 120
4.11.3 Multicast Deadlock 122
4.12 Conclusion 123

5 Circuit-Switched Coherence 124
5.1 Circuit-Switched Coherence Overview 124
5.2 Pair-wise Sharing Prediction 125
5.3 Ordering Invariants 127
5.3.1 Race Example 127
5.4 Overhead 131
5.5 Evaluation 132
5.6 Related Work 135
5.7 Conclusion 137

6 Virtual Tree Coherence 139
6.1 Overview 140
6.2 Hierarchical Coherence 141
6.2.1 Local Coherence 142
6.2.2 Multicast Directory Coherence 142
6.2.3 Multicast Broadcast Coherence 143
6.3 Ordering Invariants 146

6.3.1 Acknowledgments 147

6.4 Enforcing Consistency Models on VTC 148

6.4.1 Weak Ordering 149

6.4.2 Stronger Consistency Models 150

6.5 Coherence Actions 155

6.5.1 Data responses 156

6.6 Walk-through Example 158

6.7 Relationship between trees and regions 158

6.8 Coarse-Grain Regions 160

6.9 Overhead 163

6.10 Design Optimization 166

6.10.1 Root Selection Policies 167

6.11 Scalability 168

6.12 Discussion 169

6.13 Evaluation 170

6.13.1 Methodology 170

6.13.2 Baselines 171

6.13.3 Performance Results 173

6.13.4 With and Without VCTM 177

6.13.5 Tree Root Selection Policies 178

6.13.6 Limited Caching of Sharing Vector 179

6.13.7 Activity/Power 179

6.13.8 Interaction between VCTM and VTC 181

6.13.9 Discussion 183

6.14 Related Work 183

6.14.1 Ordered Interconnect 183

6.14.2 Unordered Interconnect 184

6.14.3 Hierarchical Coherence 186
6.14.4 Network Designs Cognizant of Cache Coherence 188
6.15 Conclusion 189

7 Conclusion 191
7.1 Contributions and Summary of Results 191
7.2 Future Research Directions 192
7.2.1 Scalability of the proposals in this dissertation 193
7.2.2 How to deal with network hotspots? 193
7.2.3 Further design of multicast and reduction networks 194

Bibliography 195

Tables

Table

3.1 HCS Simulation Parameters 54

3.2 Benchmark Descriptions 55

4.1 Percentage of network requests that can be multicast 73

4.2 VCT Table Overhead 92

4.3 Destination CAM Overhead 93

4.4 Network Parameters 96

4.5 Virtual Circuit Tree Hit Rates (%) 111

5.1 Prediction and Protocol Walk-through Example 126

5.2 Comparison between Baseline Protocol and Prediction Protocol 128

5.3 Table 5.2 (cont from previous page) 129

6.1 VTC actions for Invalid Region 156

6.2 VTC actions for Region Shared 157

6.3 VTC actions for Region Modified 157

6.4 VTC actions for Region Eviction 158

6.5 Steps Corresponding to Figure 6.6 159

6.6 Storage Comparison in KBits 166

6.7 Simulation Parameters 171

6.8 Steps Corresponding to Figure 6.9 174

Figures

Figure

1.1 Target CMP System 3

2.1 Examples of Various Interconnection Networks 15

2.2 Router Micro-architecture 21

2.3 Packet-Switched Router Pipeline 22

2.4 Optimized Packet-Switched Router Pipeline 24

2.5 Bus-Based Snooping Multiprocessor System 30

2.6 Multiprocessor system with a Directory Protocol on a Mesh Interconnect 32

3.1 Impact of Interconnect Latency on Overall Performance 40

3.2 Bandwidth Demands placed on On-Chip Network 41

3.3 On-chip misses 42

3.4 Performance of Traditional Circuit Switching 43

3.5 Hybrid Circuit-Switching Router design 45

3.6 Setup Network Router Stages 46

3.7 Circuit-Switched Router Pipeline 48

3.8 HCS Network Performance 56

3.9 HCS Latency Breakdown 57

3.10 Restricting Circuit Setup 58

3.11 Average Link Stealing Interval 59

3.12 Wait time for Packet-Switched Flits (starvation) 60

3.13 Channel Load of Setup Network 61

3.14 Overall Performance of HCS 62

3.15 Server Consolidation Performance of HCS normalized to PS 63

3.16 Performance of HCS with Uniform Random Traffic 64

3.17 Performance of HCS with Permutation Traffic 65

4.1 Performance of multicasts on packet-switched interconnect 70

4.2 Multiple Unicast approach to multicasting 75

4.3 Cumulative distribution of unique multicast destination sets 76

4.4 Percentage of multicasts in each multicast destination set size 77

4.5 Header packet format 79

4.6 Multicast Router Micro-architecture 80

4.7 Multicast Router Pipeline 82

4.8 Multicast setup and routing example 85

4.9 Difference between Non-Optimal and Optimal Tree Selection 88

4.10 Overlapping and collapsing similar trees 90

4.11 Adaptive Routing Challenge 95

4.12 SGI Origin Directory Protocol network performance 98

4.13 TokenB Coherence Slowdown in the Absence of Hardware Multicast Support 99

4.14 TokenB network performance with VCTM 100

4.15 Opteron Protocol performance with VCTM 102

4.16 Virtual Tree Coherence network performance with VCTM 103

4.17 TRIPS network performance with VCTM 104

4.18 Adaptively Routed Multicasts 106

4.19 Comparison between VCTM and Path-Based Multicast 107

4.20 VCTM under uniform random traffic with 10% multicasts 108
4.21 Comparison against aggressive (unrealistic) network 109
4.22 LRU optimization for VTC 112
4.23 TCAM and LRU optimizations 113
4.24 Dynamic Partitioning 114
4.25 Virtual Channel Restriction 115
4.26 Reduction in buffering, link and crossbar traversals 116
4.27 Minimum Spanning Tree Link Usage 117

5.1 Protocol Race Example 130
5.2 RegionTracker structures 131
5.3 RegionTracker indexing 132
5.4 Performance of Protocol Optimization + Hybrid Circuit Switching . . . 133
5.5 Region-Based Prediction Accuracy 134
5.6 Circuit Reuse 135

6.1 Example of Hierarchical Coherence with a Global Directory Protocol. . 144
6.2 Example of Hierarchical Coherence with a Global Broadcast Protocol. . 145
6.3 Store Ordering. Addresses A and B are in different regions. 152
6.4 Write Atomicity. Addresses A and B are in different regions. 153
6.5 Sequential Consistency Example: Dekker's algorithm. 154
6.6 Virtual Tree Coherence Example 159
6.7 Characterization of Sharers by Region Size 161
6.8 Region Vector Entry 164
6.9 Greedy Order Example 174
6.10 16 core Performance Comparisons 175
6.11 64 core Performance Comparison 176
6.12 Hop Count Compared to Directory Baseline 177

6.13 Root Hotspot for SPECjbb 178

6.14 Hit Rates for Limited Caching for Sharing Vectors 179

6.15 VTC Interconnect Traffic (16 core) 180

6.16 VTC Interconnect Traffic (64 core) 181

6.17 Impact of Region Size on Virtual Circuit Tree Hit Rate (16 cores) . . . 182

Chapter 1

Introduction

Computers have become pervasive throughout our society. By increasing productivity and mobility, as well as providing new avenues of entertainment, computers are commonplace in professional and personal lives. The wide array of uses and applications of computers presents numerous challenges to hardware designers but also provides us with a vast audience for consuming our innovations.

Over the course of this decade, single processor computer chips have given way to chip multiprocessors. These chip multiprocessors are becoming the primary building blocks of computer systems. The emergence of many-core architectures marks a massive shift in the way we think about designing and engineering these systems.

Particularly, the presence of multiple cores on a chip shifts the focus from computation to communication as a key bottleneck to achieving performance improvements. Extracting additional instruction level parallelism (ILP) has been a much researched way of keeping execution units busy with work and achieving higher performance. Increasingly, efficient communication between execution units or cores will become a key factor in improving performance for many-core chips. High performance on-chip communication is necessary to keep cores fed with work.

Communication requirements and patterns have been well-researched in the traditional multiprocessor domain (multi-chip) and the supercomputing domain. These solutions, while valuable, cannot be fully recycled for on-chip use. Tight area and power budgets require new, efficient solutions. Additionally, performance is more tightly coupled with router delay than to link transmission delay (the reverse of an off-chip interconnect). Bandwidth in on-chip networks is no longer pin-limited and on-chip wires are a readily available commodity. These changes require that application communication be re-examined for many-core systems.

1.1 Many-core Era

As the computer industry seeks to continuously provide customers with performance improvements, its design focus has shifted from large single core designs to many-core designs consisting of many simpler cores. This ability to place many-cores on a single die is driven by technology scaling that has scaled consistently with Moore's law.

1.1.1 Technology Projections

As scaling has continued along the projected Moore's law curve, feature sizes have shrunk to the sub-micron range allowing billions of transistors to be packed on a single chip. Historically, this increase in transistor counts has allowed the industry to drive up single threaded performance. Creating large and complex designs to extract additional ILP and thus improve execution throughput has been a predominant trend. However, wire delays are now dominating pipeline delay making it difficult to shrink cycle times and this is leading to a plateauing of single-threaded performance [7, 106].

As a result, the computer industry is shifting from building large, complex single core chips to building simpler cores and placing dozens or hundreds of these simple cores on a single chip [13, 60, 72, 114, 129, 132, 137, 142]. In 2008, designs are being produced both in academia and industry ranging from two cores to 80 cores. The use of many simpler cores shifts the focus from instruction level parallelism to thread level parallelism (TLP). Communication architectures that facilitate the coordination and communication of dozens to hundreds of threads are a central concern to architects

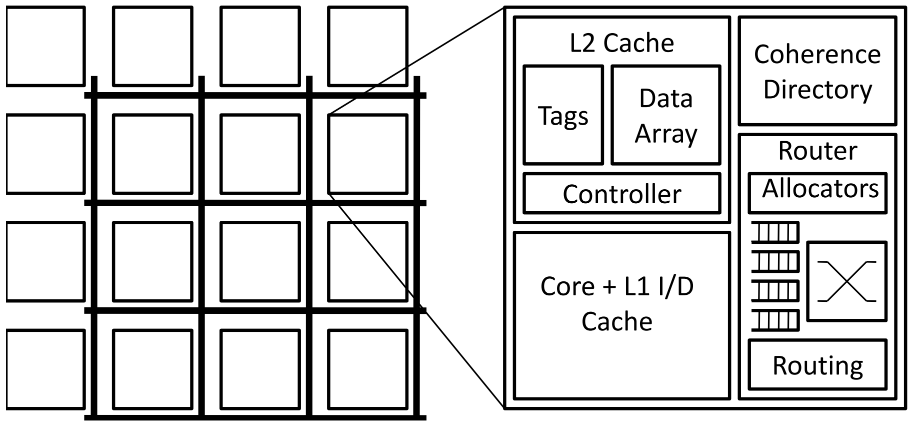


Figure 1.1: Target CMP System

moving forward.

Based on these emerging designs, Figure 1.1 depicts a high-level view of the architecture that will be used in this dissertation. This system, with cores and cache laid out as tiles and arranged in a grid fashion and connected with a network represents a likely direction for many-core architectures moving forward. This dissertation will focus on optimizations to this baseline in the areas of on-chip caches and the interconnection network.

1.1.2 Challenges of Many-Core

This industry-wide shift to many-core architectures provides numerous opportunities and challenges that need to be explored. In ILP-driven machines, parallelism can be extracted automatically by the hardware or by the compiler. Many-core architectures shift the focus from ILP-driven architectures to those focused on thread-level parallelism (TLP). Now, with TLP-architectures, there is a pressing and urgent need for parallel applications to utilize these machines.

1.1.2.1 Parallel programming

In addition to the research and design challenges faced by large scale many-core design, the industry faces the challenge of finding applications and workloads with which to benchmark and evaluate these designs. Parallel programming is challenging in both correctness and performance. Yet the demand for parallel applications is substantial. No longer is parallelism confined to a small subset of machines, namely servers and supercomputers. Rather, parallel architectures are available in laptops and desktops and will be available in handhelds and other pervasive computing devices. Parallelism is becoming a commodity feature of computing systems requiring significant performance improvements to drive the industry further. Optimizing the communication architecture of many-core chips can alleviate some of the performance burdens placed on the programmer.

Parallel applications of the future may exhibit different properties; however, one commonality will be the need for highly efficient communication between threads. Fine-grained parallelism is exhibited in existing suites such as RMS workloads [26] and the PARSEC benchmarks [14]; fine-grained parallel applications will require a fast and seamless communication substrate.

1.1.2.2 Server consolidation

An alternative class of applications for many-core architectures are server consolidation workloads. The resurgence of virtualization technology [46, 123] has allowed the consolidating of several physical servers onto one single physical, high-end server. Amongst many other benefits, servers can experience higher utilization and may dynamically meet changing demands by scaling their resource requirements [62].

Many-core architectures are well-suited to commercial workloads running in a server consolidation environment. Each workload may be given the impression of running on its own private system but in actuality, the hardware may be shared with other workloads in a variety of possible arrangements. These workloads, which tend to be multi-threaded and communication-intensive applications, are a good match for a many-core architecture's memory system, which allows for fast inter-thread communication while the multiple cores allow for concurrency. Server consolidation workloads exhibit both spatial and temporal sharing of on-chip resources such as the interconnection network and caches.

Communication requirements are also significant for server consolidation workloads; now several discrete applications are competing to use the same communication infrastructure. This competition introduces new challenges; however, the unique characteristics of server consolidation workloads also present opportunities for optimization. These workloads are characterized by a majority of communication remaining local within a virtual machine and very limited global communication across virtual machines. These characteristics will be explored and exploited in this dissertation. Cache coherence protocols and on-chip networks can be optimized for this class of communication while still maintaining the flexibility to perform well with single monolithic workloads.

1.1.2.3 Computation vs. Communication

As the industry moves toward commodity parallel applications and hardware, efficient communication becomes central to high performance. With an abundance of transistor resources, architects can create an abundance of computational units, resulting in computation that is essentially "for free". The key to keeping computational units active will be communicating data between them; communication therefore becomes central to performance improvements. This dissertation focuses on the demands placed on the communication architecture and the opportunities present to optimize the communication system.

In a multiprocessor system, the communication substrate can be viewed from two different perspectives. The first is the physical infrastructure which focuses on *how* cores communicate. Specifically, the design of this physical infrastructure looks at what resources and wiring are necessary to facilitate the efficient physical transfer of bits. The second component of the communication substrate focuses on *what* needs to be communicated. In a shared memory multiprocessor (the focus of this dissertation), the cache coherence protocol governs *what* is communicated between cores. Much of the existing research in this area, both for traditional multiprocessors and many-core chips examines these subsystems separately; however, insight and opportunities are found when we examine how these two components are intertwined.

As systems become more tightly coupled in many-core architectures, co-design of system components becomes increasingly important. In particular, coupling the design of the on-chip network with the design of the coherence protocol can result in a synergistic relationship providing superior performance.

1.2 Communication Challenges in the Many-core Era

Using dedicated ad-hoc wires to directly interconnect cores on-chip has become an intractable solution for the many-core era [35]. Dedicated wires are costly, in terms of area and delay and are an inefficient way to realize parallel communication. This inefficiency has resulted in a shift to on-chip networks, where communications between cores are multiplexed on shared wires; these on-chip networks provide the scalable solutions that many-core architectures will require.

1.2.1 Interconnect Architectures

The interconnection network design space contains many dimensions along which architects can optimize for power and performance. Topology is among the first considerations; for a modest number of cores, buses, rings and crossbars provide viable solutions. These communication substrates have been utilized in systems such as the IBM Cell [59, 64] and Sun's Niagara [72]. Shared buses and simple rings do not provide the scalability (bandwidth) needed to meet the communication demands of these future many-core architectures, while full crossbars are impractical due to their size and wiring requirements. More scalable topologies such as meshes and tori provide the scalability needed on-chip.

To date, designers have assumed a packet-switched on-chip network as the communication fabric for many-core chips [52, 132]. One downside of transitioning from dedicated wires to a network architecture is the introduction of router delay overheads. With dedicated wires, communication delay is solely a function of the physical properties of the wires; however, with an on-chip network, additional delay is added by the routers. A large on-chip network such as a mesh or torus, utilizes routers to determine the resource allocation, switching, flow control and routing of messages at each juncture in the network. In on-chip networks, routers consume significant area and power and therefore must be carefully designed.

In addition to added area and power overheads, the introduction of these routers results in increased communication latency over the use of dedicated wires. Due to this additional overhead, significant research has focused on driving down the latency overheads. The choice of switching technique contributes to the router latency experienced by messages. Packet-switching networks suffer from the overhead of router latency but efficiently multiplex communication from different sources. Alternatively, circuit-switching networks can eliminate these router overheads but are traditionally much less efficient than packet-switching networks. This dissertation asserts that circuit-switching can perform well without sacrificing bandwidth. In this dissertation, one such technique, hybrid circuit switching, is proposed and examined.

Routers are responsible for computing the path messages take from source to destination through the network. Current designs focus this routing computation on a message bound from a single source for a single destination (a unicast message). In order for low-latency, high-throughput communication to be realized, messages from a single source bound for multiple destinations (multicast messages) must also be considered. This requirement to support multicast messages is especially pressing in light of the large number of proposals and designs that generate multicast messages finding their way into many-core architectures. In this dissertation, we propose an efficient, low-cost technique to improve router handling of this class of message.

1.3 Cache Coherence for Many-Core architectures

The interconnection network architecture provides the physical resources needed to realize communication. The bits transmitted on that interconnection network architecture are determined by the application's sharing of data and instructions in memory. The shared-memory model is an intuitive way to realize this sharing. Logically, all processors access the same memory, allowing each processor to see the most up-to-date data. Practically speaking, memory hierarchies are employed to improve performance of shared-memory systems. These hierarchies complicate the logical, unified view of memory held in the shared-memory paradigm due to the presence of multiple copies in different caches. Each processor's view of memory must remain consistent with others' view of memory. Cache coherence protocols are designed to maintain one coherent view of memory for all processors. The cache coherence protocol governs what communication is needed in a shared memory multiprocessor to maintain a single coherent view of memory.

A variety of cache coherence protocols have been designed targeting different systems. In traditional multiprocessor systems designed using commodity single-core chips, the cache coherence protocol design choices need not be central to the chip design choices. As with interconnection networks, we need to adapt and re-engineer these protocols to make them appropriate for on-chip use. Two classes of protocols are the most commonly used: broadcast protocols and directory protocols. Generally speaking, there is a tension within these protocols between scalability and performance. With the tight coupling of many-cores on a single chip, the performance of the cache coherence protocol has become increasingly important. Specific trade-offs between these designs will be discussed in Chapter 2. This dissertation seeks to combine the best features of both into two novel protocols.

Very broadly speaking, broadcast protocols provide excellent performance for a modest number of cores and directory protocols provide scalability to many cores but at a performance cost. Directory protocols require additional on-chip storage in many-core systems. In traditional systems, broadcasting provides the best performance when the number of nodes is small; the same holds true for chip multiprocessors. Going forward, it is highly likely that only a small number of cores will need to be involved in coherence. Focusing coherence on this small number of cores naturally lends itself to a broadcast-based protocol. However, to date, chip multiprocessor research has focused on directory protocols as the appropriate scalable solution. We believe that systems will want some form of multicast- or broadcast-based coherence on chip (for high performance) but it must be realized in an inexpensive fashion (where the primary costs are bandwidth and storage overhead).

Specifically, this dissertation explores protocol optimizations that provide low latency requests and scalability with low power and area overheads. The first solution, circuit-switched coherence, reduces the latency of pair-wise read-sharers; this solution overcomes some of the latency problems of directories but does not address storage overheads. Through the co-design of a circuit-switched interconnect and the protocol, we challenge the predominant use of packet switching in networks and achieve superior network performance through the protocol co-design. The second solution, virtual tree coherence, simplifies ordering and achieves low latency for reads and writes with low storage requirements. Through the co-design of a multicast interconnect and a multicast coherence solution, we challenge the predominant use of directories in many-core designs by leveraging our novel high bandwidth, ordered interconnect to provide snooping-based coherence.

Both of these solutions leverage coarse-grain sharing information; data and coherence behavior are examined across multiple memory addresses to better observe and respond to common communication patterns. Optimizations can be made within the coherence protocol by providing the system with more information about communication behavior and patterns. Recent research has leveraged the benefits of coarse-grained coherence information [8,20,21,99]. Rather than simply tracking coherence information on a cache block level; information can also be tracked across multiple cache lines. Patterns can be observed better and exploited at this coarser granularity. We utilize this tracking of coarse grained information for coherence protocol improvements; specifically, coarse-grained predictions of pair-wise sharing relationship are leverage by circuit-switched coherence. Secondly, virtual tree coherence uses coarse-grain information to determine multicast destination sets.

1.4 Thesis Contributions

This dissertation re-examines and challenges some of the design assumptions that hold true for shared-memory multiprocessors when explored in the context of chip multiprocessors. As we migrate many of these design choices on-chip, it is worthwhile to examine their suitability and present novel solutions that are attractive in the unique environment of a many-core architecture. The goal of this dissertation to carefully consider communication requirements (as dictated by the coherence protocol), codesign the interconnect to better serve the coherence protocol, and improve the cache coherence protocols to better leverage the functionality of the on-chip interconnection network.

Specifically, this dissertation makes the following contributions:

- **Hybrid Circuit Switching (HCS)** explores the impact of communication latency on traditional and emerging applications for many-core designs. Communication latency is critical to the performance of these applications. Hybrid circuit switching focuses on reducing the routing latency for pair-wise communication patterns through a novel router architecture that interleaves circuit- and packet-switched messages on the same physical network. Packet-switching is typically favored over circuit-switching due to the bandwidth limitations of circuit-switching. Hybrid circuit switching overcomes this limitation through bandwidth stealing.

- **Virtual Circuit Tree Multicasting (VCTM)** characterizes the demand for on-chip multicasting across a variety of scenarios. Virtual Circuit Tree Multicasting employs a novel multicast router architecture that facilitates efficient on-chip broadcast and multicast communication. Performing the routing computation for a multicast message can complicate the router design significantly. VCTM leverages the repetitive nature of multicast communication to setup multicast routes with minimal extra hardware and then reaps throughput benefits from subsequent use of these routes. By removing redundant unicast messages, VCTM saves significant bandwidth, reduces dynamic power, and drastically improves the throughput of the network in the presence of multicast communication patterns.

- **Circuit-Switched Coherence (CSC)** leverages the presence of HCS and pair-wise sharing to design an optimized directory-based protocol. Pair-wise sharing will experience lower latency by avoiding indirections through the directory protocol. Recent sharers are predicted based on coarse-grain tracking of cache blocks. This network-protocol co-design results in superior performance for commercial workloads.

- **Virtual Tree Coherence (VTC)** utilizes the implicit ordering of the virtual trees provided by VCTM to facilitate scalable coherence for many-core architectures. A conceptually simple ordering invariant is described and demonstrated to have performance superior to directory and broadcast protocols. Tree roots are used to order multicast requests to active sharers of data. Coupling this ordering mechanism with coherence information maintained for coarse-grained address regions reduces on-chip coherence storage and improves performance over a baseline directory protocol.

- **Server Consolidation Workloads** present unique data sharing and communication behavior. Server consolidation workloads represent an emerging and important class of applications for many-core architectures. We utilize server consolidation workloads to further evaluate both proposed networks and protocols using server consolidation workloads.

Hybrid Circuit Switching and Virtual Circuit Tree Multicasting focus on improvements to the interconnection network while Circuit-Switched Coherence and Virtual Tree Coherence focus on optimizations for the cache coherence protocol. While described separately in this dissertation, HCS and CSC work together to provide superior performance while VCTM and VTC are intertwined for both performance and correctness reasons.

1.4.1 Relationship to published works

This dissertation includes work that has been previously published in four conference publications.

HCS (Computer Architecture Letters, NOCS-2) : HCS and Circuit-Switched Coherence were previously published as a Computer Architecture Letter and in the Network on Chip Symposium with co-authors Li-Shiuan Peh and Mikko Lipasti [41, 42].

Additional analysis and explanation is provided in this dissertation for a more in-depth understanding of the benefits of HCS and Circuit-Switched Coherence.

VCTM (ISCA-35): Virtual Circuit Tree Multicasting was originally published in the International Symposium on Computer Architecture with co-authors Li-Shiuan Peh and Mikko Lipasti [43]. In this dissertation, we consider further optimizations to improve the scalability of the original design as well as expand upon the description of the implementation.

VTC (MICRO-41): Virtual Tree Coherence originally appears in the proceedings of the International Symposium on Microarchitecture with co-authors Li-Shiuan Peh and Mikko Lipasti [44]. Additional protocol optimizations are explored in this dissertation to reduce network hop count and alleviate network pressure from coherence traffic.

Server Consolidation (IISWC 2007) : The simulation methodology and infrastructure for evaluating server consolidation workloads originally appeared in the IEEE International Symposium on Workload Characterization 2007 with co-authors Dana Vantrease and Mikko Lipasti [45]. In this dissertation, we utilize these server consolidation workloads and methodology to further evaluate the ideas presented here.

1.5 Dissertation Organization

This dissertation is organized as follows: Chapter 2 presents background information related to both interconnection network architectures and cache coherence protocols. Chapters 3 and 4 present the two network designs: the Hybrid Circuit-Switched network and the Virtual Circuit Tree Multicasting network along with their respective evaluations. The co-designed coherence protocols that operate in a synergistic fashion with the previously presented interconnection network designs are presented in Chapters 5 and 6. In Chapter 7, conclusions and avenues of future research are discussed.

Chapter 2

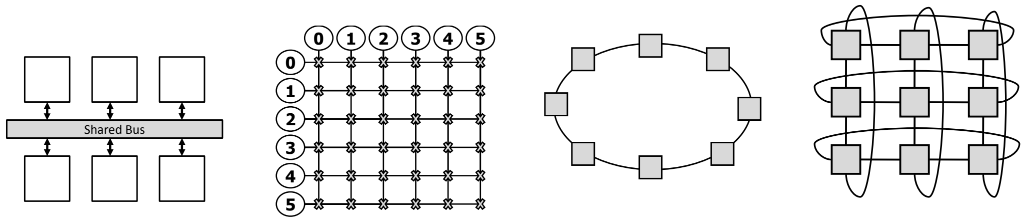
Background

This chapter presents an overview of both interconnection networks (Section 2.1) and cache coherence protocols (Section 2.2) which form the basis of the work presented throughout this dissertation. The interconnection network provides the physical medium on which communication occurs. After providing an overview of how interconnection networks are designed, the cache coherence problem is presented along with two common classes of protocols. Finally, the chapter ends with a discussion of recent research on server consolidation workloads; an emerging class of applications considered in this dissertation (Section 2.3).

2.1 Interconnection Network Overview

With the promise of many cores on a single chip, research emphasis has been placed on designing on-chip interconnection networks. Why are such network designs necessary? The alternative to building on-chip networks is to use dedicated ad-hoc wires to connect cores and communicating components. In this dissertation, we consider interconnection networks to broadly encompass buses, crossbars as well as higher bandwidth substrates, such as meshes and tori, [34] as depicted in Figure 2.1.

With a small number of components, dedicated wiring can be used to interconnect them. However, the use of dedicated wires is problematic for several reasons. First, as we increase the number of cores on chip, the amount of wiring required to directly connect



(a) Bus (b) Crossbar (c) Ring (d) Torus

Figure 2.1: Examples of Various Interconnection Networks

every component will become prohibitive. In addition to scalability, the performance of dedicated wiring is a problem. As the number of components increase, the length of dedicated wiring to connect them will also increase resulting in long latency global communication.

On-chip networks are an attractive alternative to dedicated wiring for several reasons. First and foremost, networks represent a scalable solution to on-chip communication. In addition to scalability, on-chip networks are also very efficient and are able to better utilize resources by multiplexing different communication flows on the same links allowing for significant amounts of parallel communication. On-chip networks with regular topologies have nice electrical properties and are built modularly from regular, repetitive structures easing the burden of verification [35].

To begin the discussion of interconnections networks, we must examine the key differences between off-chip and on-chip networks; these differences motivate the need for new on-chip solutions presented later in this dissertation. After discussing these differences, an overview of the different components of a network architecture will be presented.

2.1.1 On-Chip vs. Off-Chip interconnects

Substantial research has been done in the realm of off-chip networks for multiprocessors and supercomputers (for a more in-depth explanation of off-chip networks we refer the interested reader to [34]). Several key differences between off-chip networks and on-chip networks motivate the need for new interconnect solutions for the many-core era.

First, on-chip networks are designed under very tight area and power budgets. With multi- and many-core chips, now cores, caches and interconnects are competing for the same chip real estate. Integrating large number of components under tight area and power constraints poses a significant challenge for architects to create a balance between these components. The logic to support on-chip networks may consume $\sim 25\%$ of each tile in an many-core design [60]. On-chip networks consume a significant faction of total on-chip power [16,140]; up to $\sim 30\%$ for Intel's 80 core teraflops network [60,137] and 36% for the RAW on-chip network [132]. This tight integration in on-chip networks requires innovative and efficient solutions for the communication substrate.

In off-chip networks, communication latency is heavily dependent on link transmission latencies. In on-chip networks, short links can be traversed with low latency (typically a single cycle to travel a one-hop distance); this quick traversal results in router latencies becoming a more significant contributor to overall communication latency. Off-chip networks, such as the Alpha 21364 [100] can accommodate long router pipelines but the dramatic shortening of transmission delay on-chip makes this a growing problem. While communication latency is dominated by router delays, bandwidth becomes a much more abundant resource for on-chip networks. Bandwidth in off-chip networks is constrained by the pin bandwidth afforded by packaging technology. On-chip wires are cheap and plentiful resulting in significantly higher bandwidth than can be provided to off-chip network links.

Now that several key differences between off-chip and on-chip networks have been enumerated, specifically the tight area and power constraints, the relative contribution of router and link delays and the increased bandwidth available for on-chip, the remainder of this section will focus on the components of network design, specifically flow control, topology, routing algorithms and router microarchitectures, and the challenges facing on-chip network design.

2.1.2 Flit Level Flow Control

Flow control determines how resources are allocated to messages as they travel through the network. The flow control mechanism is responsible for allocating (and de-allocating) buffers and channel bandwidth to waiting packets. Resources can be allocated to packets in their entirety (done in store-and-forward and virtual cut-through [34]); however, this requires very large buffer resources making it impractical on chip. Most commonly, on-chip networks handle flow control at the flit level [1] ; buffers and channel bandwidth are allocated on the small granularity of flits rather than whole packets. Wormhole and virtual channel flow control are two types of flit level flow control. Wormhole flow control allocates buffers and channel bandwidth on a flit granularity requiring small buffering resources. With wormhole flow control, each input port has a single buffer queue. The packet travels through the network like a worm; the downside is if the head of the packet becomes blocked the body flits will still hold channel bandwidth that cannot be released to other resources since there is only a single virtual channel per link (known as head of line blocking).

The alternative to wormhole flow control is virtual channel flow control. Again, this technique allocates channel bandwidth and buffers on the granularity of flits; but now, since there are multiple virtual channels per physical channel, if one packet becomes blocked other packets can use the idle bandwidth if they have been assigned to a different

[1] A flit is a flow control unit, a subdivision of a packet

virtual channel. Buffers are distributed or shared among virtual channels. Virtual channel flow control is used in the baseline network in this dissertation since it makes effective use of channel bandwidth and buffering resources.

2.1.3 Topology

The topology determines the physical layout and connections between nodes in the network. A wide range of topologies is available for network design. Topologies can be classified as direct and indirect; with direct topologies, each router is coupled with a network node (core). In an indirect topology, there are several intermediate routers that are not connected to network nodes. Most networks use topologies that can be derived from two families, butterflies (indirect) and tori (direct). Average hop count is an important metric in topology selection; average hop count determines the average packet latency in an uncongested network. Another characteristic of topologies is path diversity; the path diversity (the number of unique paths that exist between the source and destination) of a network impacts its ability to distribute the traffic load and to tolerate faults in the network.

To date a small subset of network topologies have been explored for on-chip networks. Topologies such as a bus or a ring architecture do not provide the scalability required for many-core architectures; they provide too little bandwidth and long latency due to high hop counts as the network size increases. Two dimensional topologies such as meshes (a grid) and tori (a grid with wrap-around links) are popular with on-chip networks as they easily map to a planar substrate. To reduce hop count and thereby lessen the impact of router delay, high radix topologies such as the flattened butterfly have been proposed [69]. Another technique to reduce hop count, is to employ concentration [11]. Concentration connects several network nodes to a single router; often four network nodes are grouped to the same router. In this case, a 64-node system would utilize a 4x4 mesh rather than an 8x8 mesh; this reduces hop count considerably.

Topology is not the focus of this dissertation and the techniques proposed in Chapters 3 and 4 are agnostic to topology selection. As such, the most common on-chip topology - a 2-D mesh - is used throughout this work.

2.1.4 Routing Algorithm

The routing algorithm determines the path a message takes through the network. Routing algorithms can be deterministic, adaptive or oblivious; additionally routing algorithms can be classified as minimal or non-minimal. Deterministic routing algorithm allow only one fixed path between a source and destination pair. Adaptive routing can choose from multiple paths between a source and destination pair and factors in the current state of the network. Oblivious routing algorithms route traffic without any regard to the state of the network; for example, Valiant's randomized routing provides excellent load balancing but destroys any locality in the communication pattern and is non-minimal [136]. Minimal routing algorithms choose the shortest path between the source and the destination, while non-minimal routing algorithms may include additional hops. On-chip networks which are sensitive to communication latency typically choose minimal routing algorithms; the flattened butterfly [69] is one exception. To better distribute the load and provide better bandwidth utilization, the flattened butterfly leverages a non-minimal routing algorithm. In this case, by avoiding congestion on non-minimal routes, packet latency is kept low.

Dimension-order routing, which routes first in the X direction and then in the Y direction, is often chosen for its simplicity and deadlock-freedom properties. Limiting turns, as dimension order routing does, removes the potential for a resource cycle to occur between multiple packets in the network. Dimension order routing is both minimal and deterministic. Adaptive routing, such as turn-model routing [50] can provide more flexibility to better distribute the network load with some additional complexity while still maintaining deadlock-freedom. If the routing algorithm does not provide deadlock-freedom, escape virtual channels can be used to break deadlock cycles. When all possible turns are permitted by the routing algorithm, restrictions are placed on virtual channel allocation to create acyclic resource graphs [34].

Several adaptive routing algorithms have been proposed to avoid congestion in on-chip networks [53, 74, 117]. O1Turn [117] is an oblivious routing algorithm that distributes the load by randomly choosing between XY and YX dimension order routing at the source node. Region Congestion Aware (RCA) [53] communicates congestion information to neighboring routers via a monitoring network. This information can be used to make more intelligent routing decisions based more network state information rather just local router information. Token Flow Control [74] communicates tokens to nearby routers; these tokens help a router select the least congested path, as well as improve packet bypassing (discussed in the next section) at nearby routers for packets holding tokens.

Throughout this dissertation, dimension order routing is assumed; however, as with the choice of topology, our proposals are independent of routing algorithm and can be extended to various routing algorithms.

2.1.5 Router Micro-architecture and Pipeline

Much of the work in this dissertation focuses on optimizing the router micro-architecture. A router microarchitecture is comprised of the following components: input buffers, virtual channel router state, routing logic, allocators, and a crossbar. These primary components are shown in Figure 2.2.

Our baseline router is a 5x5 virtual channel router. The 5 input and output ports are the cardinal directions and the core injection/ejection port. Virtual channels [32] have been quickly adopted from off-chip networks into on-chip networks due to the improved throughput they provide. By creating multiple virtual channels per physical channel and associating buffers with each virtual channel, packets destined for different

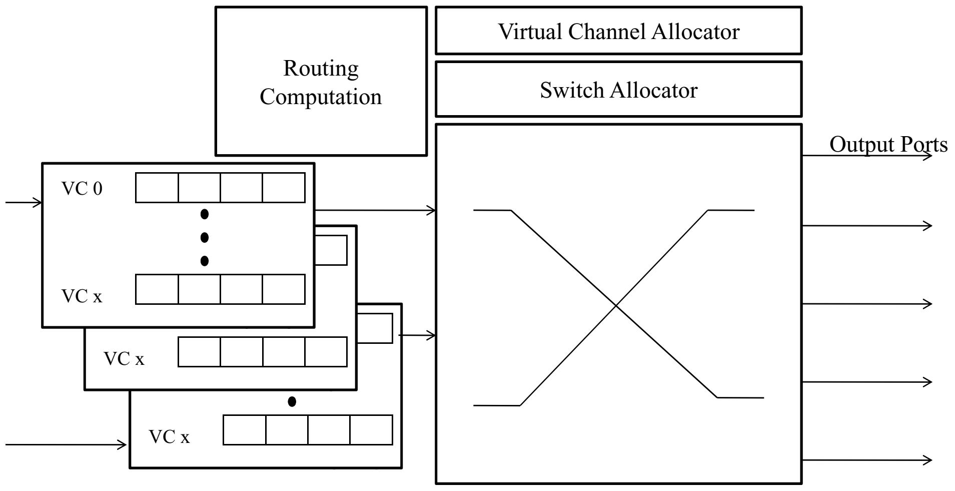


Figure 2.2: Router Micro-architecture

output ports can pass blocked packets in a router.

The router pipeline of an on-chip network consists of the following stages (shown in Figure 2.3): the buffer write (BW) stage, the routing computation stage (RC), the virtual channel allocation stage (VA), switch allocation stage (SA), switch traversal stage (ST) and link traversal (LT). When a new flit arrives at the router, it is decoded and buffered according to its specified input virtual channel (BW). Routing computation (RC) is performed in the second stage to determine the appropriate output port. In the virtual channel allocation stage (VA), the flit arbitrates for a virtual channel based on the output port determined in the previous stage. Once a virtual channel has been successful allocated, the flit proceeds to the fourth stage. Here, the flit arbitrates for access to this switch (SA) based on its input-output port pair. Once the switch has been allocated to the flit, it can proceed to the switch traversal stage (ST) and traverse the router crossbar. The link traversal stage (LT) occurs outside the router to carry the flit to the next node in its path. Each packet is broken down into several flits; as shown in Figure 2.3, the header flit is responsible for routing computation and virtual channel allocation while the body and tail flits reuse that computation and allocation.

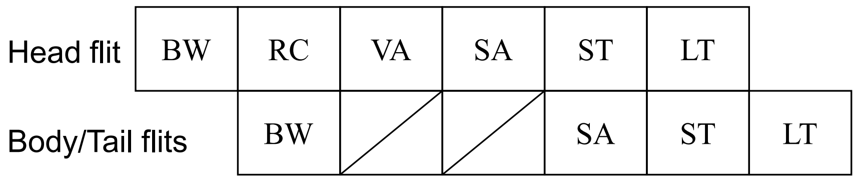


Figure 2.3: Packet-Switched Router Pipeline: Buffer Write (BW), Routing Computation (RC), Virtual Channel Allocation (VA), Switch Allocation (SA), Switch Traversal (ST) and Link Traversal (LT)

2.1.5.1 Pipeline Optimizations

Delay through each router in the network is the primary contributor to communication latency. As a result, significant research effort has been spent reducing router pipeline stages and improving router throughput. Some key techniques are discussed below.

Lookahead Routing: a single pipeline stage is devoted to computing the proper output port for the current packet (RC); this stage adds latency overhead as the packet progresses through multiple routers. Lookahead routing moves this computation from the current router to the previous router. As a result, the packet already has its routing decision in hand when it arrives at the current router; this eliminates the routing computation from the critical path of a packet. Lookahead routing was proposed in the SGI SPIDER chip [47].

Pipeline Bypassing is employed to reduce the critical path through the router. If there are no other flits in the input buffer, the incoming flit may speculatively enter the switch traversal stage (ST). In one cycle, the crossbar is setup for the bypassing flit and a virtual channel is allocated. If a port conflict is detected the bypassing must be aborted. The bypassing pipeline is shown in Figure 2.4b. Recent work [52, 73] uses lookahead signals or advanced bundles to shorten the pipeline to a single stage. While the flit is traversing the switch, a lookahead signal is traveling to the next router to perform the routing computation. In the next cycle, when the flit arrives, it can proceed directly to switch traversal, resulting in a single cycle router pipeline (ST+LT) as shown in Figure 2.4c. This bypassing is possible when there are no waiting flits in the input buffers when the advanced bundle arrives, there is no port conflict with existing flits and there is no output port conflict between advanced bundles arriving simultaneously.

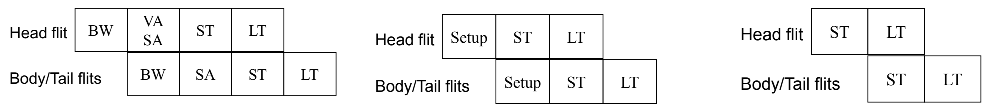
Speculation: another way to reduce the number of router pipeline stages that a packet must traverse, is to use aggressive speculation techniques [100, 102, 109]. Virtual channel and switch allocation may occur in parallel in the same stage. By speculatively performing virtual channel and switch allocation, we can reduce the critical path through the router. If speculation succeeds, the flit proceeds directly to the switch traversal stage. However, if speculation fails, the flit must repeat these pipeline stages to achieve the necessary allocations.

The benefit due to bypassing and speculation techniques is predicated on low loads; as the number of in-flight packets in the network increases, so does the likelihood of misspeculation. Failed bypassing or speculation results in a longer critical path through the router.

With the application of state-of-the-art router pipeline optimizations, the router pipeline shown in Figure 2.3 is transformed into the pipeline shown in Figure 2.4. We utilize this highly optimized router pipeline as a baseline of comparison for our hybrid circuit-switched router and the virtual circuit tree multicasting router.

2.1.6 Key Challenges Addressed in this Dissertation

The design of on-chip networks is still a relatively young field. As such, there are many open research problems that need to be addressed. As the focus of this dissertation is on network-protocol co-design, we limit our study of these open challenges to those that we believe will be most amenable to coupling with coherence protocols



(a) 3 stage optimized pipeline

(b) Bypassing pipeline

(c) Single stage pipeline

Figure 2.4: Packet-Switched Router Optimized Pipeline: Buffer Write (BW), Virtual Channel Allocation (VA), Switch Allocation (SA), Switch Traversal (ST) and Link Traversal (LT)

improvements.

2.1.6.1 Router Overhead

As noted earlier, one of the key differences between on-chip and off-chip network is the relative contribution of router delay to overall communication latency. While packet switching provides efficient use of link bandwidth by interleaving packets on a single link, it adds higher router latency overhead. Alternatively, circuit switching trades off poorer link utilization with much lower latency, as data need not go through routing and arbitration once circuits are set up. Packets traversing established circuits experience the router pipeline in Figure 2.4c. We present the design of a hybrid circuit-switched router to overcome both the router delay and the bandwidth limitations of circuit-switching (Chapter 3).

2.1.6.2 Efficient Multicast Routing

Current on-chip routers are heavily optimized for one-to-one, or unicast traffic. Routing logic is designed to route each packet to a single output port. The vast majority of current network-on-chip proposals ignore the issue of multicast communication. Proposals that might effectively leverage a multicast router either naively assume the existence of an on-chip multicast router or fail to model network contention (once contention is modeled, the need for hardware multicast support becomes abundantly clear). The presence of one-to-many (multicast) or one-to-all (broadcast) traffic can significantly degrade the performance of these designs since they rely on multiple unicasts to provide one-to-many communication. This results in a burst of packets from a single source and is a very inefficient way of performing multicast and broadcast communication. This inefficiency is compounded by the proliferation of architectures and coherence protocols that require multicast and broadcast communication. While unicast messages are likely to dominate traffic patterns, efficient on-chip multicast support is essential for many-core architectures moving forward. We present Virtual Circuit Tree Multicasting (VCTM) which adds multicast routing functionality to a state-of-the-art router with low overhead (Chapter 4).

2.1.6.3 Power Consumption

In addition to addressing issues of communication performance in on-chip networks, it is important to design power-efficient solutions to the router overhead and multicast routing problems. In this dissertation, we explore two techniques, HCS and VCTM that can reduce interconnection latency and/or improve throughput while maintaining or reducing the power consumption relative to the baseline. Streamlining communication through the addition of small router structures can have a significant impact on dynamic network power. Reducing network activity saves the dynamic power of switching activity and translates into overall power savings despite additional router overheads.

2.2 Coherence Overview

In addition to optimizing the physical medium that transfers bits around the chip, we must also consider the design of the coherence protocol that determines what data needs to be sent to whom and when. First, we present an overview of the cache coherence problem as well as some existing approaches; additional existing approaches related to this work will be presented in subsequent chapters.

Parallel programming is extremely difficult and has become increasingly important. With the emergence of many-core architectures, parallel hardware will be present in commodity systems. The growing prevalence of parallel systems requires an increasing number of parallel applications. Maintaining a global, shared address space alleviates some of the burden placed on programmers to write high performance parallel code. When a shared address space is not employed, message passing explicitly moves data between cores and address spaces but incurs substantial overheads to do so. It is easier to reason about a global address space than a partitioned one which is why many current architectures support the shared memory paradigm rather than the message passing paradigm.

Cache coherence protocols are necessary to maintain a single unified address space in the presence of cache/memory hierarchies. While hierarchies serve to substantially improve performance, they allow multiple copies of a memory location to exist on-chip. Coordinating these multiple copies falls to the cache coherence protocol.

2.2.1 Memory Consistency Overview

A memory system is coherent if it is possible to construct a hypothetical serial order of all memory operations to a given address location [31]. For programs executing on a uniprocessor all instructions appear to execute in the order specified by the programmer or the compiler. The hardware implementation may execute these instructions out of order but maintains this abstraction so that the programmer has a simple and intuitive model with which to reason about program order and program execution. What follows from this model is that a load is expected to return the value of the last store to that memory address.

With parallel programs, it becomes more difficult to reason about program order;

specifically the order of loads and stores. With a single thread, a load will return the value of the last store from that thread. But with multiple threads, the most recent store may have occurred on a different processor; therefore, the value to be returned by any given load is not as straightforward as in a sequential program. To help programmers reason about valid orderings of loads and stores by multiple threads, the hardware must adhere to a given memory consistency model. Memory consistency models specify the order a processor observes the memory operations performed on other processors.

One such memory consistency model, sequential consistency, specifies that a program execution is sequentially consistent "if the result of any execution is the same as if the operations of all processors were executed in some sequential order, and the operations of each individual processor occur in this sequence in the order specified by its program." [78]. Relaxing the constraints placed on the memory system implementation through relaxed consistency models [4] allows designers freedom to implement optimizations that overcome memory latency. Sequential consistency is the simplest and most intuitive consistency model; however, it can also restrict performance optimizations. As such, we explore the use of weaker consistency models for VTC in Chapter 6. Sequential consistency is assumed for Circuit-Switched Coherence in Chapter 5.

2.2.2 Cache Coherence Ordering Invariants and Permissions

Cache hierarchies have long been used to reduce the latency of memory operations and provide significant performance improvements. The addition of caches, however, complicate multiprocessor memory consistency. Caching allows processors to retain copies of a memory location closer to them for lower latency. If two processors both cache and subsequently read a memory location, their respective loads will return the same value. However, if one processor then writes that memory location in its private cache, subsequent loads from both processors will return different values resulting in incoherence. The mechanisms that guarantee that the value one processors writes will propagate to all processors are defined by the cache coherence protocol.

This dissertation uses as its basis, invalidation-based protocols that maintain coherence by enforcing a single writer-multiple reader invariant. Any number of cores may cache a copy of memory to read from; if a core wishes to write to that memory address, it must ensure that no other cores are caching that address.

At their basis, cache coherence protocols require three states for cache lines: *Invalid, Shared* and *Modified.* A cache line in the invalid state is not cached by the processor. When a processor holds a line in the shared state it can read from that line but cannot perform a store to that line; multiple copies of the same shared line may exist throughout the system. In order to store to a line, a processor must hold a cache line in the Modified state; this means that only one copy of the cache line is cached anywhere in the system and all or part of the cache line is dirty with respective to the value held in main memory. To perform a write to a shared line, the cache coherence protocol must invalidate externally shared copies of the cache block.

Two additional states are commonly added to invalidation-based protocols to improve their performance: *Exclusive* and *Owned.* A cache line is loaded in the exclusive state when no other copies exist on chip. On a subsequent write to the cache line, the upgrade coherence request can be saved as no other cores are caching the block. If the block was loaded in the Shared state, an upgrade request would need to be sent. A fifth coherence state, owned may be added to indicate which core will source the data. If a block is held in the owned state, that core, rather than memory will supply the data when a remote coherence request is observed. The use of the owned state also saves writebacks on dirty misses as owned can signify a block that is shared and dirty.

With these five states, a processor can write to a block that it finds in its cache in modified or exclusive and it can read from a block it finds in modified, owned, exclusive or shared. Three basics actions are required: Read-Shared, Read-For-Ownership and Upgrade. On a load or instruction miss in the last level cache, a read-shared request is issued to the interconnect and the data is returned. A processor caching the block in modified, owned or exclusive is responsible for supplying the data and will no longer have write permission to the block (transitions to owned or shared). For a store miss, a Read-For-Ownership is issued, this operation returns the data to the requesting cache and invalidates other copies that may exist. Again, to obtain the most recently written value, a cache with the block in modified, owned or exclusive will respond to this request. The protocol must have a means of indicating to the requesting processor that it is safe to assume all copies have been invalidated so that the store can complete. An upgrade is issued to convert a read-shared block to be writable by the requesting processor.

Write propagation and write serialization are two properties implied from coherence. Write propagation means that writes to a memory location are visible to all processors, while write serialization dictates that all processors see all writes to the same location in the same order.

Throughout the literature, there are two predominant ways of implementing cache coherence protocols: broadcast-based protocols and directory-based protocols. Both of these forms of coherence protocols exhibit problems for emerging many-core chip multiprocessors. Furthermore, these two protocols place vastly different demands on the interconnection network; the former exhibiting one-to-all traffic and the latter being dominated by point-to-point traffic.

2.2.2.1 Broadcast/Snooping Protocols

A widely adopted approach to cache coherence is a snooping protocol on a bus. A snooping broadcast protocol issues coherence requests to all processors via a shared, totally-ordered interconnect such as a bus. All components in the systems are connected to a bus. A bus can be viewed electrically as a set of wires or as a logical equivalent. The key advantage of using a bus for cache coherence is that it is a totally-ordered interconnect. All components attached to the bus will observe all requests in the same

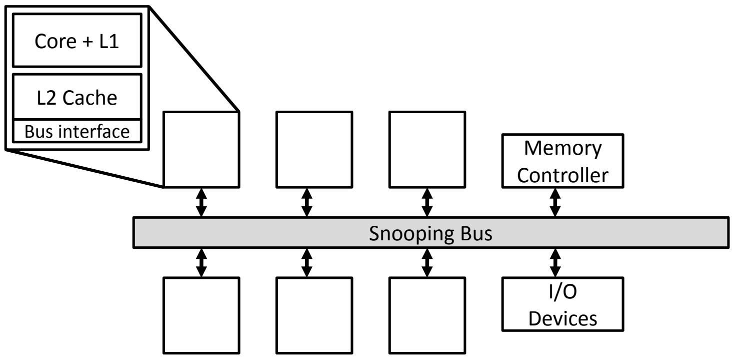


Figure 2.5: Bus-Based Snooping Multiprocessor System

logical, total order.

In addition to the total order properties of a bus, another advantage is the use of shared (wired-or) lines; any endpoint on the bus can assert a shared line and that assertion is globally visible to all endpoints on the bus. These shared lines can be useful for cache coherence. For example, a processor with a cache line in owned can assert the owned line which signifies to memory and other processors that it will source the data for the current request.

When coherence requests are placed on the bus, all caches attached to the bus will snoop the request. Each cache controller contains a state machine that responds as needed to each request. The key to snooping-based protocols is that all cores see requests in the identical order. Once a processor observes its own store request on the bus, it knows that no other outstanding request can be ordered before it. It is now safe for the processor to assume that all other cores have observed its store request and invalidated any valid copies. Sequential consistency is maintained as all other outstanding requests to the cache line have completed when a processor observes its own store request on the bus.

A bus-based multiprocessor system is depicted in Figure 2.5. Goodman [51] was the first to propose snooping-based protocols for cache coherence. Initially, electrically shared buses where used; in these early implementations the bus was held for the entire coherence transaction. Higher performing bus designs including pipelined and split transaction buses have been used to improve the performance of these protocols. Modern systems such as Sun's UltraEnterprise Servers, implement a logical bus from a hierarchy of buses [23, 24, 119]. This design relaxes the atomicity provided by a bus but continues to provide the requisite total order.

Broadcast protocols do not scale well in performance and power since they rely on a central ordering point for all coherence requests and flood the interconnect with broadcast requests. However, they offer fast cache-to-cache transfers and a simple abstraction for reasoning about coherence ordering.

2.2.2.2 Directory Protocols

Unlike broadcast protocols that rely on the total order of a bus for coherence, directory protocols use directories to serve as ordering points for coherence requests. Directory protocols rely on point-to-point messages rather than broadcasts; this reduction in coherence messages improves their scalability. Removing the centralized bottleneck of a shared bus also results in a more scalable protocol.

Directories maintain information about the current sharers of cache lines in the system and as well as state information. By maintaining sharing lists, directory protocols eliminate the need to broadcast an invalidation to the entire system; instead, point-to-point messages are sent to the subset of cores that are listed as sharers. Addresses are interleaved across directory nodes; each address is assigned a home node which is responsible for ordering and handling all coherence requests to that address.

Directories order cache requests instead of relying on the total order provided by a bus. But since directory protocols are often built on top of unordered interconnects, they can only handle one outstanding request to a cache line at a time. For example, when

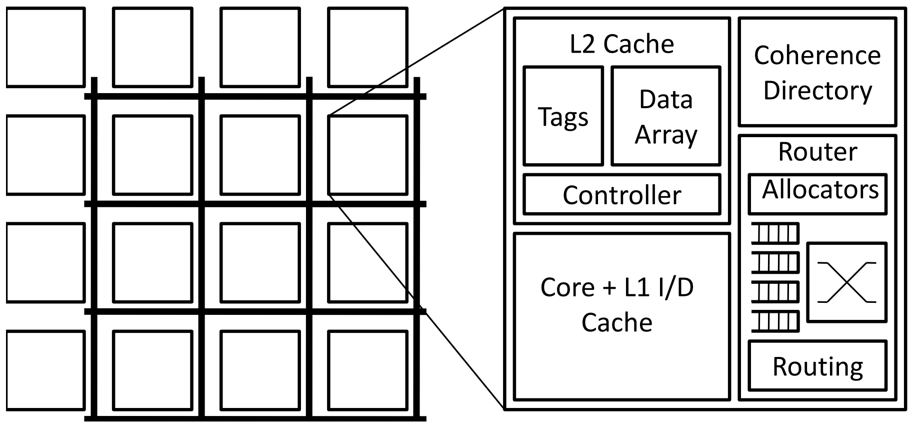


Figure 2.6: Multiprocessor system with a Directory Protocol on a Mesh Interconnect

a directory receives a store request, it sends out invalidation requests and transitions to a busy state until the store has completed. While in this busy state, the directory will negatively acknowledge (or NACK) incoming requests to that line. These NACKed requests must be retried by that processor. If multiple outstanding stores were in flight from the directory, they could be reordered with respect to each other by the interconnection network; processors would then see different orderings of stores which would break the sequential consistency model. An alternative to NACKing requests when the directory is in a pending or busy state are NACK-free directories protocols [25]; this protocol queues pending requests at the home node until they can be serviced.

Figure 2.6 depicts a system built with a directory protocol. The SGI Origin [80] and the Alpha 21364 [100] are examples of systems employing directory protocols.

Directory protocols provide greater scalability by distributing the ordering points across various directory nodes; but latency penalties are paid for traveling to and accessing these ordering points. In large multi-chip multiprocessor systems, it is fairly easy to add extra bits to memory for directory storage and use directories to build scalable coherence protocols. However, to make performance acceptable for on-chip usage, directory information must be stored on-chip (not as part of off-chip memory as is done in distributed systems). Caching directory information on-chip represents substantial overhead; this additional area could be better utilized for the core itself or for larger on-chip caches.

In [94], significant directory cache miss rates were observed for commercial workloads (up to 74%). Directory cache miss rates will become an even more significant performance problem with server consolidation workloads. Multiple server workloads sharing the same resources on chip will touch large amounts of memory, placing enormous pressure on these directory caches. So, while directory protocols do provide designers with scalability; they do so with significant performance degradation when directory indirections and directory misses are factored in.

2.2.3 Coarse Grain Coherence Tracking

The coherence protocols proposed in Chapters 5 and 6 leverage coarse-grain tracking information to improve cache-to-cache transfer latency and reduce bandwidth overheads. In this section, we briefly discuss existing work on coarse-grained coherence.

In conventional systems, information about cache coherence is maintained on a per-block granularity. However, by looking at a set of contiguous addresses (a region), more optimizations can be enabled. A region is defined as a contiguous portion of memory consisting of a power of two number of cache blocks.

Coarse Grain Coherence Tracking (CGCT) [20] has been proposed to eliminate unnecessary broadcasts in order to improve the scalability of broadcast-based systems. Requests to non-shared regions of the address space can send a request directly to the memory controller rather than order their request on the broadcast bus which is a precious and limited resource. Tracking sharing patterns on a coarser granularity than cache lines can take advantage of spatial locality and reduce the storage overhead associated with maintaining this information. RegionScout [99] makes similar observations about the benefits of tracking information on a coarse granularity for coherence purposes.

The structures required by CGCT and RegionScout have been generalized in RegionTracker [147] to scalably incorporate additional functionality. Specifically, RegionTracker replaces a conventional tag array with region tracking structures, and is shown to achieve comparable performance to a conventional fine-grained tag array with the same area budget. With RegionTracker, one structure is used to encompass the functionality of both the fine-grained tags of a conventional cache and maintain additional information about larger regions.

2.2.4 Desirable Properties

Scalable cache coherence solutions are imperative to drive the many-core revolution forward. There is tension between the need for an ordered interconnect to simplify coherence and the need for an unordered interconnect to provide scalable communication.

Even with hundreds to thousands of cores on a chip, in the common case, only a few processors need to observe a given coherence request. With the single-writer, multiple-reader protocol invariant, only cores that are actively caching a block need to be made aware of a pending write to that block. Since providing global coherence, such as a broadcast, across thousands of nodes is impractical from both a performance and a power standpoint, a logical solution is to maintain coherence amongst just the current subset of readers. Infrequently, it will be the case that every core on-chip does need to observe a coherence request; this case must be handled correctly but not necessarily quickly.

In short, to address the key shortcomings of broadcast- and directory-based protocols, the desirable properties for scalable on-chip coherence are:

- **Limit coherence actions to the necessary subset of nodes:** this will reduce the power consumption and the interconnect pressure.

- **Fast cache-to-cache transfers:** send requests to sharers as quickly as possible, avoiding the overheads of directory indirections.
- **Limited bandwidth overhead:** limit unnecessary broadcasts to the entire chip; only communicate amongst the subset of sharers where coherence needs to be maintained.
- **Limited storage overhead:** make efficient use of on-chip directory storage and cache tag array storage.

Chapters 5 and 6 propose coherence protocols targeting these four properties. In this section, we have laid the groundwork for the coherence protocols that will be discussed in Chapters 5 and 6. Next we will discuss an emerging class of workloads whose unique sharing and communication characteristics can be leveraged in the design of coherence protocols.

2.3 Emerging Workloads

Now that we have provided background on interconnection networks and cache coherence protocols, we briefly discuss the workloads considered in this dissertation. The proposals in this dissertation are evaluated with scientific, commercial and server consolidation workloads in order to examine a variety of communication and sharing behaviors. We study workloads that have been traditionally used in SMP evaluations; these are scientific workloads are from the SPLASH-2 suite [146] and commercial workloads including SPECjbb, SPECweb [124], TPC-H and TPC-W [134].

Server consolidation workloads are a class of emerging workloads that present opportunities for coherence optimizations based on sharing characteristics. With server consolidation workloads, multiple discrete server applications share a many-core CMP.

Each application runs inside its own virtual machine and the multiple applications are coordinated through the virtual machine monitor. The common case for these applications is that any cache sharing will occur only *within* a virtual machine. Sharing across virtual machines (e.g. between applications) will be very rare.

By integrating a large-scale multiprocessor onto a single chip, where several resources may be shared, interference can come from many sources; this interference occurs at a much finer granularity than in traditional multi-chip, multi-board servers. Many cores may compete for the same interconnect bandwidth, memory controllers and cache resources. Sharing these resources to provide superior performance as well as ensuring fairness has been the subject of recent research [54, 61, 103] and will continue to grow in importance. The interconnect, which must be able to provide fast and reliable communication between any two cores may be shared across many cores. Memory controllers, which are central arbiters between the chips and memory, may receive requests from multiple cores. Lastly, caches may be shared to varying degrees amongst cores. The last level cache is the final opportunity to keep requests on chip before incurring the high latency effects of going off-chip. As we focus on communication and coherence in this dissertation, there are opportunities for interference among server workloads in the consolidated environment.

For the purposes of this discussion, sharing within a virtual machine will require local coherence operations and sharing across virtual machines will require global coherence operations. Note that the term local is not meant to imply physical locality between requests but rather that local requests are limited to those nodes within a single virtual machine. Threads within a virtual machine can be scheduled to maintain physical locality, resulting in local coherence requests that only travel short distances to other cores. However, the coherence mechanisms explored in subsequent chapters are not predicated on physical locality and may ease the burden of scheduling.

To reduce memory pressure, Waldspurger proposes content-based sharing, the sharing of read-only pages across virtual machines [138]. This inter-vm sharing could precipitate more global coherence between virtual machines; however, this is limited to read-sharing. We do not model any inter-vm sharing of this kind for our coherence protocols; however, inter-vm sharing can be handled by these protocols. Virtual machine monitor code is also shared across virtual machines.

Recently, server consolidation workloads have been receiving attention in the research community. Coherence protocols such as the Virtual Hierarchies protocols [97] leverage this class of workloads for performance optimizations. Analysis has been done to explore the impact of the virtual machine overheads [10]. The interaction of multiple virtual machines has also raised quality of service issues [54, 61, 103]. Wells et al. [141] have explored scheduling of these workloads.

In the subsequent chapters, we first explore interconnect optimizations addressing the key problems of router overhead and multicast routing. Next we address the bottlenecks present in cache coherence protocols for many-core architectures.

2.4 Conclusion

In this chapter, we have provided background on interconnection networks, cache coherence protocols and emerging workloads. We focus our background discussion of interconnects on the router architectures since both HCS (Chapter 3) and VCTM (Chapter 4) modify the router architecture. The first proposes modifications for low latency while the second proposes modifications for improving routing and throughput of multicast messages. The latency associated with network routers impacts performance; specifically, the low-latency delivery of coherence requests is imperative.

Interconnection networks and cache coherence protocols are tightly coupled. First, the cache coherence protocol is responsible for the bandwidth demands placed on the interconnection network. These bandwidth demands have an impact on a given protocol's ability to scale to a large number of cores; Circuit-Switched Coherence, discussed in Chapter 5 places low bandwidth demands on the interconnect through the use of pair-wise traffic. Second, cache coherence protocols may require an ordered interconnect for correctness; there is tension between providing an ordered interconnect and offering the high bandwidth required for large systems. We explore the ability of the interconnect to provide both ordering and high bandwidth in Chapter 6.

Finally, emerging workloads such as server consolidation present new sharing patterns; we explore optimizing the coherence protocol for these applications. The vast majority of traffic in these applications is going to be local; we propose a hierarchical protocol that accelerates this local coherence with multicasting (Chapter 6). The designs proposed in the following chapters emphasize the intertwining of application behavior, cache coherence protocols and interconnection networks in chip multiprocessor architectures.

Chapter 3

Hybrid Circuit Switching

Hybrid Circuit Switching optimizes the router pipeline to streamline on-chip communications. Router pipeline delay can contribute significantly to on-chip network latency and therefore is an important candidate for improvement. Before presenting the router design, we characterize the on-chip communication that motivates this work.

3.1 Communication Characterization

This section presents characterization data to further stress the importance of solving existing communication challenges, particularly reducing router overhead. We characterize the importance of on-chip network latency to overall performance. Additionally, we demonstrate the contribution of one type of communication pattern: pairwise sharing in real workloads.

For the suite of commercial and scientific workloads evaluated (see Section 3.5.1 on page 54 for details on workloads, machine model and additional simulation parameters), the network latency of a 4x4 multicore design can have a high impact on performance (Figure 3.1) while the bandwidth demands placed on the network are relatively low (Figure 3.2). Figure 3.1 illustrates the change in overall system performance as the per-hop delay is increased from 1 to 11 processor cycles. When a new packet is placed on a link, the number of concurrent packets traversing that link is measured (including the new packet); this is the channel load [34]. The average is very close to one, illustrating

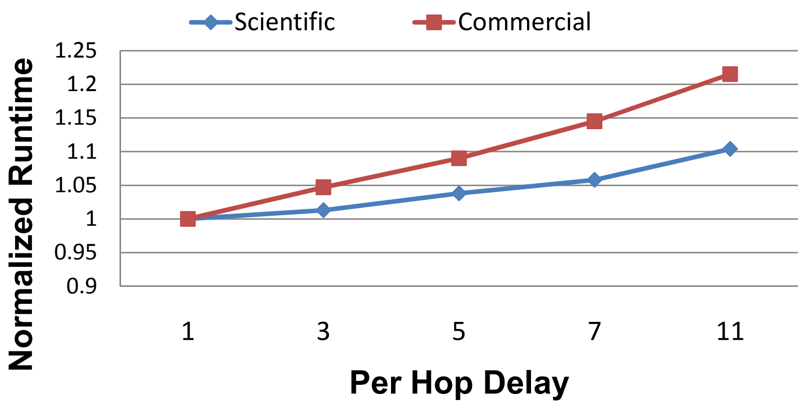


Figure 3.1: Impact of Interconnect Latency on Overall Performance

very low link contention given our simulation configuration.

Wide on-chip network channels are significantly underutilized for these workloads while overall system performance is sensitive to interconnect latency. An uncontended 5-cycle per-hop router delay in a packet-switched network can lead to 10% degradation in overall system performance. As the per-hop delay increases, either due to deeper router pipelines or network contention, overall system performance can degrade by 20% or more. With the use of simple in-order cores, the ability to tolerate this delay is limited. Looking forward, as applications exhibit more fine-grained parallelism and more true sharing, this sensitivity to interconnection latency will become more pronounced. This latency sensitivity coupled with low link utilization motivates the exploration of circuit-switched fabrics for many-core architectures.

3.1.1 Pair-wise Sharing

One of the goals in this research is to provide optimizations for certain communication patterns and types while maintaining a flexible substrate that performs well in the absence of these patterns. The first application behavior that we take note of is pair-wise sharing. Our workloads exhibit *frequent pair-wise sharing* between cores. This behavior can be due to the presence of migratory sharing and producer-consumer

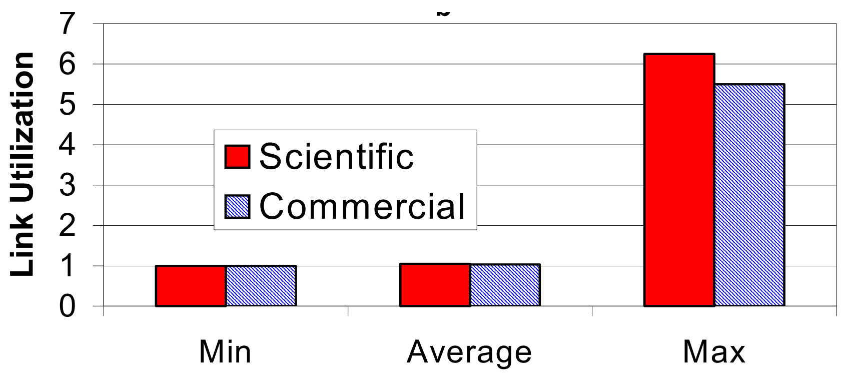


Figure 3.2: Bandwidth Demands placed on On-Chip Network

relationships. Prior work has also shown that processor sharing exhibits temporal locality and is often limited to a small subset of processors (e.g. [15, 37]). Designing a router architecture to take advantage of such sharing patterns can out-perform even a highly optimized packet-switched router.

Figure 3.3 illustrates the percentage of on-chip misses that can be satisfied by cores that recently shared data with the requester. The requester might have requested other cache blocks from the core or sent cache blocks to the other core. The categories on the x-axis indicate the number of most recent sharers and the corresponding percentage of on-chip misses that can be satisfied by one of those cores on the y-axis. For example, with the commercial workloads, the two most recent sharers (of any cache block) have a 65% chance of sourcing data for the next cache miss. The likelihood of a recent sharer supplying the data improves when we consider recent sharers for smaller spatial regions of memory.

Using the above characterization as motivation, the next section presents the hybrid circuit-switched network targeting pair-wise sharing.

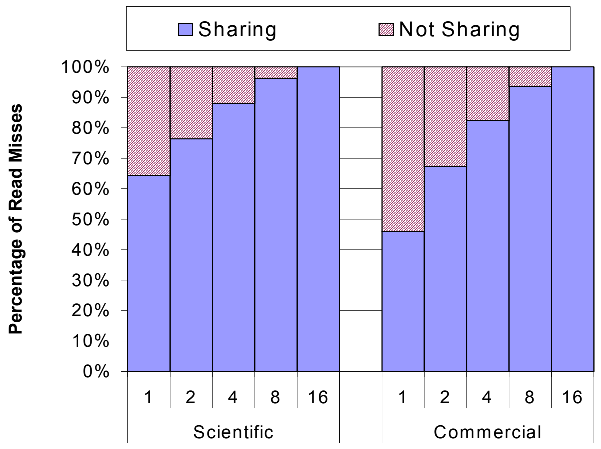


Figure 3.3: On-chip misses satisfied by recent sharer(s). On-chip misses are 54% and 70% of all misses for the scientific and commercial workloads evaluated.

3.2 Hybrid Circuit Switching Overview

Switching or flow control techniques used in networks can be broken down into buffered and bufferless. Packet-switching is a buffered flow control technique. There are several types of packet-switching that are employed in networks (distinguished by the granularity at which resources are allocated). Virtual channel flow control is utilized in our baseline router design. Buffers and channels are allocated in a per-flit basis at each hop in the network. Only the head flit is responsible for allocating the virtual channel but all flits are responsible for allocating necessary buffer space and channel bandwidth. Allocating resources on a flit granularity reduces the buffering required in the network.

Circuit-switching is a form of bufferless flow control. With circuit-switching flow control, channel bandwidth is allocated to an entire packet from the source to the destination (across multiple hops). In traditional circuit-switching, a probe (setup flit) is injected into the network to reserve the necessary channel bandwidth. Once the source receives an acknowledgment message that all channels have been acquired, it injects the circuit-switched packet into the network; other messages needing to acquire the reserved bandwidth are blocked until the current circuit-switched packet releases its resources.

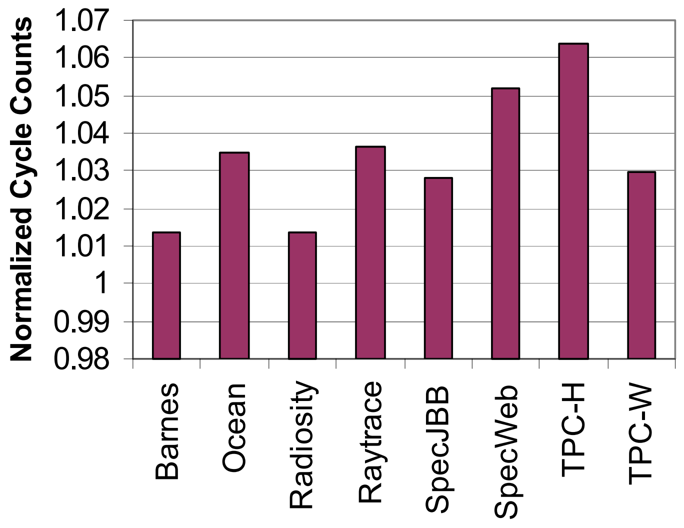


Figure 3.4: Performance of Traditional Circuit Switching

The area and power associated with buffering is removed in a circuit-switched network; however, messages pay a long startup latency (but thereafter pass through individual routers with only the switch traversal stage).

Our investigations show that traditional circuit-switched networks do not perform well, as circuits are not reused sufficiently to amortize circuit setup delay. As seen in Figure 3.4, every application saw a *slowdown* when using traditional circuit-switched networks versus an optimized packet-switched interconnect for a 16 in-order core CMP (simulation parameters can be found in Section 3.5.1). A second downside to a traditional circuit-switched network is the low bandwidth that it provides. Reserving channels across multiple hops severely restricts the bandwidth offered by the network.

These observations motivate a network with a *hybrid router design* that supports both circuit and packet switching with *very fast circuit reconfiguration*. The key design goals for this network are to avoid the circuit setup delay of traditional circuit switching and offer bandwidth comparable to that of the baseline packet-switched network.

3.2.1 Hybrid Circuit Switched Network

As shown in Figure 3.5, our design consists of two separate mesh networks: the main data network and a tiny setup network. The main data network supports two types of traffic: circuit-switched (CS) and packet-switched (PS). Circuit-switched messages and packet-switched messages are interleaved on the same physical resources within the network. The latency benefits of circuit-switching are derived from multiples uses of a circuit between reconfigurations. To increase the lifetime of a single dynamic circuit instance, we partition each link into multiple narrower physical channels. Multiple physical channels allow multiple circuits to occupy the same link concurrently which reduces the frequency of reconfiguration.

In the data network, there are C separate physical channels, one for each circuit. To allow for a fair comparison, each of these C channels has $1/C$ the bandwidth of the baseline packet-switched network in our evaluations. A baseline packet-switched network has a channel width of D data bits along with a $log_2(V)$ virtual channel ID. Flits on the data network of our hybrid circuit switched network are D/C wide, plus the virtual channel ID for packet-switched flits and an additional bit to designate the flit type (circuit- or packet-switched). A single setup network is shared by all C circuits.

3.2.2 Setup Network

Similar to a traditional circuit-switched network, the setup network handles the construction and reconfiguration of circuits. The setup network is responsible for storing the switch configuration information for active circuits. In a traditional circuit-switched network, a packet must wait for acknowledgment that a circuit has been successfully constructed before leaving the source node. A key feature that distinguishes our hybrid network from a traditional circuit-switched network is that we do not rely on acknowledgment messages. The packet payload is piggy-backed immediately behind the circuit

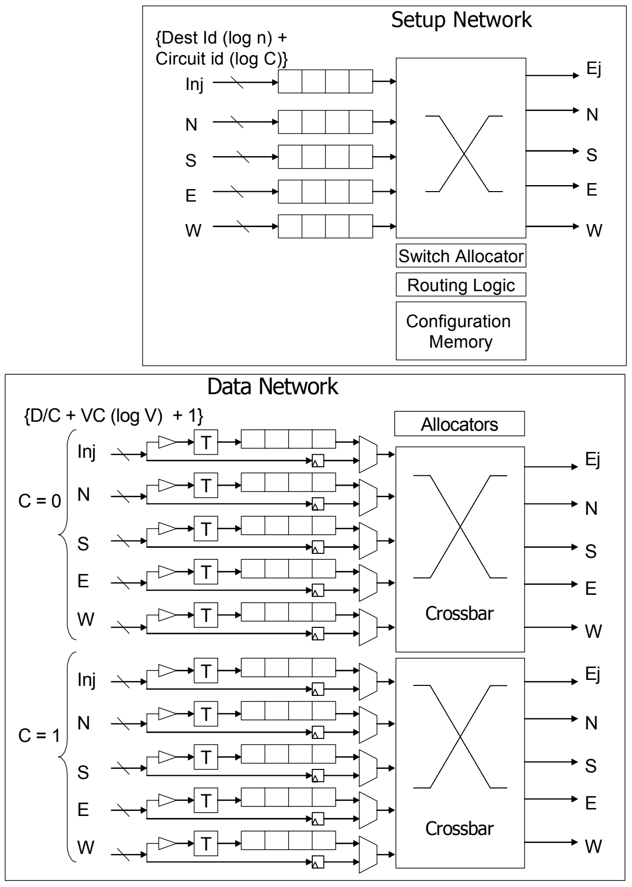


Figure 3.5: Hybrid Circuit-Switching Router design

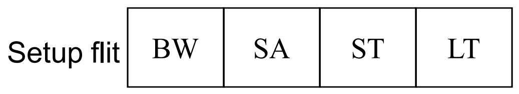


Figure 3.6: Setup Network Router Stages

setup request; this piggy-backing eliminates the latency penalty associated with setting up a new circuit.

At the time of injection, the network interface controller chooses one of the C circuit planes to be used by the circuit under construction. If there are no unused circuit planes, the least recently used circuit at the injection site will be reconfigured as packet-switched whilst the new circuit request will take over the old circuit. Incoming circuit-switched flits intended for this reconfigured circuit will henceforth be tagged as packet-switched flits and will traverse the packet-switched pipeline from this node until reaching their destination. The setup network will dispatch a control flit to the source of the old circuit notifying it of the reconfiguration. This control flit signals the source to either stop sending circuit-switched flits or re-establish the circuit. As this control flit travels upstream along the circuit-switched path, it deconfigures each segment of the circuit. This is analogous to sending credits upstream to indicate that a downstream buffer is available. The difference in this case is that the control flit slows the arrival of reconfigured circuit-switched flits by reconfiguring them closer to the source; as a result of this, we do not require additional buffering at each router to handle reconfiguration. The control flit prevents buffer overflow due to too many circuit-switched flits arriving at the reconfigured node. When a circuit has been reconfigured, the rate of arriving flits may be greater than the rate that the newly converted packet-switched flits can be consumed, which could lead to overflow if the source is not notified.

The routers in the setup network have three pipeline stages (shown in Figure 3.6), similar to stages of our baseline packet-switched router. Speculation and virtual channel allocation are removed from the setup network router. Virtual channels are unnecessary since traffic on the setup network is low and wormhole routing is sufficient and requires less overhead. When a setup flit arrives, consisting of the destination field ($log_2(N)$ bits, where N is the number of nodes) and the circuit number ($log_2(C)$, where C is the number of physical circuits), it will first be written to an input buffer (BW). Next, it will go through switch arbitration (SA), with each port having a C:1 allocator. This is followed by a circuit reservation on the data network which sets up the data network switch at that current node to route incoming circuit-switched flits correctly; successful switch arbitration determines the ordering of two setup flits requesting the same circuit resources. The setup flit then traverses the crossbar (ST) and the link (LT) towards the next router in the setup network. The width of the setup network is $log_2(N) + log_2(C)$ bits. Because the the setup network is very narrow, the switch allocation and switch traversal stages occur in a single clock cycle. This allows data trailing the setup flit to experience lower latency as its routers are setup in advance of its arrival on the data network.

The physical circuit plane C is selected at the injection router based on LRU information and the circuit number ($log_2(C)$ bits) is stored at the network interface controller for access by future circuit-switched flits injected at that router. A circuit must remain in the same physical circuit plane from source to destination. As a result of this constraint, $log_2(C)$ bits are sufficient to identify the circuit.

3.2.3 Circuit Switched Pipeline

The circuit-switched pipeline on the hybrid network is depicted in Figure 3.7. To allow circuit- and packet-switched flits to intermingle throughout the network, we add an extra bit field to each flit indicating if this flit is a circuit- or packet-switched flit. When a flit enters the router pipeline, the circuit field is checked (CFC). If the field is set, this is a circuit-switched flit and will proceed through the pipeline shown in Figure 3.7, bypassing directly to the switch traversal (ST). The switch was already

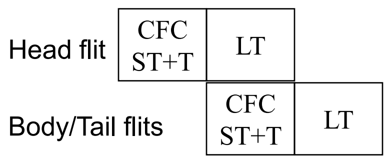


Figure 3.7: Circuit-Switched Router Pipeline. CFC: Circuit Field Check, ST: Switch Traversal, T: Tagging, LT: Link Traversal.

configured to the appropriate output port when this circuit was originally established. When enabled, the tagging stage (T) flips the circuit field bit for incoming packets. The tagging stage is enabled when a reconfiguration has taken place. By flipping the circuit field bit, flits originally intended for a circuit will now be redirected to packet buffers. In subsequent hops, the tagging stage will not be enabled but former circuit-switched flits will stay packet-switched until they arrive at their destination.

A hybrid router architecture with two circuit planes is depicted in Figure 3.5. If the circuit field is set and the reconfiguration signal has not been asserted by the setup network, then the incoming flit will take the direct path to the crossbar (the lower lane for each port shown in Figure 3.5). If the circuit field has been cleared or the circuit has been reconfigured, the flit will take the packet-switched path (upper lane) and be written into an input buffer.

This circuit-switched pipeline is nearly identical to the highly optimized single-cycle packet-switched router in Figure 2.4c. However, note that circuit-switching is able to achieve better performance than a single-cycle packet-switched router under certain communication patterns. In the packet-switched baseline, multiple incoming flits prevent bypassing; with circuit-switching, these flits can occupy different circuit planes and proceed simultaneously through the router in a single cycle. This opportunity will be explored in greater depth in the evaluation (Section 3.5).

3.2.4 Packet Switched Pipeline

If the circuit field is zero, the flit entering the router is packet-switched and will be buffered, proceeding through the packet-switched pipeline shown in Figure 2.4. The allocator in the packet-switched pipeline is designed to enable packet-switched flits to *steal* bandwidth from idle circuits. The packet-switched flit will perform speculative virtual channel and switch allocation each subsequent cycle after the buffer write stage until it is able to steal bandwidth through the switch. The router receives a signal from the input ports indicating the presence or absence of incoming flits on the circuit C that the packet-switched flit has also been assigned to and therefore is attempting to steal bandwidth from. If there are no incoming flits for that circuit, the packet-switched flit arbitrates for the switch. Once a packet-switched flit has won passage through the crossbar, it then traverses the output port and continues to the next hop. The circuit field bit remains set to zero; this flit will continue to be interpreted as packet-switched and buffered appropriately at each hop; bandwidth stealing will occur at each hop.

To prevent the unlikely scenario where packet-switched flits become starved by circuit-switched flits, a timeout mechanism is used to trigger the reconfiguration of the starved circuit plane. The timeout triggers if a flit has been waiting in a buffer for 15 cycles. This will force circuit-switched flits into the packet-switched pipeline and allow the starved packet-switched flits to make forward progress. Starvation of flits is characterized in Section 3.5.

3.2.5 Narrow Data Network

To prevent frequent reconfiguration, hybrid circuit switching subdivides the network links into multiple narrow links for multiple circuits. The same subdivision can be done for the baseline packet-switched network. A recent many-core architecture from Tilera [142] utilizes multiple packet-switched networks. In their design, the networks are assigned to different classes of traffic, e.g. processor to memory traffic, processor to processor traffic, etc. Utilizing multiple networks can improve throughput by reducing the network load but can suffer from increased serialization delays. Correct speculation rates can be improved by the presence of fewer flits in the input buffers which will enable more frequent bypassing.

3.3 Power and Area Overhead

Area and power are first order design constraints when dealing with on-chip networks. Circuit-switched flits do not trigger activity in all router stages (e.g. no buffer read/write, no allocation); as a result, circuit-switched flits consume less dynamic power as they traverse the network than packet-switched flits. We look at the area and power impact of the hybrid circuit-switched interconnection network.

Using Orion and a 70nm technology [139], a setup network router (including configuration memory) consumes less than 2% of the overall router power. An activity factor of 1 was used to provide a worst case analysis. The configuration memory in the setup network consists of 25 bits for each of the C circuits. For each of the five input ports, 5 bits drive the select signals of the crossbar to activate the correct output port. The setup network also consumes additional area due to the addition of buffers, switch allocators and wiring. However, as the setup network is very narrow, we do not expect significant area overhead.

On the data network, components of the hybrid router that increase power consumption and area are C D/C-wide multiplexers that select from either the circuit or the buffers and tagging hardware to reset the circuit bit in each flit. To reduce the area and power consumed by the C D/C multiplexers, we add an additional input to the 4:1 multiplexers in the baseline router that select between the VCs. Replacing 4:1 multiplexers with 5:1 multiplexers increases the power consumption of the router less than 1%. Power can be potentially lowered further by reducing the buffers and VCs in the hybrid router; however we chose to stick with the same configuration as the packet-switched baseline for a consistent comparison.

3.4 Discussion

In the following sections, we explore some of the potential issues and drawbacks of hybrid circuit switching. We discuss the potential for circuits to thrash in the network or to become fragmented. Policies governing when to establish circuits are explored and finally, we look at the ability of HCS to scale to larger systems.

3.4.1 Circuit thrashing

If two different source-destination pairs try to establish a circuit at the same time sharing a common link on the same circuit plane, these requests will be serialized in the setup network. The first request through the setup router will only hold the circuit for a single cycle making reuse by subsequent messages impossible. The following cycle, the second request will reconfigure the link and hold the circuit until a subsequent request wishes to claim the link. In the event that these two source-destination pairs continue thrashing by trying to establish circuits over the same link, their latency will degrade to the baseline packet-switched latency. Circuit thrashing is less likely with more available circuit planes.

3.4.2 Circuit Fragmentation

Circuit fragmentation due to reconfiguration can occur in the hybrid network. The reconfiguration mechanism only reclaims contentious links for a new circuit (not all the links of the circuit). The remaining fragments of the old circuit are left intact, thus leaving partial circuits in the network. Links in a circuit downstream from the reconfiguration site will no longer see circuit-based traffic intended for that circuit (packets intended for that circuit will have been tagged as packet-switched at the reconfiguration site). Therefore, until this link is claimed as part of a new circuit, there will always be idle bandwidth for packet-switched flits to steal.

In a scenario where a link in the circuit has been reconfigured at a node other than the source node, the source of that circuit would be unaware of the reconfiguration and could keep sending circuit-switched flits on the circuit fragment. These circuit-switched flits will be converted to packet-switched flits at the reconfiguration site; this could potentially cause a buffer back-pressure situation back to the source node if there is not enough bandwidth available on the contended link for the packet-switched flits to steal and continue on to their destination. As such, upon reconfiguration we send a notification flit back to the source of the circuit. The source of the circuit can then choose to re-establish the circuit or send packet-switched flits.

3.4.3 Setting up Circuits

Determining when and if to set up a new circuit impacts the reconfiguration frequency, circuit reuse potential and the possibility of circuit thrashing. In this work, two different policies regarding the set up of new circuits are explored. In the first policy, the decision to allow a new message to construct a circuit-switched path (if one is not already present) is made based on the nature of the message. For example, invalidation requests from the directory are not indicative of a pair-wise sharing relationship and therefore are injected as packet-switched flits. Read and store requests and data transfers will setup a circuit if one is not present. All types of messages can reuse an existing circuit between given source-destination pairs. We refer to this first policy as the limited setup policy. The second policy is to allow messages of any type to always initiate new circuit set up.

3.4.4 Scalability

Given the same traffic load on a larger network, there can potentially be higher circuit contention which would require more circuits, limiting the scalability of the proposed network. With a fixed channel width, subdividing to create more circuit planes increases the serialization delay; narrower links result in more flits per packet. However, in large systems, circuit switching has the potential for greater latency savings. As the hop count increases from source to destination, the number of cycles saved will also increase.

So far, our discussion of hybrid circuit-switching has assumed a mesh topology. There is nothing fundamental that limits hybrid circuit switching to a mesh. As such, to scale hybrid circuit switching and the interconnect in general, we suggest moving to a more scalable topology as the number of nodes in the system grows. Enriched connectivity (such as an express cube [33]) will not only lower circuit contention and thrashing, but can potentially reduce global cross-chip latency to just pure wire delay as circuits can be formed with mostly express links.

Future many-core workloads, such as server consolidation, are unlikely to require significant global communication. This limited sharing will reduce the need for large numbers of circuits that sprawl across the chip and will be particularly well suited to hybrid circuit-switching. Next we look at the impact of HCS on single multiprocessor workloads and server consolidation workloads.

3.5 Evaluation

In the following sections we present an evaluation of the hybrid circuit-switched network compared to an aggressive packet-switched baseline. Before presenting in-depth results on the strengths and weakness of HCS, we present the simulation methodology and workloads under consideration.

Table 3.1: HCS Simulation Parameters

Cores	16 in-order cores
Memory System	
L1 I/D Cache (lat)	32 KB 2-way set associative (1 cycle)
Private L2 Cache	512 KB (8 MB) total 4-way set associative 6 cycles, 64 byte lines
Shared L3 Cache	16 MB (1 MB bank at each tile) (12 cycles)
Main Memory Latency	100 cycles
Interconnection Network	
Router	8 Virtual Channels with 4 Buffers each Optimized Pipeline
Setup Network	Wormhole with 4 Buffers
Channel Width	32 Bytes

3.5.1 Methodology

To evaluate the proposals in this dissertation, a full-system simulation environment, PHARMsim is used [18,82]. Full-system simulation has the benefits of executing both operating system code and application code; additionally, the simulator faithfully models real system interactions. PHARMsim is built on SIMOS-PPC and is configured with 16 cores on a 4x4 mesh. Included in our simulation infrastructure is a cycle-accurate network model including pipelined routers, buffers, virtual channels and allocators. To mimic industry trends of providing simpler cores on-chip, we model in-order cores. For performance results, statistical simulation is used with 95% confidence intervals [9]. The parameters used to configure HCS experiments are presented in Table 3.1. The baseline interconnection network used for comparison is the optimized packet-switched pipeline presented in Chapter 2. Under low load conditions, the delay through this baseline router is a single cycle. For moderate to high loads, speculation will fail more frequently, increasing the delay to three pipeline stages.

Despite its benefits, there are drawbacks to full-system simulation. In particular it is slow and can sometimes obscure the effects of one subsystem. To focus on the

Table 3.2: Benchmark Descriptions

Benchmark	Description
SPECjbb	Standard java server workload utilizing 24 warehouses, executing 200 requests
SPECweb	Zeus Web Server 3.3.7 servicing 300 HTTP requests
TPC-H	Transaction Processing Council's Decision Support System Benchmark, using IBM DB2 v6.1, running query 12 with a 512MB database and 1GB of memory
TPC-W	Transaction Processing Council's Web e-commerce benchmark, DB Tier, browsing mix, 40 transactions
Barnes-Hut	8K particles, full end-to-end run including initialization
Ocean	512x512 full end-to-end run (parallel phase only)
Radiosity	-room -batch -ae 5000 -en 0.050 -b 0.10 (parallel phase only)
Raytrace	car input (parallel phase only)

interconnect, we have extracted our network model and also performed detailed network level simulations in isolation.

We use a variety of commercial and scientific benchmarks to perform the evaluation in this dissertation, including SPECjbb, SPECweb [124], TPC-H, TPC-W [134] and several benchmarks from the SPLASH-2 suite [146]. Descriptions of those benchmarks are provided in Table 3.2.

In addition to single workloads, we evaluate HCS in the presence of server consolidation workloads. For these experiments we create heterogeneous workload mixes of 4-core workloads of SPECjbb, TPC-H and TPC-W. For example, 4 copies of SPECjbb are run with 4 copies of TPC-W. Each machine is overcommitted [141]; we employ a load balanced scheduler. As a result, threads from the same virtual machine can be scheduled anywhere on chip resulting in changing pairwise combinations.

To supplement our full-system simulation with commercial and scientific workloads, we further evaluate HCS through the use of synthetic traffic patterns. Two synthetic traffic patterns commonly used for interconnection network evaluation are uniform random and permutation traffic. With uniform random traffic each node can

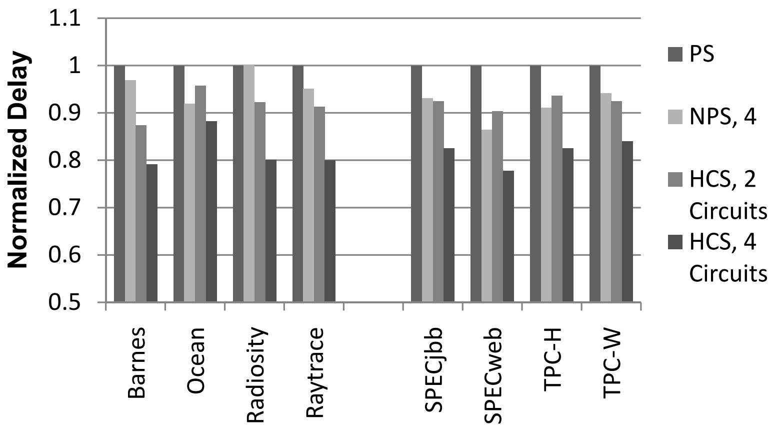


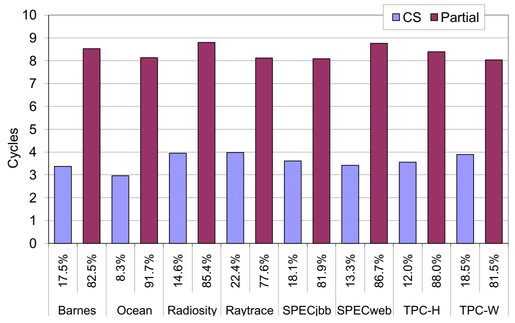
Figure 3.8: HCS Network Performance

communicate with any other node chosen at random. Under permutation traffic, each node communicates with one other node (all pair-wise communication). For both traffic patterns, we vary the injection rate to study the behavior under increasing loads.

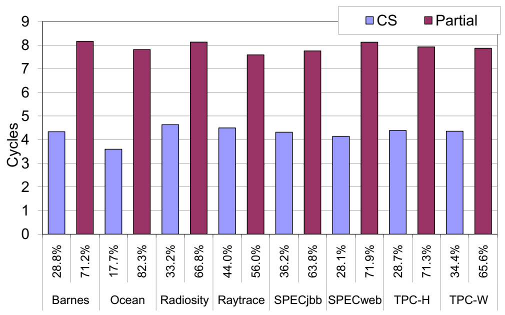
3.5.2 Network Performance Results

One of the primary goals of this work is to reduce the interconnect latency by removing the router overhead. Our hybrid circuit-switched network succeeds in reducing interconnect latency by as much as 23% as shown in Figure 3.8. We measure the average interconnect latency for two and four circuit planes as compared to the baseline packet-switched interconnect. Average packet delay is calculated as the time between the injection of the head flit and when the critical word arrives at the destination. The four circuit plane configuration gives us additional benefit as circuits can be maintained longer between reconfigurations and see more reuse, thus reaping more benefit. To combat the increased serialization delay that moving to four circuits would cause, we send the critical word first through the network. As such, for a fair comparison, we also send critical word first for HCS with two circuits and NPS with four circuits.

In addition to comparing HCS against the baseline packet-switched network, we



(a) 2 Circuits



(b) 4 Circuits

Figure 3.9: Latency Breakdown of Circuit-Switched Flits and Packet-Switched/Partial Circuit-Switched Flits

also present results for Narrow Packet-Switching (NPS); in NPS we partition the packet-switched network into four narrower networks (similar to the four narrow circuit planes in HCS). Each narrow network in NPS has two virtual channels with four buffers per VC. Total buffering is kept constants across all network configurations. When a packet is injected into the network, it selects an NPS network in a round robin fashion and continues on the same NPS network until reaching its destination. NPS provides modest improvement over packet-switching but under-performs when compared to HCS. NPS improves speculation probability over PS by distributing packets across multiple narrow channels. HCS still has the benefit of allowing pair-wise sharing packets to proceed quickly through the router.

Circuit-switched connections are designed to enable fast connections between frequently sharing cores; however, we do not want to sacrifice the performance of messages that do not involve frequent sharing. Our design is able to circuit-switch 18-44% of all flits with an average packet latency of 4.3 cycles. The remaining 56-82% of flits that are packet-switched or partially circuit-switched through the network still achieve a reasonable average interconnect latency of 7.9 cycles. Figure 3.9 gives the contribution of circuit-switched flits and non-circuit-switched flits to overall average network latency.

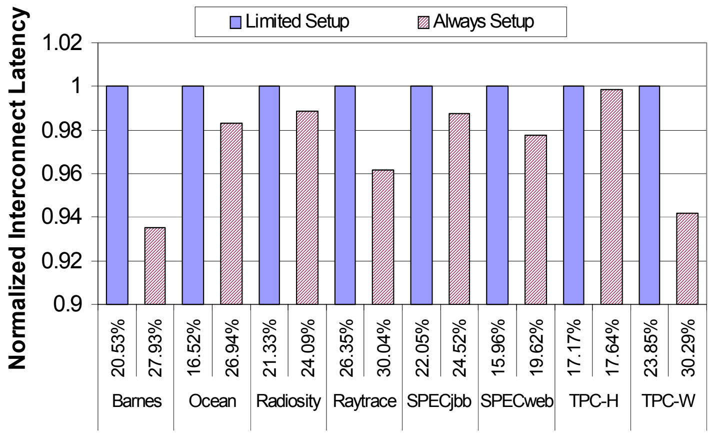


Figure 3.10: Impact on Network Latency of Restricting Circuit Setup to Certain Classes of Messages

Non-circuit-switched flits include both packet-switched flits and reconfigured circuit-switched flits (partially circuit-switched). The x-axis shows the percentage of overall network messages that are either circuit-switched or not. With four circuit planes, we see a larger contribution of purely circuit-switched flits; this is expected as reconfigurations are less frequent. This larger contribution of circuit-switched flits causes the overall average network latency to be lower. Policies that take further advantage of circuit-switched links and reduce unnecessary reconfigurations can further reduce interconnect latency.

3.5.2.1 Circuit Setup Policies

Two different policies regarding the setup of new circuits are compared in Figure 3.10. The first allows new circuits to be setup for a limited set of message classes. The second policies always sets up a new circuit when one is not present in the network. Interconnect latency in Figure 3.10 is normalized to the limited setup case. The numbers on the x-axis indicate the percentage of flits that are circuit-switched. Limiting the construction of new circuits to certain message classes causes a loss in opportunity as noted by an average increase in network latency of 3% with a maximum increase of

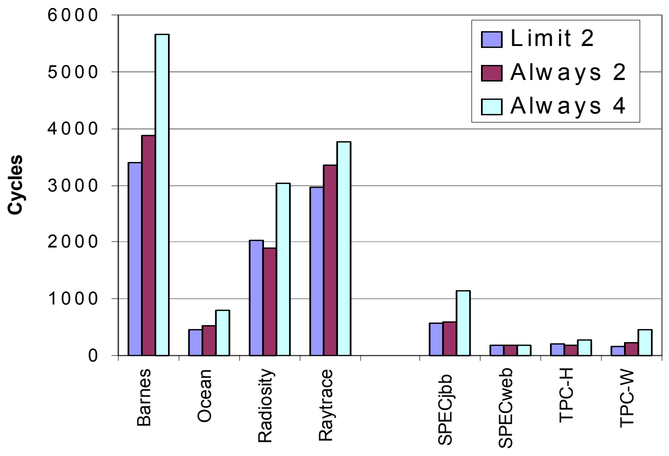


Figure 3.11: Average Link Stealing Interval. Limit 2 refers to a limited setup policy with 2 circuits. Always 2 and 4 refer to an always setup policy with 2 and 4 circuits respectively.

7%. Up to 10% more flits take advantage of circuit-switched paths when new circuits are always constructed. The limited setup policy does drive up individual circuit reuse by 30%. A policy that simultaneously achieves maximum circuit utilization and high circuit reuse could result in further performance improvement.

3.5.2.2 Packet-Switched Bandwidth Stealing

Limiting the setup of circuits increases the percentage of packet-switched flits and reduces the average interval between instances of link stealing at each input port. Scientific workloads see longer intervals between link stealing than commercial workloads due to the lower communication demands of the former as shown in Figure 3.11. In the always setup with two circuits case, Radiosity and TPC-H see lower time intervals between instances of link stealing; this can be attributed to the very frequent reconfiguration of links which causes circuit-switched flits to be transitioned into packet-switched flits and necessitates more bandwidth stealing.

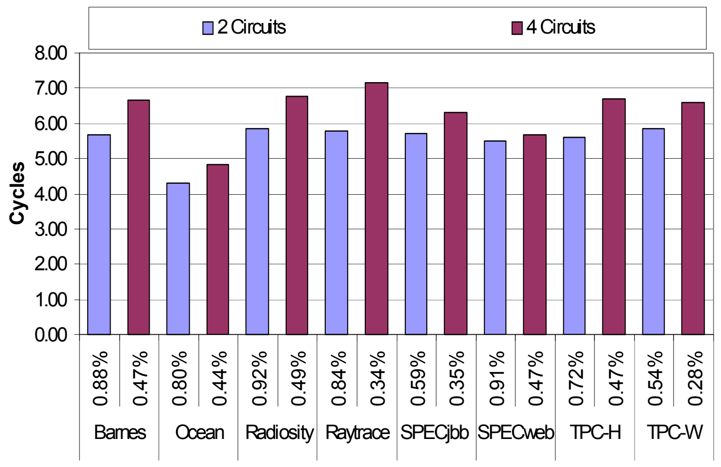


Figure 3.12: Wait time for Packet-Switched Flits (starvation)

3.5.2.3 Packet-Switched Flit Starvation

As described in section 3.2.4, to prevent the unlikely scenario of packet-switched flit starvation, a timeout mechanism forces the reconfiguration of the circuit starving the packet-switched flits to allow packet-switched flits to make forward progress. To characterize the potential for starvation, we measure the waiting time of packet-switched flits (with the 15-cycle timeout mechanism disabled) in Figure 3.12. The values on the x-axis indicate the percentage of all flits that must wait at least one cycle to steal bandwidth from a circuit; in all cases it is less than 1%. The y-axis shows the average number of cycles a request will wait; flits waiting zero cycles are omitted from the average. The average wait time for this small number of flits is less than seven cycles across all workloads. The average wait time increases slightly when the number of circuit planes is increased from two to four; this is explained by higher circuit utilization.

3.5.2.4 Setup Network Evaluation

Earlier in this chapter, we assert that a wormhole network is sufficient for the setup network. Figure 3.13 supports this claim. Utilization of the setup network is particularly low when circuit construction is limited to a subset of messages. The data

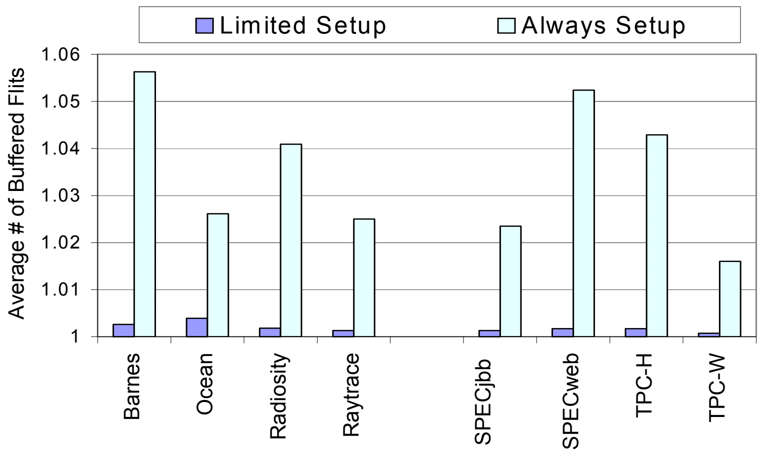


Figure 3.13: Channel Load of Setup Network

in Figure 3.13 is measured as the average number of flits waiting at a router when a new flit arrives (including the new flit); this is the channel load. The always setup policy drives up the load on the setup network but it is still at an acceptable level for wormhole switching. The average delay through the setup network is 7.33 cycles which reflects the very low utilization of the setup network (the average zero load delay through the setup network is 5.4 cycles). The setup network does not need significant buffering resources since low utilization is coupled with the small setup flit size (6-8 bits).

3.5.3 Full system results

Figure 3.14 shows the overall system performance for hybrid circuit-switching with two and four circuit planes and narrow packet-switching with four narrow networks, all normalized to the baseline packet-switched network. Moving from two circuits to four circuits shows an average of 3% reduction in execution time (up to 7%) for commercial workloads. Frequent reconfiguration prevents TPC-H from seeing any benefit from two circuits. When characterizing the sharing patterns in TPC-H, four recent sharers are needed to satisfy 64% of on-chip misses, in contrast to only two for the other commercial workloads. Ocean does not benefit from circuits since most misses go off-

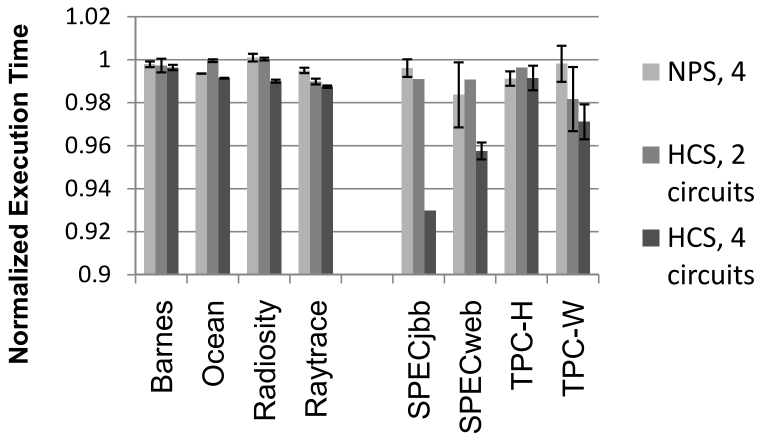


Figure 3.14: Overall Performance of HCS

chip, causing the miss latency to be dominated by the memory access time. The other scientific workloads see little benefit from hybrid circuit switching due to low overall miss rates and low numbers of coherence misses. The performance of scientific workloads is less sensitive to interconnection network latency than the commercial workloads. Increasing the number of circuits further would likely yield little benefit due to the increase in flits/packet. For most workloads, little improvement is gained from the NPS configuration; NPS does not provide the added performance improvement for pair-wise sharing that HCS targets.

3.5.3.1 Server Consolidation Results

In Figure 3.15, we examine the performance of HCS under server consolidation workloads. With these server consolidation workloads, each thread can only share with three other threads (since we consolidated multiple four core workloads and there is no inter-VM sharing). As a result, there are many fewer possible pairs than with the 16-core workloads; circuit-switched paths can persist for longer and provide lower latency between sharers. We see a degradation in the performance of TPC-W in Mix 2 (when paired with SPECjbb); however there is a net performance improvement as SPECjbb

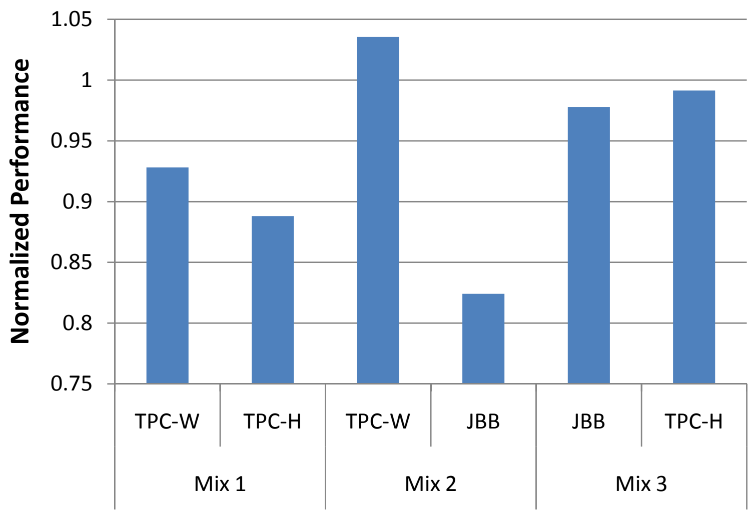


Figure 3.15: Server Consolidation Performance of HCS normalized to PS

benefits significantly from circuit-switched links.

3.5.4 Synthetic Traffic

In addition to performing full-system simulation for HCS, we further explore its performance through synthetic traffic patterns in Figures 3.16 and 3.17. Both figures show average interconnect latency in cycles across all injected packets. All simulations are run for 1 million cycles with increasing injection rates (or channel load) shown as a percentage of maximum link capacity along the x-axis.

With uniform random traffic in Figure 3.16, we reduce latency by 10-15% over packet switching across all loads prior to saturation, despite the fact that there is hardly any reuse of circuits due to the randomized traffic. HCS has lower latency at low to moderate loads primarily because circuits are always given priority on their first use – i.e. the first packet on a circuit always bypasses through the router whereas in packet-switching, bypassing within the router will not be possible if there are more than one flit in the entire router.

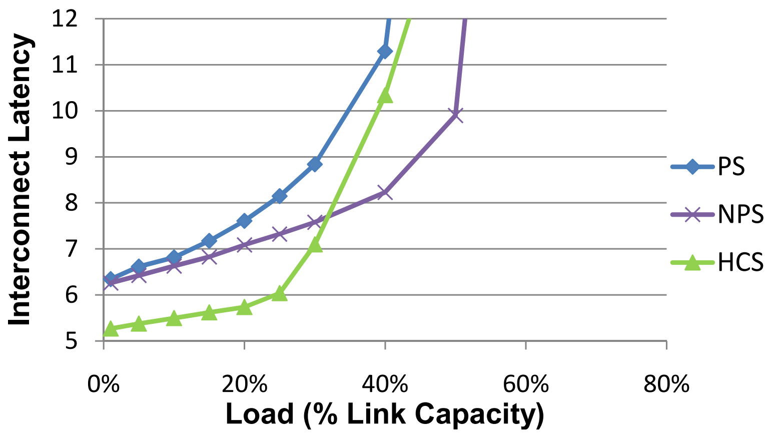


Figure 3.16: Performance of HCS with Uniform Random Traffic

At very low utilization circuit-switching out-performs the packet-switched baseline since we piggyback data flits along with the setup flits, which shaves one cycle off the serialization latency. Lookahead techniques require one extra cycle of setup in the first router or network interface controller giving packet-switched messages a longer latency even at low loads.

Figure 3.16 also shows network latency results for Narrow Packet Switching (NPS). At moderately low loads, approximately 40% more incoming packets are able to bypass directly to the crossbar with HCS than with NPS or PS. When the load reaches roughly 30%, HCS and PS bypass similar numbers of packets resulting in similar network saturation points. This attests to the effectiveness of idle bandwidth stealing for packet-switched flits in HCS. At moderate loads (20-40%), NPS is able to bypass 5-18% more packets than PS, delivering higher saturation throughput. This matches intuition since NPS trades off serialization delay with bandwidth.

In Figure 3.17, we simulate the performances of HCS under permutation traffic, where each node communicates with one other node. Since HCS specifically targets pair-wise sharing, we would expect this type of traffic to benefit from circuit reuse and perform very well. At low to moderate loads, HCS improves network latency by 20% over PS and saturates at higher utilization. NPS performs slightly better than PS at

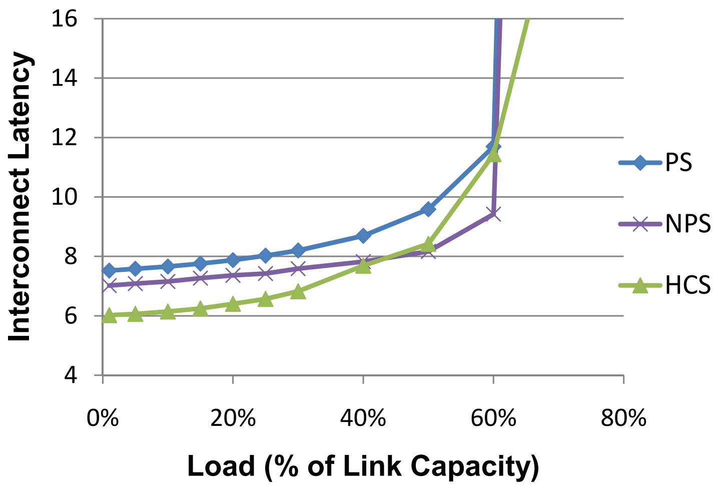


Figure 3.17: Performance of HCS with Permutation Traffic

very low utilization. As the load increases, NPS performs considerably better than PS (10-15% lower latency) since the load is distributed across multiple narrow networks allowing speculation to be more effective. With round-robin placement of packets on the NPS networks, these networks eventually saturate at a similar load to PS.

Studying permutation traffic on the HCS network leads to an interesting observation. As the load increases, a single circuit becomes saturated and it becomes beneficial to establish additional circuits for the overloaded pair. When the load reaches 25%, three packets (on average) are queued up at the injection port of a single circuit, even though three out of four circuits may be completely idle. Permutation traffic represents fully pair-wise traffic between sources and destinations leaving some extra circuit resources available for these redundant circuit paths. Distributing the load across multiple redundant circuit leads to the performance results shown in Figure 3.17.

To reiterate, Figure 3.16 represents worst-case circuit reuse behavior (we observe much higher reuse for real applications), so it is not surprising that HCS saturates somewhat earlier than the NPS case. Additionally HCS sees robust latency reductions at low utilization. In contrast, Figure 3.17 demonstrates the robustness of HCS over both PS and NPS under all traffic loads when there is significant circuit reuse.

In summary, HCS is able to reduce network latency by up to 23% and improve overall performance by up to 7% for real workloads. HCS achieves these performance gains over a highly optimized baseline packet-switched router with a single-cycle delay for low loads.

3.6 Related Work

In the following sections, we explore related works in hybrid networks and prior research targeting router delay.

3.6.1 Hybrid Networks

Hybrid Circuit Switching proposes a hybrid circuit-switched router that interleaves circuit- and packet-switched flits on the same physical network with low area and power overhead. Several other hybrid network designs have been proposed in the literature. SoCBus [143] only packet-switches configuration messages but holds the data at the source until the setup is acknowledged. All data in their proposal must be circuit-switched through the network. Wolkotte et al. [145] propose a design that has both circuit- and packet-switching; however, it is our understanding that these two networks are physically separate. The packet-switched network is used for reconfiguration and best-effort traffic while the circuit-switched network is used for guaranteed-throughput communications. Wave-switching [38] combines circuit-switching and wave-pipelining but in their design, wormhole-routed and circuit-switched data do not interact and have physically separate resources. Pipeline circuit switching [48] requires that a setup and acknowledgment message be sent and received prior to the data transfer. HCS interleaves the two types of flits on the same physical links and does not require an acknowledgment message to begin transmitting on the circuit.

Another hybrid network [113] combines a bus architecture with a switched network; the bus allows processing elements with communication affinity to transfer data quickly without high (router) overhead while the switched network provides scalability. HCS provides greater flexibility as sharing cores does not need to be physically close to experience low communication latency.

3.6.2 Router Delay

Hybrid Circuit Switching is able to remove the buffer write and virtual channel/switch allocation stage for up to 44% of flits with a four circuit configuration and always setup policy. Other recent work [75] also successfully removes a significant portion of this overhead through Express Virtual Channels (EVCs). EVCs create express virtual paths that bypass nodes for a given number of hops, allowing them to reduce both delay and energy consumption in the network. EVCs provide a general framework to accelerate messages. To achieve maximum benefit, EVCs are limited to a small number of hops; the HCS network will show increasing gains as the hop count goes up under low loads. Flit reservation flow control [108] avoids the router overhead by sending a control flit to reserve network resources ahead of data flits; however, this design suffers from increased router complexity and high overhead.

Our baseline packet-switched router is more aggressive than a recent Intel router design [76] which has a four-stage pipeline to accommodate an aggressive 16-FO4 clock cycle; at low loads, it has a single-stage pipeline. Recent routers have aggressively pursued a single-cycle pipeline, but only at low loads. The TRIPS network uses lookaheads and bypassing to realize a single-stage router [52], while Mullin's Lochnest router uses aggressive speculation to shorten the pipeline to a single cycle at a 35-FO4 clock cycle [102]. RAW's dynamic network consists of a 3-stage pipeline [132]. Hybrid circuit-switching can be seen as another technique to further shorten the router pipeline in the presence of certain sharing and communication patterns.

Router delay can be avoided through reduced hop count. Topologies such as a flattened butterfly [69] can be used to reduce the number of router topologies; however this design suffers from the use of long global wires. To evenly distribute the load in the network, the flattened butterfly also relies on non-minimal adaptive routing which can increase packet latency. HCS is able to bypass routers without the drawback of long global wiring delays. Kim et al. propose polymorphic networks [70]; these networks can be reconfigured to reduce latency for specific traffic patterns. Unlike hybrid circuit switching with reconfigures frequently and at a fine granularity, polymorphic networks incur significant overhead for reconfiguration. The granularity of reconfiguration is likely to be on application boundaries.

3.6.3 Multiple Networks

Both RAW [132] and Tilera [142] employ multiple narrow networks similar to our NPS baseline. RAW partitions its networks into static and dynamic networks. The static networks send packets along routes that are precomputed by the compiler. The dynamic networks send packets along routes that utilizes network routing and allocation of resources similar to the packet-switched baseline. Tilera implements five narrow packet-switched networks to be used by different classes of traffic.

3.7 Conclusion

In this chapter, we demonstrate the potential for circuit-switched networks for multi-core architectures. Our HCS network successfully overcomes some of the drawbacks associated with circuit-switching, specifically: avoiding setup overhead, reconfiguring circuits on-the-fly, and interleaving circuit- and packet-switched flits on the same physical resources. HCS optimizes pairwise communication; next we consider the router bottlenecks for more generalized communication behavior: multicast sharing.

Chapter 4

Virtual Circuit Tree Multicasting

We motivate interconnection designs by observing sharing behavior and the resulting communication patterns. Hybrid Circuit Switching focuses on pair-wise communication; in this chapter, we examine more widely shared data that can lead to one-to-many communication patterns.

Current state of the art on-chip networks provide efficiency, high throughput, and low latency for one-to-one (unicast) traffic. The presence of one-to-many (multicast) or one-to-all (broadcast) traffic can significantly degrade the performance of these designs, since they rely on multiple unicasts to provide one-to-many communication. We begin by characterizing the problems associated with the multiple unicast approach to multicasting; then we highlight the opportunities in existing proposals for leveraging hardware multicast. After motivating the need for multicasts, we present our solution, Virtual Circuit Tree Multicasting.

4.1 Communication Characterization

Pair-wise sharing characteristics from the previous chapter can be generalized to examine sharing among a small subset of cores: multicast sharing. Pair-wise communication is realized through unicast messages which current state-of-the-art routers handle efficiently. However, efficient support for multicast sharing is lacking in current router designs as the de facto standard is the multiple unicast approach.

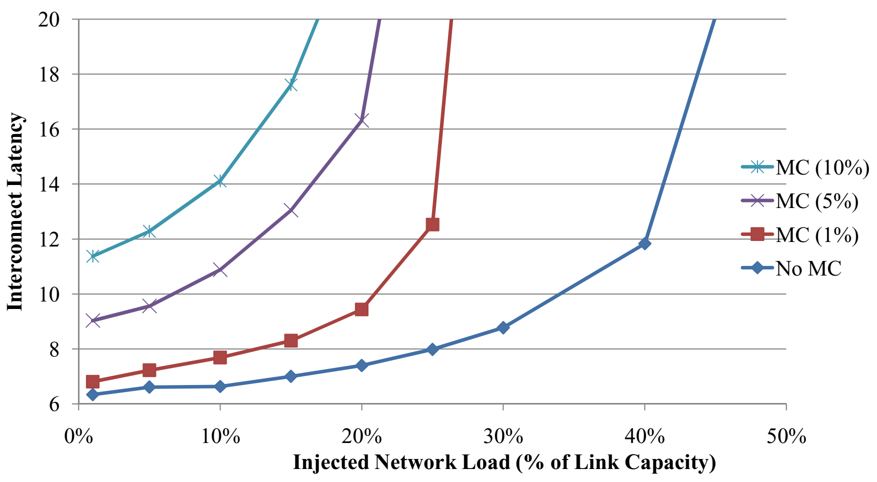


Figure 4.1: Performance of multicasts on packet-switched interconnect

Without support for multicast communication significant performance and throughput degradation is observed. Figure 4.1 shows the performance of a state-of-the-art packet-switched router in a 4x4 mesh in the presence of uniform random traffic. This router performs well when all injected packets are intended for a single destination (No MC). When we start injecting multiple packets in the same cycle, at the same source destined for multiple nodes, we see significant throughput degradation; MC 1% converts 1% of injected packets into a multicast destined for a random number of destinations (<=15). If 1% of injected packets are multicasts, the saturation throughput point drops from 40% capacity to 25% capacity. Saturation is defined to by the point at which the latency becomes asymptotic [34]. The network saturates at 20% and 15% for traffic with 5% and 10% multicasts respectively. These multicast packets are broken down into multiple unicasts by the network interface controllers as the baseline packet-switched routers are not designed to handle multiple destinations for one packet.

Even in relatively small systems, on the order of 16 nodes, multiple unicasts can significantly degrade performance, as demonstrated in Figure 4.1. The poor performance and throughput of the multiple unicast approach will be exacerbated by the presence of more one-to-many or one-to-all communication as systems grow to encompass dozens or hundreds of nodes.

4.2 Broadcast and Multicast Architectures

Several recent research proposals could leverage hardware multicast support to further enhance their performance. In this section, we will explore the opportunities within existing proposals for multicasting. Beyond these scenarios, hardware multicast support will enable new research directions that would previously have suffered from poor interconnect performance despite utilizing state-of-the-art on-chip network designs.

In general-purpose chip multiprocessors, the most logical source of multicast traffic will be messages generated by the coherence protocol. A wide variety of implemented and proposed coherence protocols will benefit from hardware multicast support.

- Broadcast-based coherence protocols
    - ∗ TokenB Coherence: the TokenB protocol requires broadcasting of tokens that maintain ordering amongst requests [92].
    - ∗ Intel's next-generation QPI protocol: supports unordered broadcasting between subsets of nodes [65].
    - ∗ AMD Opteron Protocol: order is maintained by communicating requests to an ordering point (memory controller) and then broadcasting from the ordering point [30].
    - ∗ UnCorq [127]: unordered snoop delivery through a broadcast.
- Multicast-based coherence protocols such as Multicast Snooping [15] and Destination Set Prediction [91].
- Virtual Hierarchies [97]: specifically VH-Dir-Bcast: a local directory backed by a global broadcast protocol.

- Invalidation requests in Directory Protocols [80]
  - ∗ Invalidation requests for widely shared data represent the primary need for multicasts.
- Operand delivery in scalar operand networks [114, 129, 132]
  - ∗ Operands that are consumed by multiple instructions could be multicast through the network.
- Non-Uniform Cache Architectures: Dynamic NUCA [68] utilizes a multicast to quickly locate a block within a cache set.
- New Innovations
  - ∗ Virtual Tree Coherence: a hierarchical protocol employing a first level multicast with a second level global protocol (Chapter 6).
  - ∗ New coherence protocols, prefetching, and global cache replacement policies are research areas that could benefit from the presence of on-chip hardware multicast support.

In Figure 4.1, we demonstrate that even a multicast rate of just 1% is enough to cause significant throughput degradation. Table 4.1 shows that in the subset of above scenarios under evaluation, all exceed a multicast rate of 1% and will benefit from hardware support for multicasting. More detailed descriptions of the scenarios under evaluation can be found in Section 4.10.

Without efficient multicasting on chip, these types of otherwise promising coherence protocols become much less attractive. Considering the large number of cores that will be available on chip moving forward, interconnect support for low latency one-to-many communication will be critical. The tight coupling of on-chip resources mandates that interconnection network be designed with communication behavior in mind and

Table 4.1: Percentage of network requests that can be multicast

Scenario	Percentage Multicast
Directory Protocols	5.1
TokenB Coherence	5.5
Virtual Tree Coherence	8.5
Operand Networks (TRIPS)	12.4
Opteron Protocol	3.1

that coherence protocols be designed with interconnect capabilities in mind. Providing hardware multicasting support will facilitate significant innovations in on-chip coherence.

4.3 Baseline Inefficiencies

State-of-the-art packet-switched routers can (and do) utilize multiple unicast messages to achieve multicast functionality. Decomposing a multicast into several unicast packets can consume additional cycles and cause a bottleneck at the injection port as multiple messages try to access the network in the same cycle. In the baseline router, this injection bottleneck can add several cycles to the average network latency.

Many current router optimizations, such as speculative virtual channel allocation [102, 109], are effective only at low loads. Multiple unicasts drive up the network load, even if only briefly, and can easily render these optimizations ineffective. The presence of several redundant messages waiting in virtual channels for the same output port prevent bypassing and can result in failed speculation.

An example of the problems of using multiple unicasts to realize a multicast is shown in Figure 4.2. Figure 4.2 shows a multicast originating from node X intended for nodes A,B,C and D. The network interface controller creates four identical copies of the message (1A-1D). With deterministic dimension-ordered routing, all four messages want to traverse the same link in the same cycle. Messages 1B and 1D successfully arbitrate for the output in consecutive cycles. However, the messages now intended for nodes A and C are blocked waiting for B and D to gain access to a busy channel. Messages intended for B and D will again compete for the same output port once the channel becomes free.

There are several problems with this scenario. The first issue is stalled messages 1A and 1C; this problem could be addressed with more virtual channels and buffers at each router. The second problem is competition for the same bandwidth; this can be alleviated with wider links. Both of these solutions are costly in terms of area and power and exhibit poor scalability. The bandwidth bottleneck along the links could also be alleviated through the use of an adaptive routing algorithm; this would improve throughput. For example, A and C could be routed to the North and South respectively in the first hop. However, these solutions do not address the fundamental inefficiency of the simultaneous presence of multiple unnecessary messages in the network. Speculation problems also result from the presence of many concurrent packets in the network. In a network with moderate load or approaching saturation, these additional messages, even when spread out over disjoint paths, can drive up average network latency significantly. Multiple redundant messages also consume unnecessary dynamic power. The use of multiple unicasts also causes additional congestion in the network interface controller (NIC); the competition for the injection port into the network can add significant delay to packets.

4.4 Virtual Circuit Tree Multicasting Overview

To combat the problems highlighted in the previous section, we propose Virtual Circuit Tree Multicasting. Virtual Circuit Tree Multicasting builds on existing router hardware in state-of-the-art networks, and augments it with a lookup table that performs multicast route calculations. To simplify route construction, multicast trees are built incrementally, by observing the initial set of multiple unicasts and storing the rout-

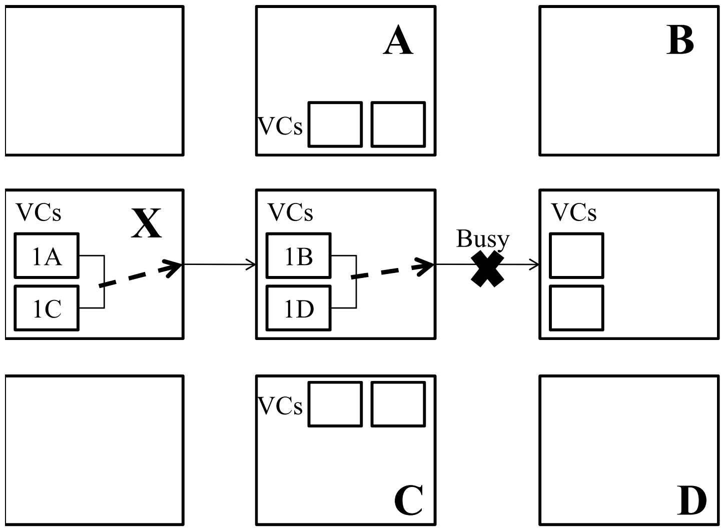


Figure 4.2: Multiple Unicast approach to multicasting

ing information in the lookup table. This approach avoids the overhead of encoding the multicast destination sets in the initial set up message, and enables an efficient tree-ID based approach for addressing multicast messages once the tree has been set up. As an additional benefit, conventional unicast traffic is unaffected by these straightforward additions to the network router.

With VCTM, each multicast first forms a virtual circuit connecting the source with the destination set, identified by a Virtual Circuit Tree (VCT) number unique to each source and destination set combination currently active in the network. In a tree-based approach, a multicast continues along a common path and branches (replicates) the message when necessary to achieve a minimal, dimension-order (e.g. one turn) route to each destination. A tree is defined by both its source and its destination set.

Once a multicast tree has been set up, packets will be routed based on this VCT number at each router. Multiple VCTs are time-multiplexed on physical links, as in traditional virtual circuit switching [34]. However, unlike virtual circuit switching where intermediate routers do flow control based on virtual circuits, and thus need to

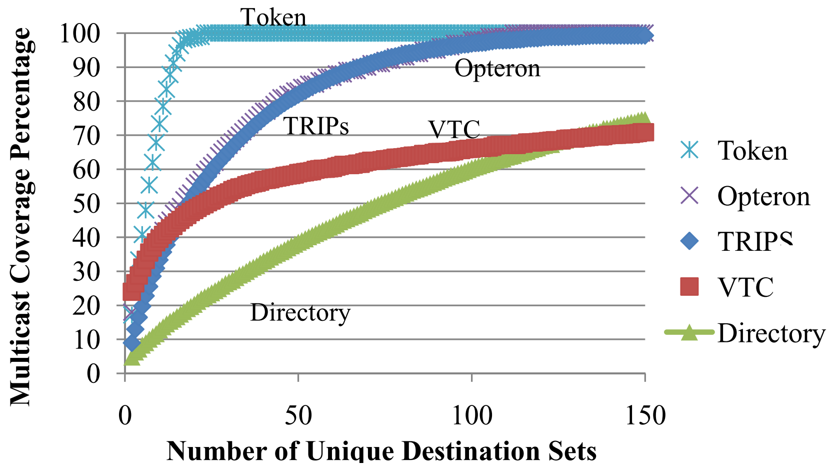


Figure 4.3: Cumulative distribution of unique multicast destination sets

support all virtual circuits, we use VCTs *only* for routing. Virtual channels are still used for flow control at the intermediate routers, with virtual channels dynamically allocated to each virtual circuit at each hop.

Routing information for each VCT is stored in a virtual circuit table at each intermediate router. The virtual circuit table is statically partitioned among source nodes; the virtual circuit tree numbers are local to each source. The virtual circuit table is partitioned into n smaller tables each needing a Read/Write port for access from the source assigned to that partition. A table with 1024 VCT entries would allocate 64 entries to each source in at 16-node system. In Section 4.10, we demonstrate that significant performance improvements can be achieved with a much smaller number of virtual trees. Multicast trees can only be evicted at the source; this prevents any multicast packets from missing in a downstream VCT table. We explore the impact of dynamic versus static partitioning, as well as various replacement strategies in Section 4.10.

The goal of VCTM (independent of its use in Virtual Tree Coherence) is a flexible, plug and play solution to multicast communication. As such, a variety of possible communication characteristics need to be supported. Figure 4.3 shows the cumulative

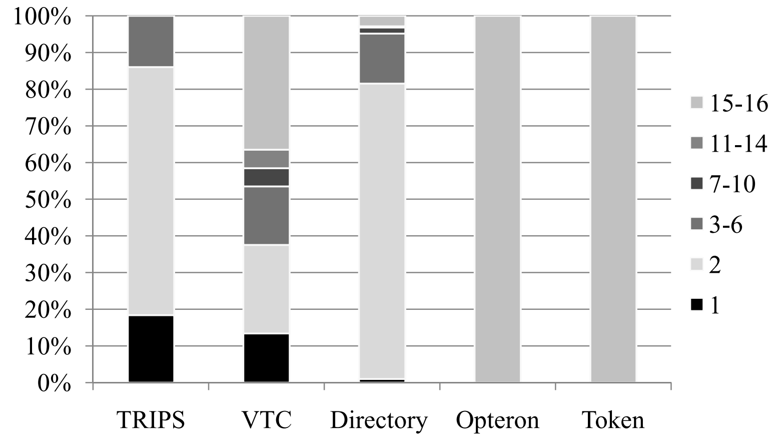


Figure 4.4: Percentage of multicasts in each multicast destination set size

distribution of unique multicast destination sets that are used throughout the execution for each of five scenarios. We see that some scenarios (TokenB, Opteron) require very few destination sets for all multicast communication. Virtual Tree Coherence and the SGI Origin Directory protocol utilize a much wider variety of destination sets. From Figure 4.3, we see there is significant reuse of destination sets. In a 16-core system, the number of possible unique trees is 65,399 [1] ; even with VTC which uses a large number of trees, 65% of all multicasts use less than 1% of possible tree combinations.

Restricting the number of concurrently active VCTs requires that there be reuse of the destination sets to see benefit. The directory and virtual tree coherence protocols have less reuse than the other scenarios; however, this figure obscures any temporal component of reuse. Even if a large number of trees are touched (needed) across the entire execution, these scenarios see benefit from the temporal reuse of some multicast trees.

Figure 4.4 shows the breakdown of the number of nodes in each multicast destination set. Multicasts in TokenB Coherence and the Opteron protocol are broadcasts,

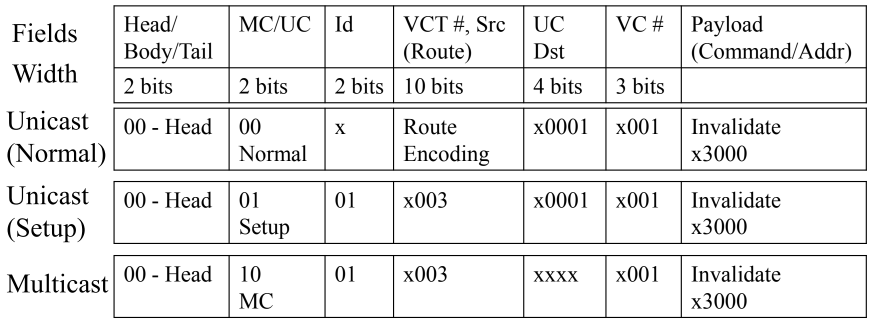
[1] The number of possible trees is $n - 2^{(}n - 1) - \binom{(n-1)}{0} - \binom{(n-1)}{1}$ with n nodes in the system. Combinations are removed from the trees that connect a root to 0 or 1 nodes; these are not trees with two or more leaves.

hence they include all possible destinations. At the other extreme, the majority of multicasts in TRIPS are to only two nodes; research has shown on average less than 15% of dynamic instruction results have three or more future uses [17]. More future uses would result in larger destination sets for TRIPS but this is not the case. Invalidations from the directory in the SGI Origin protocol go to only two nodes on average as well.

The bottom line is that architectures and protocols that require multicast support have a variety of characteristics. For example, some protocols perform broadcasts to all nodes while other have relatively small destination sets. For VCTM to be a robust multicast design, it must be able to perform well under this wide variety of conditions.

4.4.1 Router Micro-architecture

VCTM supports three different types of packets: *normal-unicast*, *unicast+setup* and *multicast*. Normal-unicast packets are equivalent to those found in a traditional packet-switched router, unicast+setup packets are used to set up a multicast tree (while delivering a useful payload), and multicast packets are sent when the desired multicast tree is already present in the network. Figure 4.5 shows the different fields encoded in each type of packet. The first field encodes if the flit is a head, body or tail flit; the second field encodes the type of flit. Each active tree has an ID number associated with it; this ID number is used to determine if a destination is being added to an existing tree or if a new tree is being constructed. When the ID bits in the packet (third field) do not match the ID bits in the router, the entry is cleared and a new tree is constructed. The fourth field encodes the virtual tree entry number and the source node number. If the packet is a normal unicast, the lookahead routing information is encoded in the 4th field instead of the virtual circuit tree number. For normal-unicast and unicast+setup packets, the fifth field encodes the destination. The destination is unnecessary for the multicast packets since a table lookup is used to route to each destination. The sixth field stores the virtual channel number assigned to the packet; the final field contains



Fields	Head/ Body/Tail	MC/UC	Id	VCT #, Src (Route)	UC Dst	VC #	Payload (Command/Addr)
Width	2 bits	2 bits	2 bits	10 bits	4 bits	3 bits	
Unicast (Normal)	00 - Head	00 Normal	x	Route Encoding	x0001	x001	Invalidate x3000
Unicast (Setup)	00 - Head	01 Setup	01	x003	x0001	x001	Invalidate x3000
Multicast	00 - Head	10 MC	01	x003	xxxx	x001	Invalidate x3000

Figure 4.5: Header packet encoding format, assuming 1024 virtual circuits, 16 network nodes and 8 virtual channels.

the message payload.

The router micro-architecture is shown in Figure 4.6. Comparing Figure 2.2 (on page 21) and Figure 4.6, it is clear that only modest changes are required to realize VCTM. The largest change comes with the insertion of the Virtual Circuit Tree Table used to store the local output ports needed to realize each tree. The width of each Virtual Circuit Tree Table entry is proportional to the router radix. One bit is needed for each output port; higher radix routers will need wider entries to accommodate the additional output ports. Each Virtual Circuit Tree Table stores one entry for each currently active tree in the network. These entries are statically partitioned among all the nodes in the network. Normal-unicast packets traverse the highly optimized pipeline shown in Figure 2.4b on page 24; they do not need to access the Virtual Circuit Tree Table and are routed via existing hardware in the router pipeline (e.g. dimension-order routing). Only multicast packets are routed via the Virtual Circuit Tree tables.

Routing a multicast packet can result in significant complexity (translating into significant hardware cost). This complexity comes from additional hardware to choose multiple output ports per packet, hardware to modify the destinations in each packet header as multicasts branch and deadlock avoidance; further discussion of these complexities can be found in Section 4.5. We avoid this complexity through the use of

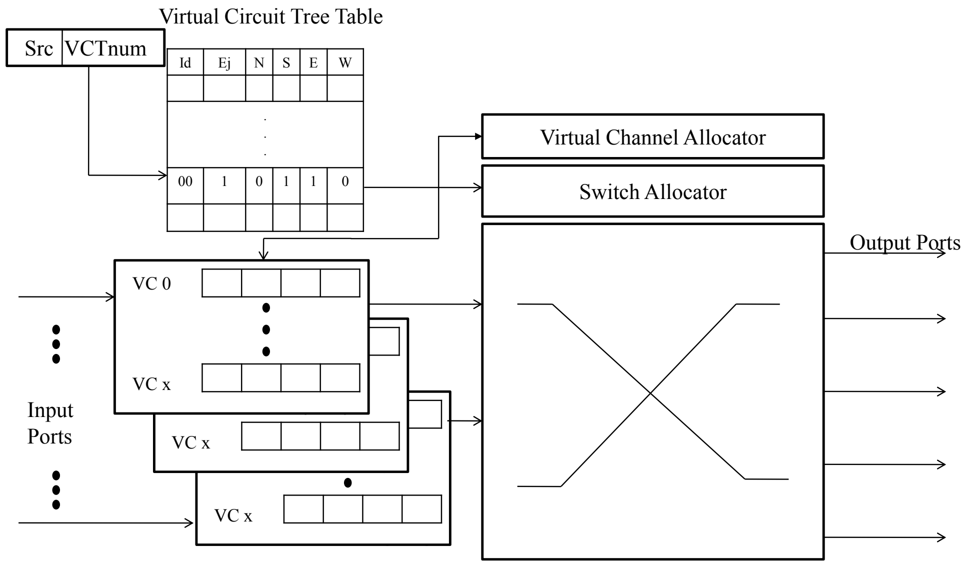


Figure 4.6: Multicast Router Micro-architecture

the unicast+setup packets to incrementally construct the multicast tree routes. Unicast+setup packets deliver a packet payload to a single destination node (in the same fashion as a normal-unicast packet); however, while delivering this packet, they also add their destination to a multicast tree. They add their destination to the multicast tree by recording a 1 in proper output port column in the VCT entry at each router.

Both the baseline router and VCTM use dynamic buffer management [130] to share buffers among the input virtual channels of one port; this reduces the amount of buffering needed to support the potentially longer occupancy of a packet in a buffer (due to a multicast tree branch). VCTM potentially lengthens the buffer occupancy at nodes where the tree branches; however, VCTM injects fewer messages into the network which reduces buffer pressure. A small 4-input adder is also added to calculate the number of output ports that a multicast is destined for; a multiplexer is added to select which entry to feed in to the adder. A flit must remain in the input buffer until the switch allocator has granted the number of requests equal to the port count computed from the recorded output port entries for that flit.

4.4.2 Network Interface Controller

The VCT routing tables are added to each router to support the lookup of multicast tree routes by in-flight packets. VCTM also requires that a structure be added to the network interface controller (NIC). A content-addressable memory (CAM) is added to the NIC to quickly search for active multicast trees. This CAM stores the destination sets corresponding to active trees. Each destination set CAM is private to the source node and contains only trees that the source is the root node for. When a new multicast arrives at the NIC, the CAM is queried; if there is a CAM hit, the requested multicast tree is active in the network. A CAM miss indicates that there is no active tree for the requested multicast and one must be constructed.

4.4.3 Router Pipeline

Figure 4.7 depicts the changes to the router pipeline originally shown in Figure 2.4a on page 24. Normal unicast packets use the original pipeline. Unicast+setup packets follow the original pipeline, with an added step of updating the Virtual Circuit Tree Table after the routing computation has been performed. This update is off the critical path for the packet. For multicast packets, the routing computation stage is replaced with the VCT table lookup. Additionally, VA/SA, ST and LT can occur multiple times if the multicast packet is branching at this node (e.g. multiple output ports need to be traversed); if this is not a multicast branching node, then each of these stages executes exactly once.

Multicast packets do not use the lookahead optimization included in the baseline router pipeline. If lookahead routing were used, at a branch in the multicast tree, a set of output port combinations would need to be recorded in the VCT Table for each fork. This would increase the size of the VCT table; additionally, the header information of the branching multicast packet would need to be updated individually with these output

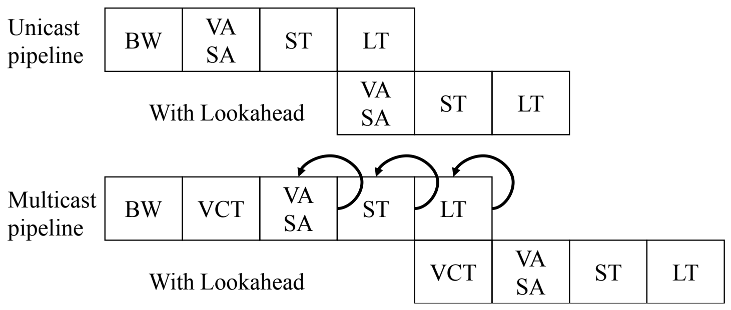


Figure 4.7: Multicast Router Pipeline

port combinations.

Speculative virtual channel allocation is still performed for multicast packets. As in the baseline, we optimistically assume that we will get the VCs needed for each output port and speculatively perform switch allocation. However, each input may now need to generate multiple output VC requests. We generate one request each cycle until all multicast outputs are satisfied. A 4-input adder is used to determine the number of output ports that must be traversed according to the ones stored in the VCT entry. Flow control is managed using virtual channels, which are dynamically allocated to virtual circuits. Therefore our design does not require a large number of VCs (which would be the case if VCs were coupled to virtual trees with each active VCT assigned its own unique VC). Switch allocation occurs in a similar fashion. Since each input VC may generate multiple switch requests, the router generates one per cycle and queues them up. At a tree branch, each output port is considered independently; we do not need to successfully allocate all needed output ports in the same cycle for the packet to proceed. Each output port can be granted in different (and non-consecutive) cycles.

Our baseline router assumes credit-based flow control; credit turnaround is lengthened by the replicating of flits at branch nodes; once the final flit at the branch node has traversed the switch, a credit can be sent to the upstream router.

4.4.4 Destination Encoding

The destination set for each multicast is encoded using a bit vector. This bit vector is stored in the destination set CAM for fast searching when a new multicast request arrives at the network interface controller. The size requirements for the bit vector will grow as the number of destinations in the network grows; however the bit vector only needs to be stored in the destination set CAM (not the network routers). Each CAM entry is N bits wide where N is the number of network nodes. Therefore the cost of this scaling is small.

We then encode the destination set into the packet by using a virtual circuit tree identifier. The virtual circuit tree identifier can be encoded with $log_2\ (NumberofVirtual CircuitTrees/N)$ bits. This encoding is a much more scalable solution than the destination encoding used in prior work (discussed in Section 4.11) while still allowing the flexibility to access all destinations and have a variety of multicast trees active concurrently.

4.4.5 Example

Now that we have described the functionality of the VCTM router architecture, we present a walk-through example in Figure 4.8 for the process of constructing a new multicast tree. When the network interface controller at Node 0 initiates a multicast message, it accesses the Destination Set CAM, which contains all of the node's currently active multicast trees (Step 1). In this example, no matching entry is found, so the node will invalidate its oldest tree (e.g. VCT 1) and begin establishing a new tree (Step 2).

Step 3 decomposes the multicast into one unicast+setup packet per destination. The packet type field is set to be unicast+setup and the ID field of former VCT 1 was 0, so we increment the ID to be 1. Matching ID bits indicate that a new node is being added to an existing tree; while differing ID fields indicate an old tree is being replaced with a new one. Each unicast+setup packet is given the same VCT number but a different destination. Additionally, all three packets contain the same payload (e.g. memory address plus command).

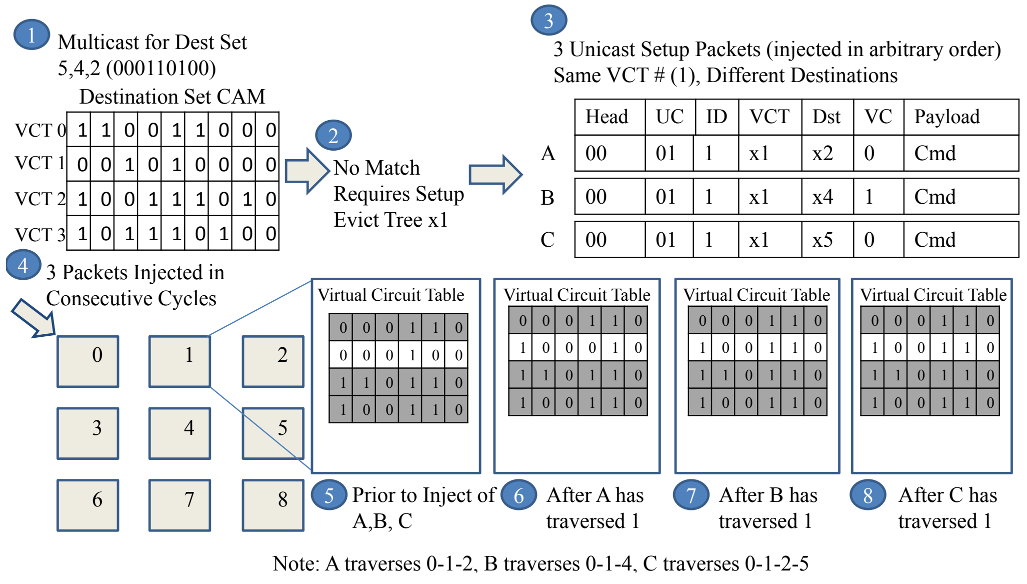
In Step 4, each packet is injected into the network in consecutive cycles. Unicast packets are dimension order (X-Y) routed with respect to the source so the resulting multicast tree will also be dimension order routed.

The updates to the virtual circuit tree table at Node 1 are shown in Steps 5-8. Step 5 shows the entries prior to the creation of new VCT 1 (highlighted row 2 corresponds to VCT 1). Packet A is routed first. Upon arrival at Node 1, A determines that the VCT entry is stale (due to differing ID fields). Packet A will clear the previous bits in the row and update the row with a 1 corresponding to its output port based on the routing computation performed by the unicast routing hardware; in this case, the East port. Packet A also stores the new ID of "1" in the first column of the VCT entry.

At Step 7, the unicast destined for Node 4 (Packet B) will update the VCT entry. The ID fields are now the same, so it will not clear the information written by Packet A. A one will be stored in the South column. Finally, Packet C traverses the link to Node 1; Packet C will also use the East output port. Since Packet A already indicated that this multicast will use the East port, Packet C does not need to make any updates to the VCT entry at Node 1.

Similar updates will be performed at each Virtual Circuit Table along the route. Updates to the Virtual Circuit Table do not impact the critical path through the router for normal unicast packets as they do not perform any computation needed by the unicast packet to reach its destination.

When a subsequent multicast destined for 2, 4 and 5 arrives at Node 0, it will find its destination set in the CAM. In this case, the network interface controller will form a multicast packet for VCT 1. The virtual circuit tree number will index into the portion of the Virtual Circuit Table at each router assigned to source node 0. The table



VCT 0	1	1	0	0	1	1	0	0	0
VCT 1	0	0	1	0	1	0	0	0	0
VCT 2	1	0	0	1	1	1	0	1	0
VCT 3	1	0	1	1	1	0	1	0	0

	Head	UC	ID	VCT	Dst	VC	Payload
A	00	01	1	x1	x2	0	Cmd
B	00	01	1	x1	x4	1	Cmd
C	00	01	1	x1	x5	0	Cmd

Virtual Circuit Table (5)

0	0	0	1	1	0
0	0	0	1	0	0
1	1	0	1	1	0
1	0	0	1	1	0

Virtual Circuit Table (6)

0	0	0	1	1	0
1	0	0	0	1	0
1	1	0	1	1	0
1	0	0	1	1	0

Virtual Circuit Table (7)

0	0	0	1	1	0
1	0	0	1	1	0
1	1	0	1	1	0
1	0	0	1	1	0

Virtual Circuit Table (8)

0	0	0	1	1	0
1	0	0	1	1	0
1	1	0	1	1	0
1	0	0	1	1	0

Figure 4.8: Multicast setup and routing example

will output the correct output ports that this packet needs to be routed to. All three destinations share a common route to Node 1 where the first fork occurs. After the first fork, one packet is routed to Node 4 and one packet is routed to the East toward nodes 2 and 5. At Node 2, the packet forks again, with one packet being delivered to the ejection port and another packet continuing on to Node 5.

4.5 Discussion

VCTM requires multiple unicast+setup messages (one per destination) to construct a multicast tree. Alternatively, a single setup phase could be employed; however, such an approach is subject to additional complexities. Multi-header multicasts have been proposed [28, 88]; in this type of approach, each destination is allocated its own header flit. The header flit is routed to the appropriate output port and the data is then sent to as many output ports within the switch as necessary. Work by Malumbres et al. [88] requires that sufficient storage at the router is available to buffer the entire packet payload; this amount of buffering would be impractical in on-chip networks.

The multi-header approach also wastes significant bandwidth; although in a 16 node system, only 4 bits are necessary to specify the destination, it will consume an entire flit of link bandwidth. For a broadcast to $N-1$ cores, $N-2$ additional header flits are added (our approach requires a single header flit). A coherence request can fit in a single flit including the header; now this request requires N flits; this wastes tremendous on-chip bandwidth.

VCTM encodes the destination set by using the virtual circuit tree entry number; this is a simple approach to encoding. Alternatively, a bit-string encoding can be used; this approach requires N bits, where N is the number of nodes in the system. Routing hardware can be replicated to handle computing routing information to multiple destinations or can be iterated over multiple times; multiple iterations will increase the number of clock cycles spent routing. New headers will be formed using this routing information with a separate header for each output port now containing a subset of destinations. No branch of the multicast can proceed to the next router until all routing calculations are computing, which will increase the latency experienced by these packets. In contrast, VCTM requires a single cycle to lookup the routing information for all output ports in the VCT table.

An alternative to performing multiple routing calculations is to store reachability information. Storing reachability information with a vector for each output port has been proposed to determine which output ports to route these bit-string encoded destinations to [107]. This work has been proposed for multistage interconnection networks with a path diversity of one where there is a single path through the network from source to destination; intermediate nodes cannot reach all destinations. A reachability vector encodes a bit-string of destinations that can be routed to from a given output port. In a direct network, since every node can route to every other node, we need to remove destinations from the header as they are routed to; this will prevent messages from cycling through the network infinitely. To remove destinations, we will need to send a modified header to each output port at a tree branch; this new header information can be achieved by ANDing the destination bit-vector with the reachability bit vector for each output. We believe that constructing new header flits will insert an extra pipeline stage into each router pipeline as well as additional buffering to create these new header flits. Work on applying reachability vectors to off-chip networks with path diversity inserts uses an additional cycle to compute the new header information [121]. We are unaware of the application of these reachability vectors to on-chip networks. The VCTM routing scheme is simple to realize; these other possible schemes described may result in additional complexity; we leave a detailed study of the hardware and performance of such schemes to future work.

4.6 Optimizations

Several optimizations to the original VCTM proposal are possible. In this section, we discuss the following possible optimizations: replacement policy, optimal tree selection, tree collapsing and dynamic table partitioning. These optimizations target improving tree utilization, reducing the power consumption and storage overhead required by VCTM.

4.6.1 Tree Replacement Policy

Several replacement policies can be employed to select a tree entry when setting up a new tree. We explore two replacement policies: local round robin and local least recently used (LRU). LRU replacement can be used to prevent frequently used trees from being evicted while they are still useful, but LRU replacement requires additional bits to track the LRU information at each source node. Both of these policies assume a static partitioning of VCT entries per node. A global replacement scheme would need to be considered if dynamic partitioning of the VCT tables was employed.

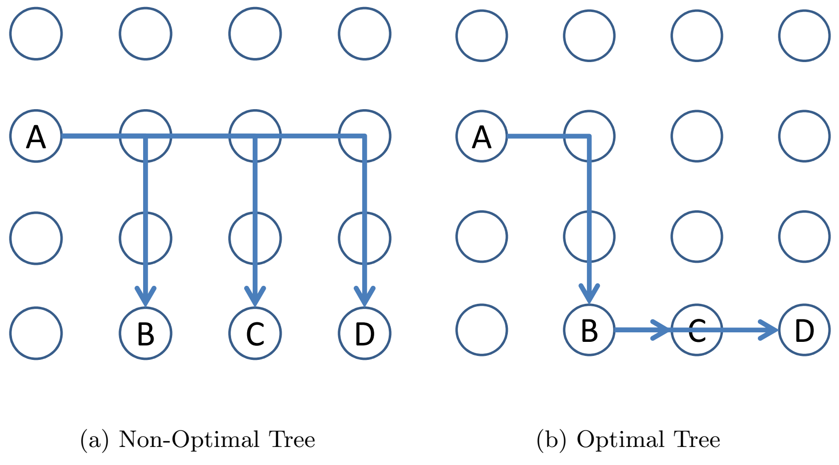


Figure 4.9: Difference between Non-Optimal and Optimal Tree Selection

4.6.2 Optimal Tree Selection

VCTM produces minimally routed trees: all destinations are routed to in the fewest number of hops. However, it is possible to construct trees that are minimally routed and utilize fewer physical resources, that is construct minimal spanning trees that have the lowest network cost. Figure 4.9 depicts an example of the non-optimal routes that can currently result from relying on dimension order routing versus an optimal route that can be realized in an idealized way. In Figure 4.9a, the tree from A to B,C and D is realized with 9 link traversals. However, in Figure 4.9b, the tree is realized in 5 link traversals.

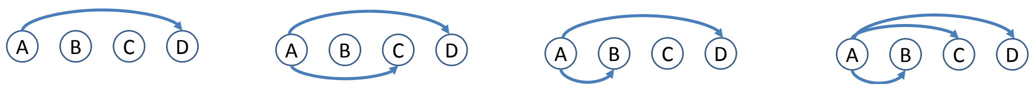
There are several complexities associated with searching for these optimal tree routings. The first is the cost of the search itself; determining the optimal route would be very expensive. The hardware cost associated with constructing a minimum spanning tree would be prohibitive. Relying on software would likewise represent too much overhead given the fine granularity of reconfiguration in our network. Prim's algorithm [110] for example can find a minimum spanning tree in $O(E + Vlog(V))$ time (where E is the number of edges and V is the number of vertices. For trees that receive substantial reuse, exist for a long duration, and could be realized with substantially lower cost, the overhead of relying on software to produce a minimum spanning tree may be warranted.

The second is the potential for deadlock; routing restrictions may be violated to achieve an optimal route. Using minimum spanning trees has the potential for resource cycles; a single minimum spanning tree will be acyclic, however, no guarantees can be made across multiple trees. Currently, we rely on the deadlock-freedom property of dimension order routing; if we remove routing restrictions to find optimal tree layouts, deadlock may be introduced. Alternatively, we could rely on virtual channel escape paths. Using virtual channel escape paths will break a deadlock cycle but will also break the ordering restrictions imposed on our network for use by Virtual Tree Coherence (discussed in Section 4.9 and Chapter 6). We find the performance and throughput benefits of optimal tree layouts to be very small; therefore, we do not feel that the complexity is warranted. Results showing the possible link usage savings will be presented in Section 4.10.5.2.

4.6.3 Tree Collapsing

With static partitioning of virtual trees to source nodes, the size of the Virtual Circuit Tree tables grows with the size of the network, assuming the demand each source node places on the network remains the same. The number of possible unique trees is approximately 2^n for n network nodes; this is a prohibitive number of trees as the system grows (18 quintillion trees for 64 nodes). However, we believe reuse behavior will continue to be exhibited by larger systems.

We propose a solution to improve scalability beyond relying on the reuse factor. To combat this potential scalability problem, we explore collapsing similar trees into one tree. Certain virtual trees subsume other virtual trees; a multicast request can utilize an existing tree that subsumes it rather than setting up a new tree (and evicting an old tree). The result of tree collapsing is that fewer unique trees are needed to



(a) Tree 1 (b) Tree 2 (c) Tree 3 (d) Tree 4

Figure 4.10: Overlapping and collapsing similar trees

achieve significant performance benefits which results in smaller Virtual Circuit Tree tables at each router. An example of similar trees that can be collapsed is shown in Figure 4.10. All four of these destination sets require the same link traversals. Choosing Tree 4 to route to the destination sets in Trees 1-3 would result in the same number of link traversals without requiring the setup of a new tree. The downside is extra switch allocations for the extra (unnecessary) output ports. Routing a multicast to extraneous destinations must not confuse the application. In a coherence protocol, extra destinations will generate extra snoops but not effect the correctness. Extra snoops will consume additional power; this extra power needs to be factored into the decision to use precise or imprecise multicast trees. In scalar operand networks, handling the presence of extra operands may be non-trivial.

To realize this functionality, we replace the Destination Set CAM with a Ternary CAM (TCAM). A TCAM differs from a CAM in that three values, 0, 1, and “don’t care” can be specified in the search register and/or in the stored data [5]. Within the destination set TCAM, all destinations must be precisely specified (e.g. a 1 or a 0); however we can add “don’t care” values into the search register. Now, a search of the destination set TCAM will return a match that includes all of the requested destinations but may include extra destinations.

A tree that results in a broadcast (e.g. every node is a leaf node) subsumes all other trees. So a multicast with two destinations specified and fourteen destinations specified as “don’t cares” could result in a broadcast. This result is less than ideal as it adds significant traffic to the network, consuming bandwidth and power. As a result, we limit the number of "don't care" bits in the query register to a threshold based on the extra link traversals required over a tree that is an exact match.

To do this a table is added to the NIC to determine which "don't care" bits result in zero or one extra link traversals for a given destination. From the example in Figure 4.10, B and C would be potential "don't care" candidates when constructing a multicast to reach destination D. This "don't care" lookup table can be used by the NIC to construct the search destination set. Results for incorporating this functionality are presented in Section 4.10.4.1.

4.6.4 Dynamic Table Partitioning

Static partitioning of the virtual circuit tree tables provides a simple solution; however, this may lead to large tables and wasted table entries. Dynamically partitioning the virtual circuit tree tables among all cores will likely provide higher utilization and reduce the overall size. For example, with server consolidation workloads, communication across virtual machines will be very rare. As such, routers within a virtual machines will not require storage for trees dedicated to other virtual machines (this assumes threads are scheduled with spatial locality). Without dynamic table partitioning, these entries would be idle.

The drawback to a dynamic scheme is increased complexity. Currently, each core manages its own pool of virtual trees. Dynamic table partitioning requires some global coordination among cores; a core must not issue multicast packets to a tree that has been reallocated to another source node. Doing so would result in dropped or misrouted packets. Dynamic partitioning also increases the size of the destination set CAMs since now each source node can utilize the full VCT table and must have a CAM entry per VCT entry. The potential for dynamic partitioning is explored in Section 4.10.4.3.

Table 4.2: VCT Table Overhead

Number of entries	Area (mm^2)	Energy (nJ)	Time (ns)
512	0.024	0.0018	0.43
1024	0.041	0.0023	0.44
2048	0.078	0.0030	0.46
4096	0.101	0.0037	0.51

4.7 Power and Area Analysis

Area and power overheads are associated with the addition of virtual circuit tables and destination set CAMs. While these additional structures in our VCTM router consume additional power, this is offset by the power savings due to reduced switching activity shown in Section 4.10.5.1.

We use Cacti [131] to calculate the area and power overhead associated with adding a virtual circuit tree table to each router shown in Table 4.2. Several different VCT table sizes are calculated for a 70nm technology; energy is reported as dynamic energy per read access. With both static and dynamic table partitioning, we assume that idle entries can be gated off to reduce power consumption. Each entry is 7 bits wide; these results are estimates as Cacti cannot produce the exact geometry of our tables. Assuming a 1 ns clock period, each table can be accessed in less than half a cycle.

In Section 4.10, we demonstrate that a small number of entries (512) is sufficient to achieve significant benefit. We can further reduce the table size by using a TCAM for the destination set CAM and by dynamically partitioning the VCT tables among the source nodes.

We also added a Destination Set CAM to each network interface controller. This *small* CAM is searched for the VCT number matching the given destination set and can be overlapped with the message construction so as to not add additional latency to the

Table 4.3: Destination CAM Overhead

Number of entries	Area (mm^2)	Energy (nJ)	Time (ns)	Total Bytes 16 nodes (25)
32	0.018	0.007	0.87	64 (96)
64	0.021	0.010	0.90	128 (192)
128	0.029	0.017	1.09	256 (384)
256	0.077	0.040	1.53	512 (768)

critical path. The NIC speculates that an active tree will be found and inserts the VCT number returned by the CAM search. In the event of a misspeculation, it is reasonable to assume that decomposing a request into multiple unicast+setup packets will take a couple of cycles, one per destination.

Each CAM size in Table 4.3 corresponds to the number of VCT entries in Table 4.2 partitioned evenly among 16 nodes. Both 32 and 64 entry CAMs can be accessed in under a cycle and will give each source a reasonable number of concurrently active multicast trees. We do not foresee each core needing significantly more trees as the system scales.

Replacing the Destination Set CAM with a TCAM will double the storage requirements. A TCAM has three possible states, 0, 1 and “don’t care” requiring 2 bits per destination. Doubling the size of the destination set CAM pays off with a significant reduction in the number of active trees, which reduces the entries needed in the VCT tables. The TCAM design also adds a 32 byte table (for a 16 core system) to each NIC to determine the “don’t care” bits for the search register. The potential reduction in active trees is examined in Section 4.10.4.1.

4.8 Extension to other topologies and routing algorithms

VCTM is built on top of the existing routing algorithm of the baseline interconnection network. As such, VCTM extends seamlessly to other topologies. A topology that provides greater path diversity may result in less benefit; there may be less redundancy to eliminate in tree paths. With more scalable topologies (e.g. tori, express cubes), the impact on network throughput of the multiple unicast approach to multicasting will likely be less severe. However, VCTM will still provide benefit through reducing switching activity and improving network power consumption.

VCTM is currently implemented with dimension-order X-Y routing. There is no fundamental dependence between these two. Overlaying VCTM on alternative routing algorithms would also result in benefit. The tree setup can be adaptively routed, but subsequent uses of that tree will follow the identical path. So, while an optimal path to avoid congestion may be selected during the setup phase, this may no longer be the least congested path on successive uses. Some additional complexity is occurred when using an adaptive routing algorithm. If a multicast forks at an early node in the tree and joins at a later node in the tree due to adaptive routing, duplicate messages will be spawned at the join node; this problem is illustrated in Figure 4.11. In this example, we assume a west first turn model routing algorithm [50]; a packet must be first routed in the west-bound direction but can freely choose between north, south and east afterwards. If during tree setup, a packet destined for B is routed east then south and a packet destined for C is routed south then east, the tree will fork at the source node (A) and join at the shaded node. Later when a multicast is routed along this tree, two messages for B and two messages for C will be created at the shaded node. The first set for B and C will be created when the multicast enters on the north port and the second set when the multicast enters on the west port. This problem does not induce deadlock cycles in the network but can increase the amount of traffic limiting the benefit of VCTM. We leave the solving of this problem for future work.

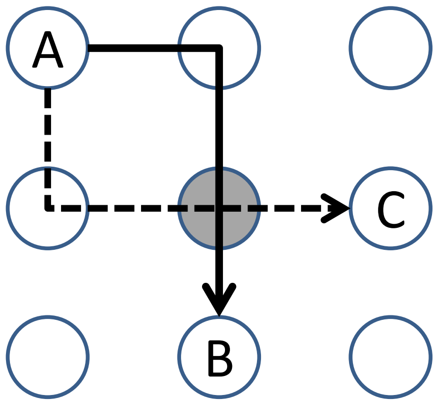


Figure 4.11: Potential for duplicate messages being spawned in tree from adaptive routing

4.9 Network Ordering

VCTM was first designed without consideration to inter-message ordering within the network. In Chapter 6, we will present the design of the Virtual Tree Coherence (VTC) protocol. The VTC protocol requires certain inter-message ordering be maintained within the network.

Specifically, VTC requires that separate multicasts utilizing the same virtual tree do not become re-ordered with respect to each other prior to reaching the leaf nodes. For the specific case of VTC, we restrict a virtual tree to utilize the same virtual channel throughout its network traversal and for subsequent tree traversals. This virtual channel is assigned at the time that the tree is setup. The network still contains multiple virtual channels so different virtual trees can take advantage of them for higher throughput. This virtual channel restriction results in some performance degradation in the network but it is fairly insignificant compared to the overall hop count reduction that preserving network order achieves for the coherence protocol.

Table 4.4: Network Parameters

Topology	4-ary 2-mesh 5-ary 2-mesh
Routing Algorithm	X-Y (Dimension Order) Routing
Channel Width (flit size)	16 Bytes
Packet size	1 flit (Coherence messages) 5 flits (Data) 3 flits (TRIPS)
Virtual Channels	4
Buffers per port	24
Router ports	5
VCTs	Varied from 16 to 4K (1 to 256 VCTs/core)

4.10 Evaluation

The methodology for evaluating VCTM differs slightly from the other evaluation methodologies used in this dissertation. One of the goals of this portion of the evaluation is to demonstrate VCTM's flexibility and amenability to a variety of scenarios. To that end, we leverage various research groups' infrastructures to collect network traces to understand the network latency and bandwidth impact of using VCTM. These results do not provide the complete picture that full-system simulation achieves but do demonstrate how powerful a mechanism VCTM is. VCTM is evaluated in a full-system simulation environment in conjunction with VTC later in Chapter 6.

To study the impact of VCTM on a variety of architectures and coherence protocols, we leverage traffic traces collected from several simulation environments. Traces for the directory and VTC protocols were generated with PHARMsim [18]. These traces are collected from end-to-end runs of the 8 workloads detailed in Table 3.2 (found on page 55). To collect traces for TokenB Coherence and the Opteron protocol, GEMS 2.1 [89] with Garnet [6] full system simulation environment was used; each SPLASH-2 workload was run for the entire parallel phase. Finally TRIPS traces use SPEC [124] and MediaBench [98] workloads. They were collected on an instantiation of the Grid Processor Architecture containing an ALU execution array and local L1 memory tiles connected via a 5x5 network [2] . For all scenarios, we use the network parameters found in Table 4.4.

As with the HCS evaluation, we utilize synthetic traffic to further stress our router design. With a uniform random traffic generator, we can adjust the network load as well as the percentage of multicasts.

4.10.1 Performance of Various Multicast Scenarios

With VCTM, the potential for performance improvement comes from two main factors, the reduction in network load (improved throughput) and reduced contention for network injection ports. In the following sections, we will describe each scenario in more detail and then present the performance improvements.

4.10.1.1 Directory Protocol

Directory-based protocols are often chosen in scalable designs due to the point-to-point nature of communication; however, they are not immune to one-to-many style communications. Directory protocols, such as the SGI Origin Protocol [80] send out multiple invalidations from a single directory to nodes sharing a cache block. As such, invalidation requests in a directory protocol could leverage hardware multicast support. In current implementations, the network interface controller must decompose the sharing list from the directory into multiple redundant packets each destined for a different sharer. While not necessarily on the critical path, these requests can be frequent and can waste power and hurt the performance of other network requests that are on the critical path.

[2] Traces for TokenB Coherence and the Opteron protocol were collected by Niket Agarwal. TRIPS traces were provided by the TRIPS group and Noel Eisley provided assistance in using these traces.

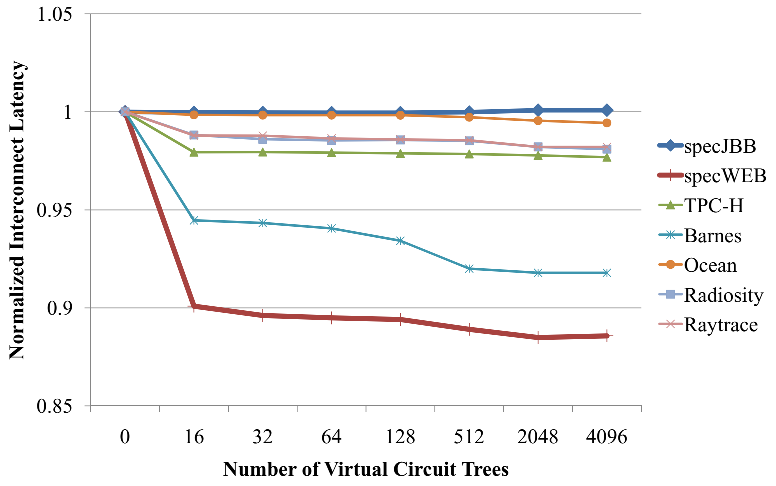


Figure 4.12: SGI Origin Directory Protocol network performance

Characterization of these invalidation messages in PHARMsim [18] with commercial and scientific workloads [124, 134, 146] shows that invalidation messages have an average network latency of up to two times the overall average network latency. The percentage of total requests that are invalidation multicasts is shown in Table 4.1 (on page 73).

For the directory protocol, we simulate 32KB L1 caches and private 1MB L2 caches. Addresses are distributed across 16 directories, with one directory cache located at each processor tile. The reduction in network latency for the directory protocol is shown in Figure 4.12. Invalidation requests represent approximately 5% of network requests for the directory protocol. However, since the network load for this protocol is low for applications like SPECjbb, TPC-H, Raytrace and Ocean, VCTM is unable to realize substantial benefits. SPECweb has a slightly higher load which translated into more benefit from VCTM (up to 12%).

4.10.1.2 TokenB Coherence

Token Coherence is designed to decouple protocol performance from protocol correctness. In broadcast-based protocols, ordering (for correctness) is often implicit

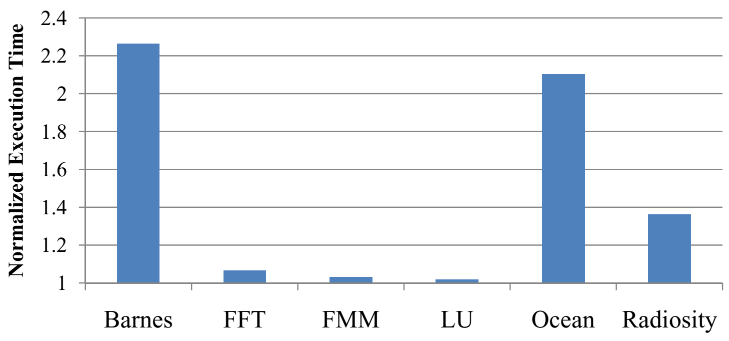


Figure 4.13: TokenB Coherence Slowdown in the Absence of Hardware Multicast Support

through the use of a totally-ordered interconnect. To improve scalability, Token Coherence removes this implicit assumption and instead uses token counts to ensure proper ordering of coherence requests. A processor must hold at least 1 token to read a cache line and must hold all tokens to write to a cache line. In the event of a race, requests must re-issue.

The TokenB protocol broadcasts for tokens; broadcasting for these tokens can be a significant bottleneck. Originally intended as a chip-to-chip protocol with off-chip multicast support, the absence of multicast functionality on-chip hurts the performance of token coherence and calls in to question the feasibility of this otherwise attractive solution.

Figure 4.13 shows the slow-down of the TokenB Coherence protocol when the assumption of hardware multicast support is removed. GEMS [89] was used to generate this data for a 16-core system[3] . This release of GEMS models only link level contention, ignoring router pipelining and internal router contention (prior to the incorporation of Garnet [6]). The router is composed of a single pipeline stage with infinite buffering. Despite the unrealistically aggressive router model, substantial slow down is already

[3] This result was provided by Niket Agarwal, a graduate student at Princeton University

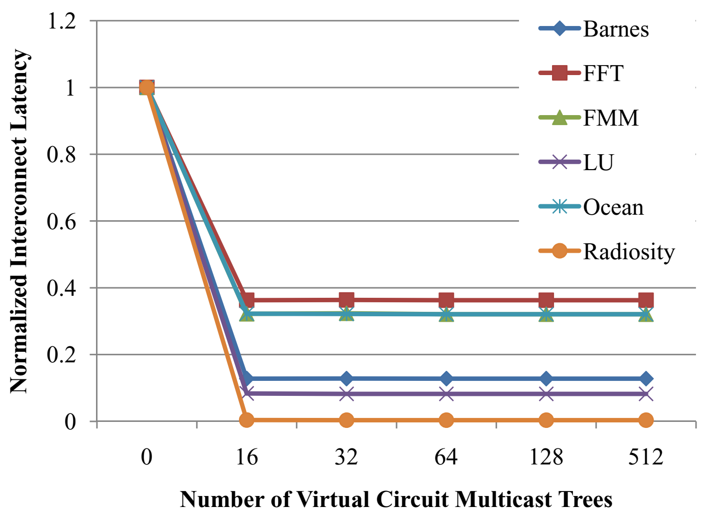


Figure 4.14: TokenB network performance with VCTM

observed due to NIC and link bandwidth bottlenecks.

TokenB coherence simulations were configured with 64KB L1 caches and 1MB private L2 caches with a MOESI TokenB protocol. Normalized interconnect latency is presented in Figure 4.14. To request tokens, the source broadcasts to all other nodes on-chip; as a result, only one virtual circuit tree is needed per node. If the source node sent out a multicast with a variable destination set, a VCT count larger than 16 would be useful. Radiosity has reached network saturation with the baseline configuration; relieving the pressure caused by multiple unicasts results in substantial latency improvement of close to 100%. Barnes and LU are also very close to saturation, leading to close to 90% saving in network latency.

We further evaluate VCTM against a path-based multicast routing. For workloads near or at saturation, a path-based multicast can also effectively relieve network pressure and reduce latency by an average of 70%; however, for workloads not nearing saturation (FFT, FMM and Ocean), the path-based multicast increases network latency by 48% over the baseline due to non-minimal path routes to reach some destinations.

4.10.1.3 AMD Opteron (HT) Protocol

AMD's Opteron protocol [30] has been designed for maintaining coherence between chips in a traditional multiprocessor system. Coherence requests are sent to a central ordering point (memory controller) and then broadcast to all nodes in the system. The feasibility and potential for moving this style of protocol on-chip is directly tied to the performance provided by the interconnect, and can be improved with multicasting. Recent research proposals have compared themselves to an on-chip Opteron style protocol [127].

Traces were generated for the Opteron protocol with 64 KB L1 caches and a 16 MB fully shared L2 cache. In Figure 4.15, the Opteron protocol sees steady improvement with the addition of Virtual Circuit Trees; once 512 VCTs are available performance levels off with a savings of 47% in network latency. Some filtering of destination occurs in this protocol resulting in the need for a larger number of trees than the TokenB protocol. In all cases, path-based multicasting performs worse than the baseline and VCTM. A high percentage of all multicasts in the Opteron protocol go to 15 destinations so a path-based multicast has to snake through the chip to each node; if a non-optimal path is chosen, latency will be high. An addition downside to path-based multicast, not reflected in the network latency results is that the requester will have to wait until the last node in the path has received and responded to the snoop. VCTM delivers all snoops in a more timely fashion resulting in faster snoop responses.

4.10.1.4 Virtual Tree Coherence

In Chapter 6 we will explain the workings of Virtual Tree Coherence in detail. Briefly, Virtual Tree Coherence maintains lists of sharers for coarse-grained memory regions and then multicasts coherence requests to those region sharers. For the trace-based evaluation, we use 2KB regions; each region covers 32 cache lines. Figure 4.16

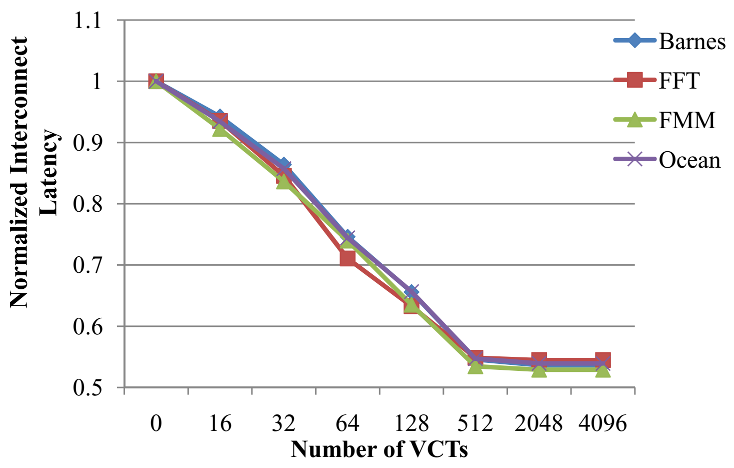


Figure 4.15: Opteron Protocol performance with VCTM

present the improvement in network latency for virtual tree coherence. This coherence protocol needs more simultaneous multicast trees than other scenarios to see substantial benefit. The destination sets used by Virtual Tree Coherence vary much more widely; however, the overhead of supporting additional trees is low; 512 to 2048 total VCT entries would be feasible design points and would reduce network latency by up to 65%. Without multicast support, this type of protocol would see prohibitive network latency for sending out snoop requests.

4.10.1.5 Operand Networks

Architectures such as TRIPS [114], RAW [132] and Wavescalar [129] use an operand network to communicate register values between producing and consuming instructions. The result of an instruction is communicated to consumer tiles which can then wake up and fire instructions that are waiting on the new data. If the result of one instruction is consumed by multiple subsequent instructions on different tiles, operand delivery could be expedited by a multicast router. Studies have shown that 35% of dynamic values generated by an application have 2 or more future uses [17]. The cost of on-chip communication can significantly impact compiler decisions in this style of

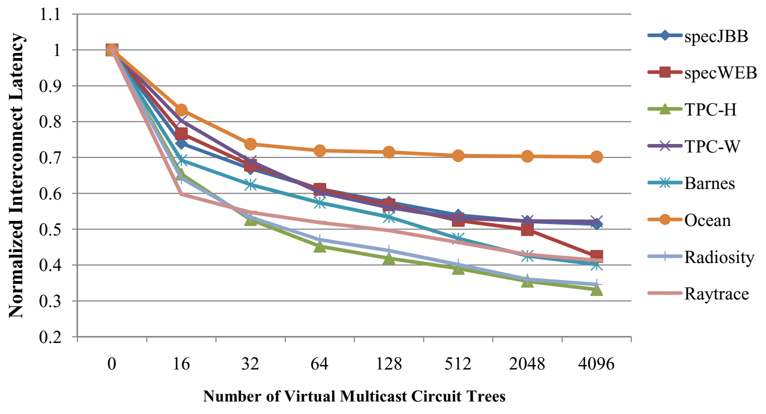


Figure 4.16: Virtual Tree Coherence network performance with VCTM

architecture [71].

Figure 4.17 shows benchmarks with varying degrees of improvement due to multicast support for the TRIPS architecture. Art sees the most benefit due to the network load approaching saturation and a significant reduction in the number of packets by using VCTM. The majority of workloads see less than 20% improvement due to the low number of nodes in the destination set (the average is 2). Additionally, many multicast trees constructed for these workloads branch at the source node. If a tree branch occurs at the source node, little or no benefit is seen as there will be little to no reduction packets over the multiple unicast approach. Path-based multicast outperforms VCTM for art, bzip2 and swim by 4% due to more effectively reducing the network load.

4.10.1.6 Adaptively Routed Multicast

In [88] multicasting is performed using a multiheader approach; each destination is encoded in a separate header with a single payload. Using a multiheader approach eliminates the need for a setup phase; each header flit is routed at each node and the payload is fanned out to all necessary ports. The downside to using this approach is that it significantly increases the size of multicast packets to accomodate multiple headers per packet (as discussed in Section 4.5 on 85). In all scenarios evaluated except

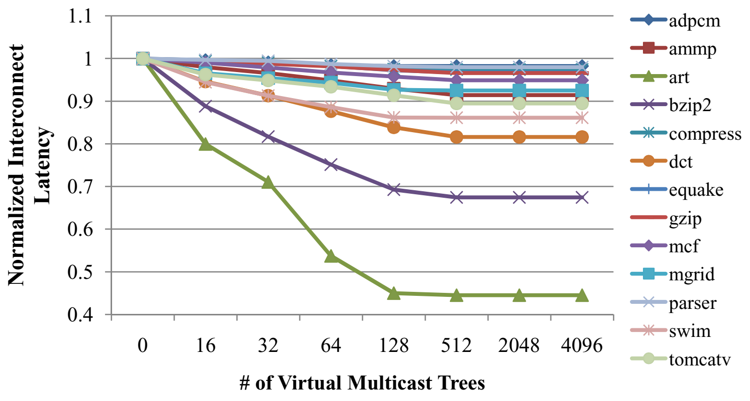


Figure 4.17: TRIPS network performance with VCTM

for TRIPs, a VCTM multicast packet consists of a single flit (3 flits for TRIPs). In the coherence scenarios, such as TokenB, a multiheader approach now sends a single multicast packet consisting of $n + 1$ flits (where n is the number of destinations in the multicast) rather than n single-flit multicasts. With TRIPs, for a multicast destined for two destinations, the multiheader approach would send four flits instead of six flits in the baseline; the larger payload for TRIPs packets allows potential traffic savings.

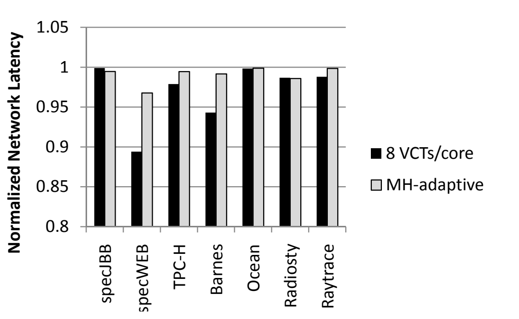
The potential benefit to the multiheader approach is the flexibility and ease of applying adaptive routing algorithms to multicasting. In this section, we apply a west-first turn model routing algorithm [50]; to when multiple output ports are available, the number of credits is used to determined the less congested route. More sophisticated adaptive routing algorithms (e.g. [53, 74]) could be considered for additional benefit but are outside of the scope of this work. Applying a simple adaptive routing algorith such as west-first to VCTM is not straight-forward (as discussed in Section 4.8 on page 93); modifying an adaptive routing algorithm to be suitable for use with VCTM is left for future work.

The results of the multiheader adaptive multicasting (MH-adaptive) is shown in Figure 4.18 for all five scenarios with network latency normalized to the baseline multiple unicast approach. The directory protocol sees modest performance improvements from adaptive routing in Figure 4.18a but VCTM outperforms it by reducing traffic. For the TokenB and Opteron scenarios (Figure 4.18b and c) MH-adaptive does not reduce the traffic over the multiple unicast approach but is able to better distribute the traffic on different network channels resulting in performance improvements over the baseline. VCTM with 8 VCTs per core still outperforms this approach. Like TokenB and the Opteron protocol, VTC (Figure 4.18d) experiences some improvement from adaptive routing but VCTM provides greater improvements across all workloads. With the TRIPs scenario (Figure 4.18e), MH-adaptive improves performance over both the baseline and the 8 VCT case for most benchmarks. TRIPs is marked by small destination sets and MH-adaptive is able to reduce the traffic over the baseline while not paying the setup overhead associated with VCTM.

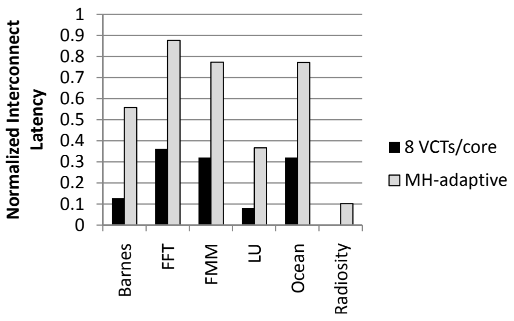
4.10.1.7 Path-Based Multicast

Figure 4.19 compares each scenario with a path-based multicast and VCTM with 16 VCT entries per core. The path algorithm we consider forms a single packet that visits each destination in turn; dual-path algorithms that allow two packets to start at the source node cut down on path length and are discussed in Section 4.11. Both path multicast and VCTM are averaged across all applications within the scenario and normalized to the multiple unicast baseline. Long path lengths impact VTC and the AMD Opteron protocol causing performance degradation. Significant performance improvement is observed with TokenB Coherence since path-based multicasting does alleviate network pressure despite long path lengths. Small performance improvement over the baseline is observed for TRIPS; consuming instructions are usually scheduled for locality, which reduces the likelihood of long path lengths.

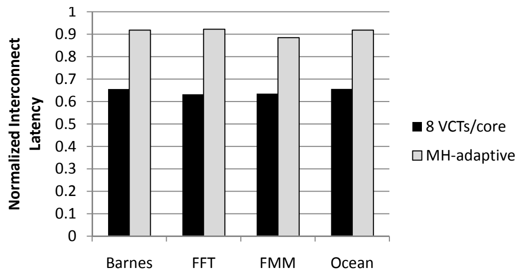
In summary, we see that VCTM provides a range of benefits for the different scenarios evaluated; the impact of VCTM is tightly coupled to the multicast behavior



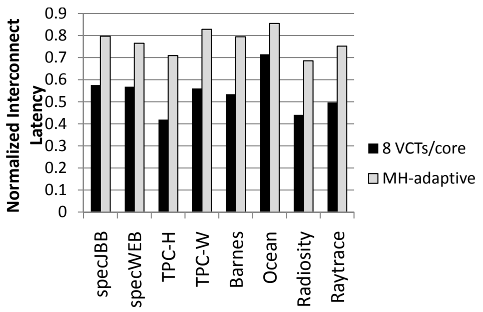
(a) Directory Coherence



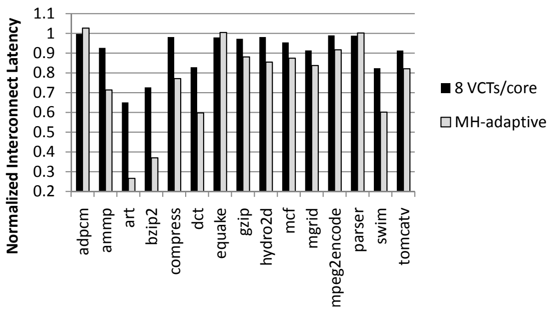
(b) TokenB Coherence



(c) Opteron Protocol



(d) Virtual Tree Coherence



(e) TRIPs Operand Network

Figure 4.18: Adaptively Routed Multicasts

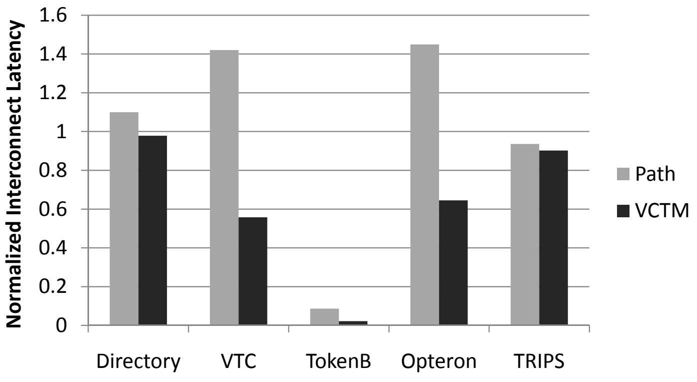


Figure 4.19: Comparison between VCTM and Path-Based Multicast

of the applications examined. The directory protocol sees the least benefit; for this scenario, the network load is low leaving less room for improvement. With TokenB coherence, token requests are broadcast to all nodes, so only one virtual tree is needed per node. Several of these benchmarks are running very close to network saturation resulting in substantial latency savings with VCTM. Virtual Tree Coherence requires more simultaneous multicast trees than other scenarios to see substantial benefit. Without multicast support, this type of coherence protocol would see prohibitive network latency for sending out snoop requests. TRIPS sees only a small average improvement with VCTM. The average destination set size for TRIPS is two leaving little opportunity for improvement. The Opteron protocol sees steady improvement with the addition of virtual circuit trees; once 512 VCTs are available, performance levels off with an average savings of 47% in network latency. Some filtering of destinations occurs in this protocol resulting in a larger number of trees than the TokenB protocol. VCTM provides the most significant improvement for networks that are operating at or near saturation and for larger destinations sets; further modifications to the baseline design are needed to provide improvements for small destination sets which are found in scalar operand networks and directory protocols.

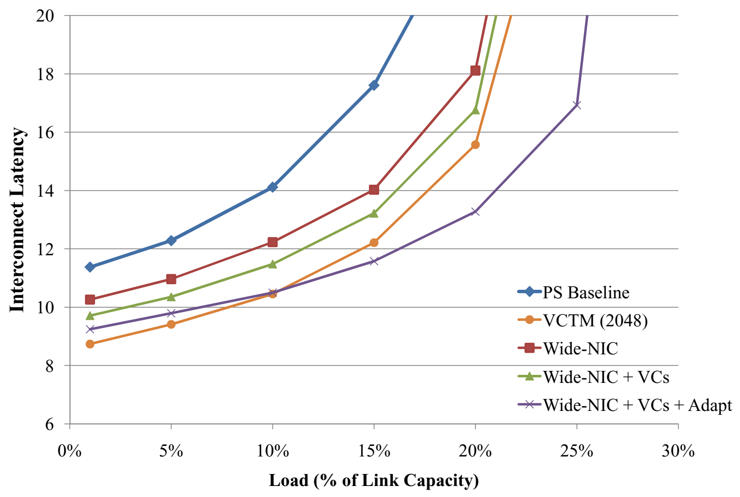


Figure 4.20: VCTM under uniform random traffic with 10% multicasts

4.10.2 Injection Port Contention

The VCTM router reduces the number of message injected into the network from the size of the destination set to a single message for each multicast. Injection port contention accounts for up to 35% of packet latency in the baseline. On average, alleviating injection pressure reduces the cycles a message spends in the network interface controller by 0.2 (directory), 6 (TokenB), 5 (VTC), 0.5 (TRIPS) and 3.5 (Opteron) cycles, where the average network latency is 14 cycles.

4.10.3 Synthetic Traffic Performance

Next, we use synthetic traffic to further explore the benefits of VCTM. Several aggressive packet-switched networks are evaluated in Figure 4.20 for their ability to approach the performance of VCTM. The PS baseline represents the same baseline packet-switched router configuration used in previous performance studies. The network interface (NIC) represents a substantial bottleneck for multicasting, so the Wide-NIC configuration allows the NIC to inject as many packets are are waiting in a cycle (in the baseline, only one packet can be accepted per cycle). We add to the Wide-NIC

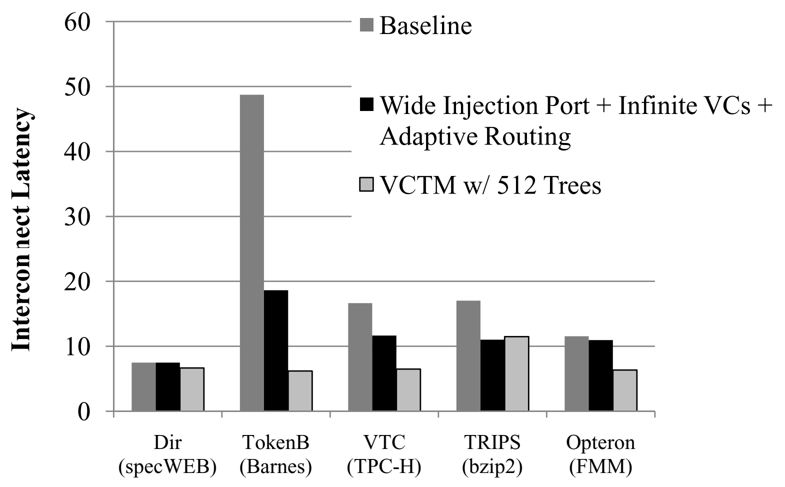


Figure 4.21: Comparison against aggressive (unrealistic) network

configuration nearly infinite Virtual Channels and buffers (wide-nic+vcs). To better distribute the network load, we utilize adaptive routing (wide-nic+vcs+adapt). Finally, we compare each of these scenarios to VCTM with 2048 VCTs (128 VCTs per core). With uniform random traffic, we see much less tree reuse than is seen in actual workloads which diminishes the benefits of VCTM.

Here approximately half of the latency penalty associated with the multiple unicast approach is paid in the NIC; creating a wider issue NIC would likely result in significant overhead and complexity; more so than our proposal. Wide-NIC+VCs+Adapt outperforms VCTM for moderate loads; performance improvements in VCTM are predicated on tree reuse which is very low for uniform random traffic. Building a design with a very large number of VCs would have a prohibitive cost (area and power); we believe our design is much more practical.

Figure 4.21 illustrates that with real workloads, VCTM outperforms the highly aggressive (unrealistic) packet-switched router. We compare one benchmark running with 512 VCTs from each scenario to Wide-NIC+VCs+Adapt. In the presence of reuse, our simple hardware solution achieves greater benefit than the aggressive network. In

Figure 4.20 there is very little reuse of trees; so even with VCTM, multicasts consume significant bandwidth. In contrast, with real scenarios, reuse is very high causing VCTM to out-perform the aggressive, adaptively routed network. TRIPS is the one exception but again, VCTM is much less expensive than the aggressive network and performs comparably in this case.

4.10.4 Impact of VCTM Optimizations and Modifications

Next, we examine modifications made to the original VCTM design. To improve scalability and overall performance of VCTM, we propose replacing the Destination CAM with a TCAM, explore other replacement algorithms, and evaluate dynamic partitioning of the VCT tables. Lastly, we restrict the number of virtual channels in the network for the baseline packet-switched design and for VCTM. Utilizing VCTM with the Virtual Tree Coherence protocol requires that a single tree use the same virtual channel at each hop in the network.

4.10.4.1 Destination Set CAM hit rate

For VCT tables with 512 entries (32 per core), we are able to achieve significant reuse across all scenarios. Individual benchmark VCT table hit rates range from 62% to 100%; with directory coherence seeing the lowest hit rates overall. In Tables 4.5a and b, we provide the average hit rates for each scenario for a range of Destination Set CAM and TCAM entries.

Remember that replacing the CAM with a TCAM doubles the size of tree storage in the NIC. However, we can reduce the VCT tables at each router by a factor of four or eight resulting in a net reduction in storage for the same performance gains. The hit rates for 16 entries with a TCAM are similar to those with a 64 entry CAM. This reduces the overhead in the baseline design and improves VCTM's ability to scale to larger network sizes. There is additional potential for improvement with more sophisti-

Table 4.5: Virtual Circuit Tree Hit Rates (%)

Destination CAM Entries per core	Directory	TokenB	VTC	TRIPS	Opteron
4	46	100	63	30	41
16	68	100	84	92	91
32	78	100	88	98	99
64	86	100	91	99	100

(a) CAM Hit Rates

Destination TCAM Entries	Directory	TokenB	VTC	TRIPS	Opteron
4	56	100	69	82	95
16	78	100	90	99	99
32	85	100	92	99	100
64	90	100	95	99	100

(b) TCAM Hit Rates

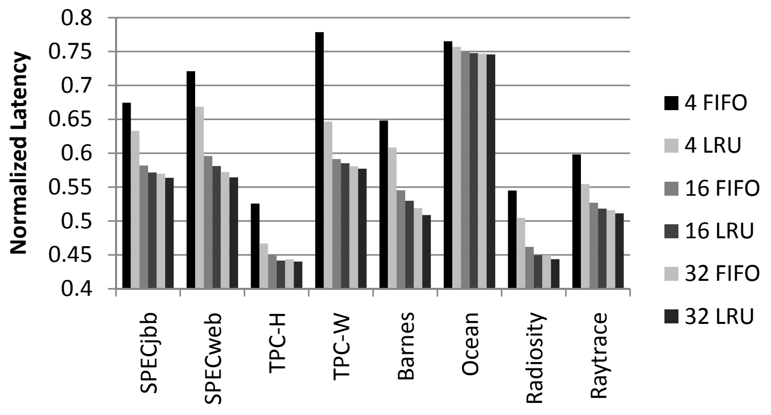


Figure 4.22: LRU optimization for VTC

cated matching heuristics. However, as previously mentioned imprecise trees may cause problems depending on the application; imprecise trees may also consume additional power due to extra cache accesses at leaf nodes.

4.10.4.2 VCT replacement policy

To improve the hit rate of frequently used multicast trees, we explore a LRU replacement policy versus the baseline FIFO policy. When LRU replacement is applied to VCTM with our real workload traces, we see improvements ranging from 1% to 13% for 4 VCT entries per core. The most significant performance improvements comes for a small number of trees (4 per core). Tracking LRU information for a large number of trees yields only very minor benefits (1% improvement for 16 entries per core). Improvements for the VTC scenario with LRU replacement are presented in Figure 4.22.

With uniform random synthetic traffic (with 10% multicasts), we found the switch from FIFO replacement to LRU to have no impact on performance; however, when we couple the LRU replacement policy with the TCAM optimization, we see lower latency by between 13% and 31% (depending on network load) and improved throughput. Saturation throughput is improved by approximately 5% with the LRU+TCAM optimizations. Uniform random traffic does not exhibit the reuse property that VCTM is

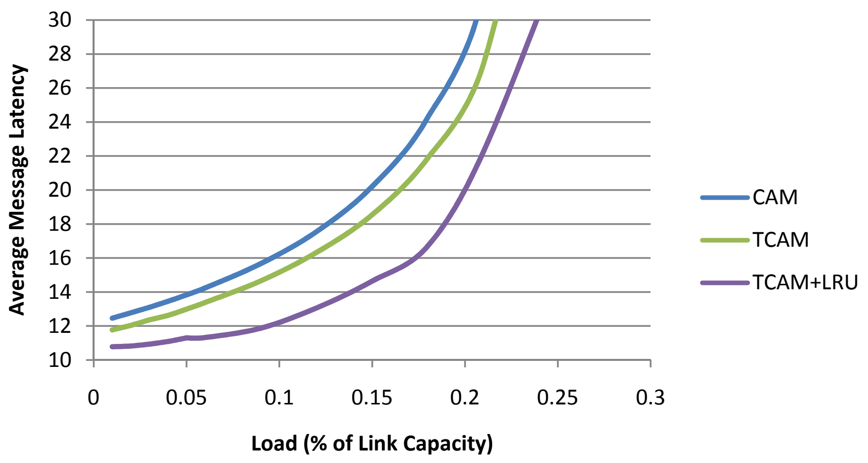


Figure 4.23: Performance Impact on Uniform Random Traffic with 10% Multicast using the TCAM and LRU optimizations with 16 VCT entries per core.

predicated on but by maintaining frequently used TCAM entries, we are able to capture reuse of similar destination sets and see throughput and performance benefits. Therefore, even with low-reuse scenarios, we can achieve benefit in the presence of similar destination sets. The Virtual Circuit Tree hit rate improves from 0% to 11% with the TCAM and to 33% with the TCAM+LRU configuration when the network load is 10%.

4.10.4.3 Static vs. Dynamic Table Partitioning

We explore the opportunity to use dynamic virtual circuit tree table partitioning to reduce the virtual circuit tree table sizes at each router. Figure 4.24 provides the minimum, average and maximum tree usage per core for a one million cycle window sampled every 10,000 cycles throughout the execution for the Virtual Tree Coherence scenario. We focus on the VTC scenario since it has the highest tree demand of all the scenarios. We find that there are periods of high demand from a small number of cores as well as periods where one or more cores are exhibiting very little tree usage. This data suggests that a dynamic partitioning scheme could more effectively use the virtual circuit tree entries; however, we leave the details of such a scheme for future

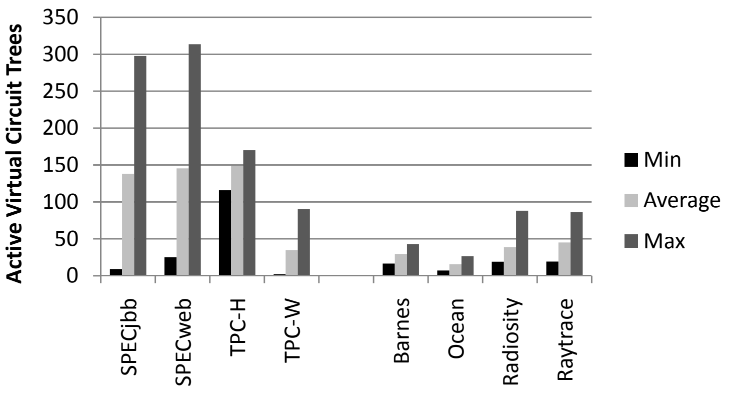
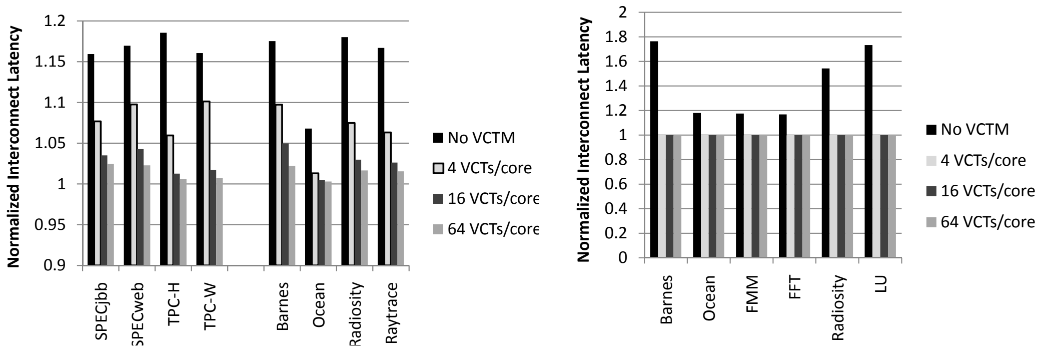


Figure 4.24: Dynamic Partitioning

work. Dynamic partitioning increases the size of the destination set CAMs which could negate the improved utilization of virtul circuit tree tables.

4.10.4.4 Impact of reducing Virtual Channels

To maintain in-network ordering of messages, a virtual tree is allowed to only use one virtual channel; however with a modest number of available virtual trees (16 trees per core), we are able to use network bandwidth efficiently and only see a degradation of 3% in end-to-end network latency (versus an unrestricted virtual channel allocation). In Figure 4.25, we illustrate the impact of a varying number of virtual channels on the baseline network and the VCTM network. Figure 4.25 includes results for Virtual Tree Coherence (a variable multicast scenario) and for the TokenB scenario (a full broadcast scenario); we believe these two scenario capture the range of virtual channel impact. TokenB has the same results for 4, 16 and 64 VCTs/core since it is a broadcast scheme; each source node needs only 1 VCT to reap the full benefit. Without VCTM support, virtual channels play a much more critical role in the performance and throughput of the network; this matches intuition as the baseline network experiences a much higher load from the multiple unicast approach. With VCTM, we see that virtual channels have some impact but it is much smaller and restricting the virtual channels does not



(a) Virtual Tree Coherence (b) TokenB Coherence

Figure 4.25: Virtual Channel Restriction: Interconnect Performance with 1 VC normalized to 16 VC

result in significant degradation. Therefore, the in-network ordering of virtual trees comes at a small network performance cost making VCTM effective for Virtual Tree Coherence ordering.

4.10.5 Activity/Power

The virtual circuit tree table increases the power consumption of our multicast router over the baseline packet-switched router; however this increase in power consumption is offset by reducing redundant link switching and buffering. As shown in Tables 4.2 and 4.3 (on page 92), the per-access energy consumed by our added hardware structures is small. For 32 VCTs/core, a read consumes 0.0018 nJ; while a read of a 32-entry DCAM consumes 0.007 nJ. Using the power models from Orion [139], the energy consumed by a link traversal is 0.049 nJ. This number was calculated using a 70 nm technology and 5mm, 16-byte links. The energy saved for each link traversal is significantly more than the energy spent performing the VCT lookup (13x for table with 4096 entries); for a small energy cost associated with the tables, significant savings are possible.

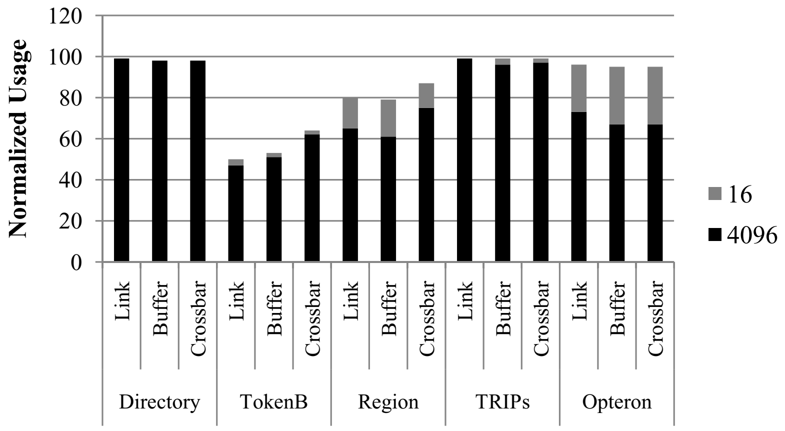


Figure 4.26: Reduction in buffering, link and crossbar traversals

4.10.5.1 Reduction in Network Usage

Figure 4.26 shows the reduction in buffering, link and crossbar traversals across the five scenarios under evaluation compared to the baseline router. These three components consume nearly 100% of the router power [140]; the reduction in switching shown will translate into significant dynamic power savings over the use of multiple unicast messages. This figure shows the activity reduction for a very small number of multicast trees (16) and a very large number of trees (4096). As performance levels off at or before 4K VCTs, these numbers represent the maximum activity reduction that can be achieved with our technique. Substantial savings can be achieved with a more modest number of trees.

Buffer accesses are already reduced in our baseline through the use of bypassing in state-of-the-art routers. VCTM is able to further reduce overall accesses by removing redundant packets from the network. Removing redundant packets allows more packets to bypass input buffers than in the base case.

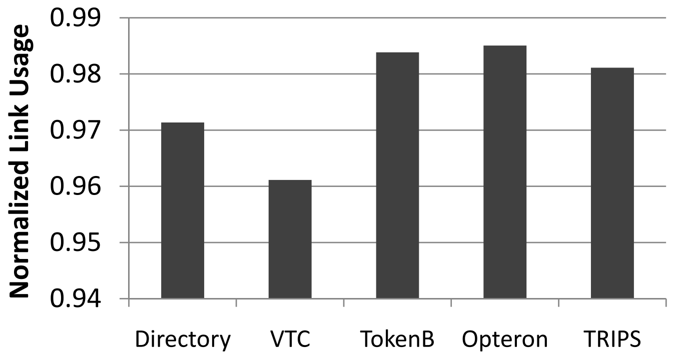


Figure 4.27: Average Link Usage with Minimum Spanning Trees relative to XY Trees with 16 VCTs/core

4.10.5.2 Impact of Optimal Trees

Virtual Circuit Trees are constructed along X-Y routing paths. The use of X-Y routing results in deadlock-free tree formations; however, X-Y routing does not necessarily produce optimal trees. All destinations are routed to in a minimal fashion in VCTM but alternative routing algorithms may produce trees that utilize fewer links to reach those destinations (these minimum spanning trees would be considered optimal). To determine how close to optimal our trees are, we remove the X-Y routing restriction; as a result minimum spanning trees can be found that use up to 60% fewer links. Despite the large possible reduction, the average reduction in link traversals is only 2% when compared to the saving achieved with X-Y routing as shown in Figure 4.27. This figure depicts the average link usage for each scenario when minimum spanning trees (MST) are constructed normalized to the link usages with dimension order X-Y trees. Both MST and XY results utilize 16 VCT entries per core. As a result, the power savings shown in Figure 4.26 are close to the maximum possible savings.

4.11 Related Work

Network design for multi- and many-core chips is an area of significant research effort. In the following sections, we examine the related work for VCTM, focusing on routing, encoding and deadlock. As many on-chip network solutions have leveraged off-chip designs, we also explore relevant off-chip network research.

4.11.1 Multicast Routing

There are two techniques for routing multicast messages: path-based routing and tree-based routing. In path-based routing, each destination is visited sequentially until the last node is reached [83]. Paths must be carefully selected to avoid deadlock; cycles can occur in the network even in the presence of dimension-order routing. Path-based multicasting also has the added complexity of finding the shortest path that visits all nodes in the destination set; this is done by ordering the destinations in the network interface controller. This ordering phase of message formatting can be quite costly; it has a software cost of $O(n \times log(n))$ [88] which would add considerable latency to the total message transmission latency.

Several multicast routing algorithms based on Hamiltonian paths have been proposed [84]. Dual-path and multi-path routing algorithms partition the destination set into disjoint sets and then route to each set using a path-based multicast. The destination set can be partitioned based on various heuristics. These algorithms fork at the source node but not at subsequent nodes; in their message passing environment, message replication is avoided because it is very expensive. Based on routing restrictions to prevent cycles in the paths, not all nodes are routed to in a minimal fashion which can lead to high network latencies; VCTM reaches all leaf nodes with a minimal path to provide a low multicast latency.

Very little prior work focuses on the design of on-chip multicast routers. In [87], the authors construct circuits for multicasting in a wormhole network. Path-based routing plus the requirement of setup and acknowledgment messages results in a long latency overhead for their approach. Alternatively, VCTM focuses on a tree-based routing; path-based multicasting was explored for the various scenarios. In these scenarios, we found that path-based multicast does improve performance by relieving congestion. The latency for the multicast to arrive at the final node is high for path-based multicasting; since requests in a coherence protocol may have to wait until they receive responses from all nodes, we believe path-based multicasting is unattractive in these systems. VCTM delivers all requests in minimum hop counts so that all responses will be returned in a timely fashion.

With a path-based multicast, current lookahead routing mechanism can be used as only one destination is being routed to at a time. Path-based multicasting is attractive for its routing simplicity; implementing a path multicast would require only minor modifications to the current packet-switched router. We have demonstrated that network latency is a critical factor for commercial workloads on CMPs; therefore it is preferable to avoid the sequential latency associated with a path-based routing approach. Our design is able to leverage lookahead techniques by using slightly wider bundles to remove the VCT lookup from the critical path.

A multicast router design [63] for DNUCA caches [68] is constrained to match very specific characteristics of the DNUCA design space, i.e. many routes are unused. Additionally, the details of how they realize their multicast router are sparse. Circuit-switching and time-division multiplexing have been used for on-chip networks that provide multicast functionality [85], but suffer from the constrained bandwidth of circuit-switching.

Significant work has been done for off-chip multicast routers. Several proposals target multistage interconnection networks [128, 135]. Domain-specific requirements of off-chip networks (e.g. ATM switches) can be quite different [135]; this work targets a switch with 1000 input ports. Routers with a large number of ports are prohibitively expensive for on-chip networks making this type of solution unattractive. While a lookup table is also indexed to find the proper output port mappings, VCTM utilizes a much smaller lookup table that is more suitable for on-chip designs. Additionally, their work adds and removes nodes incrementally to a multicast tree while our work creates a new tree at low latency and overhead and hence does not support addition or deletion of nodes. Adding nodes to an existing tree in VCTM is fairly straight forward; a unicast+setup packet is formed in the NIC with the VCT number and the ID field corresponding to the existing. As this unicast+setup packet is routed its output ports are added to the tree as needed. This feature could be easily achieved with VCTM but assessing its usefulness is left to future work. Deleting nodes from an existing tree is decidedly more complicated since an output port at any given router could be utilized by multiple destinations and there is not enough information inside the network to determine which destinations need which output ports. Another design [128] also uses a lookup table to determine the routes but advocates using software to set up this table; this approach would work well for fixed routes that endure for a significant amount of time. For fine-grained on-chip parallelism, software approaches to routing may incur too much overhead.

Recently, Anton [118], specifically designed for molecular dynamics, uses off-chip multicasting; multicasts can be sent to limited sets of nodes. The Piranha architecture's novel technique, *cruise missile invalidations* limits the number of messages injected into the off-chip network from a single request [13]; each invalidate multicasts to a subset of nodes. As with VCTM, Piranha sees improved invalidation latencies by reducing the number of messages. They also reduce the number of acknowledgments; we defer study of similar reduction operations to future work.

4.11.2 Multicast Destination Encoding

Another significant challenge with on-chip multicasting is the destination encoding within the header flit. There are several approaches to destination encoding [29] including all-destination encoding and bit-string encoding. The all-destination approach encodes each destination node number into the header. The header can be of variable size with an end of header character to delineate it from the payload. Bit string encoding uses a single bit for each possible destination. If the node is included in the destination set, the bit will be set to one. The header size needed to encode the number of destinations is fixed. The number of bits to encode the number of destinations grows as the network grows for each of these approaches.

Nodes can also be partitioned into regions and multicasts sent to all destinations within each region. The header size can be fixed to limit the number of possible destinations that a multicast can reach; however, our solution is more flexible. The 16 bits necessary to encode all possible destinations in a 4x4 mesh could reference 2^{16} different trees. Virtual Circuit Tree Multicasting is thus a much more scalable solution than destination or bit-string encoding. If extra messages can be handled properly by the application (coherence protocol or operand network), then coarse vectors can be used for encoding the multicast destination similar to what has been done to reduce the required directory storage in traditional DSM multiprocessors [55]. The Dir_iCV_r approach for directories specifies i pointers and a region size of r. I destinations can be specified precisely, but when the number of destinations exceeds i a coarse bit vector is used. This approach could be applied to multicasting; additional logic would be required within the router to determine the output ports for a coarse region.

In an off-chip network design, multi-port encoding [122] determines when to replicate a packet in each state of a multistage network; this restricts the destination sets that can be reached by a packet. Some multicasts require multiple passes through the network to reach all destinations; due to the latency sensitivity of our workloads, multiple passes are undesirable. Multiple output ports are encoded in the header flits; a different output is used in each pass until all destinations are reached. This approach is closely tied to the multistage topology and would be difficult to implement in a two-dimensional mesh. VCTM efficiently routes multicasts in a straightforward manner. Their work also acknowledges the inefficiencies of the multiple unicast approach but in an off-chip setting.

There is a continuum of ways to realize a multicast. At one end of the spectrum you have the multiple unicast approached; the other extreme is a single multicast. The unicast approach requires $nx1$-destination messages; the full multicast approach requires a $1xn$-destination message. In between you have a range of options; a multicast could be realized by $\frac{n}{2}$ messages with two destinations each (bicasts). Another example would be $\frac{n}{4}$ multicast messages with four destinations each. The multi-port encoding scheme [122] requires multiple serial phases to deliver a multicast to all destinations; this approach lies between the two extremes of the continuum. Exploring alternative ways of encoding and realizing multicast through multiple multicasts may provide additional benefit for our schemes that have small destination sets; we leave this exploration as future work. Realizing a multicast destination set with multiple trees may also reduce the amount of tree storage required by our VCTM design.

4.11.3 Multicast Deadlock

Routing decisions are more complex with multicast routing than unicast routing. In tree-based routing difficulties arise when a branch occurs in a multicast tree. Routing to multiple destinations simultaneously would require either replication of of the routing hardware for all n possible destinations of a multicast packet or would require n iterations through the routing logic. Lookahead routing could encode multiple output ports for the next hop in the network, however, the destination set would need to be properly partitioned. At a branch, a subset of the destinations would need to be encoded for each output port. Failing to prune destinations from the header flit or improperly pruning destinations would result in deadlock. VCTM uses the VCT number to avoid partitioning and pruning the destination set to avoid these deadlock problems.

Research into deadlock-free multicast tree routing [88] uses pruning to prevent deadlock in a wormhole-routed network. Their work targets small messages such as invalidates in a DSM system. In VCTM, for a given tree, all of the leaf destinations are reached via dimension-order routing with respect to the source node; therefore no cycles can occur within a single multicast tree instance. If a tree were allowed to adaptively route with respect to a branch (an intermediate route point) deadlock would be a problem. Path-based multicast routing can also result in deadlock in a wormhole-routed network. The ordering of destinations within the path can result in a cycle within the route of a single multicast message.

At a branch in the network, multiple output ports will need to be traversed by the same packet. In VCTM, a multicast packet does not need to wait for all output ports to be available in a single cycle, but can instead route to each output as it becomes available. Requiring a packet to wait for all output ports to be available in the same cycle would degrade performance and possibly induce deadlocks across multiple active multicast trees.

4.12 Conclusion

In this chapter, we present the second of two interconnection networks, Virtual Circuit Tree Multicasting. Hybrid Circuit Switching is a technique to streamline the router pipeline used in on-chip communication. It is particularly effective for pair-wise communication patterns. Virtual Circuit Tree Multicasting attacks a different part of the router architecture. Rather than try to optimize the entire pipeline, VCTM focuses on a single stage, the routing computation. By providing efficient routing for multicast messages, VCTM causes such messages to flow through the rest of the router pipeline in a streamlined fashion. There are some commonalities in two interconnect designs developed in this dissertation; both HCS and VCTM rely on communication reuse for benefit. Both designs also overlap the set up phase in the network with useful work to keep packet latency low.

Chapter 5

Circuit-Switched Coherence

In the previous two chapters, we proposed optimizations to the on-chip interconnection network. Hybrid Circuit Switching and Virtual Circuit Tree Multicasting are interconnection network designs that optimize *how* data is communicated assuming certain application and protocol behaviors. In these two chapters we will focus on *what* is communicated and design protocols that map well to these two interconnect designs. This chapter looks at co-designing the first of two protocols with our interconnect optimizations in order to improve overall communication performance.

We couple the hybrid circuit-switched network with a directory coherence protocol based on the SGI Origin 2000 [80]. As noted earlier, directory-based protocols are more scalable than broadcast-based ones; however, a key performance disadvantage of directory protocols is the indirection through the directory that is required to request data from another processor. With a pairwise interconnect such as HCS, we want a protocol that is going to best maximize pair-wise communication.

5.1 Circuit-Switched Coherence Overview

Our protocol extensions streamline coherence requests for load and instruction misses by predicting pairs of processors that frequently share data and directly requesting data from one cache to another, via a circuit-switched link without first consulting the directory. Sequential consistency is maintained by ordering all coherence events at the directory, but latency is improved by overlapping the circuit-switched data transfer with the directory access.

To allow directory indirections to be streamlined, we modified the directory protocol as follows:

- Allow a cache to send a request directly to another cache.
- Notify the directory that a sharer has been added without having the directory forward the request to the owning cache or initiate a memory request.
- Retry the request to the directory if the direct request fails due to an incorrect prediction or a race condition.

The directory does not need to be aware of which circuit-switched paths are active as long as it is notified to add a new sharer to the sharing list for each cache line. The protocol implementation is decoupled from changes to the interconnection network.

5.2 Pair-wise Sharing Prediction

Coarse grain coherence information is used to predict the likely pair-wise sharer of data. Each processor stores information about sharers of an address region alongside its last level cache. When data is sourced from another processor, the predictor stores the identity of the sourcing processor for the address region of the data. The next time a cache access misses to an address in that region, we predict that the same processor will again source the data. Unlike the region tracking that will be done for VTC, these regions just serve to provide a prediction and are not required to be correct. At the time of the prediction, if no circuit-switched path exists between the requesting and predicted processor, one is established.

If the predicted core cannot provide the cache line requested, it responds to the requesting processor to indicate an incorrect prediction. The requesting core must then

Table 5.1: Prediction and Protocol Walk-through Example

1. Processor 1 misses to Address A
   First miss to A $\rightarrow$ No Prediction Available
2. Send Request to Directory
3. Directory forwards request to Processor 4
4. Processor 4 responds with data
5. Processor 1 receives data and stores Prediction(Region A) = 4
6. Processor 1 misses to address $A + 8$ $\rightarrow$ Predicts Processor 4
   Send request to Processor 4
   Notify Directory (in parallel)
7. Processor 4 receives predicted request
   Prediction is correct $\rightarrow$ responds with data
8. Directory adds Processor to sharing list for Address $A + 8$
9. Processor 1 receives data from Processor 4 and ACK from Directory

retry to the directory. The latency of the mispredicted request is thus the round trip latency from cache to cache on the interconnect plus the indirection latency of the conventional directory request. In some cases, the directory will be able to detect an incorrect prediction and initiate first level actions to reduce the latency impact of mispredictions. The prediction array is then updated to reflect the core that actually sourced the data for that cache line. An example of the protocol and prediction mechanism is given in Table 5.1.

In the example in Table 5.1, if Processor 1's prediction had been incorrect, the directory will have already added Processor 1 to the sharing list due to the decoupling of the data request from the ordering request. The directory protocol already supports silent evictions of shared lines, so having Processor 1 added as a sharer will not result in any incorrect coherence actions (even though Processor 1 will have to retry and is not currently sharing that cache line). The sharing list must include all processor caching that block but can contain additional processors at the expense of more invalidation messages should an upgrade or store request occur. Processor 1 will retry its request to the directory once Processor 4 has acknowledged Processor 1's incorrect prediction.

5.3 Ordering Invariants

We build these protocol modifications on top of an existing protocol [80]. In both the baseline and the modified protocol, the directory serves as the sole ordering point for requests. Our modifications decouple the data request message from the ordering message. The data response may be received prior to the acknowledgment that the requests has been properly ordered from the directory, but the requesting processor cannot consume the data without that acknowledgment. The directory will send a negative acknowledgment if the decoupled ordering request reaches the directory in a busy state. The busy state indicates that a store or upgrade was ordered prior to this request and the read must be retried. To illuminate the protocol changes as well as demonstrate that ordering properties have been preserved, we include pseudo-code for a read request for both the baseline and modified protocols in Table 5.2.

5.3.1 Race Example

Figure 5.1 illustrates the necessity of the directory acknowledgment message included in Table 5.2. This acknowledgment (or negative acknowledgment) message ensures that all requests complete in the correct order, thereby maintaining a coherent view of memory. If a directory notification arrives at the directory while it is in a busy state, the directory must respond with a negative acknowledgment so that the requesting processor does not acquire a shared block without being successfully added to the sharing list.

Consider the example in Figure 5.1a, in which P0 initiates a shared request directly to P1 who currently holds a shared copy of the data. At time 1, P0 sends out its request and directory notification so that it can be added to the list of sharers. However, at time 2, an upgrade request arrives from P2 and is ordered before the directory notification is received at time 3. P1 responds to the P0's request with the shared data

Table 5.2: Comparison between Baseline Protocol and Prediction Protocol

Baseline Protocol - Read Request RdReq	**Prediction Protocol - Read Request RdReq**
1. RdReq goes across network to home directory cache	1a. RdReq goes across network to predicted sharer 1b. Directory update goes across network to home node
Directory Actions	**Decoupled Directory Actions** **(cont. from 1 b)**
2. Directory cache performs look up **if** Directory state is Unowned **then** initiate memory request When memory request completes send data to requester **if** Directory state is Exclusive **then** Transition to Busy Send intervention to owner Add requester to sharing list Transition to Shared when cache response is received **if** Directory state is Shared **then** Forward request to owner Add sharer to sharing list **if** Directory state is Busy **then** Send NACK to requester	2. Directory cache performs lookup **if** Directory state is Unowned **then** Prediction is incorrect → initiate memory request When memory request completes send data to requester **if** Directory state is Exclusive **then** **if** Prediction is correct **then** Transition to Busy (wait for ack from predicted node) Send ACK to requester **else** Prediction is incorrect Transition to Busy & send intervention to owner Transition to Shared when cache response received Add requester to sharing list **if** Directory state is Shared **then** **if** Prediction is correct **then** Add sharer to sharing list & send ACK to requester **if** Prediction is incorrect **then** Forward request to owner & send NACK to requester **if** Directory state is Busy **then** Send NACK to requester
Cache Response to Directory request	**Decoupled Cache Actions** **(cont. from 1a)**
3. Owning cache receives intervention Send data to requester & transition to Owned Send ACK to directory	3. Owning cache receive predicted request **if** Prediction is correct **then** Send Data to Requester **if** Block is Exclusive or Modified **then** Send Ack to directory **else if** Prediction is incorrect **then** Send NACK to Requester
Cont. on next page	

Table 5.3: Table 5.2 (cont from previous page)

Baseline Protocol - Read Request RdReq	**Prediction Protocol - Read Request RdReq**
Requesting Node	**Requesting Node**
4. Receives Data from either Directory or Owning Cache Transitions to Shared & forward data to L1 RdReq Complete	4. **if** Data received from Predicted Cache **then** **if** Directory NACK received **then** Discard data (stale copy) & wait for valid response **else if** Directory ACK received **then** Forward Data to L1 & RdReq Complete **else if** NACK received from Predicted Cache **then** Re-initiate RdReq without prediction

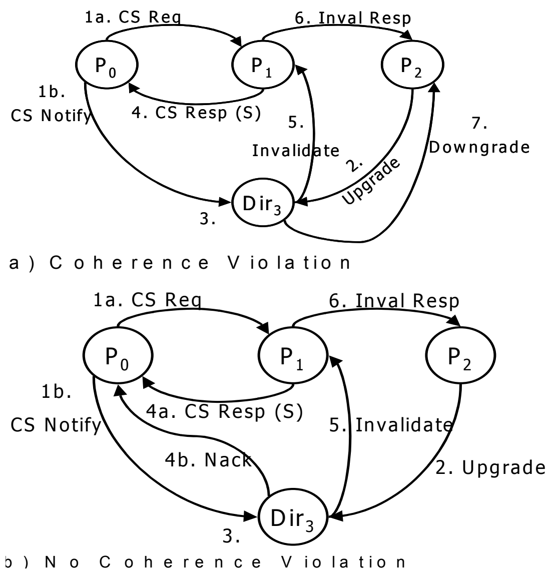


Figure 5.1: Protocol Race Example

at time 4 and subsequently sees an invalidate message from the directory at time 5. P0 will not receive an invalidate from the directory since it has not yet been added to the sharing list at the time of P2's request. P1 invalidates its copy of the block and sends an invalidation response to P2 at time 6. P2 now believes it has the only copy of the block and can complete its write to the block. In response to P0's directory notification, the directory sends a downgrade message which P2 sees at time 7. Now, both P0 and P2 believe they have shared copies of the block, but P0's copy is stale.

An acknowledgment has been added to the example in Figure 5.1b. At time 3, when the directory receives the notification from P0, it knows that P1 is no longer sharing the data (due to the upgrade) and that P0 will receive stale data from P1. The directory responds to the notification with a negative acknowledgment so that P0 will know not to use the data it receives at time 4 from P1.

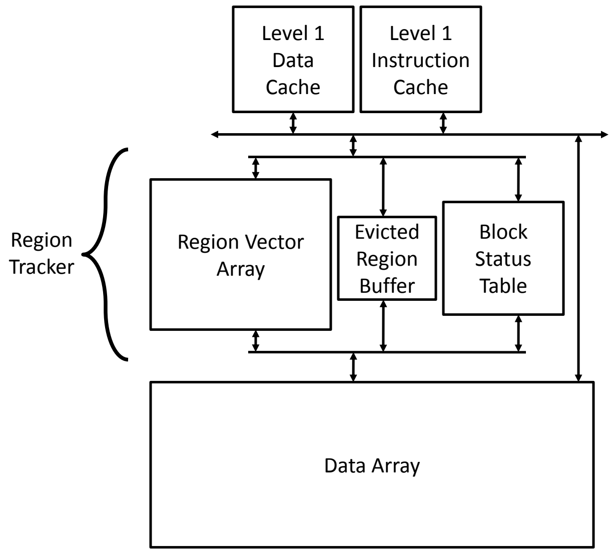


Figure 5.2: RegionTracker structures

5.4 Overhead

RegionTracker [147] has been proposed as a framework to efficiently track coarse-grained region information for various optimization purposes. Figure 5.2 illustrates the region vector structure that is used to track the presence of regions in the cache and provide location information for those blocks in the data array. RegionTracker also requires an evicted region buffer which minimizes the cost of region evictions. Since not all cache blocks within a region are going to reside in the cache, the block status table tracks per-block information such as LRU status and coherence state, for those blocks that are present.

Figure 5.3 illustrates how a 50-bit address indexes into the region vector array. Here we assume 1KB regions. We've augmented each Region Vector Entry to include a prediction field. The prediction mechanism in Circuit-Switched Coherence tracks the most recent sharer for each region. This requires an additional $log_2(N)$ bits per region vector entry (where N is the number of cores).

The resulting overhead for adding this information is calculated using Equation

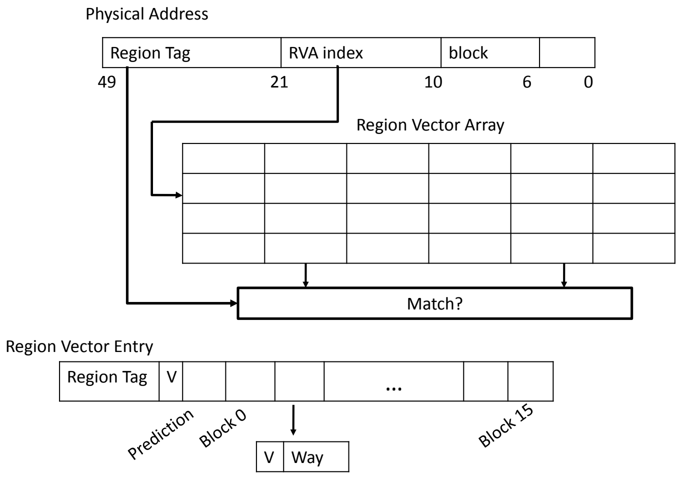


Figure 5.3: RegionTracker indexing

5.1. Rather than utilize a separate structure with a separate set of tags to store the region predictions, RegionTracker requires a single set of tags for both the L2 data array and the region information. This results in smaller overhead relative to the conventional L2 cache design. Assuming the L2 cache size of 512KB (same as in Table 3.1 in Section 3.5 on page 54) and 1024 region sets and 4 region ways, we calculate the overhead for predictions to be 36% compared to the conventional tag array. A separately tagged prediction structure (rather than RegionTracker) would result in 47% overhead compared to the baseline. When placed in the context of total on-chip cache storage (8 MB), the additional storage for RegionTracker represents only a 2% increase in bits required for cache storage.

$$\begin{split} RegionArraySize = (Tag + 1 + log_2(N) + (RegionSize/CacheLineSize) \times 4) \\ \times RegionSets \times RegionWays \end{split} \tag{5.1}$$

5.5 Evaluation

Here we evaluate the circuit-switched coherence protocol with the hybrid circuit-switched interconnection network. The same methodology and machine configuration

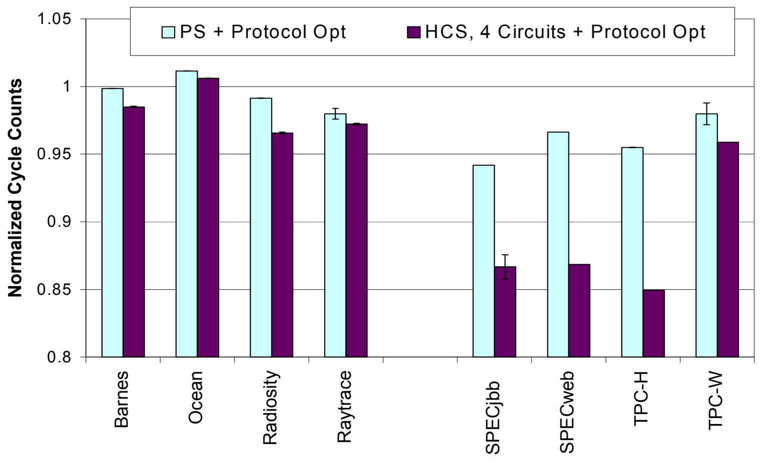


Figure 5.4: Performance of Protocol Optimization + Hybrid Circuit Switching

are used as in Chapter 3 (found on page 54). The baseline protocol is a directory protocol modeled after the SGI Origin protocol [80].

Overall system performance when hybrid circuit switching with four circuit planes is combined with the protocol optimization is shown in Figure 5.4. Since the protocol optimizations are largely independent from the interconnect design, we compare against the baseline packet-switched network augmented with the protocol optimizations with results normalized to the packet-switched baseline. The HCS results are given for four circuit planes and the *always setup* policy described in Section 3.4.3.

We see up to 15% improvement in overall system performance with an average improvement of 12% for commercial workloads and 4% for scientific workloads. The commercial workloads see greater benefit due to their higher miss rates and greater sensitivity to miss latency. Looking at the results for the packet-switched network with the protocol optimization, we see small improvements over the baseline. Commercial workloads see an average improvement of 5%; scientific workloads again see less benefit due to less sensitivity to miss latency. TPC-H derives most of its benefit as a result of 82% of misses being satisfied on-chip coupled with a very low contribution of store misses to the overall miss rate. As our protocol optimization only targets load and

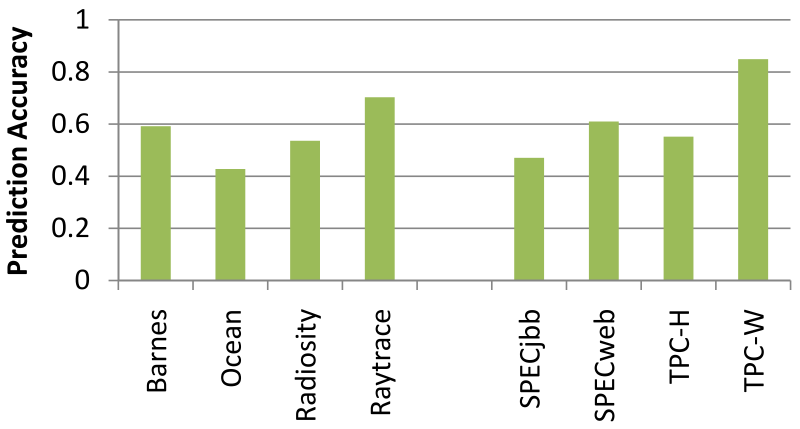


Figure 5.5: Region-Based Prediction Accuracy

instructions misses, TPC-H will see more benefit than those benchmarks that have a larger contribution of store misses. Ocean's performance is dominated by off-chip misses; the injection of additional traffic due to the protocol optimization further delays these misses resulting in a slight performance degradation.

Prediction accuracy is shown in Figure 5.5. The average accuracy of our region based prediction is 60%. Incorrect predictions result in extra interconnect traffic but do not add latency over the baseline protocol performance. The decoupled ordering request to the directory includes the prediction made by the requesting processor. When the directory detects this prediction is incorrect, it forwards the request to the owner of the cache line. This forwarded request will have the same latency as the baseline indirection through the directory.

Comparing Figures 3.14 (page 62) and 5.4, we see that in many cases, the improvement of HCS with the protocol optimization given four circuit planes is greater than the sum of HCS alone and the protocol optimization with packet switching. Given our co-design of both the coherence protocol and the interconnect, hybrid circuit-switching has more opportunities for circuit reuse when the protocol optimization is added, resulting in superior performance; circuit reuse increases by up to 64% (with an average increase of 16%) when HCS is combined with the protocol optimizations (shown in Fig-

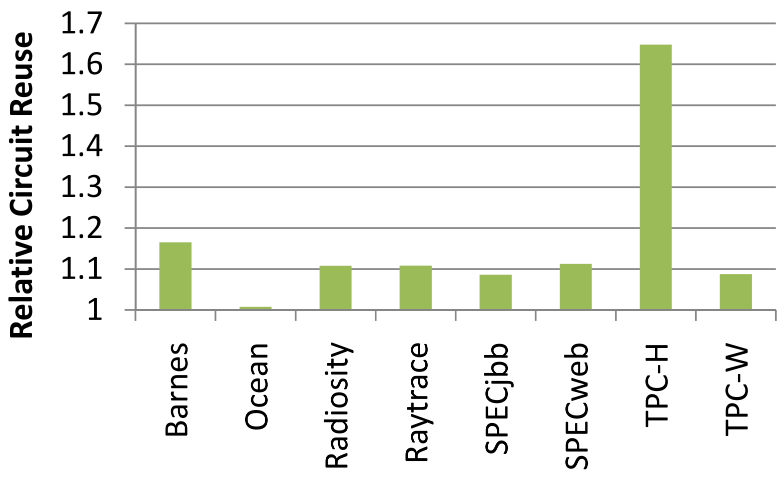


Figure 5.6: Circuit reuse with protocol optimization (relative to no protocol optimization)

ure 5.6). The protocol optimization reduces on-chip cache miss latency by 50% when using the packet-switched network. Using hybrid circuit switching, those miss latencies are further reduced by an additional 10%.

5.6 Related Work

Here we discuss several prior works studying directory protocol optimizations. Mukherjee and Hill [101] propose using prediction to accelerate directory coherence protocols using a predictor based on a Pap-style branch predictor. They predict the upcoming coherence actions based on the recent history of coherence requests. The Memory Sharing Predictor [77] improves on the accuracy of coherence predictors by limiting predictions to memory requests (reads, stores and upgrades) rather than all coherence messages (acknowledgment and invalidations are eliminated). Remote access latency can be reduced by having the directory initiate these coherence requests speculatively. Circuit-Switched Coherence focuses on predicting who will source the data for a given request to accelerate that transfer via a circuit-switched connection. With the HCS interconnect, we also explored the benefit of only establishing connections based on coherence requests (not responses), believing that those message are more indicative of sharing patterns. For HCS, limiting path construction based on coherence message types resulted in reduced coverage; for coherence predictors this limiting results in increased accuracy. Instruction-based prediction has been used to accelerate coherence [66, 67]. Work by Acacio et al. [3] also looks at taking the directory access off the critical path for distributed shared memory designs. In their work, only lines held in the exclusive or modified state can be accelerated through prediction; Circuit-Switched Coherence is extended to include shared cache lines and complements the HCS interconnect design. These other prediction schemes could also be coupled with the HCS interconnect to improve overall performance. PATCH [111] uses direct requests as performance hints to improve performance, while token counting and directory actions ensure correctness and forward progress. Their direct requests can be either pair-wise or multicast based on destination set prediction mechanism [91]. Additionally, their direct requests can be dropped due to interconnect congestion; the dropping of these messages will not impede forward progress.

To reduce the cost of coherence predictors, the Coherence Predictor Cache has been proposed [105]. This work builds on existing but expensive coherence predictors but leverages the reuse of a small fraction of cache blocks and coherence actions. This temporal locality is exploited to associate only a small number of predictors with these highly reused blocks. In this dissertation, we attempt to reduce storage overhead but consider memory behavior on a coarser granularity than single cache lines, through region based optimizations.

To eliminate the latency of the directory indirection, Cheng et al. [27] propose relocating the home node to be the producer-node in the presence of producer-consumer relationships. Now consumers of this data will receive the data directly from the new home node rather than experience the extra indirection of the baseline protocol. We believe this protocol would also make good use of our hybrid circuit-switched interconnection network.

5.7 Conclusion

In this chapter, we explore a coherence protocol that uses prediction coupled with fast interconnect paths to improve the latency of cache-to-cache transfers. This protocol places the directory ordering request in parallel with the data request to an on-chip cache. One of the limitations of Circuit-Switched Coherence is that it focuses solely on read-shared misses and does not provide optimizations for stores or upgrade requests.

Circuit-Switched Coherence places low bandwidth requirements on the interconnection network due to the point-to-point nature of requests. Traffic may increase slightly due to incorrect predictions but overall the traffic is still low and comparable to the baseline directory protocol. The bandwidth requirements of circuit-switched coherence are low due to the unicast nature of messages lending itself to bandwidth scalability.

Circuit-switched coherence relies on the reuse of circuit-switched paths between frequent sharers to reduce cache-to-cache transfer latency. As systems grow, additional pressure may be placed on these links resulting in more frequent reconfiguration and bandwidth stealing; however, if virtual machines and operating systems are able to maintain physical locality among sharing cores, circuit-switched paths will likely continue to be effective.

If the average hop count to the directory begins to far exceed the hop count to a pair-wise sharer, this protocol will suffer as it waits for the directory acknowledgment before it can consume the data from a sharer. Optimizing the path between the requester and the directory will become increasingly important. Modifications could be made to partially overlap this latency by consuming the data before the acknowledgment and relying on speculation support in an out-of-order core to roll back in the event of a negative acknowledgment. Another downside to this protocol is that it is predictive in nature which can lead to additional latency on mispredictions. Circuit-switched coherence also requires additional storage to track predictions as well as directory storage overheads which may limit its scalability for large systems.

In Chapter 2, we describe four desirable properties for on-chip cache coherence protocols. The first three are partially addressed by Circuit-Switched Coherence. First, Circuit-Switched Coherence communicates in a pair-wise fashion which limits the cores involved in coherence requests. Second, cache-to-cache transfer latencies are reduced by avoiding directory indirections and leveraging fast circuit-switched paths. Third, bandwidth requirements are low due to the point-to-point nature of requests. The final property, low storage overhead, is not addressed by this protocol, however, Region-Tracker is leveraged to minimize the additional storage overhead that is needed. A second protocol, Virtual Tree Coherence, that addresses these four characteristics will be discussed in the next chapter.

Chapter 6

Virtual Tree Coherence

To fully realize the massive computation power of many-core architectures, the communication substrate must be carefully examined and streamlined. There is tension between the need for an ordered interconnect to simplify coherence and the need for an unordered interconnect to provide scalable communication. In this chapter, we propose a second co-designed coherence protocol, Virtual Tree Coherence, that relies on a virtually ordered interconnect.

As discussed in Chapter 2, most cache coherence protocols rely on an ordering point to maintain correctness. In directory-based protocols, the directories serve as an ordering point for requests. With bus-based schemes (broadcast protocols), the bus serves as the ordering point for requests. While a bus is able to provide a total order, recently several proposals have leveraged the partial ordering properties of a ring to facilitate cache coherence [96, 126, 127]. Exploiting the partial ordering properties of a ring eliminates some of the disadvantages of a bus; however there are still some downsides. A ring represents a significantly less scalable topology than a mesh or a torus. Logically embedding a ring into a mesh topology [127] provides a higher bandwidth communication substrate for data transfers but still suffers from large hop counts to order requests and lower bandwidth for coherence requests as they must all traverse the same embedded ring.

6.1 Overview

Virtual Tree Coherence demonstrates that ordering can be achieved through structures other than a ring or a bus, in particular through a tree. Specifically, rather than realizing ordering through a *physical* tree interconnect [49], Virtual Tree Coherence maintains coherence through *virtual* trees. These trees are embedded into a physical network of arbitrary topology by VCTM, with a virtual tree connecting the sharers of a region. The root of this virtual tree now serves as the ordering point in place of a directory protocol's home node. To prevent a single tree from becoming a bottleneck, trees are assigned to coarse grained regions giving VTC a high bandwidth ordering substrate.

To improve scalability and performance, Virtual Tree Coherence is implemented as a hierarchical protocol. At the first level, multicast snooping is achieved through logical trees. At the second level of the protocol, the global protocol provides the caches with information about which processors must be involved in the first level snooping.

In a nutshell, every request is first sent to its virtual tree root node (obtained from the local region tracking structure explained in Section 6.8). This root node orders the requests in order of receipt, then multicasts requests to all sharers of the region through a VCTM virtual tree for that region. The above works for all *current* sharers. *New* sharers, however need to take a two-level approach: first they have to go to the directory home node to obtain the identity of the tree root or broadcast to all cores to find the root node. This new sharer will then be updated into the local region tracking structure and added onto the virtual tree.

Virtual Tree Coherence results in several benefits due to our first level multicast protocol. First, the root node can be strategically selected to be one of the sharers to cut down on latency. In a conventional directory protocol, the home node for an address is statically defined for a given address, and is thus not necessarily a sharer. Second, latency is low, comparable to a broadcast-based protocol, since we do not need to collect acknowledgments nor wait for a directory lookup; messages that reach the root node begin their tree traversal immediately. Third, bandwidth demand is low compared to a broadcast-based protocol, as only the current sharers are involved in the multicast, rather than all nodes; meanwhile, with an unordered interconnect offered bandwidth remains high.

6.2 Hierarchical Coherence

With the anticipated prevalence of many-core architectures, cache coherence can potentially become a significant system bottleneck. For these large systems, providing global coherence across all cores will become very expensive. For directories, the expense lies in increasing hop counts to order requests. For broadcasts, the expense comes in the dynamic power and high bandwidth requirements associated with frequent system-wide broadcasts.

Leveraging hierarchies is one technique to combat these problems. Having a local level that can reduce the latency of coherence requests among a subset of cores (be it cores participating in the same virtual machine or cores accessing the same regions of memory) coupled with a global tier that can coordinate memory accesses across the entire system as needed, offers substantial opportunities for improvement in many-core architectures.

As a recent example, Virtual Hierarchies [97] has been proposed as a hierarchical protocol to improve performance for server consolidation workloads on many-core architectures. Two different hierarchies are evaluated in this work, a first level (local) directory with a second level (global) broadcast (VH-Dir-Bcast) and a first level directory with a second level directory (VH-Dir-Dir).

Virtual Tree Coherence looks at exploiting the same properties as Virtual Hierarchies for performance improvement but with different hierarchies than VH-Dir-Bcast and VH-Dir-Dir. Specifically, we examine a first level multicast with a second level directory (VTC-Mcast-Dir) and a first level multicast with a second level broadcast (VTC-Mcast-Bcast).

6.2.1 Local Coherence

For the local tier of the hierarchy, we propose using a multicast. Multicasting can achieve cache-to-cache transfer latencies similar to those of broadcast-based protocols. Additionally, by multicasting, the bandwidth requirements are significantly reduced over the broadcast protocol. At the local level, we track sharers on a region granularity and then multicast coherence requests to those sharers rather than the entire system.

6.2.2 Multicast Directory Coherence

When local information is not available, global coherence actions are required. In the case of VTC, a region miss indicates that the local processor has no coherence information about the larger region to which the requested cache line belongs. Two coherence mechanisms are considered for the global portion of the hierarchy. For the first, global coarse-grained directories maintain this sharing information.

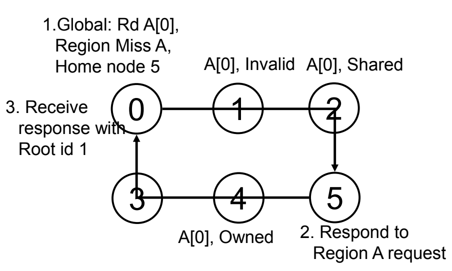
When a second level operation is required due to a region miss by a processor, the directory supplies the sharing list and the identity of the root node to the requesting processor. Once the directory response is received, the processor will send its coherence request to the root node. The root adds the requester to its region sharing list. The root node in turn performs the first-level coherence actions by sending the coherence request to all leaf nodes (current sharers). A region miss results in two levels of indirection; the first indirection goes to the directory and the second indirection goes to the root node. Cache requests that result in region misses incur significantly higher latency than those that result in a region hit and can go directly to the root node. Some of the latency associated with this two layer indirection could be removed by allowing the directory to forward the cache request directly to the root node while responding to the requester with the root information in parallel. As a second benefit of this optimization, this would also refetch the sharing information to the root node which may evicted the information. Eviction of sharing information by the root is rare; however, if the cache request arrived at the root node with no sharing information available, it would be delayed by a third indirection to reobtain that information from the directory. However, as global operations are infrequent, we do not evaluate the potential benefit of this optimization.

The coarse-grained global directories in VTC are very simple compared to the baseline directories and to other global directories in hierarchical protocols. The VTC directories are non-blocking; these directories do not order coherence requests as conventional directories do. They are simply responsible for providing sharing vector information and the identity of a region's root node. First level operations are then used to provide the ordering. This extra indirection consumes additional latency (but is not the common case) but results in a much simpler global layer. Fortunately, we find this extra indirection is infrequent.

An example of the two level coherence actions are illustrated in Figure 6.1. Steps 1-3 represent actions of the global coherence mechanism in Figure 6.1a, while steps 4-7 in Figure 6.1b represent local coherence actions. These actions will be described in more detail later in this chapter. At a high level, on a region miss, Core 0, queries the home node to obtain the identity of the root node. The directory responds; then core 0 initiates local actions by sending its read request to the root (Core 1). The sharing cores receive the multicast and Core 4 responds with the data.

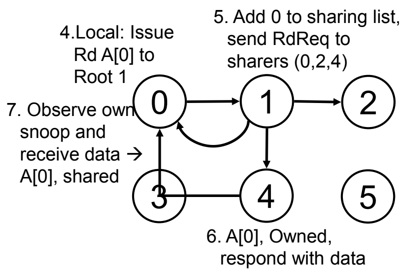
6.2.3 Multicast Broadcast Coherence

The most significant downside of using directories to maintain global coherence in the multicast-directory protocol is the on-chip storage required to maintain the directories themselves. To reduce the coherence storage requirements, we propose a second



1.Global: Rd A[0], Region Miss A, Home node 5
A[0], Invalid
A[0], Shared
3. Receive response with Root id 1
0
1
2
3
4
5
A[0], Owned
2. Respond to Region A request

(a) Steps 1-3



4.Local: Issue Rd A[0] to Root 1
5. Add 0 to sharing list, send RdReq to sharers (0,2,4)
7. Observe own snoop and receive data → A[0], shared
0
1
2
3
4
5
6. A[0], Owned, respond with data

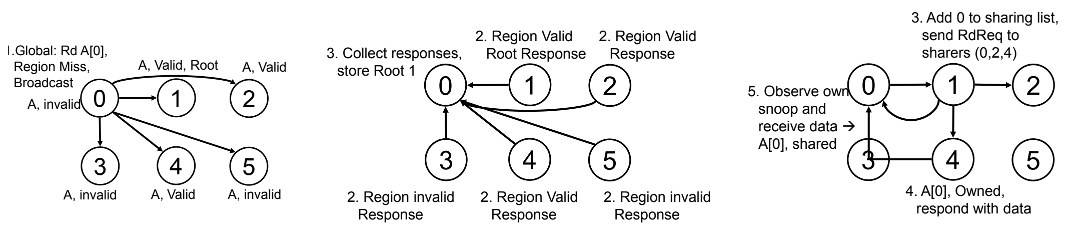
(b) Steps 4-7

Figure 6.1: Example of Hierarchical Coherence with a Global Directory Protocol.

variety of virtual tree coherence, the multicast-broadcast protocol. With multicast-broadcast, local coherence is still maintained via a multicast along a virtual tree.

Rather than rely on global coarse-grained directories for the second level of coherence, on a region miss, the core broadcasts its request to all cores to collect region information. All cores must acknowledge the region miss request; this way, the requesting core will receive information about the root node and the current sharers. If all cores send a *region invalid* response, the requesting core will become the region root as it is the only core caching any lines in the region. Multiple cores can broadcast for region information simultaneously; if there are no current sharers of the region, these simultaneous broadcasts will not find a root node. When this occurs, the root node is determined by a static assignment based on the region address. This is done in the same manner that a static home node is determined for the address. Unfortunately, this may result in a sub-optimal root selection.

When the root node observes the broadcast for region information, it can immediately forward the individual cache line request to the leaves of the region tree. This represents a slight performance optimization over the multicast-directory protocol. On a region miss, in the multicast-directory protocol, a core must wait until it receives the region information from the directory before forwarding the request to the tree root. As


1.Global: Rd A[0], Region Miss, Broadcast
A, Valid, Root
A, Valid
A, invalid
0
1
2
3
4
5
A, invalid
A, Valid
A, invalid
3. Collect responses, store Root 1
2. Region Valid Root Response
2. Region Valid Response
2. Region invalid Response
2. Region Valid Response
2. Region invalid Response
3. Add 0 to sharing list, send RdReq to sharers (0,2,4)
5. Observe own snoop and receive data → A[0], shared
4. A[0], Owned, respond with data

(a) Step 1 (b) Step 2 (c) Steps 3-5

Figure 6.2: Example of Hierarchical Coherence with a Global Broadcast Protocol.

such, it experiences two indirections to order its request.

An example of the hierarchical protocol using a global broadcast protocol is depicted in Figure 6.2. Steps in Figure 6.2a and b represent global coherence actions, while the steps in Figure 6.2c represent local coherence actions. On a region miss, Core 0 broadcasts to all cores to obtain region information. All cores must send *region valid, region invalid* responses; additionally Core 1 includes its status as the root node in its *region valid* response. Core 1 initiates the appropriate local coherence actions which are the same as depicted in Figure 6.1b. Further details on these coherence actions are discussed in subsequent sections in this chapter.

The potential downside to using a global broadcast is the extra traffic it will inject into the network. Comparing Figure 6.1 and Figure 6.2, this extra traffic is quite apparent from the number of arrows which indicate communication. A virtual tree can be used by this request to reduce the traffic pressure and network power consumption. However, as RegionTracker should be sized to make region misses infrequent, it is unlikely that either of these differences from the multicast-directory protocol will have a significant impact on performance and power.

In the next section we discuss the ordering invariants associated with the first level multicast protocol.

6.3 Ordering Invariants

Virtual Tree Coherence provides the following ordering invariants:

- Ordering Point: Each memory region is mapped to a single ordering point so all requests to the same address will go to the same ordering point. This is achieved by assigning a single virtual tree per region, and having the root of that virtual tree serve as the ordering point. Requests are then unicast to the tree root. This is similar to the use of a directory as an ordering point.

- Sharers observe the same ordering of requests: requests multicast from the root node must arrive at leaf nodes in the same order. Logically, the tree needs to maintain the ordering of a bus: sending a request to the root node of the tree is equivalent to arbitrating for the bus. All requests sent out from the root of the tree will then be ordered with respect to each other. In other words, requests to the same virtual tree must not be reordered by the underlying physical interconnect. This is achieved by modifying VCTM to ensure that each virtual tree is tied to a single virtual channel.

- Cores caching a block must see all coherence requests to that block: a multicast must contain all current sharers. Additional non-sharing cores can be included but never fewer. This is achieved either with the second-level directory always having a complete list of sharers or a full broadcast to obtain sharing information. When a non-sharer requests a block, it must first get the sharing list from the directory and be added to the sharing list at that time. Then when the Tree Root multicasts the *new sharer's* request to all sharers, the *current* sharers will add the requester to their sharing list so that the region sharing lists at the L2 cache are kept up-to-date. Strictly speaking, only the root node needs to maintain the sharing list and non-root sharers need to maintain the identity of the root node. In Section 6.9, we examine the storage overhead of having all sharers maintain the sharing list versus just having the root maintain the sharing list.

- Write serialization: Ordering through the root node serializes all writes to the same address region. Requests to the same virtual tree maintain a total order in the network. Since write order is maintained from the ordering point to the leaf nodes, invalidation acknowledgments are not required
- Write propagation: A write can complete once it sees its ordered request returned from the root node; this guarantees that any subsequent request to that cache block by any processor will receive the new value written. It is essential that all cores caching the region be included in the virtual tree; stale values are invalidated when the root forwards the write request to all leaf nodes (sharers).

6.3.1 Acknowledgments

Broadcasting can be performed on arbitrary (e.g. unordered topologies); AMD's Opteron protocol is one example of such a system [30]. Requests are sent to the ordering point (a memory controller) and then broadcast to all nodes; to maintain a total order, the ordering point then blocks on the address so that subsequent requests for this address do not race with the outstanding requests. Acknowledgment messages are then sent from all processors to the requester; once the requester has collected all acknowledgments and the data it sends a completion message to the ordering point. Once the completion message is received, the ordering point unblocks and can process the next buffered request. The ordering point must block since the interconnect provides no ordering guarantees.

In contrast, VTC does not require the interconnect to block or wait for a completion acknowledgment since we maintain a total order within a virtual tree in the network. Conceptually, multicasts to the same tree can be numbered (when they are forwarded by the root node) so that each destinations receives multicasts in an increasing order. They cannot be *renumbered* between when they pass through the root and when they reach any leaf node. A logical total order is maintained among multicasts to the same region; they do not need to arrive and be processed at leaf caches at the same physical time but in the same logical order. If we assume that memory is encompassed by a single region, this would be correct; however, regions are much smaller and we consider ordering across regions with limited acknowledgments in the next section.

6.4 Enforcing Consistency Models on VTC

As discussed in the previous section, coherence ordering is maintained via tree order; however, this is not sufficient to maintain consistency across region boundaries. This section will be devoted to a discussion of ordering relationships maintained between multiple regions.

Before we discuss the enforcement of various consistency models in VTC, we first define a new communication operation: *tree fence*. A *tree fence* specifies a region address and is sent to the tree root associated with that region address; the tree root then forwards the *tree fence* to all leaves of the tree for that region. The root also supplies the requestor with the number of sharers so it knows how many acknowledgments to wait for. When each leaf receives the *tree fence* it responds to the requesting processor with an acknowledgment. Given that a total tree order is maintained as explained in the previous section, previous invalidations sent on the tree must have been sunk when the cache processes and responds to the tree fence. When acknowledgments have been collected from all tree leaves, the *tree fence* is complete. All memory operations issued from the root node prior to the *tree fence* must now be complete. Next, these *tree fences* are employed to enforce different consistency models starting with weak ordering.

6.4.1 Weak Ordering

Weak ordering was first defined by Dubois, Scheurich and Briggs [39]. They define weak ordering as follows:

> In a multiprocessor system, storage accesses are weakly ordered if (1) accesses to global synchronizing variables are strongly ordered, (2) no access to a synchronizing variable is issued by a processor before all previous global accesses have been performed, and if (3) no access to global data is issued by a processor before a previous access to a synchronizing variable has been globally performed.

PowerPC, a weakly ordered consistency model enforces memory ordering with *sync* instructions. When a *sync* is encountered, all prior (older) memory operations must be complete before the *sync* can be committed. This enforces an order between memory operations (both reads and writes) before the *sync* and after the *sync*.

VTC uses the *tree fence* coherence primitive to achieve the *sync* functionality. *Tree fences* are used to determine that no prior (older) memory operations are still incomplete. The *tree fence* operation cannot be reordered on the tree with respect to older operations. Collecting acknowledgments on the *tree fence* indicates that older memory operations are complete.

A single *tree fence* is not sufficient since that *tree fence* only indicates completion of memory requests that used that tree (references to a single region). To enforce ordering between all operations (regardless of region) before and after a *sync*, regions accessed in an externally visible manner must be tracked by each cache. A region is considered accessed and externally visible if a cache miss occurs to the region or if an upgrade miss occurs to the region. When a sync is encountered in the program, one *tree fence* per accessed region must be executed and completed before the sync can be committed. Since regions encompass multiple cache lines, the number of *tree fences* will likely be significantly less than the number of cache lines accessed.

With infrequent *sync* operations, the number of accessed regions can grow large. Tracking a large number of regions incurs both storage overhead and increases the cost of the *sync* operation. To lessen this impact, we track a small number of unfenced regions in a FIFO buffer; when a new entry needs to be inserted into the buffer, a *tree fence* is executed for the oldest entry. By eagerly executing *tree fences* prior to the *sync,* only a small number of *tree fences* will need to be issued when the *sync* is encountered.

6.4.2 Stronger Consistency Models

VTC can also be adapted to stronger consistency models as discussed in this section. To explain how VTC supports stronger consistency models, we start with a baseline directory protocol. A directory protocol is a well establish coherence and consistency solution; we modify the protocol step by step to resemble the VTC protocol.

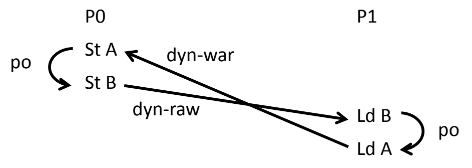
First consider a region to be composed of a single cache line (we will relax this constraint later in the discussion). In a directory protocol, store misses are ordered by collecting invalidation acknowledgments from all processors caching the block. A store miss cannot complete until all acknowledgments have been received. This has been shown to produce a sequentially consistent execution.

Next, we relax the criterion for completion of the store. Again, the store miss sends its request to the directory, the directory forwards the invalidation requests to all sharers and responds to the store miss with the number of acknowledgments that the store must receive. However, we allow the store to complete prior to collecting the invalidation acknowledgments. The store has completed but we prevent the new value from being observed by the system until all acknowledgments have been received. (Observing the new value from the store occurs when a remote processor issues a read to that cache line.)

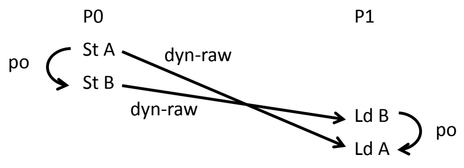
When a new value written by a store is due to be observed by the occurrence of a remote load, that remote load cannot be satisfied until all acknowledgments are collected. In this slightly modified directory-based protocol, those acknowledgments would be in-flight in the network since a store miss requires invalidation acknowledgments. The cache must now wait for those acknowledgments to finish propagating to it through the network before allowing the new value to be observed. In VTC, remote caches do not send acknowledgments on invalidation requests. As such, further action must be taken over the directory-based protocol. On a remote load (the observing action), the processor who stored the data must send out a request and collect acknowledgments on its last store to that address using a *tree fence*.

Delayed collection of acknowledgments for the store enforces the store to load order; however, we need to enforce write ordering across multiple stores. When a store is observed by a remote load, we must ensure that all previous stores executed by the processor responding to the current remote load have been observed. We must delay allowing the remote load to observe our store until we have collected acknowledgments on all previous store misses that have yet to collect acknowledgments. This write order violation is illustrated by an example in Figure 6.3a. This example uses the constraint graph model originally proposed by Landin et al. [79] and extended by Cain [19]. The edges in the constraint graph are labeled by the ordering relationship; *po* indicated the program order of a single thread. *Dyn-raw* and *dyn-war* indicate dynamic ordering relationships between threads (dynamic read after write and dynamic write after read). Executing *tree fences* on regions A and B (as shown in Figure 6.3b) when P0 receives P1's remote load of B correctly enforces store ordering.

An example of a write atomicity violation is shown in Figure 6.4a. In this example, we see that store invalidates can be delayed on the tree enroute to some but not all leaves. The resulting execution violates write atomicity since P1 reads the new value of A while P2 reads the stale value of A after reading the new value of B. In Figure 6.4b, inserting a *tree fence* when P0 observes P1's dynamic read after write to A forces the



(a) Consistency Violation: This dynamic ordering of operations violates sequential consistency (illustrated by the presence of a cycle in the constraint graph). P1 reads the new value of B but reads the stale/old value of A. This occurs because St A and St B have not been ordered on the same tree.



(b) No Consistency Violation: To break this cycle, when P0 observes P1's read request to B (dynamic read after write) it must ensure that all previous stores have completed. A *tree fence* on Region A and Region B will be executed by P0 and the remote read will be deferred until the *tree fences* have completed.

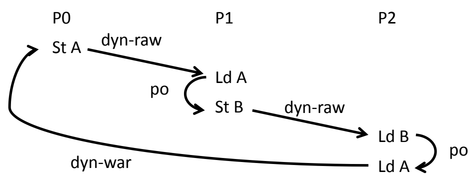
Figure 6.3: Store Ordering. Addresses A and B are in different regions.

store to A to complete with respect to all processors before P1's remote read is satisfied. Likewise, when P1 observes the dynamic read after write to B from P2, a *tree fence* will be executed on Region B. As a result, now both P1 and P2 will read the new value of A.

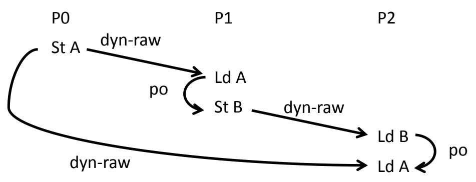
In addition to executing a *tree fence* when a remote load is going to observe the new value, the VTC protocol must also initiate a *tree fence* when a block in a dirty region is written back to memory. Without a *tree fence* prior to a writeback, a remote processor could load the written back (up to date) value from memory before other remote processors have sunk the previous store.

At first glance, the distinction between performing the store and observing the store does not provide us with any benefit. Each store miss now requires two actions: first a request is issued in the network to order the store with respect to other references on the same tree and then a second request is issued to the network to collect acknowledgments before allowing the store to be observed. This second request is the *tree fence* described above.

Given the other properties of our VTC protocol, we do not require two actions for every store miss as we will demonstrate. If we expand the size of a region to encompass



(a) Write Atomicity Violation: This dynamic ordering of operations violates write atomicity (required by stronger consistency models). Again, the violation is detected by a cycle in the constraint graph. P1 reads the new value of A but the propagation of the store is delayed along the tree enroute to P2. P2 reads the new value of B but reads the old value of A. Again this occurs when accesses to A and B are ordered on different trees (since they are not in the same region).



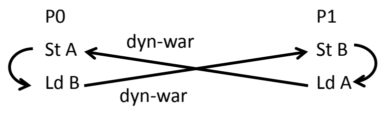
(b) No Write Atomicity Violation: To break this cycle, when P0 observes P1's read request (dynamic read after write) it must ensure that the store to A (and any previous stores have completed). A *tree fence* on Region A will be executed by P0 and the remote read of A will be deferred until the *tree fence* has completed. Likewise, P1 will initiate a *tree fence* on Region B when P2's remote read is detected.

Figure 6.4: Write Atomicity. Addresses A and B are in different regions.

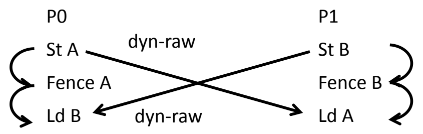
multiple cache lines, we can coalesce acknowledgment collection for multiple stores to different lines in the same region. Assume one processor performs multiple stores to the same region (including to different cache lines within that region); then when a remote load needs to observe the new value to one of those stores, a single *tree fence* will collect outstanding acknowledgments for all of the previous stores to that region.

We further lessen the potential performance impact of delaying remote reads by performing eager *tree fences*. Again, a small FIFO can be used to track unfenced regions; when a new region is stored to, the oldest region in the FIFO will be fenced to reduce the penalty imposed by a remote read. At one extreme, a FIFO size of one can be employed; when a processor transitions from storing in region A to storing in region B, it eagerly executes a *tree fence* on region A. Due to the eager fencing, when a remote read occurs there can be at most one dirty region that needs to initiate a *tree fence*.

In Figure 6.5, Dekker's algorithm is shown to illustrate how sequential consistency can be supported with VTC and *tree fences*. If both processors hold shared copies of regions A and B initially, P0 and P1 will experience upgrade misses to A and B



(a) Sequential Consistency Violation: without a mechanism to enforce store to load order a cycle will be created where both processors load stale values of A and B.



(b) Sequential Consistency: by inserting a *tree fence* between the store and load sequential consistency is maintained.

Figure 6.5: Sequential Consistency Example: Dekker's algorithm.

respectively. For a sequentially consistent execution, P0's load of B cannot retire until P0's store of A has been observed by all processors. For example, P0 could load the stale value of B and P1 could load the stale value of A due to delays on the virtual trees as illustrated in Figure 6.5a by a cycle in the constraint graph. To enforce store to load program order between different addresses a *tree fence* on previously stored regions must be executed and completed before the local load can retire. Executing *tree fences* to maintain store to load program order across different addresses would decrease the ability to coalesce multiple stores to the same region into a single *tree fence*. As a result, it seems unlikely that the use of *tree fences* would provide any benefit over simply acknowledging every store. One exception would be the execution of many consecutive stores, such as during initialization phases; these code segments would see benefit from coalescing stores into fewer *tree fences*.

As such, we assume that our system does not enforce store to load program order between different addresses. As a result, the above *tree fence* ordering constraints achieve a weaker consistency model than sequential consistency such as processor consistency or SPARC TSO consistency.

6.5 Coherence Actions

Coherence information is maintained at the processor at two granularities. The local last-level cache (L2) maintains coherence state information on a cache block granularity. Coherence information is then maintained by RegionTracker on a region granularity encompassing multiple contiguous cache blocks. On a per-region granularity we track which external cores are caching the region and the location of the root node for this region.

Coarse grain coherence tracking (CGCT) was first proposed for SMP systems; on clean-shared misses a request would go directly to the memory controller rather than waste precious bus bandwidth. Since VTC focuses on many-core architectures, we want requests to stay on-chip to save miss latency and conserve off-chip bandwidth. Additionally, VTC is implemented on a much higher bandwidth substrate than a bus-based SMP system. Therefore, clean-shared misses are multicast to other cores caching the region. Tables 6.1, 6.2 and 6.3 give an overview of the steps taken based on the region state for loads, stores and upgrades.

When there is no region state available, two levels of indirection must be performed. First a global operation is required to gather region information (Table 6.1). This global operation can be either a directory access or a system-wide broadcast; with the trade-offs previously discussed in Section 6.2. Once region information is obtained, local coherence operations are performed based on Tables 6.2 and 6.3. If valid copies of blocks in the region exist in remote caches, coherence requests must be ordered via the tree order mechanism. These steps are illustrated in Table 6.2. If no valid copies of the region exist, a subsequent load or store miss goes directly to main memory and upgrade requests are immediately satisfied by the local cache since there are no processors that need to be invalidated. These actions are shown in Table 6.3.

When a core replaces a region, it notifies the root node for that region to remove it from the sharing list to reduce false sharing (shown in Table 6.4). The root node will then construct a new tree for that region with one fewer leaf node. The other cores caching the region do not need to be notified since only the root is responsible for sending coherence requests to all sharers.

When the root receives a remote region eviction notification, it can upgrade its region to modified if no other cores are caching the region. By upgrading the region, unnecessary broadcasts are eliminated and individual lines can be immediately upgraded when the local processor issues a store request.

Table 6.1: Virtual Tree Coherence - Region Invalid: Local cache has no information about remote copies of the region

Cache Miss	**VTC Coherence Actions**
Load/Store	1. Request Region Destination Set Information from Directory 2. Directory responds with region sharing list 3. Region state set to Exclusive/Modified if sharing list is null else Region State is set to shared 4. Load/Store actions performed according to steps in Tables 6.2 and 6.3

6.5.1 Data responses

The directory maintains an owner bit for each line in the region. If set, the owner bit indicates that memory will supply the data for the coherence request; if not set, one of the cores on chip will be responsible for supplying the data. If multiple cores are caching one block, the most recent core to receive the block is designated as the next supplier. The supplier state migrates with the block to reduce the possibility that the supplier will be evicted from the cache. Somewhat similar, the SL state in the IBM Power 4 [133] indicates that a cache can source the data to a requester on the same chip. The Forward (F) state [58] has also been proposed to allow one shared copy to be responsible for responding to a cache request. The cache block in F state then

Table 6.2: Virtual Tree Coherence - Region Shared: Remote cores are caching clean/dirty lines in this region

Cache Miss	VTC Coherence Actions
Load	1a. Send Read Request to Root Node 1b. Request data from memory: speculative memory request Partially overlaps memory latency with ordering 2. Request is ordered by Root Node and forwarded to region sharers 3. Observe own request - ordered w.r.t. other requests to this address 4. Multicast sharers caching data, respond to Read Request with Data 5. If data not on chip, wait for memory response.
Store	1a. Send Store Request to Root Node 1b. Request data from memory: speculative memory request 2. Request is ordered by Root Node and forwarded to region sharers 3. Observe own request 4. Region sharers caching data, response to Store Req. w/ Data and invalidate own copy 5. Receive data from multicast sharer or wait for memory response if not cached on chip Once observed own request and received cache line, *is safe to perform store*
Upgrade	1. Sent Upgrade Request to Root Node 2. Root Node Forward Upgrade to all sharers 3. Region sharers caching data observe upgrade request and invalidate cache block 4. If Observe other store/upgrade request, another request ordered before own Invalidate cache line, now request that was ordered prior to mine will supply fresh data 5. else if Observe own request, Upgrade complete

Table 6.3: Virtual Tree Coherence - Region Modified: No remote cores are caching any lines in this region

Cache Miss	VTC Coherence Actions
Load	1. No other cores caching region - request does not need to be ordered 2. Send Read Request to Memory
Store	1. No other cores caching region - request does not need to be ordered 2. Send Store Request to Memory
Upgrade	1. Can upgrade without sending message

Table 6.4: Virtual Tree Coherence - Region Eviction

VTC Coherence Actions
1. Invalidate all copies cached in region
2a. Send Region Invalidate Acknowledgment to Directory
2b. Notify root of invalidation
3. Directory removes sharer from sharing list for region

transitions to the shared state and the new sharer is placed in F state.

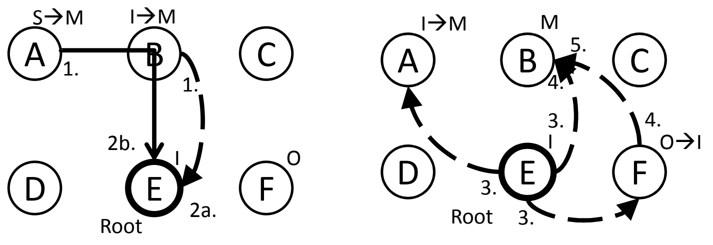
6.6 Walk-through Example

An example of Virtual Tree Coherence's handling of two racing store requests is illustrated in Figure 6.6 with the corresponding step descriptions presented in Table 6.5. In this example, Node E is the *root* of this region's tree; as such, all requests to addresses within that region must be ordered through Node E.

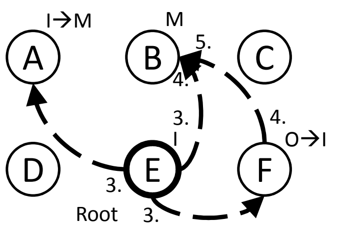
In the example, A and F are initially caching the block in question (A has the block in shared state and F has the block in owned state). Both invalidate their blocks when they see B's request from the root node. This prevents A from reading a stale copy of the block after B has written it. Invalidation acknowledgments are unnecessary with VTC for writes to complete since the virtual trees are snoop-based. This is analogous to the lack of acknowledgments in a snooping bus protocol. With VTC, a write can complete when it sees its own request returned from the root node.

6.7 Relationship between trees and regions

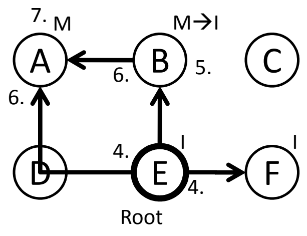
To maintain coherence, all cores caching a region must see coherence requests to that region. A single tree is maintained at the root node for that region. Remember, an address maps to only one region; so that address participates in a single tree connecting all sharers. All requests use this tree; it is not possible for a single region to map to



(a) Time 1-2



(b) Time 3-5



(c) Time 4-7

Figure 6.6: Virtual Tree Coherence Example: This example illustrates two exclusive requesters to the same address in the tree-order protocol. Dashed and curved arrows represent messages originating at or intended for B, solid and straight arrows represent messages originating at or intended for A. E is the root node for the region being accessed.

Table 6.5: Steps Corresponding to Figure 6.6

(1)	Both A and B issue requests to the root to modify a block owned by F A,B,E,F are caching this region, E is the root node for this region
(2a) (2b)	E receives B's root request and it becomes ordered. E receives A's root request, it becomes ordered after B
(3)	E forwards B's request to all leaf nodes of the region tree A sees B's request has been ordered prior to its own request A knows it will receive data from B after B has completed A must invalidate its existing copy so as not to read stale data
(4)	E forwards A's request to all leaf nodes of the region tree B sees its own request forwarded from E. B knows its request has been ordered B does not need to wait for acknowledgments F sees B's request and responds with the Data
(5)	B receives the Data response from F and completes its transition to Modified State A,B,F see A's ordered request
(6)	B invalidates its Modified copy and sends the data to A
(7)	A receives the data from B and completes its transition to Modified State

multiple trees. If this were allowed, it would mean that different cores had different sharing lists for that region and incoherence would result. Multiple regions can map to the same tree; this simply means that multiple regions are being shared by the same set of processors with the same root node. If all cores are caching a region (for example, a lock variable), a single tree will be constructed at the root node with all cores as leaf nodes.

Recall from Chapter 4 (on page 95), VCTM restricts each tree to one virtual channel throughout its traversal of the network. This virtual channel restriction is sufficient to maintain a total order within a single virtual tree, and since all requests to the same cache line will use the same virtual tree this provides a total order among requests to the same cache line. Messages bound from the same source to the same destination in an unordered network can pass each other in two ways. If these messages are assigned to different virtual channels they can be reordered within the network. The other way messages can pass each other is if they are assigned to different routes. To take care of the first case, we restrict a tree to a single virtual channel. The second case is not permissible in our system as adaptive routing is disallowed and trees follow a fixed, static route from source to destination.

Virtual channels are assigned to new trees in a round robin fashion; this distributes the virtual trees among all available virtual channels. When a new sharer is added to a tree, the tree maintains the same virtual channel mapping as the old tree. This restriction prevents messages on the new tree from passing in-flight messages on the old tree.

6.8 Coarse-Grain Regions

Virtual Tree Coherence utilizes coarse grained regions for determining the destination sets for multicasting coherence actions. Further distinguishing VTC from previous proposals that used region tracking structures to filter away unnecessary broad-

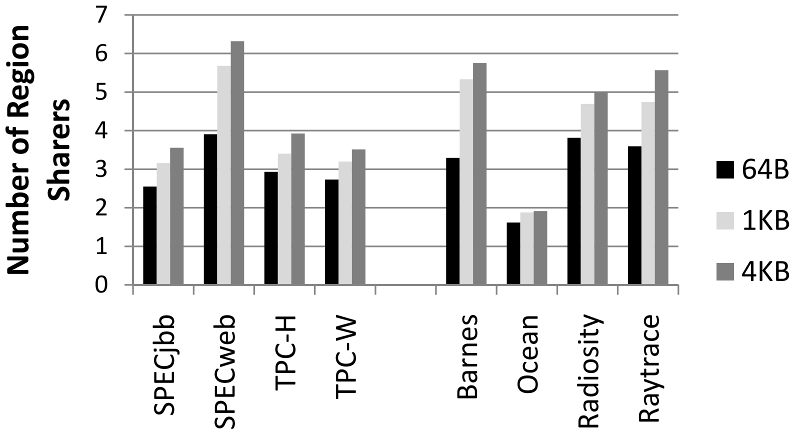


Figure 6.7: Characterization of Sharers by Region Size

casts [20, 99], perform DRAM speculation [8], and prefetching [21], Virtual Tree Coherence leverages these structures to keep track of the current set of sharers and root node of a region. Additional bits are thus added to the Region Vector Array (RVA) of RegionTracker [147] to track the current region sharers and the region root node. The low-overhead design of RegionTracker allows us to optimize for region based sharing with only a modest area increase over a conventional L2 cache design.

In Figure 6.7, we present the sharing patterns for a variety of scientific and commercial workloads based on various region sizes. Similar to what has been previously observed [15, 91], the number of sharers for a cache block (cache block size = 64B) is small. As the region size increases (16 and 64 cache lines per region), the number of sharers increases slightly; this increase is the result of false sharing. Sending multicasts to an increased number of cores will utilize additional bandwidth but at a substantial area savings for tracking this information. Previous work leveraging regions has used a 1KB region size; similarly we believe this is a good trade-off between area overhead and unnecessary multicasting of coherence requests. Further evaluation of different region sizes will be presented in Section 6.13.

The original CGCT protocol specifies three remote region states: Invalid, Clean and Dirty. For regions that are externally invalid or externally clean, a local read request can go directly to memory without snooping any other processors. From a coherence perspective, these broadcasts are unnecessary since memory has a clean copy of the block. When a region is externally dirty, a broadcast must be sent to all processors for the read request so that the local core will obtain an up-to-date copy of the line as memory may contain a stale copy. Store and upgrade requests must be broadcast to remote cores with regions that are clean or dirty so that invalidations are properly handled.

For the purposes of VTC, we seek to satisfy as many requests as possible from other on-chip caches (rather than accessing memory). As such, we do not need to distinguish between remote clean and remote dirty. Remote region state in VTC is simply tracked as remote valid and remote invalid. Multicasts are initiated on loads, stores and upgrades whenever the region is in a remote valid state. In CGCT, requests that were cached in remote clean regions could also go directly to memory without requiring a broadcast. VTC does not leverage this optimization; since VTC is designed for a many-core architecture, cache-to-cache transfers will experiences substantially lower latency than accessing main memory. As a result, VTC does multicast for clean-shared data. Since VTC leverages a high bandwidth, scalable interconnect this traffic does not cause a performance problem.

Coarse Grain Coherence Tracking removes unnecessary broadcasts to improve the scalability of broadcast-based systems. By eliminating a substantial percentage of broadcast requests, less pressure is placed on the bus which is a considerable bottleneck for scaling broadcast systems. Likewise, VTC experiences benefit from the removal of some broadcasts. Regions that are held exclusively by one processor do not need to broadcast on a cache misses and can go directly to memory. VTC also derives benefit from fast upgrades to lines that are in exclusive regions (remote invalid).

6.9 Overhead

The storage overheads of Virtual Tree Coherence stem largely from three components (1) the RegionTracker structure in each core, (2) the global coarse-grained directory cache (distributed across all cores) and (3) the VCTM overheads discussed in Chapter 4. But as RegionTracker obviates the need for L2 tags, L2 tag array storage overhead is saved. Additionally, the global coarse-grain directory caches replace the fine-grained directory caches required in the baseline protocol. The sizes of the first two components clearly depend on the size of a region R. Here we use a region size of 1KB.

The original RegionTracker proposal consumes area comparable to a conventional L2 tag array assuming 1KB regions and 8MB data array. We've added additional bits of information, specifically n bits for the multicast sharing vector, where N is the number of cores in the system and $log_2(N)$ bits to track the multicast root node. The region size per core is determined by Equation 6.1 in terms of N and region geometry (e.g. $RegionSets$, number of sets in region, $RegionSize$, size of each region, $RegionWays$, associativity of the region array). Each entry consists of the Region Tag, 3 bits of state, N bits for the multicast sharing vector, $log_2(N)$ bits to identify the root node and 4 bits of state per cache line ($validbit + way$) in the region.

$$\begin{aligned} RegionArraySize = (Tag + 3 + N + log_2(N) + (RegionSize/CacheLineSize) \times 4) \\ \times RegionSets \times RegionWays \end{aligned} \tag{6.1}$$

For example, assuming a 50 bit address, 1KB regions, 1024 region sets, 8 region ways and 16 cores in the system, we find the region array size to be 116KB. Our evaluation assumes a L2 cache of 1MB where the size of the conventional tags would be 74KB.

Only the root node needs to cache the sharing vector for a given region; as a re-

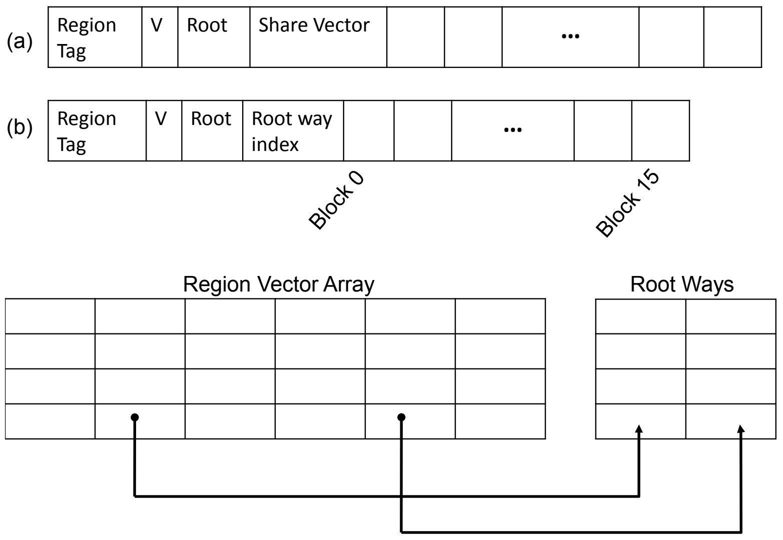


Figure 6.8: Region Vector Entry with Sharing Vector (a) and without (b) and Region Vector Array with Root Ways

sult, our baseline RegionTracker implementation wastes storage by associating a sharing vector with each region. To reduce the storage impact, we propose limiting the number of ways that can cache sharing vector information. This modified storage arrangement is depicted in Figure 6.8 and the overhead is calculated using Equation 6.2. For a 16-core system, the savings is modest, storage of the RegionTracker is reduced by 5% with four Root Ways and by 9% with two root Ways per set. However, as the sharing vector size increases with the number of nodes in the system, the savings become more pronounced for 64 cores; here, this modification saves 18% and 28% with four and two root ways respectively.

$$\begin{aligned} RegionArraySize = ((Tag + 3 + log_2(RootWays) + log_2(N) + \\ (RegionSize/CacheLineSize) \times 4) \times RegionSets \times RegionWays) + \\ (N \times RootWays \times RegionSets) \end{aligned} \tag{6.2}$$

The global coarse-grained directories (used only in VTC-Mcast-Dir) also consume area; however, compared to a conventional fine-grained directory, a coarse-grained directory has significantly greater memory reach which improves performance by reducing the directory miss rate. Computing the directory size (in bits) is done using Equation 6.3, assuming each directory entry contains N bits for the sharing vector and $log_2(N)$ bits to identify the owner node (in a conventional directory) or the root node (for VTC). The sharing vector is maintained in the directory to reduce the cost of a root eviction. If an ordering request arrives at the root and experiences a region miss, the root requests the sharing vector from the directory to reinstall its region state. If sharing vectors were not backed up by the coarse-grained directory, a root eviction would force all caches to evict that region. Directory sizes assuming 2048 directory sets and 16 directory ways are presented in Table 6.6.

$$DirectorySize = (Tag + N + log_2(N) + 3) \times DirectorySets \times DirectoryWays \tag{6.3}$$

Bit requirements for the baseline configuration (a conventional L2 + fine-grained directory cache) are compared against the VTC overheads in Table 6.6, with the parameters detailed above. We see VTC-Mcast-Dir has a 11% storage overhead over the baseline directory protocol.

However, this storage overhead can be tuned by reducing the number of entries in the coarse-grained directories, since a single entry covers $16x$ more memory than a corresponding entry in a fine-grained directory, with a region size that is $16x$ a cache line size. So, we can reduce the coarse-grained storage to trade off the storage overhead of the RegionTrackers, while still being able to cache and cover more memory than the conventional fine-grained directories. Setting the number of bits of the fine grained directory + conventional L2 tag array equal to the coarse grained directory + RegionTracker, we have enough bits for a smaller coarse-grained directory composed of 1678 sets and 16 ways. With 1678 sets and 16 ways, the small (area-equivalent) coarse-grained directory can cache 27MB of memory versus 2MB of memory that is cache-able with the fine-grained directory.

Replacing the global directory protocol with a broadcast protocol eliminates the need for directory storage. In this case, the RegionTracker has 53% more overhead than the conventional tag array but we have reduced storage 62% over the baseline directory protocol.

Table 6.6: Storage Comparison in KBits

Conventional Directory-Based Protocol	
Conventional L2 Tag Array	592
Fine-Grained Directory	1760
Virtual Tree Coherence	
RegionTracker	936
Coarse-Grained Directory	1696
Area-equivalent Coarse-Grained Directory	1448

(a) Storage Breakdown

Conventional L2 Tag Array + Fine-Grained Directory	2352
RegionTracker + Coarse-Grained Directory	2632
RegionTracker + Area-equivalent Coarse-Grained Directory	2352

(b) KBit Totals

6.10 Design Optimization

Virtual Tree Coherence reduces ordering latency of the directory by assigning the tree root node as a sharer of the cache line. As VTC is designed to improve performance through reduced cache-to-cache transfer latency, the selection of the root node is a key element in the protocol design. In this section, we explore options for selecting the root nodes.

6.10.1 Root Selection Policies

The indirection required in directory-based protocols is expensive in terms of performance. Virtual Hierarchies [97] remaps the directory home node to be a local node within the virtual machine accessing that memory address. Similarly, we migrate the ordering point for each region to one of the sharers of the region.

In order for VTC to demonstrate performance improvements over a directory protocol, the choice of root node becomes very important. Ideally, the root node should be in the sharing list for the region. It would also be beneficial to choose a sharer that is centrally located with respect to other sharers. Several policies are considered for determining the root node.

First touch. The first policy considered is to assign as the root node, the first cache to access that region. The benefit of this approach is that the root is a sharer and will be able to very quickly order its own requests as additional sharers are added to the tree. There are several downsides to this approach. If one core performs initialization operations for the application, one core will become the root of many trees irrespective of future access patterns within that application. This will also result in a network hotspot at that core and will put significant pressure on the virtual circuit trees available to that root.

Limited Migration. If a region held in the exclusive or modified state is evicted, the directory resets its root entry allowing the next cache accessing the line to become the new owner. In this manner, the root can migrate without additional complexity since migration occurs when no core is actively caching the region. When a region is evicted, and only the one sharer remains, the region can be upgraded and the root can be migrated without additional complexity. Evaluations in Section 6.13 use a baseline policy of first touch plus limited migration.

Migration to Reduce Hop Count. The root is assigned to the first core to cache any line in the region. As additional cores are added as sharers of the region, a more optimal root node can be selected. We propose selecting as the new root node the sharer with the minimal hop count to all other sharers of the region. Also, as regions are evicted, the root can again migrate to select an optimal root node for the remaining sharers. Ties among root node candidates can be broken arbitrarily or based on the number of lines in the region that each candidate is caching; the candidate with the most lines wins. The impact on ordering hop counts of this policy are considered in Section 6.13.5.

Root selection can have a significant impact on network load; policies may need to be modified with additional migration to offset network load (balance root nodes among all network nodes). With the first touch policy and the migration to reduce hop count, we observe an imbalance in the frequency of network nodes being selected as root nodes. Migrating to reduce hop count is going to drive more root nodes toward the center nodes of the network. This migration can potentially create network hotspots and place too much pressure on the VCTM mechanisms (this is illustrated in Section 6.13.5). Network hotspots can create quality of service issues within the network but we leave the exploration of this and techniques to migrate roots to offset this to future work.

When the root is migrated, cores caching the region need to be notified of the new root node. The coarse directory also needs to be updated. Additionally, in-flight requests to the old root node need to be redirected. A small root victim cache at the old root node is used redirect these in-flight requests until all sharers have received the root update message.

6.11 Scalability

In Chapter 4, we discussed a TCAM technique to improve the scalability of VCTM for larger systems. For the other added hardware required for VTC, we consider scalability issues as the system grows. First, we expect that much larger regions will provide benefit for systems running server consolidation workloads. Coarser address regions will include more false sharing but will keep virtual machines isolated from one another providing performance benefits and scalability. Region granularity could be configurable to adapt the chip to different workloads; multiple region sizes could be supported in much the same way that systems support multiple page sizes. We leave the exploration of the necessary hardware modifications for configurable region granularity for future work.

A concentrated mesh (CMESH) [11] has been proposed for large systems. A CMESH groups four cores to one single router; so a 64-core system would require a 4x4 mesh. This clustering can be applied to our storage structures as well: four cores can share a region array, last level caches and a coarse grain directory to reduce the amount of storage required. A CMESH will also reduce pressure on the VCTM hardware; a multicast can be routed to network nodes and then broadcast to the four tiles connects to each router. Essentially, to scale the system, we propose adding a layer of physical hierarchy to the virtual hierarchy that is created with VTC.

6.12 Discussion

Looking back the desirable properties of a scalable on-chip protocol, VTC targets and achieves these in the following manner:

- **Limit coherence actions to the necessary subset of nodes:** We use virtual trees to connect and order sharers, with requests multicast through these virtual trees so that coherence actions are usually limited to true sharers and not broadcast to all nodes. These virtual trees are the virtual circuit trees from VCTM and as such support efficient on-chip multicasting; VCTM trees can be mapped to onto any unordered interconnect, thus enabling scaling to many-core chips.

- **Fast cache-to-cache transfers:** The root of each virtual tree is used as the ordering point where all requests are first sent to and ordered, much like the home node of a vanilla directory protocol. However, as the root is one of the sharers and not a statically mapped home node that may be far away, the cost of indirection can be reduced considerably.

- **Limited bandwidth overhead:** By multicasting coherence only to current sharers, VTC avoids the bandwidth overhead of broadcast-based coherence protocols, while approaching their benefit of fast cache-to-cache transfers.

- **Limited storage overhead:** VTC allocates virtual trees for sharers of coarse-grained regions, rather than per cache line. The sharers of each region are tracked in local structures which are guaranteed to completely capture all current sharers of a region. Tracking on a coarse granularity allows VTC to reduce the state that needs to be maintained. Coherence state can be further reduced by relying on a global broadcast rather than global directories when local information is insufficient for coherence.

6.13 Evaluation

The following sections present the evaluation methodology, two baseline protocols and results for Virtual Tree Coherence.

6.13.1 Methodology

We use the same benchmarks to evaluate VTC as used in Chapters 3 and 5, specifically: TPC-H and TPC-W [134], SPECweb99 and SPECjbb2000 [124] and several Splash2 workloads [146]. We compare Virtual Tree Coherence against two baselines, a directory protocol and a greedy-order protocol, which are explained below. The machine model used in this portion of the evaluation is detailed in Table 6.7. Statistical

Table 6.7: Simulation Parameters

Cores	16 in-order & 64 in-order cores
Memory System	
L1 I/D Caches (lat)	32 KB 2 way set assoc. (1 cycle)
Private L2 Caches	1 MB (16 MB total) 8-way set assoc. (6 cycles), 64 Byte lines
RegionTracker (associated with each L2)	1024 sets, 8 ways, 1KB regions
Memory Latency	500 cycles
Interconnect	
Packet Switched Mesh	3 Pipeline Stages 8 VCs w/4 Buffers per VC
VCTM	64 Trees per source node (1024 total)

simulation is used to quantify overall performance with 95% confidence intervals [9].

In addition to the single server workloads, we have configured our simulation environment to support server consolidation workloads [45, 97] for up to 64 cores. For the server consolidation workloads, we create homogeneous combinations of each of the commercial workloads listed in Table 3.2; e.g. we run 4 copies of SPECjbb to create a 64-core workload. Each virtual machine is scheduled to maintain affinity among the threads of its workload.

For the server consolidation workloads, applications are isolated from one another through virtual machines; each virtual machine runs a private copy of the AIX 4.3.1 operating system. Each virtual machine is statically assigned its own portion of physical memory and has a completely private address space; no data is shared across virtual machines.

6.13.2 Baselines

Two protocol baselines are used to compare against VTC. The first baseline, a directory protocol, was also used for the evaluation of Circuit-Switched Coherence. The second, a greedy-order protocol provides another comparison point for Virtual Tree

Coherence.

6.13.2.1 BaselineI: Directory

Directory protocols are commonly used in many-core architecture proposals due to their perceived scalability. Our first baseline is a standard directory protocol modeled after the SGI Origin protocol [80]. This protocol suffers from the latency overheads associated with an indirection through a directory on each cache miss. Additionally, to make this protocol amenable to a many-core architecture, directory caches are used. A directory cache maintains recently accessed directory entries from main memory on chip. Misses to these directory caches suffer the latency overhead of going off chip to memory and can be quite frequent for server workloads and even more frequent for server consolidation workloads. For a set of commercial workloads, miss rates between 22% and 74% have been observed [94].

6.13.2.2 BaselineII: Greedy Order Region Coherence

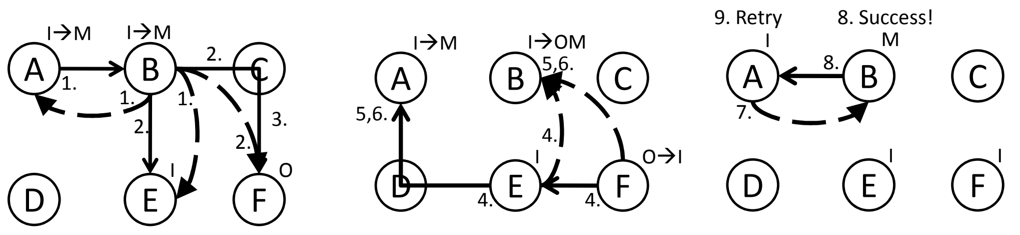
Greedy order protocols have been proposed for ring interconnects [12, 96, 120] and overlaid atop unordered interconnection networks [127]. Here, as a second baseline for comparison against VTC, we map and optimize a greedy order protocol that can leverage the region tracking structures and throughput benefits of the VCTM network that VTC also uses. The key difference is that VTC relies on the virtual trees for ordering, while greedy order does not.

Ordering. In greedy order protocols, requests are ordered by the current owner. Requests are live as soon as they leave the requester; in other words, they do not need to arbitrate for a shared resource such as a bus or pass through a central ordering point such as a directory. A request becomes ordered when it reaches the owner of the cache block (another cache or memory). In the common case when no race occurs, these requests are serviced very quickly because they do not require the additional latency of an indirection through a directory. In [96], requests complete after they have observed the combined snoop response that trails the request on a ring. Based on the combined response, a request is either successful or must retry. Mapping and extending Greedy-Order protocols onto an unordered interconnect such as a mesh requires acknowledgments to be collected from all relevant processors. However, extending to regions rather than individual cache lines, reduces the number of acknowledgments that have to be collected from all cores to only cores that are caching that particular address region. Therefore, the number of acknowledgments that are expected is derived from the sharing vector in the RegionTracker. The collection of acknowledgments is similar to the combined response on the ring but requires more messages. This is also similar to the process of collecting invalidates in a directory protocol. The owner sends an *owner acknowledgment* signifying the transfer of ownership. If no *owner acknowledgment* is received, then another request was ordered before this one and this request must retry. Greedy-Order can be applied in a broadcast fashion as well, where no sharers are tracked and acknowledgments are gathered from all processors.

Example. An example of Greedy-Order is depicted in Figure 6.9 with the corresponding step descriptions presented in Table 6.8.

6.13.3 Performance Results

In the following sections, we present quantitative performance results comparing Virtual Tree Coherence against our two baselines. Additionally, we present a comparison between Virtual Tree Coherence-Multicast-Directory (VTC-Mcast-Dir), Virtual Tree Coherence-Multicast-Broadcast (VTC-Mcast-Bcast) and a Broadcast using virtual trees (VTC-Bcast). With VTC-Bcast, a virtual tree connects all nodes; however, regions are used to designate and distribute root nodes around the network so that there is not a single root bottleneck. Significant network bandwidth and dynamic power can be saved by limiting coherence actions to multicasts instead of broadcasts, as we do in



(a) Time 1-3 (b) Time 4-6 (c) Time 7-8

Figure 6.9: Greedy Order Example: This example illustrates two exclusive requesters in the greedy-order region protocol. Dashed and curved arrows represent messages originating at or intended for B, Solid and straight arrows represent messages originating at or intended for A. Time is progressing from left to right in the figure.

Table 6.8: Steps Corresponding to Figure 6.9

(1)	Both A and B issue requests to all processors caching region to modify a block owned by F. A,B,E,F are caching this region
(2)	A's request reaches B and is replicated and forwarded to E and F B's request reaches E and F
(3)	A's request reaches F B's request reached F (Owner) first, so B's request will win
(4)	E and F respond with acknowledgments to A and B's request. B gets an owner acknowledgment from F and F transitions from owned to invalid
(5)	A receives E's acknowledgment, B receives E's acknowledgment
(6)	A receives F's acknowledgment, B receives F's owner acknowledgment
(7)	B knows its Modified request will succeed, it sends a negative acknowledgment to A
(8)	A receives a negative acknowledgment from B, it has now collected all acknowledgments and did not succeed so it will acknowledge B's request and it must retry its own request
(9)	B collects its final acknowledgment from A and successfully transitions to Modified State.

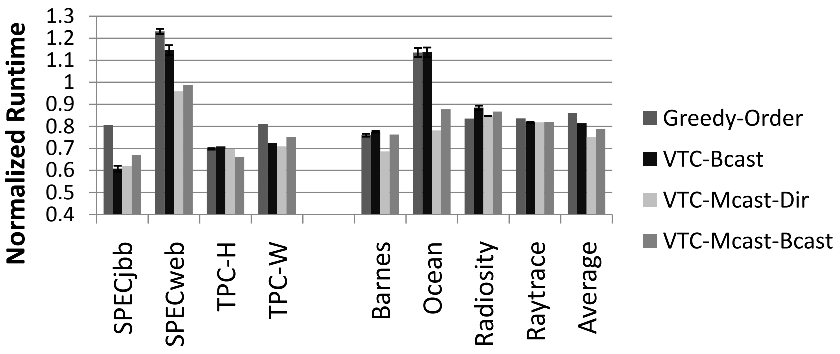


Figure 6.10: Performance Comparison for VTC against various coherence protocols with 16 cores. Results are normalized to the baseline directory protocol.

VTC-Mcast-Dir and VTC-Mcast-Bcast.

In Figure 6.10, results are presented for Greedy-Order Multicasting, Virtual Tree Coherence Broadcast and Virtual Tree Multicast Coherence with both a global directory (VTC-Mcast-Dir) and a global broadcast (VTC-Mcast-Bcast) protocol layer. All results are normalized to BaselineI: Directory Coherence. Overall, significant performance gains are achieved by VTC-Bcast and VTC-Mcast-Dir, up to 39% and 38% respectively (19% and 25% on average) when compared to the directory protocol. Virtual Tree Coherence (VTC-Mcast-Dir) outperforms Greedy-Order by up to 31% with an average improvement of 11%. In a 4x4 system, the differences between VTC-Bcast and VTC-Mcast-Dir are minor; however, as systems scale, the difference between these two becomes much more pronounced with favorable results for VTC-Mcast-Dir (Figure 6.11.)

In a couple of instances, notably, TPC-H and SPECjbb, VTC-Bcast outperforms VTC-Mcast-Dir. With larger memory footprints and irregular access patterns, these workloads experience much larger region miss rates which incur additional overhead to re-fetch region information from the second-level directory. For SPECjbb and TPC-H, 21% and 18% of L2 misses also result in region misses; the rest of the workloads

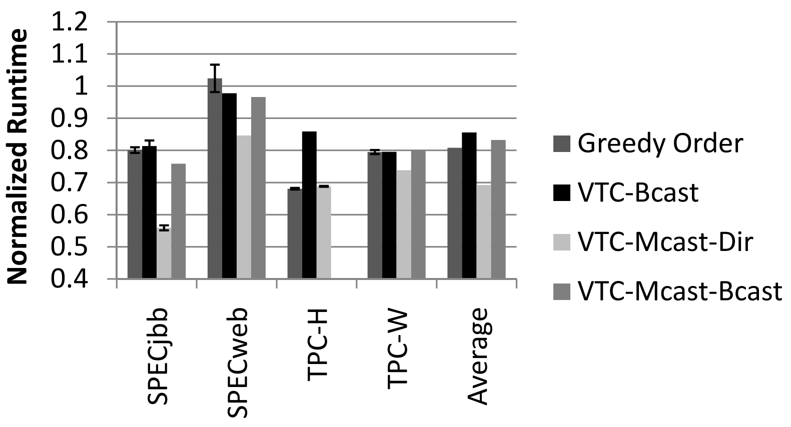


Figure 6.11: Performance Comparison for VTC against various coherence protocols with 64 cores

experience region miss rates of less than 10%. SPECweb sees only a small performance improvement from VTC-Mcast-Dir (5%); SPECweb sees the sharpest increase in traffic which limits the performance improvement. Techniques to improve the region hit rate and lower the number of false-sharers will lead to further performance improvements for SPECweb.

In Figure 6.11, the difference between VTC-Bcast and VTC-Mcast-Dir becomes more pronounced, VTC-Mcast-Dir outperforms VTC-Bcast by an average of 11% and up to 16%. With a 64-core system, broadcasting becomes more expensive (both in performance and power). VTC-Mcast-Dir provides more isolation for the virtual machines; coherence requests are only sent to nodes involved in sharing. VTC-Bcast unnecessarily broadcasts to all nodes (across multiple virtual machines).

6.13.3.1 Impact of Global Level Coherence

Figure 6.10 also compares the VTC-Mcast-Dir and VTC-Mcast-Bcast protocols. VTC-Mcast-Bcast offers similar performance with substantially lower overheads than the VTC-Mcast-Dir protocol by eliminating the coarse-grained directories, instead broadcasting for region information on a region miss. Some performance is lost with

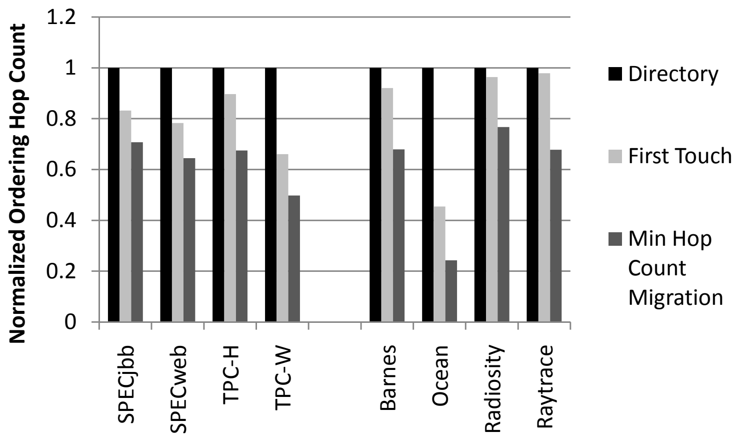
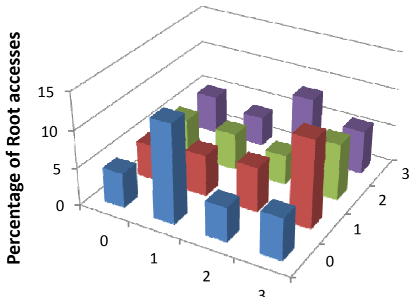


Figure 6.12: Hop Count Compared to Directory Baseline

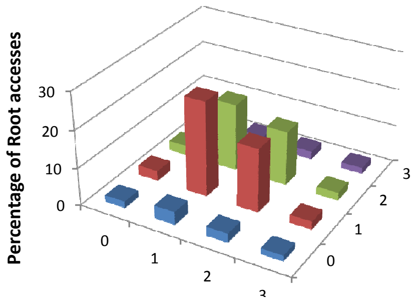
the longer latency to obtain root and sharing information for regions with VTC-Mcast-Bcast. VTC-Mcast-Dir improves performance by an average of 25% over the baseline directory protocol; that improvement shrinks slightly to 21% with a global broadcast protocol. This small loss in performance comes at significant area savings.

6.13.4 With and Without VCTM

In Figures 6.10 and 6.11, Greedy-Order, VTC-Bcast, VTC-Mcast-Dir and VTC-Mcast-Bcast, all leverage the benefits of VCTM. On a non-VCTM packet-switched mesh, the performance of VTC-Mcast-Dir degrades by an average of 15%. Without VCTM, no ordering guarantees can be made for VTC-Bcast, VTC-Mcast-Dir and VTC-Mcast-Bcast. VCTM maintains a total order for multicast requests within the network and is also essential for performance improvements and scalability. Without VCTM, in a 64-core system, VTC-Bcast sends out 63 coherence packets for each cache miss which would likely saturate the network. Greedy-Order, which places significant pressure on the interconnection network due to retries sees performance degradations up to 48% when VCTM is removed; Greedy-Order does not rely on VCTM for ordering.



(a) First Touch



(b) Minimum Hop Count Migration

Figure 6.13: Root Hotspot for SPECjbb

6.13.5 Tree Root Selection Policies

Both VTC and the directory baseline require the indirection to the ordering point for coherence requests. VTC derives performance benefit in part from reducing the cost of these indirections. With VTC-Mcast-Dir, the hop count to the ordering point is reduced by 19% for 16 cores and 50% for 64 cores since the root node is a region sharer. Furthermore, on average, 4.2*x* more coherence requests are ordered in zero hops with VTC-Mcast-Dir than with the directory protocol.

In Section 6.10.1, we discuss migration of the root node to achieve minimal ordering hop counts. Despite impressive ordering hop count reductions (shown in Figure 6.12), migration of the root does not result in noticeable performance improvements over the first touch with no migration policy. In some cases (e.g. SPECjbb), performance degradation is observed. This migration optimization concentrates 72% of root accesses at the four nodes in the center of the 4x4 mesh network resulting in network congestion and additional pressure on the VCTM entries for these four nodes. This 72% of root accesses is compared to just 20% for the first touch policy with limited migration policy. This root distribution is depicted in Figure 6.13. It is clear from this analysis that network congestion and hot spots need to be factored into the root selection policy.

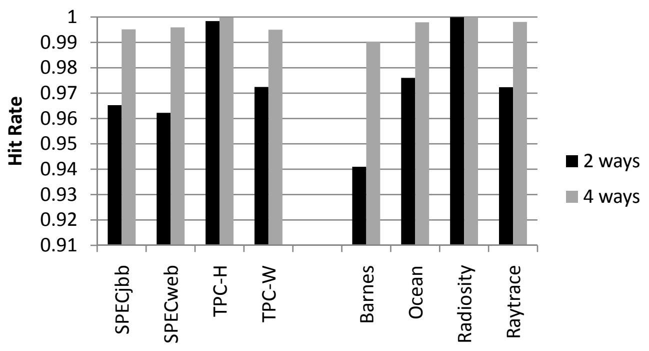


Figure 6.14: Hit Rates for Limited Caching for Sharing Vectors

6.13.6 Limited Caching of Sharing Vector

In Section 6.9, we propose limiting the number of region vector entries that can cache the sharing vector to reduce the overhead of the RegionTracker structures. This reduces the RegionTracker storage by 18% and 28% for four and two root ways in a 64-core system. The downside of limiting the number of entries per set that can cache sharing vectors, is that an ordering request to the root node might experience a sharing vector miss. Figure 6.14 shows the hit rates for caching sharing vectors for two and four roots per set. An average sharing vector hit rate of 97% is observed with two root ways per set; 3% of ordering messages would require an indirection to the global directory to reobtain the sharing information. In the case of a global broadcast layer, a broadcast would be initiated to recollect the sharing vector information.

6.13.7 Activity/Power

In addition to the performance of VTC, we consider the power (in terms of network activity) that each protocol consumes. Figure 6.15 shows the network activity (based on link traversals by flits) for each coherence protocol relative to the directory protocol. As expected, directory coherence has the lowest interconnect traffic since

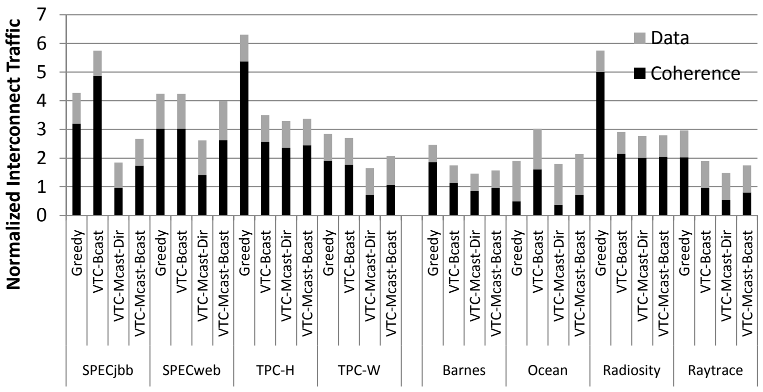


Figure 6.15: Interconnect Traffic Comparison Normalized to Directory for 16 cores (measured in link traversals by flits)

nearly all of the messages are unicasts (invalidation requests from the directory are the exception). Data traffic is similar for each protocol; the main difference lies in the required coherence traffic. Greedy-Order requires the most interconnection network bandwidth of all the protocols; on average 3.8 times the number of link traversals as the directory protocol. VTC-Mcast-Dir consumes less network bandwidth than VTC-Bcast, 35% activity reduction on average, with the most significant reduction of 68% for SPECjbb and 40% for Ocean. A large fraction of Ocean's references are memory misses; VTC-Mcast-Dir will optimize and go directly to memory if no other cores are caching the region. These memory misses are broadcast to all cores in VTC-Bcast resulting in a bandwidth spike when compared with VTC-Mcast-Dir and Greedy-Order. VTC-Mcast-Bcast requires slightly more traffic than VTC-Mcast-Dir resulting in only a 21% activity reduction over VTC-Bcast. The interconnect traffic difference between VTC-Bcast and VTC-Mcast-Dir grows from 35% with 16 cores to 68% with 64 cores as shown in Figure 6.16. VTC-Mcast-Dir requires 1.6x more traffic than a directory protocol for 64 cores.

In some cases, the traffic requirements of VTC-Bcast, VTC-Mcast-Dir, VTC-Mcast-Bcast are very similar. We attribute this to the robustness of the underlying

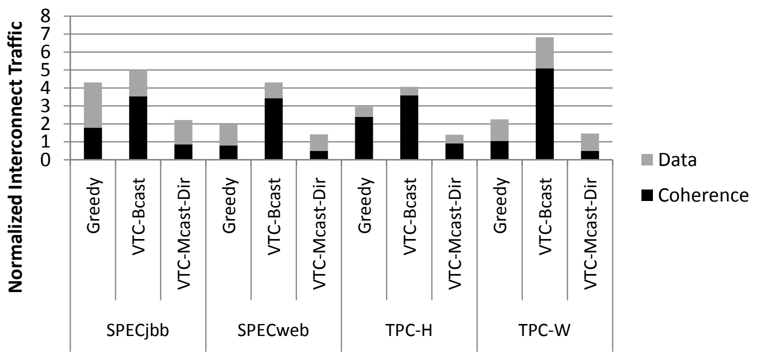


Figure 6.16: Interconnect Traffic Comparison Normalized to Directory for 64 cores (measured in link traversals by flits)

VCTM network. If you refer back to Figure 4.26 on page 116, the TokenB protocol (a broadcast-based protocol) sees a 50% reduction in interconnect bandwith with introduction of VCTM. Removing the VCTM interconnect network would drive up the interconnection utilization significantly for VTC-Bcast (as well as break one of our ordering invariants).

Network activity is only part of the story for power consumption differences in VTC-Bcast, VTC-Mcast-Dir and VTC-Mcast-Bcast. VTC-Bcast will consume significantly more power since all caches will snoop all coherence requests; VTC-Mcast-Dir and VTC-Mcast-Bcast eliminate a significant fraction of cache accesses required with VTC-Bcast (VTC-Mcast-Bcast does require more cache accesses on global coherence actions than VTC-Mcast-Dir). The retries in Greedy-Order also increase the number of cache accesses required.

6.13.8 Interaction between VCTM and VTC

The interplay between region size and the efficiency of a VCTM network is an interesting motivation for the need to co-design the coherence protocol and the interconnect. Choosing small regions results in a much larger number of unique trees that

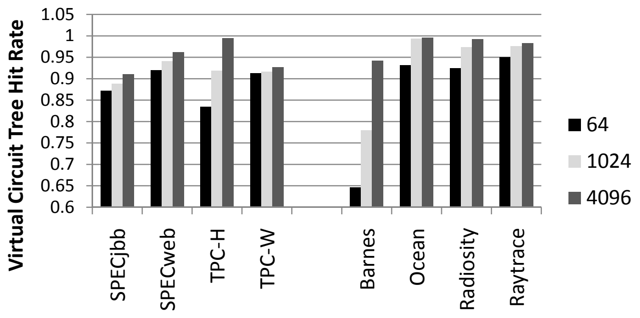


Figure 6.17: Impact of Region Size on Virtual Circuit Tree Hit Rate (16 cores)

are needed; this large number of unique trees causes the virtual circuit trees to thrash in the network. The virtual circuit tree hit rate in the interconnect ranges from 78% to 99% for 1KB regions; the hit rate drops to 65% to 95% for 64B regions as depicted in Figure 6.17. With a lower hit rate there are more tree setups in the network; recall from Chapter 4 in the VCTM design, tree setup requires replication of the multicast message into many unicasts resulting in a short burst of traffic during the setup phase; this impacts interconnection network latency and throughput.

As a result, 1KB regions, which suffer from a modest amount of false sharing compared to 64B regions, actually have 3% less interconnection network traffic by making better use of virtual circuit trees. With 4KB regions, performance is similar to 1KB regions, with slight degradations observed for SPECweb and Ocean.

For our 16-core system, we found that varying the region size had only a small impact on performance. In a system with constrained bandwidth, one would expect that increasing the number of multicast sharers would significantly degrade the performance of VCTM; we did not find this to be the case with VTC. The VCTM design is found to be very efficient for both broadcasting and multicasting (as demonstrated in Chapter 4). As a result, network performance is not impacted by false sharing except for the impact on tree hit rates noted above. The increase of false sharing with increased region size will affect the power consumed due to unnecessary cache snoops. We believe the impact of region size will become more pronounced as system size grows.

6.13.9 Discussion

Largely, the performance results for Virtual Tree Coherence with a local Broadcast and a local Multicast are very similar. We attribute this similarity to the robust nature of the underlying virtual circuit tree multicast network. The power consumption of the two protocols is different and the broadcast protocol results in a significant number of unnecessary cache lookups for snoops though.

6.14 Related Work

Cache coherence research has been of significant interest in both single and multi-chip multiprocessor systems. A variety of protocols have been proposed/implemented to achieve performance and scalability on both ordered and unordered interconnects. Additionally, we discuss various systems that have employed hierarchical protocols. We contrast these prior works with Virtual Tree Coherence in the following sections.

6.14.1 Ordered Interconnect

Multicast snooping and destination set prediction [15, 91] use prediction mechanisms to determine which processors will likely need to see a coherence request. In contrast, VTC determines exactly who must be included in a multicast. Extra cores might be contained in the destination set but never fewer cores than necessary. These protocols rely on a totally ordered physical interconnect for sending out multicast requests. Our design relaxes this constraint, permitting a higher performance interconnect. With the virtual channel restriction in place for VCTM, it is our belief that multicast snooping and destination set prediction could leverage the VCTM network and achieve the required total order for multicasts (the requirements for VTC are the same). Requests to the same address region use the same virtual tree and are restricted to using the same virtual channel as prior requests to the same address region. This virtual channel restriction prevent messages from becoming reordered with respect to each other in the network.

Bandwidth Adaptive Snooping [93] employs a hybrid protocol that achieves the latency of broadcasting when bandwidth is plentiful but converts to a directory style protocol when bandwidth is limited. This work relies on a totally ordered interconnect but overcomes some of the pressure that large snooping systems can place on the interconnect.

6.14.2 Unordered Interconnect

Token coherence provides the token abstraction to decouple performance from correctness [92]. Several variants of Token Coherence have been proposed including one based on broadcasts and one on directories. TokenB, the broadcast Token protocol requires more bandwidth than multicasting with Virtual Tree Coherence. Extensions have been proposed for multi-chip CMP-based systems in [95].

In-Network Coherence [40] replaces directories by embedding sharing information in tree structures within the network. These virtual trees (different from VTC's virtual circuit trees) are used to locate data on-chip. When a request is en-route to the home node, it can bump into a tree which will redirect the request to the appropriate core that is sharing the cache block. This optimization targets the directory indirection latency and can lead to fewer interconnect hops to find a valid cache line.

However, with In-Network Coherence, depending on the route taken by a request, a sharer may be nearby, but the request may miss it and still have to make its way several hops to the directory. In such scenarios, VTC will perform better since the tree root is a region sharer. Cache misses that do bump into a tree in In-Network Coherence will be satisfied more quickly than requests that have to travel a significant distance to the root node in VTC. It should be noted though that in VTC, it is always the case that the root node is a sharer of the region, which may be closer than the statically-mapped home node in In-Network Coherence. Also, VTC utilizes coarse-grained tracking which requires less storage overhead than the per-line, per-hop storage needed in In-Network Coherence.

PATCH [111] reduces cache-to-cache transfer latency over a directory protocol by leveraging direct requests. It presents a more elegant solution than persistent requests which require a full broadcast in Token Coherence [90] through the token tenure mechanism. Token tenure causes requests that have not been activated to return tokens to the home node after a bounded time. They leverage the prediction mechanisms from destination set prediction [91] to issue direct requests as performance hints; however, they rely on a best-effort multicast (predictive requests are given low priority) rather than the guaranteed throughput of VCTM. Stale direct requests can be dropped in the network; in their evaluation, they assume that requests in the network longer than 100 cycles are stale. A best-effort, low-priority multicast prevents these predictive requests from interfering with critical traffic; however, the network resources (specifically buffers) to support such a long time-out would be very costly but is not considered by their work. VTC leverages VCTM to keep the bandwidth demands low and allows ordered snoop delivery with low latency in our protocol.

UnCorq [127] broadcasts coherence requests on an unordered interconnect (e.g. a mesh) and then orders snoop responses via a logical ring. Similarly, we utilize the ordering implied by logical trees to maintain coherence; however, we order requests via virtual trees rather than responses. And additional difference is the use of multicasting instead of the full broadcast used by UnCorq. Greedy-Order bears some similarity to UnCorq; requests are sent to sharers quickly without regard to order. UnCorq then orders response via a logical ring, whereas Greedy-Order uses the owner to order requests.

Embedded ring protocols [125] leverage ring ordering properties but with a higher bandwidth substrate (e.g. a mesh). With such protocols, modifications to the network must be made to ensure that coherence order is maintained. These protocols require that a FIFO order on network links is maintained among coherence requests (or responses, in the case of UnCorq) to the same address. This FIFO ordering could be enforced through virtual channel restrictions or some other mechanisms; the router details are missing from this work.

Trees have also been leveraged in previous proposals to build more scalable directories for large distributed shared memory machines [22, 104]. These trees are used to reduce storage overhead but the directories still serve as the ordering points.

6.14.3 Hierarchical Coherence

Virtual Hierarchies [97] propose cache coherence variations targeting server consolidation workloads running on chip multiprocessors. One proposal utilizes two levels of directories to provide fast local coherence and correct (and substantially slower) global coherence (with the observation that global coherence is rare). The other proposal also utilizes local directories for fast coherence within a server application, and a backing broadcast protocol for global coherence. The alternative of utilizing a local broadcast backed by a global directory protocol is mentioned but not explored. The VTC-Mcast-Dir coherence mechanism in this work is similar to the latter case. Since VTC examines a different hierarchy than what is proposed in Virtual Hierarchies, we do not provide a direct quantitative comparison; however, we do highlight some features that further distinguish VTC.

Some of the performance improvements of Virtual Hierarchies are predicated on the ability of the scheduler to provide locality between communicating and sharing cores or threads. Virtual Hierarchies will work when locality is not preserved; however, we believe that the coupling of multicast coherence with a fast multicast substrate (VCTM) results in superior performance. Virtual Tree Coherence will support flexible placement and scheduling of communicating threads, whereas the benefits achieved with Virtual

Hierarchies are predicated on physical proximity.

The actions of the second level directories in Virtual Tree Coherence are very simple unlike directories in the virtual hierarchies protocols. Virtual Hierarchies requires a very large number of states and transitions in the coherence protocol to accommodate two levels of directories; this is not the case for Virtual Tree Coherence. The directories contain the sharing list of each region that is cached anywhere on-chip, the identity of the tree root for that region, and whether a block is owned on-chip or if memory is the owner.

Many traditional multiprocessor systems have been built by aggregating small bus-based SMP nodes [57, 80, 81, 86]. These systems employ first level broadcast protocols (on-node) and directory protocols for node-to-node coherence. The key difference between VTC and these systems is the dynamic nature of the snooping layer of the hierarchy for VTC. Also, with these physically hierarchical systems composed of multiple SMPs, the cost of utilizing the directory level of the coherence protocols is much more expensive than local actions that remain on-chip. The Sun Wildfire [57] provides a hierarchical affinity scheduler to optimize the snooping level of coherence; maintaining physical locality is important given the cost of migration to another board. With VTC, a node can be involved in virtual multicast layer with various sets of nodes rather than a fixed set of processors.

Hierarchical coherence schemes that employ multiple levels of snooping protocols such as the Encore Gigamax [144] have also been explored. In this system, the levels of hierarchy are joined to form a tree, with each node having inclusive knowledge of the cache blocks in all processors below it. Coherence requests are filtered so that lower levels not caching blocks need not experience extra coherence traffic.

The Hierarchical DDM system [56] was a cache-only memory architecture that employed snooping coherence for a set of processors, caches and attraction memories and built a physical hierarchy of directories to maintain coherence. These directories form a tree and racing requests are ordered at the lowest level directory that they have in common. In this system, the directories maintain information about data that is stored in attraction memories below it; the processors in this hierarchy are fixed whereas with VTC, the set of processors connected via a virtual snooping tree is dynamic. As with VTC, as requests traverse the interconnect toward the top node (root in our case), bandwidth can become a bottleneck.

Pruning caches have been proposed by Scott and Goodman [116] as a means to filter invalidation requests and combine acknowledgments at various levels of hierarchy. They note that these messages can cause interconnect and performance bottlenecks so they use pruning caches to only propagate necessary requests; the use of pruning caches improves the scalability of both the protocol and the interconnect for large multiprocessor systems. The VCTM router targets this same problem by preventing redundant messages from traversing the interconnect.

6.14.4 Network Designs Cognizant of Cache Coherence

In addition to relevant work in the domain of cache coherence, VTC also examines requirements that are placed on the interconnect to maintain correctness. These requirements have also been examined in prior work. The Rotary Router [1,2] provides mechanisms to maintain the ordering of coherence requests in the network and prevent coherence deadlock within the interconnect. Buffer resource allocation is divided up between dependent messages to prevent dependent messages from deadlocking in the network due to unavailable resources. In-order delivery is guaranteed by forcing ordered messages to traverse the same path.

Another common solution to avoiding deadlock and preventing message reordering is to dedicate a separate virtual network to each class of coherence message (e.g. requests vs. responses), where each virtual network has distinct virtual channels. This technique is employed by the Alpha 21364 [100]. VTC forces the underlying network to deliver coherence requests in order by restricting a virtual tree to use a single virtual channel.

Isotach networks [112] are a class of networks that provide very stringent guarantees on message ordering and timing; this includes providing totally ordered multicasting. These networks can enforce atomicity and sequential consistency for cache coherence protocols. Isotach networks provide ordering properties independent of any particular topology.

6.15 Conclusion

This chapter proposes Virtual Tree Coherence which multicasts requests to sharers to lower cache-to-cache transfer latency. The indirection to the directory is replaced with an indirection to the tree root to order requests. The tree root is selected based on sharing rather than statically assigned; as a result, the latency to order requests may be reduced. The bandwidth of broadcasting or multicasting on-chip is prohibitive as we move to large systems; to combat this problem, we couple VTC with VCTM to reduce the bandwidth overheads and provide a high performance interconnect substrate. To reduce the storage requirement, the global directories can be eliminated and replaced with a global broadcasting mechanism. This enhances the likely scalability of the VTC protocol. VTC improves not only read-shared misses (as is the case with Circuit-Switched Coherence) but also improves stores and upgrades resulting in significantly higher performance.

Both Circuit-Switched Coherence and Virtual Tree Coherence aim to improve the performance of on-chip cache misses. Their ability to do so is tightly coupled with the optimized interconnection network architectures described in Chapters 3 and 4. Circuit-Switched Coherence relies on the hybrid circuit-switched network for performance improvements; Virtual Tree Coherence relies on the virtual circuit tree multicasting interconnect for both performance and correctness.

Interconnection network architectures offer substantial opportunity for performance improvement. However, when coupled with coherence protocols that leverage the particular properties of the network, further opportunities are available as evidenced in this dissertation with both Circuit-Switched Coherence and Virtual Tree Coherence.

Chapter 7

Conclusion

In this chapter, we review the primary findings of this dissertation (Section 7.1). Next we present future directions for interconnect and coherence protocol design based on the work done in this dissertation (Section 7.2).

7.1 Contributions and Summary of Results

This dissertation explores the efficacy of interconnection network and cache coherence co-design. We proposed two designs that optimize for different communication behavior while maintaining a general and flexible substrate that works well under a variety of communication demands. The properties of the two interconnection networks are then leveraged in the design of two coherence protocols.

This dissertation addresses two problems in the area of existing on-chip network architectures. Router latency contributes significant overheads to communication as designers move from dedicated wires to networks. Network latency has a large impact on system performance. Furthermore, we observe frequent pair-wise sharing between cores. Here, we contribute the Hybrid Circuit-Switched router architecture that removes a significant portion of this latency for frequent pair-wise sharers.

Our second contribution is motivated by the observations that the multiple unicast approach to multicasting results in significant throughput degradations in on-chip networks. Furthermore, we present a comprehensive characterization of a variety of scenarios that could benefit from on-chip multicast support. To address this problem, we propose VCTM, low-cost multicast router solution that both reduces power consumption and improves network performance and throughput.

With the two optimized interconnection networks described, we then go on to design coherence protocols that leverage the characteristics of these networks. The third contribution of this dissertation leverages the pair-wise circuits to improve coherence latency. Circuit-Switched Coherence removes the directory indirection from the critical path. The directory indirection interrupts the pair-wise relationship between cores. By decoupling the data request from the ordering request, we are able to take further advantage of our network circuits for performance improvements.

A significant challenge facing coherence protocol design is the tension between the need for an ordered interconnect to simplify coherence and the need for an unordered interconnect to provide scalable communication. To this area, we contribute a high bandwidth ordering substrate by overlaying an ordering constraint on the trees of the VCTM network in order to build a snooping-based multicast coherence protocol. Virtual Tree Coherence multicasts requests to sharers to lower cache-to-cache transfer latency; requests are ordered at a tree root rather than a directory home node.

7.2 Future Research Directions

The many-core evolution will continue to present researchers with opportunities and challenges for many years to come. Throughout the course of this research, I have explored various facets of both the interconnection network and the cache coherence protocol for these many-core architectures. These are rich fields of research with many areas yet to be explored. Specifically, we focus future research on scalability of both the interconnect and cache coherence protocols, the potential for network hotspots due to coherence traffic, and the further exploration of multicast and reduction network designs.

7.2.1 Scalability of the proposals in this dissertation

As larger scale systems are built and parallel applications emerge to run on them, communication demands will continue to be a central component to providing performance and scalability in these systems. The mechanisms proposed in this dissertation will hold for a variety of communication scenarios but need to be evaluated against these new applications to highlight further opportunities for improvement. Scalable and tractable simulation infrastructures are of paramount importance to drive this research forward; many of the results here are presented for only 16-core systems which is a limitation of this research.

As mentioned in Chapters 3 and 4, both interconnect proposals are agnostic to topology choice. These designs can be applied to more scalable topologies to improve their scalability to the first order. VCTM currently relies on tree reuse to achieve benefit; the number of possible trees will grow as 2^n for systems with n nodes. Exploring multicast routing techniques that do not rely on hardware tables is necessary to support a large number of concurrent trees.

7.2.2 How to deal with network hotspots?

In both Circuit-Switched Coherence and Virtual Tree Coherence, there is the potential to develop network hot spots. All cores could have the same pair-wise sharer or all regions could have the same tree root node; more research is needed in routing and resource allocation to deal with these hot spots as they arise. Selecting root nodes that result in the smallest hop count forces many root nodes to the cores at the center of the chip resulting in hot spots. Despite lower hop counts, these coherence ordering actions experience longer latency to access the root and be forwarded to sharers in turn. Root migration to reduce hot spots may improve overall network performance and reduce ordering latency at the expense of additional complexity in the protocol.

Quality of service mechanisms can be explored to improve latency and throughput to these hotspots.

7.2.3 Further design of multicast and reduction networks

Through this dissertation research, support for on-chip multicasting has proven quite promising in terms of both performance and power. In Chapter 4, we discuss a continuum of approaches to solving the multicast problem ranging from multiple unicasts to a single multicast. Overall, we found that VCTM provides the most performance improvement and activity savings for designs that utilizes multicasts with large destination sets. Exploring multicast network designs along this continuum may prove promising for designs with small multicast destination sets.

The inverse type of communication, many-to-one communication also has significant applications for on-chip networks. This potential has not yet been explored and hardware support is lacking; off-chip designs [115] have considered reduction and barrier networks and there is reason to believe that they will be useful in on-chip environments. Pruning caches [116] have been proposed in traditional multiprocessors to limit the propagation of invalidation messages and to collect acknowledgments to alleviate the pressure that many-to-one communication places on the network. The collection of acknowledgment messages in directory protocols and the AMD Opteron protocols are two examples. Reduction networks have other applications to cache coherence protocols as well as applications to higher level programming constructs such as barrier implementations and map-reduce programs [36], making them an important area of future study.

Bibliography

[1] P. Abad, V. Puente, J. A. Gregorio, and P. Prieto, "Rotary router: an efficient architecture for CMP interconnection networks," in Proceedings of the 35th Annual International Symposium on Computer Architecture, San Diego, CA, June 2007.

[2] P. Abad, V. Puente, and J. A. Gregorio, "Reducing the interconnection network cost of chip multiprocessors," in Proceedings of the 2nd Annual IEEE International Symposium on Network on Chip, Newcastle-Upon-Tyne, UK, April 2008.

[3] M. E. Acacio, J. Gonzalez, J. M. Garcia, and J. Duato, "Owner prediction for accelerating cache-to-cache transfer misses in a cc-NUMA architecture," in Proceedings of ACM/IEEE conference on Super Computing, 2002.

[4] S. Adve and K. Gharachorloo, "Shared memory consistency models: a tutorial," IEEE Computer, vol. 29, no. 12, pp. 66–76, December 1996.

[5] B. Agarwal and T. Sherwood, "Ternary CAM power and delay model: Extensions and uses," IEEE Transactions on Very Large Scale Integration Systems, vol. 16, no. 5, May 2008.

[6] N. Agarwal, L. S. Peh, and N. Jha, http://www.princeton.edu/ niketa/garnet.html, 2008.

[7] V. Agarwal, M. S. Hrishikesh, S. W. Keckler, and D. Burger, "Clock rate versus IPC: The end of the road for conventional microarchitectures," in Proceedings of the 27th Annual International Symposium on Computer Architecture, Vancouver, BC, June 2000, pp. 248–259.

[8] N. Aggarwal, J. Cantin, M. Lipasti, and J. E. Smith, "Power-aware DRAM speculation," in Proceedings of the International Symposium on High Performance Computer Architecture, February 2008.

[9] A. R. Alameldeen and D. A. Wood, "Variability in architectural simulations of multi-threaded workloads," in Proceedings of the 9th International Symposium on High Performance Computer Architecture, 2003.

[10] P. Apparao, R. Iyer, X. Zhang, D. Newell, and T. Adelmeyer, "Characterization and analysis of a server consolidation benchmark," in Proceedings of the International Conference on Virtual Execution Environments, 2008.

[11] J. Balfour and W. Dally, "Design tradeoffs for tiled CMP on-chip networks," in Proceedings of the International Conference on Supercomputing, 2006.

[12] L. A. Barroso and M. Dubois, "The performance of cache-coherent ring-based multiprocessors," in Proceedings of the 20th Annual International Symposium on Computer Architecture, 1993.

[13] L. A. Barroso, K. Gharachorloo, R. McNamara, A. Nowatzyk, and S. Qadeer, "Piranha: A scalable architecture based on single-chip multiprocessing," in Proceedings of the 27th Annual International Symposium on Computer Architecture, Vancouver, B.C, June 2000, pp. 282–293.

[14] C. Bienia, S. Kumar, J. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," Princeton University Technical Report TR-811-08, Princeton University, Tech. Rep., January 2008.

[15] E. E. Bilir, R. M. Dickson, Y. Hu, M. Plakal, D. J. Sorin, M. D. Hill, and D. A. Wood, "Multicast snooping: A new coherence method using a multicast address network," in Proceedings of 26th Annual International Symposium on Computer Architecture, Atlanta, May 1999.

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

[17] J. A. Butts and G. Sohi, "Characterizing and prediction value degree of use," in Proceedings of 35th International Symposium on Microarchitecture, 2002.

[18] H. Cain, K. Lepak, B. Schwarz, and M. H. Lipasti, "Precise and accurate processor simulation," in Workshop On Computer Architecture Evaluation using Commercial Workloads, 2002.

[19] H. W. Cain, "Detecting and exploiting causal relationships in hardware shared-memory multiprocessors," Ph.D. dissertation, University of Wisconsin - Madison, Madison, Wisconsin, 2004.

[20] J. F. Cantin, M. H. Lipasti, and J. E. Smith, "Improving multiprocessor performance with coarse-grain coherence tracking," in Proceedings of the 32th Annual International Symposium on Computer Architecture (ISCA-32), Madison, WI, June 2005.

[21] ——, "Stealth prefetching," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 2006.

[22] Y. Chang and L. N. Bhuyan, "An efficient tree cache coherence protocol for distributed shared memory multiprocessors," IEEE Transactions on Computers, vol. 48, no. 3, 1998.

[23] A. Charlesworth, "Starfire: Extending the SMP envelope," IEEE Micro, vol. 8, no. 1, pp. 39–49, January 1998.

[24] ——, "The Sun Fireplane SMP interconnect," in Proceedings of Supercomputing 2001, November 2001.

[25] M. Chaudhuri and M. Heinrich, "The impact of negative acknowledgments in shared memory scientific applications," IEEE Transactions on Parallel and Distributed Systems, vol. 15, no. 2, February 2004.

[26] Y.-K. Chen, J. Chhugani, P. Dubey, C. J. Hughes, D. Kim, S. Kumar, V. W. Lee, A. D. Nguyen, and M. Smelyanskiy, "Convergence of recognition, mining, and synthesis workloads and its implications," Proceedings of the IEEE, vol. 96, no. 5, May 2008.

[27] L. Cheng, J. B. Carter, and D. Dai, "An adaptive cache coherence protocol optimized for producer-consumer sharing," in Proceedings of the International Symposium on High Performance Computer Architecture, February 2007.

[28] C. M. Chiang and L. M. Ni, "Deadlock-free multihead wormhole routing," in Proceedings of the 1st High Performance Computing-Asia, 1995.

[29] C. Chiang and L. Ni, "Multi-address encoding for multicast," in Proceedings of the Workshop on Parallel Computing, Routing and Communication, 1994.

[30] P. Conway and B. Hughes, "The amd opteron northbridge architecture, present and future," IEEE Micro Mag., Apr. 2007.

[31] D. Culler and J. Singh, Parallel Computer Architecture: A Hardware/Software Approach. Morgan Kaufmann Publishers, Inc., San Mateo, CA, 1999.

[32] W. J. Dally, "Virtual-channel flow control," in Proceedings of the International Symposium on Computer Architecture, 1990.

[33] W. Dally, "Express cubes: Improving the performance of k-ary n-cube interconnection networks," IEEE Transactions on Computers, vol. 40, no. 9, 1991.

[34] W. Dally and B. Towles, Principles and Practices of Interconnection Networks. San Francisco, CA: Morgan Kaufmann Pub., 2003.

[35] ——, "Route packets not wires: On-chip interconnection networks," in Proceedings of the 38th Annual Design Automation Conference (DAC-38), 2001, pp. 684–689.

[36] J. Dean and S. Ghemawat, "Mapreduce: Simplified data processing on large clusters," in Proceedings of the Symposium on Operating System Design and Implementation, December 2004.

[37] Z. Ding, R. Hoare, A. Jones, D. Li, S. Shao, S. Tung, J. Zheng, and R. Melhem, "Switch design to enable predictive multiplexed switching in multiprocessor networks," in Proceedings of 19th IEEE International Parallel and Distributed Processing Symposium, 2005.

[38] J. Duato, P. Lopez, F. Silla, and S. Yalamanchili, "A high-performance router architecture for interconnection networks," in Proceedings of the International Conference on Parallel Processing, 1996.

[39] M. Dubois, C. Scheurich, and F. Briggs, "Memory access buffering in multiprocessors," in Proceedings of the 13th International Symposium on Computer Architecture, June 1986, pp. 434–443.

[40] N. Eisley, L.-S. Peh, and L. Shang, "In-network cache coherence," in Proceedings of the 39th Annual International Symposium on Microarchitecture, Orlando, FL, December 2006.

[41] N. Enright Jerger, M. Lipasti, and L.-S. Peh, "Circuit-switched coherence," IEEE Computer Architecture Letters, vol. 6, no. 1, 2007.

[42] N. Enright Jerger, L.-S. Peh, and M. Lipasti, "Circuit-switched coherence," in Proceedings of the 2nd Annual IEEE Network on Chip Symposium, Newcastle-Upon-Tyne, UK, April 2008.

[43] N. Enright Jerger, L.-S. Peh, and M. H. Lipasti, "Virtual circuit tree multicasting: A case for on-chip hardware multicast support," in Proceedings of the International Symposium on Computer Architecture (ISCA-35), Beijing, China, June 2008.

[44] ——, "Virtual tree coherence: Region-based multicasting for many-core chips," in Proceedings of the International Symposium on Microarchitecture (MICRO-41), Lake Como, Italy, November 2008.

[45] N. Enright Jerger, D. Vantrease, and M. H. Lipasti, "An evaluation of server consolidation workloads for multi-core designs," in Proceedings of the IEEE International Symposium on Workload Characterization, Boston, MA, September 2007.

[46] R. Figueiredo, P. A. Dinda, and J. Fortes, "Guest editors' introduction: Resource virtualization renaissance," Computer, vol. 38, no. 5, pp. 28–31, 2005.

[47] M. Galles, "Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip," in Proceedings of Hot Interconnects Symposium IV, 1996.

[48] P. Gaughan and S. Yalamanchili, "A family of fault tolerant routing protocols for direct multiprocessor networks," IEEE Transactions on Parallel and Distributed Systems, vol. 6, no. 5, May 1995.

[49] K. Gharachorloo, M. Sharma, S. Steely, and S. V. Doren, "Architecture and design of AlphaServer GS320," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 2000.

[50] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in Proceedings of the 19th Annual International Symposium on Computer Architecture, June 1992, pp. 278–287.

[51] J. R. Goodman, "Using cache memory to reduce processor-memory traffic," in Proceedings of the 10th Annual International Symposium on Computer Architecture, June 1983, pp. 124–131.

[52] P. Gratz, C. Kim, R. McDonald, S. Keckler, and D. Burger, "Implementation and evaluation of on-chip network architectures," in International Conference on Computer Design, October 2006.

[53] P. Gratz, B. Grot, and S. W. Keckler, "Regional congestion awareness for load balance in networks-on-chip," in Proceedings of the 14th IEEE International Symposium on High-Performance Computer Architecture, February 2008.

[54] F. Guo, H. Kannan, L. Zhao, R. Illikkal, R. Iyer, D. Newell, Y. Solihin, and C. Kozyrakis, "From chaos to QoS: case studies in CMP resource management," SIGARCH Computer Architecture News, vol. 35, no. 1, pp. 21–30, March 2007.

[55] A. Gupta, W.-D. Weber, and T. Mowry, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," in Proceedings of the International Conferenece on Parallel Processing, 1990, pp. 312–321.

[56] E. Hagersten, A. Landin, and S. Haridi, "DDM - a cache-only memory architecture," IEEE Computer, vol. 25, no. 9, pp. 44–54, 1992.

[57] E. Hagersten and M. Koster, "WildFire: A scalable path for SMPs," in Proceedings of the IEEE Symposium on High Performance Computer Architecture, January 1999, pp. 172–181.

[58] H. H. J. Hum and J. R. Goodman, "Forward state for use in cache coherency in a multiprocessor system," United States Patent 6,922,756, July 2005.

[59] IBM, "Unleashing the cell broadband engine processor," http://www-128.ibm.com/developerworks/power/library/pa-fpfeib/, IBM, Tech. Rep., November 2005.

[60] Intel, "From a few cores to many: A Tera-scale computing research overview," 2006. [Online]. Available: http://download.intel.com/research/platform/terascale/ terascale_overview_paper.pdf

[61] R. Iyer, L. Zhao, F. Guo, R. Illikkal, S. Makineni, D. Newell, Y. Solihin, L. Hsu, and S. Reinhardt, "QoS policies and architecture for cache/memory in CMP platforms," in Proceedings of the 2007 ACM SIGMETRICS International Conference on Measurement and Modeling of Computer Systems, San Diego, CA, 2007.

[62] J. Jann, L. M. Browning, and R. S. Burugula, "Dynamic reconfiguration: Basic building blocks for autonomic computing on IBM pSeries servers," IBM System Journal, vol. 42, no. 1, pp. 29–37, 2003.

[63] Y. Jin, E. J. Kim, and K. H. Yum, "A domain-specific on-chip network design for large scale cache systems," in Proceedings of the International Symposium on High Performance Computer Architecture, 2007.

[64] J. Kahl, M. Day, H. Hofstee, C. Johns, T. Maeurer, and D. Shippy, "Introduction to the cell multiprocessor," IBM Journal of Research and Development, vol. 49, no. 4, 2005.

[65] D. Kanter, "The common system interface: Intel's future interconnect," http://www.realworldtech.com/page.cfm? ArticleID=RWT082807020032, 2007.

[66] S. Kaxiras and C. Young, "Coherence communication prediction in shared-memory multiprocessors," in Proceedings of 6th International Symposium on High Performance Computer Architecture, 2000.

[67] S. Kaxiras and J. R. Goodman, "Improving CC-NUMA performance using instruction-based prediction," in Proceedings of 5th International Symposium on High Performance Computer Architecture, Orlando, January 1999.

[68] C. Kim, D. Burger, and S. W. Keckler, "An adaptive, non- uniform cache structure for wire- delay dominated on-chip caches," in Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating System, 2002.

[69] J. Kim, J. Balfour, and W. Dally, "Flattened butterfly topology for on-chip networks," in Proceedings of the 40th International Symposium on Microarchitecture, Chicago, IL, December 2007.

[70] M. M. Kim, J. D. Davis, M. Oskin, and T. Austin, "Polymorphic on-chip networks," in Proceedings of the 35th Annual International Symposium on Computer Architecture, Beijing, China, June 2008.

[71] M. M. Kim, S. Swanson, A. Peterson, A. Putnam, A. Schwerin, M. Oskin, and S. Eggers, "Instruction scheduling for a tiled dataflow architecture," in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 2006.

[72] P. Kongetira, K. Aingaran, and K. Olukotun, "Niagara: A 32-way multithreaded Sparc processor," IEEE Micro, vol. 25, no. 2, pp. 21–29, 2005.

[73] A. Kumar, P. Kunda, A. Singh, L.-S. Peh, and N. K. Jha, "A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS," in Proceedings of the International Conference on Computer Design, 2007.

[74] A. Kumar, L.-S. Peh, and N. K. Jha, "Token flow control," in Proceedings of the 41st International Symposium on Microarchitecture, Lake Como, Italy, November 2008.

[75] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, "Express virtual channels: Toward the ideal interconnection fabric," in Proceedings of 34th Annual International Symposium on Computer Architecture, San Diego, CA, June 2007.

[76] P. Kundu, "On-die interconnects for next generation CMPs," in Workshop on On- and Off-Chip Interconnection Networks for Multicore Systems, December 2006.

[77] A.-C. Lai and B. Falsafi, "Memory sharing predictor: The key to a speculative coherent DSM," in Proceedings of the 26th Annual International Symposium on Computer Architecture, 1999.

[78] L. Lamport, “How to make a multiprocessor computer that correctly executes multiprocessor programs,” IEEE Transactions on Computer, vol. 28, no. 9, pp. 690–691, 1979.

[79] A. Landin, E. Hagersten, and S. Haridi, “Race-free interconnection ntworks and multiprocessor consistency,” in Proceedings of the 18th Annual International Symposium on Computer Architecture, 1991.

[80] J. Laudon and D. Lenoski, “The SGI Origin: a ccNUMA highly scalable server,” in Proceedings of the 24th Annual International Symposium on Computer Architecture, 1997.

[81] D. Lenoski, J. Laudon, K. Gharachorloo, A. Gupta, and J. Hennessy, “The directory-based cache coherence protocol for the DASH multiprocessor,” in Proceedings of the 17th Annual International Symposium on Computer Architecture, May 1990, pp. 148–159.

[82] K. M. Lepak, H. W. Cain, and M. H. Lipasti, “Redeeming IPC as a performance metric for multithreaded programs,” in Proceedings of 12th Symposium on Parallel Architecture and Compilation Techniques, 2003, pp. 232–243.

[83] X. Lin, P. K. McKinley, and A. H. Esfahanian, “Adaptive multicast wormhole routing in 2d-mesh multicomputers,” in Proceedings of Parallel Architectures and Languages Europe, June 1993.

[84] X. Lin, P. K. McKinley, and L. M. Ni, “Deadlock-free multicast wormhole routing in 2-d mesh multicomputers,” IEEE Transactions on Parallel and Distributed Systems, vol. 5, no. 8, pp. 793–804, 1994.

[85] J. Liu, L.-R. Zeng, and J. Tenhunen, “Interconnect intellectual property for network on chip,” Journal of System Architecture, 2004.

[86] T. Lovett and R. Clapp, “STiNG a CC-NUMA computer system for the commercial marketplace,” in Proceedings of the 23th Annual International Symposium on Computer Architecture, May 1996.

[87] Z. Lu, B. Yin, and A. Jantsch, “Connection oriented multicasting in wormhole-switched networks on chip,” in Proceedings of Emerging VLSI Technologies and Architectures, 2006.

[88] M. P. Malumbres, J. Duato, and J. Torrellas, “An efficient implementation of tree-based multicast routing for distributed shared memory multiprocessors,” in Proceedings of the IEEE International Symposium on Parallel and Distributed Processing, 1996.

[89] M. Martin, D. Sorin, B. Beckmann, M. Marty, M. Xu, A. Alameldeen, K. Moore, M. Hill, and D. Wood, “Multifacet’s general execution-driven multiprocessor simulator (GEMS) toolset,” Computer Architecture News, September 2005.

[90] M. M. K. Martin, “Token coherence,” Ph.D. dissertation, University of Wisconsin - Madison, Madison, Wisconsin, 2003.

[91] M. M. K. Martin, P. J. Harper, D. J. Sorin, M. D. Hill, and D. A. Wood, "Using destination-set prediction to improve the latency/bandwidth tradeoff in shared-memory multiprocessors," in Proceedings of the 30th Annual International Symposium on Computer Architecture, June 2003.

[92] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token coherence: Decoupling performance and correctness," in Proceedings of the International Symposium on Computer Architecture, June 2003.

[93] M. M. K. Martin, D. J. Sorin, M. D. Hill, and D. A. Wood, "Bandwidth adaptive snooping," in Proceedings of the International Symposium on High Performance Computer Architecture (HPCA-8), 2002.

[94] M. R. Marty, "Cache coherence techniques for multicore processors," Ph.D. dissertation, University of Wisconsin - Madison, Madison, Wisconsin, 2008.

[95] M. R. Marty, J. D. Bingham, M. D. Hill, A. J. Hu, M. M. K. Martin, and D. A. Wood, "Improving multiple-CMP systems using token coherence," in Proceedings of the International Symposium on High Performance Computer Architecture, February 2005.

[96] M. R. Marty and M. D. Hill, "Coherence ordering for ring-based chip multiprocessors," in Proceedings of the 39th International Symposium on Microarchitecture, Orlando, FL, December 2006.

[97] M. Marty and M. Hill, "Virtual hierarchies to support server consolidation," in Proceedings of the 34th Annual International Symposium on Computer Architecture, San Diego, CA, June 2007.

[98] http://www.eecs.umich.edu/mibench.

[99] A. Moshovos, "Regionscout: Exploiting coarse grain sharing in snoop-based coherence." in Proceedings of the 32th International Symposium on Computer Architecture, Madison, WI, June 2005.

[100] S. S. Mukherjee, P. Bannon, S. Lang, A. Spink, and D. Webb, "The alpha 21364 network architecture," IEEE Micro, vol. 22, no. 1, pp. 26–35, 2002.

[101] S. S. Mukherjee and M. D. Hill, "Using prediction to accelerate coherence protocols," in Proceeding of the International Symposium on Computer Architecture, 1998.

[102] R. Mullins, A. West, and S. Moore, "Low-latency virtual-channel routers for on-chip networks," in Proceedings of the 31st Annual International Symposium on Computer Architecture, June 2004.

[103] K. J. Nesbit, J. Laudon, and J. E. Smith, "Providing QoS with virtual private machines," in Proceedings of the 1st Workshop on Chip Multiprocessor Memory Systems and Interconnects, Feb 2007.

[104] H. Nilsson and P. Stenstrom, “The scalable tree protocol - a cache coherence approach for large-scale multiprocessors,” in International Symposium on Parallel and Distributed Processing, 1992.

[105] J. Nilsson, A. Landin, and P. Stenstrom, “The coherence predictor cache: A resource-efficient and accurate coherence prediction infrastructure,” in Proceedings of the International Parallel and Distributed Processing Symposium, 2003.

[106] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Change, “The case for a single-chip multiprocessor,” in Proceedings of the 7th International Symposium on Architectural Support for Programming Languages and Operating Systems, Cambridge, MA, 1996.

[107] D. K. Panda and R. Sivaram, “Fast broadcast and multicast in wormhole multistage networks for multidestination worms,” Technical Report OSU-CISRC-4/95-TR21, Department of Computer and Information Science. The Ohio State University, Tech. Rep., 1995.

[108] L.-S. Peh and W. J. Dally, “Flit reservation flow control,” in Proceedings of the 6th International Symposium on High Performance Computer Architecture, February 2000.

[109] ——, “A delay model and speculative architecture for pipelined routers,” in Proceedings of the International Symposium on High Performance Computer Architecture, January 2001, pp. 255–266.

[110] R. C. Prim, “Shortest connection networks and some generalisations,” Bell System Technical Journal, vol. 36, pp. 1389–1401, 1957.

[111] A. Raghavan, C. Blundell, and M. M. K. Martin, “Token tenure: PATCHing token counting using directory-based cache coherence,” in Proceedings of the 41st Annual International Symposium on Microarchitecture (MICRO-41), Lake Como, Italy, November 2008.

[112] P. F. Reynolds Jr., C. Williams, and R. R. Wagner Jr., “Isotach networks,” IEEE Transactions on Parallel and Distributed Systems, vol. 8, no. 4, pp. 337–348, 1997.

[113] T. D. Richardson, C. Nicopoulos, D. Park, N. Vijaykrishnan, Y. Xie, C. Das, and V. Degalahal, “A hybrid SoC interconnect with dynamic TDMA-based transaction-less buses and on-chip networks,” in In Proceedings of the 19th International Conference on VLSI Design, 2006.

[114] K. Sankaralingam, R. Nagarajan, H. Liu, J. Huh, C. Kim, D. Burger, S. Keckler, and C. Moore, “Exploiting ILP, TLP, and DLP using polymorphism in the TRIPS architecture,” in Proceedings of the 30th Annual International Symposium on Computer Architecture, June 2003.

[115] S. L. Scott, “Synchronization and communication in the T3E multiprocessor,” in Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems, 1996.

[116] S. L. Scott and J. R. Goodman, "Performance of pruning-cache directories for large-scale multiprocessors," IEEE Transactions on Parallel and Distributed Systems, vol. 4, no. 5, pp. 520–534, May 1993.

[117] D. Seo, A. Ali, W.-T. Lim, N. Rafique, and M. Thottenhodi, "Near-optimal worst-case throughput routing in two dimensional mesh networks," in Proceedings of the 32nd Annual International Symposium on Computer Architecture, Madison, WI, June 2005, pp. 432–443.

[118] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, C. R. Ho, D. J. Ieradi, I. Kolossvary, J. L. Klepeis, T. Layman, C. McLeavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang, "Anton, a special-purpose machine for molecular dynamics simulation," in Proceedings of the 34th Annual International Symposium on Computer Architecture, San Diego, CA, June 2007.

[119] A. Singhal, D. Broniarczyk, F. Cerauskis, J. Price, L. Yaun, C. Cheng, D. Doblar, S. Fost, N. Agarwal, K. Harvery, E. Hagersten, and B. Liencres, "Gigaplane: A high performance bus for large SMPs," in Proceedings of the 4th Hot Interconnects Symposium, August 1996, pp. 41–52.

[120] B. Sinharoy, R. Kalla, J. Tendler, R. Eickemeyer, and J. Joyner, "Power5 system microarchitecture," IBM Journal of Research and Development, vol. 49, no. 4, 2005.

[121] R. Sivaram, D. K. Panda, and C. B. Stunkel, "Multicasting in irregular networks with cut-through switches using tree-based multidestination worms," in Proceedings of the 2nd Parallel Computing, Routing and Communication Workshop, Atlanta, GA, June 1997.

[122] ——, "Efficient broadcast and multicast on multistage interconnection networks using multiport encoding," IEEE Transactions on Parallel and Distributed Systems, vol. 9, no. 10, October 1998.

[123] J. E. Smith and R. Nair, "The architecture of virtual machines," Computer, vol. 38, no. 5, pp. 32–38, 2005.

[124] SPEC, "SPEC benchmarks," http://www.spec.org.

[125] K. Strauss, "Cache coherence for embedded ring multiprocessors," Ph.D. dissertation, University of Illinois, 2007.

[126] K. Strauss, X. Shen, and J. Torrellas, "Flexible snooping: Adaptive forwarding and filtering of snoops in embedded ring multiprocessors," in Proceedings of the 33rd Annual International Symposium on Computer Architecture, Boston, MA, June 2006.

[127] ——, "Uncorq: Unconstrained snoop request delivery in embedded-ring multiprocessors," in Proceedings of the 40th International Symposium on Microarchitecture, Chicago, IL, December 2007.

[128] C. B. Stunkel, J. Herring, B. Abali, and R. Sivaram, "A new switch chip for ibm rs/6000 sp systems," in Proceedings of Supercomputing, 1999.

[129] S. Swanson, K. Michelson, A. Schwerin, and M. Oskin, "Wavescalar," in Proceedings of the 36th International Symposium on Microarchitecture, 2003.

[130] Y. Tamir and G. Frazier, "Dynamically allocated multi-queue buffers for VLSI communication switches," IEEE Trans. on Computers, vol. 41, no. 6, June 1992.

[131] D. Tarjan, S. Thoziyoor, and N. P. Jouppi, "Cacti 4.0," HP Technical Report, Hewlett Packard, Tech. Rep., 2006.

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

[133] J. Tendler, J. Dodson, J. J.S. Fields, H. Le, and B. Sinharoy, "Power4 system microarchitecture," IBM Journal of Research and Development, vol. 46, no. 1, pp. 5–26, 2002.

[134] TPC, "TPC benchmarks," http://www.tpc.org.

[135] J. Turner, "An optimal nonblocking multicast virtual circuit switch," in Proceedings of Infocom, 1994.

[136] L. G. Valiant and G. J. Brebner, "Universal schemes for parallel communication," in Proceedings of the 13th Annual ACM Symposium on Theory of Computing, 1981, pp. 263–277.

[137] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar, "An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS," in Proceedings of the IEEE International Solid State Circuit Conference, 2007.

[138] C. A. Waldspurger, "Memory resource management in VMware ESX server," in Fifth Symposium on Operating System Design and Implementation, December 2002.

[139] H.-S. Wang, X. Zhu, L.-S. Peh, and S. Malik, "Orion: A power-performance simulator for interconnection networks," in Proceedings of 35th International Symposium on Microarchitecture, 2002.

[140] H. Wang, L.-S. Peh, and S. Malik, "Power-driven design of router microarchitecture in on-chip networks," in Proceeding of the 36th International Symposium on Microarchitecture, November 2003.

[141] P. M. Wells, K. Chakraborty, and G. S. Sohi, "Hardware support for spin management in overcommitted virtual machines," in Proceedings of Parallel Architectures and Compilation Techniques (PACT), 2006.

[142] D. Wentzlaff, P. Griffin, H. Hoffman, L. Bao, B. Edwards, C. Ramey, M. Mattina, C.-C. Miao, J. B. III, and A. Agarwal, “On-chip interconnection architecture of the tile processor,” IEEE Micro, pp. 15–31, 2007.

[143] D. Wiklund and D. Liu, “SoCBus: Switched network on chip for hard real time embedded systems,” in Proceedings of 17th IEEE International Parallel and Distributed Processing Symposium, 2003.

[144] A. W. Wilson Jr, “Hierarchical cache/bus architecture for shared memory multiprocessors,” in Proceedings of the 14th Annual International Symposium on Computer Architecture, June 1987, pp. 244–252.

[145] P. T. Wolkotte, G. J. Smit, G. K. Rauwerda, and L. T. Smit, “An energy efficient reconfigurable circuit-switched network-on-chip,” in Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium, 2005.

[146] S. Woo, M. Ohara, E. Torrie, J. Singh, and A. Gupta, “The SPLASH-2 programs: Characterization and methodological considerations,” in Proceedings of the 22th Annual International Symposium on Computer Architecture, June 1995.

[147] J. Zebchuk, E. Safi, and A. Moshovos, “A framework for coarse-grain optimizations in the on-chip memory hierarchy,” in Proceedings of the 40th International Symposium on Microarchitecture, Chicago, IL, December 2007.