OPTICAL TOKENS IN MANY-CORE PROCESSORS

by

Dana M. Vantrease

A dissertation submitted in partial fulfillment of
the requirements for the degree of

Doctorate of Philosophy

(Computer Sciences)

at the

UNIVERSITY OF WISCONSIN–MADISON

2010

ACKNOWLEDGMENTS

This thesis would not have been possible without the support of my family, friends, and colleagues over the past eight years.

I am grateful to my advisor, Mikko Lipasti, for being a great advisor, teacher, and role model. His wisdom, guidance, and weekly doses of encouragement have been crucial to maintaining my research momentum. Through example, he has taught me the value of balancing work and life (a skill that has undoubtedly kept me sane). I am also thankful to my thesis committee members for their objective feedback: Professor Mark Hill, Professor David Wood, Professor Leon McCaughan, and Professor Parameswaran Ramanathan. My research and presentation skills have improved over the years thanks to Mark Hill's critiques, which are some of the toughest, most concise, and best critiques I have ever received. Also, I admire David Wood, for his youthful enthusiasm and great wisdom.

I would like to thank my team of collaborators at Hewlett-Packard Laboratories for giving my research real traction. I would especially like to thank: My mentor, Nathan Binkert, who guided me, taught me some valuable coding skills, and brought me much-needed fro-yo during paper deadlines; Marco Fiorentino, for patiently explaining the physics of nanophotonics; Professor Al Davis, for propping me up and believing that I am one "rocking dudette;" Jung Ho Ahn and Matteo Monchiero, for simulator help; Nidhi Agarwal, for bringing my attention to the internship and lending a sympathetic ear when my pet chicken died; Moray McLaren, for being a wonderful manager; and Norm Jouppi, for giving me the opportunity to work with such a great team.

The friends and colleagues I have met during graduate school have really made the experience rewarding. I would like to thank the many who helped to maintain the UW-architecture community through their efforts in organizing Reading Group, Affiliates, Architecture Lunch, Architecture Seminar, and Architecture Beer. Thanks to all of my lab-mates for making lab a fun (and geeky) place to work, especially to Mitch Hayenga for the shop-talk and for being a fellow fast-talker; to Atif Hashmi and Andy Nere for lending their problem solving skills and mostly avoiding me during rubber band shoot-outs; Erika Gunadi for defense advice; Natalie Enright-Jerger for keeping dibs on me; Gordon Bell for keeping the cluster alive; Jason Cantin for not allowing me to drop out; and Trey Cain for pumping good tunes while burning the midnight oil. I spent my early years in the Computer Sciences building and returned there often. I would like to thank Luke Yen for being my qual study-buddy; Dan Gibson for the enjoyable conversations and a memorable kayaking trip; Matt Allen for great presentation critiques and habanero chili; Yasuko Watanabe and Polina Dudnik for showing the world how awesome female architects can be; Mike Marty for making conferences more fun (+Google Wave Invite #1); Phillip Wells for being a great Reading Group moderator (+Google Wave Invite #2); Jayaram Bobba for helping with Graduates Anonymous; All of my CS polar plunge teams for keeping me cold in February.

Last but not least, I thank Kenley Maddux, for sticking with me and giving me his unwaivering support. I also thank my ma, pa, and siblings for believing in me and so much more.

TABLE OF CONTENTS

Page

LIST OF TABLES . . . vi

LIST OF FIGURES . . . vii

ABSTRACT . . . ix

1 Introduction . . . 1

1.1 The Many-Core Era . . . 1
1.1.1 Many-Core Computation . . . 1
1.1.2 Many-Core Communication . . . 2
1.1.2.1 Communication Uses . . . 2
1.1.2.2 Technology's Effect on Communication Architectures . . . 3
1.2 Thesis Contributions . . . 3
1.2.1 Contributions . . . 4
1.2.1.1 Optical Tokens . . . 4
1.2.1.2 Photonic Arbitration, Flow Control, and Fairness . . . 4
1.2.1.3 Atomic Coherence: Photonic Assisted Decoupling of Races from Coherence . . . 5
1.2.2 Relation to Previously Published Work . . . 5
1.3 Dissertation Structure . . . 6

2 Background . . . 7

2.1 Global Communication Technology in Many-Core Processors . . . 7
2.1.1 Electricity in Copper Wires . . . 7
2.1.2 Light in Free-Space and Silicon Waveguides . . . 8
2.1.2.1 Background . . . 8
2.1.2.2 Properties . . . 9
2.2 Silicon Photonics . . . 10
2.2.1 Devices . . . 10
2.2.2 Challenges . . . 13
2.2.2.1 Device Considerations . . . 13
2.2.2.2 System Considerations . . . 15
2.3 Alternative Global Technologies . . . 16
2.4 Summary . . . 17

3 Related Work . . . 18

3.1 Optical Interconnect Control . . . 18
3.1.1 Optical Transmission Fidelity . . . 18

Page

3.1.1.1 Topology Approaches 20
3.1.1.2 Access Policies 20
3.1.2 Flow Control 22
3.1.3 Fairness 22
3.2 Cache Coherence Complexity 23
3.2.1 Cache Coherence 23
3.2.2 Evolution of Protocol Complexity 24
3.2.3 Ordering Techniques 24
3.2.4 Protocol Verification Methods 28

4 Methodology 29

4.1 Optical Assumptions 29
4.1.1 Area 29
4.1.2 Latency 29
4.1.3 Power 29
4.2 Simulation 31
4.2.1 Workloads 31
4.2.1.1 Descriptions of Workloads 31
4.2.1.2 Running of Workloads 31
4.2.2 System Setup 33
4.2.2.1 Components 33
4.2.2.2 Configuration 38
4.3 Summary 40

5 Optical Tokens 41

5.1 Overview 41
5.2 Token Inject and Detect 44
5.3 Fixed-Injector and Variable-Injector Systems 44
5.4 Token Clocking and Refresh 45
5.5 Latency 45
5.6 Power 47
5.6.1 Laser Power 47
5.6.2 Ring Power 50
5.6.3 Laser and Ring Power 52
5.7 Comparison To Traditional Token Rings 52
5.8 Summary 52

6 Photonic Interconnect Control 54

6.1 Photonic Control 54
6.1.1 Arbitration 56
6.1.1.1 Token Channel 56
6.1.1.2 Token Slot 56
6.1.2 Flow Control (FC) 58
6.1.3 Fairness 59
6.1.3.1 Fair Token Channel 59

Page

6.1.3.2 Fair Token Slot . . . 61
6.1.4 Multiple Channel Control . . . 65
6.2 Evaluation . . . 65
6.2.1 Experimental Setup . . . 65
6.2.2 Results . . . 67
6.2.2.1 Performance . . . 67
6.2.2.2 Power . . . 69
6.3 Discussion . . . 74
6.3.1 Timing Assumptions . . . 74
6.3.2 Supporting Prioritization . . . 76
6.4 Summary . . . 76

7 Atomic Coherence . . . 77

7.1 Introduction . . . 77
7.2 Atomic Coherence . . . 77
7.2.1 Atomic Coherence Definition and Discussion . . . 78
7.2.2 Enforcing Atomicity with Mutexes . . . 79
7.2.3 Complexity Implications of Atomic Coherence . . . 81
7.2.4 Performance Implications of Atomic Coherence . . . 81
7.2.4.1 Performance Pitfalls . . . 81
7.2.4.2 Performance Opportunities . . . 82
7.3 A Photonic Atomic Substrate . . . 87
7.4 Evaluation . . . 88
7.4.1 Experimental Setup . . . 88
7.4.2 Results . . . 89
7.4.2.1 Complexity . . . 89
7.4.2.2 Performance . . . 91
7.4.2.3 Complexity-Effective Performance . . . 94
7.4.2.4 Power . . . 94
7.4.2.5 Sensitivity Analysis . . . 94
7.4.2.6 Performance Optimization Results . . . 96
7.5 Discussion . . . 97
7.5.1 Other Opportunities for Aggressive Coherence . . . 97
7.5.2 Relation to Token Coherence . . . 97
7.5.3 Data-Race-Free . . . 99
7.5.4 Memory Consistency Implications . . . 99
7.6 Summary . . . 100

8 Summary and Reflections . . . 101

8.1 Summary . . . 101
8.2 Reflections . . . 102
8.2.1 Speculation Freedom . . . 102
8.2.2 Applicability of Contributions to Other Technologies . . . 102

LIST OF REFERENCES . . . 104

LIST OF TABLES

Table Page

2.1 Definition of Optical Symbols . . . 13

3.1 Cache Coherence Complexity Taxonomy . . . 27

4.1 Optical Device Power Consumption . . . 30

4.2 Benchmarks . . . 34

4.3 System Configurations . . . 38

5.1 Ring Resonator Losses . . . 50

LIST OF FIGURES

Figure Page

1.1 Optical Token Overview . . . 4

1.2 Atomic Coherence Overview . . . 5

2.1 Approaches to On-Chip Photonic Integration . . . 9

2.2 A Ring Resonator's Interaction with Light . . . 10

2.3 Nanophotonic Building Blocks . . . 11

2.4 3D Stacking . . . 16

3.1 Optical Interconnect Topologies . . . 19

3.2 Protocol Finite State Machines . . . 25

4.1 Neural Network Example . . . 32

4.2 Four Wavelength Corona-like Data Channel . . . 35

4.3 Corona's Interconnect Layout . . . 36

4.4 Data Interconnects Used in Evaluation . . . 39

5.1 Optical Token Injection and Detection . . . 42

5.2 Fixed-Injector And Variable-Injector Token Systems . . . 43

5.3 Optical Token Dateline . . . 46

5.4 Optical Token Scalability . . . 46

5.5 Laser Power as Function of Distance on a Straight Waveguide with No Rings . . . 48

5.6 Token Power Results . . . 49

5.7 Laser Power Sensitivity to Scattering and Insertion Losses . . . 51

5.8 Traditional Token Ring Vs Optical Tokens . . . 53

6.1 Multiple Writer Single Reader Topology . . . 55

Figure Page

6.2 Token Channel 57

6.3 Token Slot 57

6.4 Token Channel with Fast Forwarding 60

6.5 Fair Slot Finite State Machine 62

6.6 Fair Slot Implementation 63

6.7 Arbiters for 4-channel MWSR 66

6.8 Synthetic Workloads: Bandwidth, Latency, and Worst Service for Arbiters 68

6.9 Max-Min Token Slot Results 70

6.10 Max-Min Token Channel Results 71

6.11 SPLASH-2 Workloads: Achieved Bandwidth for Arbiters 72

6.12 Arbiter Power 73

6.13 Layout of Multiple Arbiters 75

7.1 SGI-Origin Writeback Race 80

7.2 Parallel and Serial Miss Timing Diagrams 83

7.3 PushS and ShiftF Overview 84

7.4 Implicit Acknowledgements in Atomic Coherence 86

7.5 Complexity analysis for Atomic Coherence 89

7.6 Performance and Overhead for Atomic Coherence 92

7.7 Complexity Vs Normalized Runtime 93

7.8 Sensitivity Study for Atomic Coherence 95

7.9 Atomic Coherence Optimizations' Speedup 96

7.10 Atomic Coherence Traffic Breakdown for `MOEFSI` 98

ABSTRACT

As core count grows, demand for high-bandwidth, low-latency on-chip interconnects increases. While fast transmission of raw bits is one necessary property of future high performance interconnects, it is not sufficient; interconnects must also manage coordination of shared entities, such as resources and data. This thesis uses nanophotonics to perform many-core coordination that enhances system performance and decreases complexity.

Many have advocated silicon-based nanophotonics for future many-core data communication. It is my goal to show its applicability to other many-core needs, namely many-core coordination. The same optical properties that benefit data communication may benefit many-core coordination: low cross-chip latencies, high bandwidth, and low power consumption. However, to take full advantage of these benefits, it will be necessary to accommodate the technology's limitations. One limiter is repeater inefficiency. Converting an optical signal to and from the electrical domain is relatively slow and also power-inefficient. Thus, it is in the interest of performance to keep the signal in optical form from source to destination.

I propose *optical tokens* as a way to manage coordination while coping with repeater inefficiencies. In most cases, our optical tokens can be passed from any interested party to any other interested party with no repeats, thus allowing for flexible communication at speed-of-light propagation. When a token is viewed as usage rights for a resource, it becomes a fast traveling mutex.

I show two applications of optical tokens. The first application employs optical tokens in the arbiters of an optical data interconnect. The data interconnect is fast and requires fast arbitration for high utilization. Also, like the optical tokens, it avoids repeating data transmissions. These complimentary repeater-less designs allow a token to maintain pace with its associated bandwidth allocation and achieve very high utilization. I further extend the proposal to support flow control and fairness.

The second application employs optical tokens to address cache coherence protocol complexity. Cache coherence protocol complexity arises from the presence of concurrency races, which though rare, must be detected and resolved. My proposal, called Atomic Coherence, simplifies protocols by serializing potential races before they issue to the interconnect. Without races, sophisticated protocols are substantially easier to design, debug, and validate. Race serialization comes at some cost to performance, but is kept to a minimum with low-latency optical tokens.

Chapter 1

Introduction

The multi-core era has arrived. Duo- and quad- core systems are commonplace. Researchers are bracing for tens, even hundreds, of cores on a single chip [1]. For a long time we thought in terms of housing a single core on a chip; now a chip can house an entire neighborhood of interconnected cores and memories. With this new landscape, we must carefully consider interaction between all parts to achieve a balanced high-performance system.

The chip's parts work together to perform three main duties: computation, communication, and storage. In theory, gains in computational potential are governed by Moore's law [2]. In reality, these gains can only be realized with a communication system capable of keeping the computational units fed with data. Unfortunately, Moore's law does not directly improve communication, making communication the bottleneck in systems of multiple cores.

Thus, to continue reaping the benefit's of Moore's law, our future processors need to support powerful computation *and* powerful communication. Part of our pursuit must be to improve the communication technology, so that raw data can travel from source to destination more quickly and in larger volume. We must also pursue strategies in effective use of the technology, so that we may achieve high channel utilization and efficiently meet cache coherency and memory consistency contracts.

Finally, we cannot forget that parallel software will be what ultimately unlocks the performance of future many-core processors. So far, the burden of parallelizing software has fallen largely on the programmer, who must reason about difficult things, such as finding parallelism, non-determinism, and parallel performance tuning. Future hardware must ease the programmer's job.

1.1 The Many-Core Era

1.1.1 Many-Core Computation

Since the early 1960s, Moore's law has driven processor performance by doubling the maximum number of on-chip transistors every 18 months. During much of this time, chip architects efficiently translated their increased transistor budgets into single-threaded performance. However, the time came that larger budgets resulted in diminishing returns; the architects simply had more transistors than a single thread could use. With little progress being made on single-threaded performance, architects turned to multi-threaded performance.

Taking a divide and conquer approach to the plethora of transistors, monolithic single-core designs began to be replaced with multi-core designs (2-8 cores) [3, 4]. Now, multi-core has started to give way to many-core (8+ cores) [5, 6]. These many-core designs have simpler cores than their predecessors, choosing to emphasize higher aggregate throughput over any single core's throughput [7, 8]. Simpler cores also translate to lower aggregate power. Only transistors that can efficiently benefit performance are built into a core; many architecturally advanced techniques, like out-of-order issue, speculative execution, and large instruction windows, have been stripped away.

Simply organizing transistors into simple cores is not enough to realize the performance potential of Moore's law. Arguably, the biggest problem facing many-core processors is usability. Programs must be written that can harness the parallelism. Unfortunately, programming parallel applications is hard to reason about and even harder to optimize. Fortunately, many people across systems, compilers, and architecture are working hard on this problem. The other major problem facing many-cores is how to connect the cores to provide efficient communication. We discuss the needs of many-core communication next.

1.1.2 Many-Core Communication

1.1.2.1 Communication Uses

In many-cores, communication is used for data transfer, cache coherence, and synchronization.

Data Transfer. Communication is necessary to transfer data between cores.

The many-core data interconnect, or Network on Chip (NoC), must meet the needs of its many simple (or light-weight) cores. These new light-weight cores have different needs than their heavy-weight predecessors. Regarding bandwidth, a light-weight core requires less memory bandwidth; it executes less aggressively and generates requests less frequently. Regarding latency, a light-weight core is more latency sensitive; it is unable to overlap as many (if any) misses and will expose the miss serialization latency. A light-weight core, then, requires the interconnect to provide data with low latency but also at low bandwidth. However, when assessed as a system, the bandwidth supply must be increased to satisfy all cores. In the end, although demands of a single core have decreased, the aggregate demands on the interconnect have not. Future many-cores will require an interconnect with high bandwidth and low latency.

Caches can help these demands by keeping copies of data near the cores. When the data is not cached locally, it must be retrieved from another on-chip cache or from memory. Retrieving from another cache is likely faster than retrieving from memory, but some extra latency may be necessary if the system enforces cache coherence. Cache coherence provides a coherent flat view of memory in the presence of replicated data.

Cache Coherence Communication also serves cache coherence protocols, which manage replicated data to maintain a consistent view of memory.

Coherence approaches fall into two camps: snoopy and directory. Generally speaking, directory protocols are viewed as the scalable solution for many-cores while snoopy protocols serve smaller core-counts. In a directory protocol, each miss visits the directory. The directory supplies information about the cached state of the block, such as its state and its sharing set. If the directory allows, there can be as many sharers as caches. To track the sharers precisely incurs an overhead that increases proportionally with the number of caches. Furthermore, when the number of sharers is large (as large as the number of caches), either invalidating or updating these sharers can be quite costly for latency and bandwidth. Addressing these overheads is necessary if future many-cores are expected to provide cache coherence.

Synchronization Finally, in a system of large scale concurrency, communication can help to synchronize coordination between two or more entities. Program threads synchronize executions with barriers or locks. Hardware synchronizes for ordering at memory fences and for arbitrating shared resources.

In many-core systems, synchronization needs to scale to many participants. Many-cores considering hardware synchronization methods can take a lesson from Symmetric Multi-processors (SMPs), where dedicated hardware barriers have been around for a long time [9, 10, 11]. Hardware barriers can be quite heavy weight, as exemplified by Cray's choice to abandon its barrier network between the T3D and T3E [12]. However, we believe that if synchronization methods were more light-weight, they could be used more often, by programmer and hardware alike, to manage massive concurrency.

1.1.2.2 Technology's Effect on Communication Architectures

An interconnect should use the technology at hand to build the best communication network possible. To stay competitive, the interconnect must constantly adapt to changes in technology. For example, global wires have become more costly, in term of latency and power. Interconnects have adapted by replacing global interconnects with short point-to-point interconnects that "route packets not wires" [13]. Communication systems have also adapted by putting an extra emphasis on creating locality [14, 15].

The emergence of a new interconnect technology, like nanophotonics, has different challenges and opportunities than electronics, offering new design exploration. Its benefits, which become pronounced when communicating across distances greater than 20 micrometers [16] have fueled a re-emergence in bus, crossbar, and other global interconnect research [17, 18, 19]. Furthermore, this research aims to show how nanophotonics can allow for low-cost coordination that improves many-core performance and complexity.

1.2 Thesis Contributions

This thesis investigates how nanophotonics can benefit many-core performance through the use of light-weight optical tokens.

1.2.1 Contributions

We contribute *optical tokens* as a way to exploit silicon-based nanophotonic technology for low-latency, low-power, and high-bandwidth mutual exclusion. We also show how optical tokens provide low-latency interconnect arbitration and allow for high-performing simple cache coherence protocols.

1.2.1.1 Optical Tokens

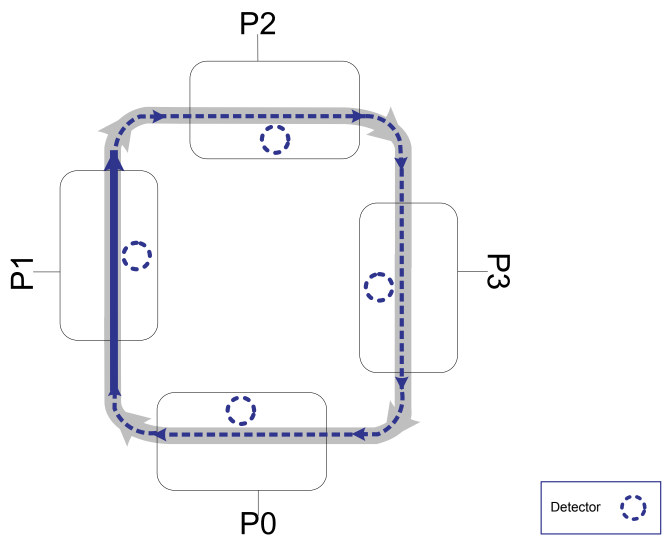


Figure 1.1 **Optical Token Overview.** The token (solid line) circulates around the ring's path (dotted line) without repeat. Any detector may seize the token as it passes.

Optical tokens are a low-latency low-power way to enforce mutual exclusion. Each token symbolizes a resource, either physical or virtual. Tokens circulate around a ring-shaped interconnect and are rarely repeated. The circulation of a token is shown in Figure 1.1.

When a node seizes a token, it has exclusive rights to the corresponding resource. When a node releases the token, it releases its rights to the resource.

Because the token is rarely repeated, we can achieve speed-of-light mutual exclusion that scales with the length of the ring instead of the number of nodes.

1.2.1.2 Photonic Arbitration, Flow Control, and Fairness

We use optical tokens to provide arbitration for optical data interconnects.

Optical tokens allow for a distributed approach to arbitration that is able to keep pace with the data interconnect. Unlike prior token proposals, our optical tokens are not repeated at non-requesting nodes; only requesting nodes receive the token. This allows for very low-latency arbitration that allows for a full matching of sources to destinations every cycle without pipelining the arbiter.

We further provide simple optical extensions that allow for flow control and enforce fair arbitration. Our arbiters exploit optics to simultaneously achieve low latency, high utilization, and fairness.

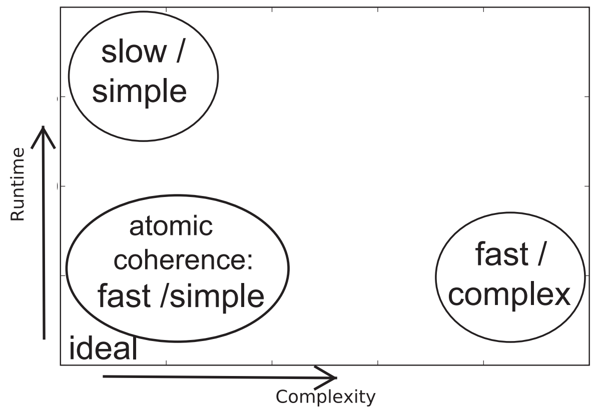


Figure 1.2 **Atomic Coherence Overview.** Complexity Versus Runtime. Most modern protocols fall into the "fast/complex" region while older protocols fall into the "slow/simple" region. The goal of Atomic Coherence is to approach the axis-intersection with "fast/simple" protocols.

1.2.1.3 Atomic Coherence: Photonic Assisted Decoupling of Races from Coherence

We use our optical tokens to resolve races in cache coherence protocols.

Atomic Coherence is a way to reduce the complexity of specifying, designing, and verifying cache coherence protocols. Atomic Coherence decouples race resolution from coherence protocol processing, leading to protocols that only need to follow expected, mostly sequential paths to satisfy requests for coherence permissions and data. This eliminates all unexpected, complex, and difficult-to-verify race transitions from the protocol state machine. By doing so, it aims to strike a balance between complexity and performance, as shown in Figure 1.2.

Atomic Coherence uses address-mapped optical tokens to provide mutual exclusion and atomicity to outstanding coherence transactions. Independent requests are allowed to proceed in parallel, but races are serialized at token acquisition.

1.2.2 Relation to Previously Published Work

This thesis encompasses work that has appeared in two conference publications and one workshop publication:

- **Corona: System Implications of Emerging Nanophotonic Technology (ISCA '08).** We use a Corona-like data interconnect to evaluate our work in Section 6 and Section 7. The Corona paper was written during an internship I had at Hewlett-Packard Labs and was co-authored by Robert Schreiber, Matteo Monchiero, Moray

McLaren, Norman P. Jouppi, Marco Fiorentino, Al Davis, Nathan Binkert, Raymond G. Beausoleil, and Jung Ho Ahn.

- **Light Speed Arbitration and Flow Control for Nanophotonic Interconnects (MICRO '09).** This thesis presents the findings of this MICRO paper. It also presents solutions for pipelining and prioritizing arbitration. The paper was co-authored by Nathan Binkert, Robert Schreiber, and Mikko Lipasti.

- **Nanophotonic Barriers (PICA '09).** The fair arbiter we propose in Section 6 uses an optical wired-OR to detect when a node has not been serviced fairly. This wired-OR functionality was inspired by the wired-OR nanophotonic barrier we presented in 2009 at the Workshop on Photonic Interconnects and Computer Architecture (Held in Conjunction MICRO-42). It was co-authored with Nathan Binkert, Al Davis, Mikko H. Lipasti, and Robert Schreiber.

1.3 Dissertation Structure

Chapter 2 presents background information on electrical and nanophotonic technologies as applied to on-chip interconnects. Chapter 3 discusses the prior work related to this thesis. Chapter 4 describes the methods and experimental setups used in our evaluation. Optical Tokens are a recurring theme that are detailed in Chapter 5. Their application to arbitration and cache coherence are presented in Chapters 6 and 7, respectively. Finally, Chapter 8 concludes the thesis.

Chapter 2

Background

This chapter presents an overview of global communication in many-core processors. Global communication is useful to future many-core chips. The number of cycles it takes to get from chip edge to chip edge is growing with each technology generation, making communication latencies highly variable. This non-uniformity means programmers must increasingly pay attention to locality. However, if global communication was fast and plentiful, we could ease programmer burdens. And, as this thesis shows through optical technology, we can also improve system throughput and reduce complexity.

We give an overview of two technologies for global communication, one electrical and one optical in Section 2.1. In Section 2.2, we flush out the details of our technology-of-choice, which uses silicon waveguides and silicon ring resonators.

2.1 Global Communication Technology in Many-Core Processors

2.1.1 Electricity in Copper Wires

Copper resistance/capacitance (RC) lines are used in on-chip interconnects. The effective rise time τ of a simple RC wire depends on its resistance, capacitance, and length, as characterized by the formula [20]:

$$\tau \approx R_l C_l l^2$$

R_l	$=$	Resistance per unit length
C_l	$=$	Capacitance per unit length
l	$=$	length

As we approach the many-core era, diminishing feature sizes will put pressure on wires to scale down. Simple three-dimensional wire scaling causes R_l to grow quadratically, for resistance is inversely proportional to the quadratically shrinking cross-sectional area. Local wires may effectively counteract R_l's increase with reduced l. Global wires, on the other hand, span an absolute (or nearly absolute) distance l, allowing the growth of R_l to dictate performance. Reducing R_l by inversely scaling wires is possible [21] but is not bandwidth efficient, requires many metal layers, and may cause significant cross talk [22].

Increasing global wire delay may also be combatted with the insertion of repeaters to break up long lines. Repeaters allow delay to grow linearly, rather than quadratically, with distance, but also consume power and area [23]. Thus, a fundamental tradeoff exists, between wire length and repeater count. This tradeoff can be used to determine the optimal repeater size and spacing to achieve a minimum delay [24]. The minimum delay is effectively fixed around 20 ps/mm for global wires (corresponding to 1 mW/mm) through 2017 [25, 26].

Global wires are valuable to inter-core communication, but, to be useful, global wire delay must be kept to a minimum. Designs that try to minimize this latency, by either inversely scaling wires or not scaling them at all, do so with increasing costs. At the end of the day, transistors are scaling, making global wires appear relatively slower, bulkier, and more power-hungry.

2.1.2 Light in Free-Space and Silicon Waveguides

Silicon nanophotonics is an interesting alternative for global communication. It has properties useful to global communication, such as low-loss high-speed long-distance dense communication. And, unlike wires, it does not suffer from resistive loss physics.

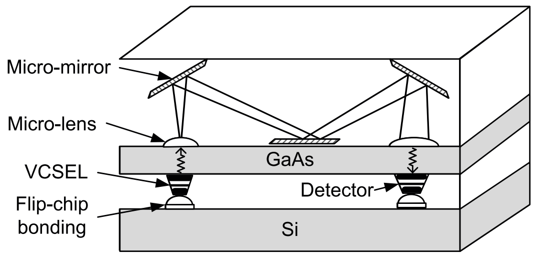
Optics has traditionally had high termination costs, which, for a long time, could only be amortized over long distances. These costs have begun to decrease, allowing cost-effective communication at smaller distances. Recently, optics has become viable for HPC processor-to-processor interconnects [27]. Intra-processor optics may soon be possible since the successful demonstrations of individual CMOS-friendly nano-scale devices [28, 29]. However, to be ultimately successful, the technology needs to move from laboratory demonstrations to reliable, cheap, and high-yield manufacturables.

2.1.2.1 Background

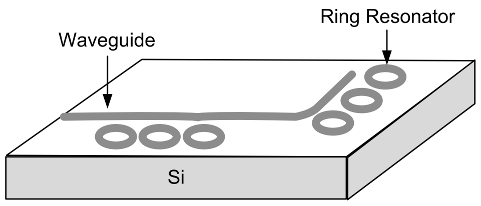
Although the focus of this dissertation is silicon photonics with waveguides and ring resonators, free-space photonics is another approach being pursued in on-chip interconnects [30].

Free-Space optics employs Vertical-Cavity Surface-Emitting Lasers (VCSELs) that are built into a Gallium Arsenide (GaAs) wafer and emit light perpendicular to the chip plane. The light travels in free-space, interacting with lenses and/or mirrors, until it reaches the detector. VCSELs allow for a single optical package, but require non-silicon materials, such as GaAs.

A silicon waveguide-based approach uses an all-silicon process on-chip. Light is routed and manipulated on-chip with silicon waveguides and ring resonators. However, because purely silicon lasers have not been realized (silicon has poor gain due to its indirect band gap [31]), the laser must originate off-chip. Figure 2.1 shows examples of VCSEL and silicon waveguide systems.



(a) Free-Space with VCSELs



(b) Silicon with Waveguides and Rings

Figure 2.1 **Approaches to On-Chip Photonic Integration**

2.1.2.2 Properties

Both VCSEL and ring resonator technologies promise low latency, low loss, and high bit density communication systems.

Low Latency. The propagation speed of light (c) in photonic interconnects is simply dependent on the speed of light in the given material as determined by the refractive index of the material: $c \approx c_0/3$ for silicon and $c \approx c_0$ for free-space integration.

Low Losses. An optical signal can have small losses over long distances. A signal in free-space experiences very little loss. Silicon waveguides experience more losses due to light scattering between the waveguide walls; currently on the order of 2.5 dB/cm [32] but predicted to approach 0.3-1.0 dB/cm [33].

High Bit Density. Bit density is the transmission rate divided by transmission medium's width. It depends on the clocking speed, pitch size, and the degree of wavelength multiplexing.

Already, free-space and silicon modulation methods have exceeded 10 GHz speeds [29, 34]. These speeds should be sufficient for designs targeting 2017 [35, 19, 36], because they meet (or exceed) predictions for processor frequency in the next decade [37]. Thus, we can assume 10 GHz speeds for our bit density calculations.

Free-space optics have pitch determined by the 20μm device-to-device spacing of the VCSELs [34]. Clocked at 10 GHz, free-space systems have the potential to achieve 0.5 (Gb/sec)/μm density, which is comparable to future electrical interconnects [38].

Silicon waveguide pitch is chosen in a way that is appropriate for the mode size and minimizes cross-talk [25]. A 0.5 μm wide waveguide might have a 3.5 μm pitch [26]. Clocked at 10 GHz, the bit density is 2.85 (Gb/sec)/μm. Multiplexing multiple wavelengths on this waveguide with Dense Wave Division Multiplexing (DWDM) increases this

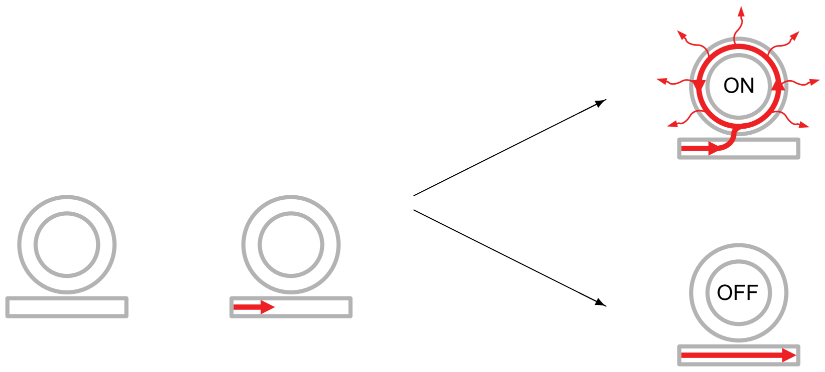


Figure 2.2 **A Ring Resonator's Interaction with Light.** The light is either diverted by the ring (top) or passes by (bottom).

value linearly. Thus, a 64-DWDM estimate would increase this density to 183 (Gb/sec)/μm. For a rough comparison with electronics, a 15 mm repeated wire at 32 nm has densities up to ˜1.5 (Gb/sec)/μm [39]..

2.2 Silicon Photonics

To capture the performance potential of optics, we need efficient physical components to transport, modulate, and detect the laser power. Continued advancements in physical design are promising steps towards realizing full photonic integration, but many challenges remain at the device and system level.

In this section we turn our focus to silicon photonics. All of our proposals leverage silicon nanophotonics to quickly transmit optical tokens (see Chapter 5). Alternative on-chip optical options, like VCSEL-based designs, are omitted because of current incompatibility with optical tokens.

2.2.1 Devices

Below I provide functional descriptions of the nanophotonic building blocks. An interconnect consists of a power source, transmission medium, and operating devices. Nanophotonic transmission works by electrically or thermally altering the refractive index of material, which in turn alters the path of the light.

- **Power Source - Laser.** All power comes from an off-chip multi-wavelength Si/III-V evanescent laser, which uses a silicon-waveguide laser cavity wafer-bonded to a III-V gain medium and has precisely controlled frequency spacings [40]. One advantage of using this type of laser is that if only one of the frequency channels is servo-locked to an on-chip standard cavity, then all of the other frequency modes will track the controlled mode.

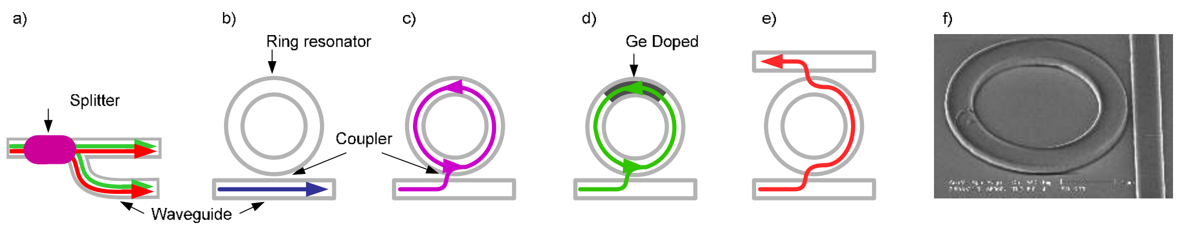


Figure 2.3 **Nanophotonic Building Blocks (Top View).** **(a) Splitter.** The passive splitter splits a fraction of the light, across all wavelengths, and couples it to the bottom waveguide. It is fundamental to distributing power and broadcasting. **(b) Off-resonance Ring.** The ring resonator is coupled to a waveguide through evanescent coupling. Off-resonance wavelengths are transmitted through. **(c) Diverter.** A resonant wavelength is coupled in the on-resonance ring and eventually gets attenuated by losses. **(d) Detector.** A resonant wavelength is coupled in the ring and absorbed by a Ge doped portion of the ring. **(e) Injector.** A resonant wavelength in the input (lower) waveguide is coupled into the ring and out through the output (top) waveguide. **(f)** SEM image of a 3 μm diameter resonator ring. Image courtesy of Qianfan Xu, HP Labs Palo Alto. (Figure appears in [19])

Another advantage is that any temperature change in the environment will cause approximately the same refractive index shift in the laser cavity and the silicon waveguides and resonators that form the DWDM network. This simplifies wavelength locking. A wavelength locking scheme that is robust against temperature changes is one of the key implementation challenges for a DWDM network-on-chip.

- **Transmission Medium - Waveguides.** Waveguides made of crystalline silicon (refractive index ˜3.5) and silicon oxide (refractive index ˜1.45) contain and route the light. The waveguides are typically ≈ 0.5 μm in width.

- **Operating Devices - Splitters, Ring Resonators.** The devices described below operate either on all wavelengths in a waveguide or single out a specific wavelength.

  Splitter. A broadband splitter is a passive device that transfers a fraction of the laser's power, across all wavelengths, between two diverging waveguides as shown in Figure 2.3(a). The unsplit portion is unaffected by the splitter.

  Ring Resonator. A ring resonator, as photographed in Figure 2.3(f), is a switch built from a small ($\approx 3\mu$m diameter) ring-shaped waveguide that is wavelength specific. As Figure 2.2 shows, an on-resonance ring resonator removes the majority of the resonant wavelength from the waveguide it is coupled to, while an off-resonance ring resonator removes a negligible amount. The ring is brought into and out of resonance when the ring's electrical charge or temperature are adjusted, which adjusts the ring's index of refraction.

  Below I describe how resonators can be configured to perform the tasks of diversion, detection, and injection. Diversion, for instance can be used to modulate data. By quickly bringing the resonator in and out of resonance, the presence of light, or its absence, is transmitted down the waveguide to encode 0s and 1s. The modulated data is wave pipelined, or latch-less. The three flavors of ring resonator are as follows:

  – Diverter. When a ring resonator is coupled next to a waveguide, the light circulates within the ring and eventually dissipates through scattering imperfections of the ring (Figure 2.3(c)).

  – Detector. When a ring resonator is doped with Germanium (Ge), the wavelength goes through a photoelectric conversion as it circulates in the ring, translating the signal into the electrical domain (Figure 2.3(d)).

  – Injector. When an on-resonance ring resonator is coupled between two parallel waveguides, a specific wavelength is injected from one transmission waveguide to the other (Figure 2.3(e)).

The figures contained in this dissertation follow some conventions that we lay out in Table 2.1 Note that a ring resonator, either germanium doped or not, can be in one of three states:

- A ring resonator can be off-resonance (empty circle).

▬	Unlit waveguide.
🟥 🟩 🟦	Specific wavelengths (e.g. red, green ,blue) lit.
○	Off-resonance ring resonator.
⊕	On-resonance non-coupling ring resonator.
◉	On-resonance coupling ring resonator.
◌	Off-resonance detector.
⊕	On-resonance non-coupling detector.
◉	On-resonance coupling detector.

Table 2.1 Definition of Optical Symbols

- A ring resonator can be on-resonance and *not coupling* any light (because no light is present) (cross in circle).
- A ring resonator can be on-resonance and *coupling* light (because light is present) (filled circle).

2.2.2 Challenges

Despite the great progress silicon nanophotonics has made in recent history, many challenges remain.

2.2.2.1 Device Considerations

Ring Resonators. Below we describe ring resonator challenges regarding capacitance, thermal variation, fabrication precision, and area.

- Capacitance. To realize the latency and power benefits of optical transmission, we must efficiently perform optical-to-electrical (OE) and electrical-to-optical (EO) conversions. A well integrated ring resonator with low capacitance is important to realize speed while achieving acceptable power consumption [39]. Advancements in analog amplifiers, positively affected by transistor scaling, can also help improve the conversion efficiency.
- Thermal Sensitivity. A ring resonator's dimensions determine its coupling wavelength. To resonate with a very high extinction ratio, the ring must be an integral number of half-wavelengths in circumference. However, the circumference of the ring can change with temperature by ≈ 0.11 nm/K, causing a mismatch in resonance [41].

The mismatch can be dynamically compensated for with local circuitry controlling the silicon temperature [42]. Depending on the range of temperatures experienced by the rings, this circuitry can be expensive.

- Fabrication. It may also be necessary to compensate for size variation that results from fabrication error. Although a ring resonator is a few microns in diameter, only a few nanometers may separate two adjacent wavelengths in a given free spectral range. High-resolution lithography may obtain such precision, as recently demonstrated by the telecom industry with $\approx 1.8 \pm 0.1$ nm wavelength spacing for ring resonators [43]. More precision will be needed for smaller 0.45 nm targets (as discussed below). In the case that the lithography cannot meet such stringent requirements, the same methods used for thermal tuning may be applied to correct for fabrication.

- Area. Each ring resonator operates at a different wavelength and is sized according to its operating wavelength. The laser's spectral range and degree of DWDM largely determine the wavelength spacing, which in turn determines a ring resonator's size. The fabrication precision can also play a role in a ring's dimensions, as described below.

  The gain spectrum of the laser, divided by the degree d of DWDM, indicates the maximum spacing between adjacent wavelengths. All wavelengths are equally spaced. A typical semiconductor laser has a spectral range of 30-40 nm [40]. We calculate that an aggressive DWDM design with $d = 64$ and a laser with a spectral range of 30 nm would then require λ spacing on the order of 0.47 nm.

  Because a ring resonates when an integral number of wavelengths (n) fit in its circumference, the minimum circumference difference between two wavelength adjacent rings (λ_i, λ_{i+1}) is 0.47nm and the minimum diameter difference ($\Delta d = d_{i+1} - d_i$) is a mere 0.15 nm. This minimum occurs when $n = 1$, and is uncomfortably small for even advanced lithography (on the magnitude of a few atoms). Fortunately, the Δd between λ-adjacent rings grows linearly with n. Thus, the value of n should be large enough to tolerate lithography variation between adjacent rings.

  The value of n should also be set so that a given ring resonator only resonates with one frequency. [1] In other words, if a ring closely obeys $d_i \bmod \lambda_i = 0$, all other λ_j's, where $j \neq i$, must obey $d_i \bmod \lambda_j \neq 0$. This may also play a role in limiting the spectral range to $[\lambda_0 * n, \lambda_0 * (n+1))$.

  This work assumes a laser with a spectral range of [1300 nm, 1320 nm], and 64 DWDM. Wavelengths have 80 GHz spacing (0.45 nm). Given the above considerations, ring resonators are between 3.5 μm and 4.0 μm in diameter and have a value of n around 32 for any given ring resonator.

[1]This work assume single wavelength-selective ring-resonators. Ring resonators that resonate across multiple frequencies are also possible with larger sized rings [44].

So, while transistor sizes are largely determined by the technology scale, ring resonator dimensions are largely determined by the coupling wavelength. Ring sizes may shrink with improvements in wavelength, but improvements are limited; estimates indicate that rings start losing effectiveness at radii $< 1.5\mu$m [28].

Laser. The off-chip laser supplies a continuous source of power for all on-chip communication. Thus, the laser is a static cost and provisioned for the maximum amount of concurrent communication. When the interconnect is not at maximum utilization, which is the common case, then the unused portion of laser power is wasted. The degree of this waste depends on the interconnect architecture.

Waveguides. It is important that the silicon waveguides have minimal propagation losses. Propagation losses occur when light scatters off of rough side-walls, which are formed with dry-etching into the resist mask [31]. Continued research into high-quality fabrication using e-beam lithography or oxidation techniques can minimize roughness [45, 46]. If necessary, laser power can be increased to compensate for such losses.

2.2.2.2 System Considerations

Up until now, we have only looked at single devices. To build a nanophotonic interconnect, we must consider the system issues that arise when many devices are integrated. Below we consider system issues of power, layout, and error rate.

Power. The laser and the ring resonators are the major contributors to system power.

The laser power must be large enough to overcome losses due to waveguide crossings, waveguide curves, light scattering, device insertions, and signal conversions [47]. Losses incurred along the length of the waveguide can be especially dramatic, because they compound. One such loss, incurred by slight attenuation when a signal passes by an off ring resonator, determines the scalability of our proposals (discussed more in Section 5.6 Page 5.6). Generally speaking, designers should be cautious of implementing a waveguide with many crossings, many tight bends, or many off-resonance rings.

Ring resonators are the other major power contributor. Power is applied to the ring resonator to trim it (hold it off-resonance), bias it (bring it on-resonance), or modulate it (quickly alternate its resonance). The power a ring consumes depends on its state (see Table 4.1 Page 30 for power quantities). We calculate system ring power by assuming each ring operates independently.

Layout. These photonic devices, with the exception of the off-chip optical power source, are built in a silicon layer, which may be integrated with CMOS electronics. Already, manufacturers of transceivers for telecom applications have begun integrating silicon photonics and electronics in the same die [48].

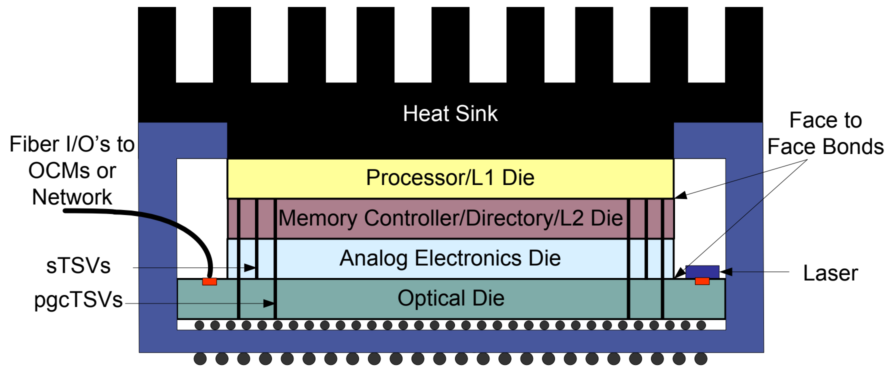


Figure 2.4 **An Example of 3D Stacking**

Prior work has advocated both same-die and separate-die integration of optical components [49, 50, 51, 49]. Monolithic integration has less interfacing overhead and higher yield than 3D stacking, but requires the optical components, which are relatively large, to consume active die area. 3D stacking, on the other hand, follows trends of future interconnects occupying separate layers and allows the CMOS and photonic processes to be independently optimized. The optical layer need not have any transistors, consisting only of patterning the waveguides and rings, diffusion to create the junctions for the modulators, germanium for the detectors, and a metalization layer to provide contacts between layers. An example of what 3D stacking might look like is shown in Figure 2.4.

Our optical token proposals are evaluated in systems with a separate optical die but are light-weight enough that they could feasibly be integrated onto the same die as the electronics.

Cost. The economics of fabrication are key to the eventual success of nanophotonics. Currently, wires are very cheap and optics very expensive. For nanophotonics to succeed, many devices must be fabricated cheaply and reliably.

2.3 Alternative Global Technologies

Transmission lines and radio frequency are alternative technologies for global communication.

Transmission Lines. Transmission lines are a promising solution that avoid the issues RC wires experience by having delay dominated by inductive impedance rather than resistive impedance. In other words, they use wave transmission signalling rather than voltage signalling. Delays near the speed of light have been demonstrated for cross-chip interconnects [52]. Transmission lines do not need repeaters, but consume more area than even the widest global wires [53].

Radio Frequency. Radio frequency (RF), like transmission lines, also use transmission of waves to signal. Unlike transmission lines, but similar to photonic waveguides, several frequencies may share the same (metal) waveguide with Frequency Division Multiple Access (FDMA).

RF devices are larger than optical devices but, unlike optics, do benefit from scaling with technology. Estimates for the inductor, which consumes a dominant portion of the transceiver area, is $50\mu m \times 50\mu m$ at 20 GHz [54].

Finally, RF suffers from electromagnetic interference (EMI) and must be shielded for reliability. On the other hand, optics is immune to EMI.

2.4 Summary

This chapter began with a discussion of electrical wire challenges and ended with a discussion of silicon photonic challenges. While both technologies have significant challenges ahead, silicon photonics is a young technology that has been making significant progress. We are hopeful researchers can sustain, or even improve, this rate of progress.

In the middle of the chapter we described how the basic nanophotonic building blocks work. With waveguides, ring resonators, and a laser, we have all the components necessary to build our proposals.

Chapter 3

Related Work

This chapter summarizes the architectural work related to this dissertation. Section 3.1 discusses interconnect control and Section 3.2 discusses cache coherence.

3.1 Optical Interconnect Control

So far, architects have mostly advocated on-chip optics for global data transmission, transferring zeros and ones from a source to a destination. Optical data interconnects have the potential to provide high frequency, high bandwidth, global communication while consuming modest power. However, to realize the potential, a few requirements must be met:

1. The interconnect's global nature requires a means to setup communication for cross-chip source-destination pairs.

2. The interconnect's speed and high bandwidth allow for single-cycle messages, but also requires fast reconfiguration.

Nearly every global data channel will need adjustments on nearly every cycle.

Frequent cross-chip reconfiguration may be possible with electronic control, but faces distinct challenges because global communication in electronics is expensive and needs to be segregated into local decision domains [55]. We believe optical tokens have the opportunity to meet control needs at a global scale.

3.1.1 Optical Transmission Fidelity

If in-flight data transmissions may collide, they must either be restricted from doing so, or else the collisions must be detected and recovered. Below we discuss how optical interconnects have prevented collisions with physical topology and access policies.

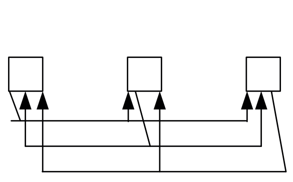
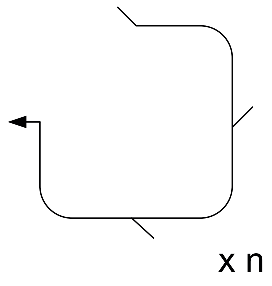
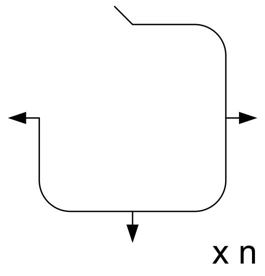
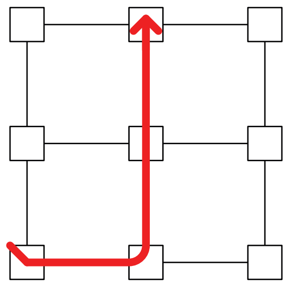
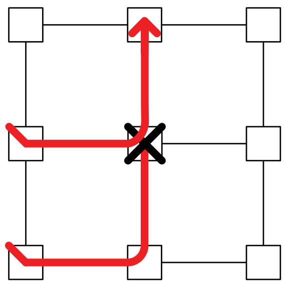
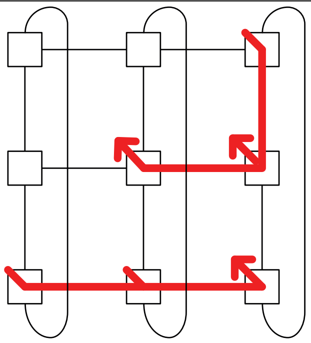
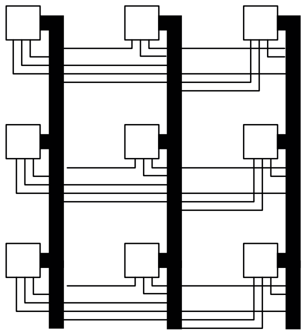
 Kirman et al [18] Topo: Optical Bus Arb: Finite State Machines	 Corona [19] Topo: Multiple Writer Single Reader Arb: Optical Tokens	 Firefly [56] Topo: Single Writer Multiple Reader Arb: Broadcast
 Shacham et al [57] Topo: Circuit-Switched Mesh Arb: Elec Worm Circuit	 Phastlane [35] Topo: Mesh with Buffered Collisions Arb: None	 Kirman and Martinez [58] Topo: Circuit Switched Sub-Mesh Arb: Optical Handshake Msgs
	 Sun. Zheng et al [59] Topo: Dedicated Point-to-Point Arb: Not Applicable	

Figure 3.1 **Optical Interconnect Topologies (Topo) and Arbitration Policies (Arb).**

3.1.1.1 Topology Approaches

The easiest way to avoid transmission collisions is to build a topology where they do not exist. Providing dedicated channels between every source-destination pair achieves this, as proposed by Sun Microsystems' point-to-point network [59]. Their approach allows a non-blocking topology with no per-packet setup time, but it is also restricts flexible use of bandwidth and requires n^2 channels. Topologies consisting of master-slave channels, where there is only one writer but may be many readers, also avoid write collisions [56]. Such single-writer-multiple-reader (SWMR) channels avoid writer-collisions but need a way to avoid reader-collisions. Reader-collisions are a problem in optics, caused by the ring-resonators having a destructive-read effect on the signal. The only proposal to avoid such reader collisions uses heavy-weight broadcast messages that instruct a single reader to activate its detectors [56].

Topologies that allow for multiple writers (or multiple masters) on a channel are susceptible to writer collisions and are the focus of Chapter 6. The topologies we study support channels with many writers, allowing for a potentially large collision problem (but also high channel utilization through sharing). If the channel only supports a small number of writers, the collision problem is not so large and the channel is less vulnerable to collisions (but also less utilizable). Kirman and Martinez use the latter approach [58]. In their design, a subset of nodes share a channel and optical messages are exchanged to configure the channel for a source-destination pair. The source-destination may communicate until the channel is reconfigured for a new pair. They find reconfigurations happen rarely for scientific workloads, and their approach has comparable throughput to prior proposals.

Finally, topologies that have intersecting channels (e.g. meshes) can have routing collisions when two or more messages are simultaneously routed onto the same physical channel. Joshi et. al [50] propose at an optical Clos network that always buffers a packet before sending it on, thereby avoiding contention for an output port. Shacham et. al [57] pro-actively avoid collisions in a mesh by requiring a circuit be set up between a source and destination before transmitting. They rely on long-lasting circuits to recover the setup latency. Phastlane [35] also uses a mesh, but reacts to collisions at the router, buffering all but one of the collisions. If buffers are not available, the message is dropped. Their design also requires a very aggressive router in addition to optical and electrical circuits with picosecond delay.

Figure 3.1 shows the variety of optical topologies discussed above.

3.1.1.2 Access Policies

Collisions may also be avoided with access policies that either restrict or permit a source from transmitting. Some common access approaches include:

Time/Frequency Division Multiplexing. Multiplexing avoids collisions by restricting a source to a certain dimension of the transmission space. In time division multiplexing (TDMA), a source may only transmit 1 out of every n network cycles, where n is determined by the number of possible source-destination pairs. In frequency division multiplexing (FDMA), a source and destination communicate exclusively over a fixed set of frequencies. Both TDMA and FDMA have been proposed for optical NoCs [18, 59].

Bipartite Matching. Matching protocols create collisionless source-destination pairs by solving an instance of the bipartite matching problem [60]. The problem input is a set of edges (requests) that connect two sets of vertices (sources and destinations). The output is a set of matchings, represented by a subset of the input edges.

In a centralized one level bus [61], there is at most one match at a time (one source wins the bus). In crossbars, there can be many simultaneous matchings (many sources to many destinations). Crossbar matching is most widely used in router chips, like Tiny Tera [62], but can also found in high performance computing chips, like the DEC EV7 [63, 64]. The goal of a matching is to provide as many matches as possible with low latency [65].

Maximal matching, whereby the matching cannot be extended by adding another pair, requires interaction between the sources and destinations [63]. Maximal matching can be achieved by optically communicating all arbitration information to-and-from a centralized arbiter, as proposed by Minkenberg et. al [66] and Krishnamurthy, Franklin, & Chamberlain [67]. It can also be achieved in a distributed manner, as proposed by Qiao and Melhem [68] for SMPs. Requesting nodes send optical pulses downstream while detecting pulses that may be coming from an upstream requester. At the end of an arbitration, a successful requester detects no light (no requesters) from upstream.

Token Protocols. Token protocols have been used for a long time in distributed systems to provide mutually exclusive access to channels [69, 70, 71]. A node that seizes a token is guaranteed a mutually exclusive allocation of the bandwidth; no other node's data transmission will collide with it. Token protocols are naturally distributed and allow for continual resource arbitration. While token protocols work on any physical topology, they work best on a ring topology.

In optics, Ha and Pinkston [72] and Kodi and Louri [73], have advocated token-based protocols to arbitrate for optical SMP data interconnects. Their proposals process tokens electrically at each node by converting the optical signal to an electrical signal, processing the token, and converting it back to an optical signal. Repeating the token at every node is mandatory. Similarly, Marsan et. al [74] propose a collisionless arbitration strategy for optical LAN and MAN rings that relies on inspecting message slot headers, a technique analogous to requesting tokens. In Section 6, we show the latency of repeating a token cannot be tolerated in optical NoCs which are highly sensitive to latency. We argue that on-chip optical interconnects will likely have repeat-free transmission and will benefit most from repeat-free tokens.

The token protocols we propose use optical tokens to minimize token repeats. The protocols work best on many-writer interconnects that physically resemble the ring or bus topology, like [18, 19, 74, 75]. Our proposals are not directly applicable to meshes, like Shacham et. al's [57] and Cornell's Phastlane [35].

3.1.2 Flow Control

Optical interconnects are fast and have the potential to fill buffers at the destination quickly. The rate of data flow must either be throttled to guarantee the destination always has a buffer for an incoming transmission, or may be unrestricted with a buffer-overflow/retransmission mechanism in place.

To keep the system in a steady state, the rate that a node services its buffers should be greater than or equal to the rate that the buffers are filled. For some topologies, the buffer fill-rate can be very high (even scaling with the number of nodes), making buffer overflows very likely. For example, Sun's dedicated optical channels [59] and Firefly's SWMR channels [56] allow for a node to potentially receive from all on-chip nodes at once.

Such bursty behavior can overwhelm buffers and cause undesirable thundering herds [76] in the face of retransmission. For other topologies, where the peak receive bandwidth is fixed under scaling of nodes, buffer flow is still an issue that may be caused by queueing back ups. Irrespective of topology, throttling the flow can improve bandwidth utilization by avoiding failed transmissions due to receive buffer overflows.

The most common approach to flow control is credit based [77]. Each buffer is associated with a credit, and only a node that seizes a credit may seize the corresponding buffer. Our proposal, as well as a similar proposal [75], uses credit-based flow control and circulates the credits optically. We only evaluate designs with flow control.

3.1.3 Fairness

Finally, the interconnect must guarantee management such that all nodes make forward progress. The ring topology has daisy-chain ordering. In Chapter 6, we find that a node's position on the ordered ring can impact the service it receives.

Many ring arbitration schemes in LAN systems have had to address issues of access fairness when a node's location on the ring matters. Metaring [78] uses a so-called SAT(isfied) token that travels upstream, against traffic, and refreshes transmission quotas at each hop. Starving nodes hold the SAT token, causing other nodes to exhaust their quotas, and eventually guaranteeing the starving node is serviced. Other approaches use fairness epochs that operate in a "stop-and-go" fashion [79, 80]. These methods aim to over time balance the varying bandwidth needs of nodes, but adjust the needs too slowly for unpredictably dynamic and fine grain on-chip communication.

We overcome ring-ordering issues by providing a means for an under-serviced node to communicate that it is starving. Elsewhere, Flexishare [75] also encountered unfairness problems with its optical channels. The authors solved Flexishare's unfairness with a two-pass scheme, where only a single node could be serviced on the first pass and if unused, any node could be serviced on the second pass. Although very similar to our scheme, their scheme guarantees a lower bound on service while our scheme comes very close to providing max-min service, a harder service guarantee (see Section 6.1.3 on page 59 for more discussion of max-min). A lower bound sets the minimum service a node will receive, but makes no statement about the service received above this bound. Flexishare enforces the minimum service on the first pass, but the second pass still suffers from daisy-chain ordering. Our max-min goal, on the other hand, states that service above this bound should be equally distributed across all the nodes.

3.2 Cache Coherence Complexity

More cores mean more caches. Caches improve performance because they allow for low-latency access to data by keeping copies of the data near the cores. However, in the presence of many caches, systems must keep a consistent view of memory. Cache coherence protocols do just this by managing reads and writes to shared memory locations, usually at the granularity of a block. Given a memory location, a protocol should deliver read values that can be interleaved into a total order of writes to that same location. Unfortunately, as we describe below, cache coherence protocols have become more complex over time, which motivates our Atomic Coherence work (Chapter 7).

3.2.1 Cache Coherence

Hardware often implements cache coherence, allowing for performance benefits over software. Hardware implementations need no involvement from the compiler and programmer, allowing for a flat-view of memory.

There are two major approaches to how blocks are maintained under replication. In the *update* approach, all copies of a block may be *updated* when a processor alters (writes) any one copy. In the *invalidate* approach, all copies, except the one being altered, must first be evicted from the caches before the remaining lone copy is written. The update approach allows reading and writing of a shared block. It has the advantage of propagating writes for subsequent use by the sharers, but also consumes extra bandwidth and makes write atomicity difficult.

Whether or not an update or invalidate approach is used, there is still an issue of finding sharers so that they may be updated or invalidated. The sharers may be found with *snoops*, a *directory*, or a combination of the two [81]. Snoops are messages that probe every cache for copies and take appropriate action on the block. Snooping protocols work well for small systems, but do not scale to large systems because the bandwidth to a cache and the number of snoops that a cache must process in a given cycle scales with N. Directory protocols are more scalable by only probing caches that have the block. Requests can look in the directory to find an entry containing sharer information. The main drawback to directories is that the directory introduces a level of indirection, which adds latency. Another drawback that has been receiving recent attention [82] is that directories are limited by their metadata, which must scale with N for precise tracking of sharers. And while imprecise tracking reduces metadata overhead [83, 84, 85], it sacrifices precision of the sharing set.

Regardless of the approach taken to managing caches, there are complexity issues that arise due the concurrent nature of caches operating independently. We discuss complexity issues next.

3.2.2 Evolution of Protocol Complexity

The literature usually represents cache coherence protocols as state machines with events that cause atomic transitions between stable states (e.g.. `M, S, I`) [86]. In early bus-based machines, these *stable transitions* could be accomplished atomically, since the bus, once allocated, was held until the transition completed and prevented the initiation of any other concurrent transitions. Such blocking shared buses made it easy to provide coherence, since arbitration for the shared resource implicitly specified a total order for all memory references. The buses atomically serialized all racing requests, enabling a trivial construction of a coherent system-wide interleaving of reads and writes to a common address.

Unfortunately, performance concerns soon forced designers to abandon blocking shared buses, adopting non-blocking buses and other high performance interconnects instead [88]. Such interconnects separate a request from its associated responses, freeing the bus in the interim to deliver other commands and providing much higher utilization and bandwidth. However, coherence transitions can no longer be accomplished atomically, but must instead be decomposed into a series of *split transitions*. Split transitions correspond to the completion of various steps of the end-to-end state transition (e.g. request, coherence response, data response). This is an unavoidable result of exposing greater concurrency in the processor-memory interconnect.

Unfortunately, absent arbitration for a single shared resource, split transition protocols are no longer immune to races between conflicting requests to the same addresses. Coherence guarantees require that protocols adapt to provide a means for detecting and resolving these interconnect-induced races. If the race is detected after two or more nodes in the system have initiated conflicting coherence state transitions, the protocol must include additional *race transitions* to correctly resolve the situation. The result is an explosion of state-event space, adding complexity to the protocol but also adding performance.

Figure 3.2 shows an example of the state-explosion for a simple MSI protocol while Figure 3.2 shows the state-explosion for a more complex MOETSI protocol. More complicated protocols have even larger state explosions; the Cray-X1 has been reported to have as many 214 million reachable states [89]. As the figures show, many races result in self-transitions (returning to the same state). For brevity, the figure does not show that each self-transition may take one of many actions (e.g. buffer, `NACK`, `ACK`, etc). Also, the figures only show valid state-event pairs. When designers specify the protocol, they must figure out from all possible state-event pairs, which pairs are valid and require specification. Reasoning about valid versus invalid state-pairs can be difficult in the presence of races.

3.2.3 Ordering Techniques

There are many techniques for establishing and maintaining coherence request order. Once an order has been created, the order must be obeyed and non-atomic interleavings must be dealt with.

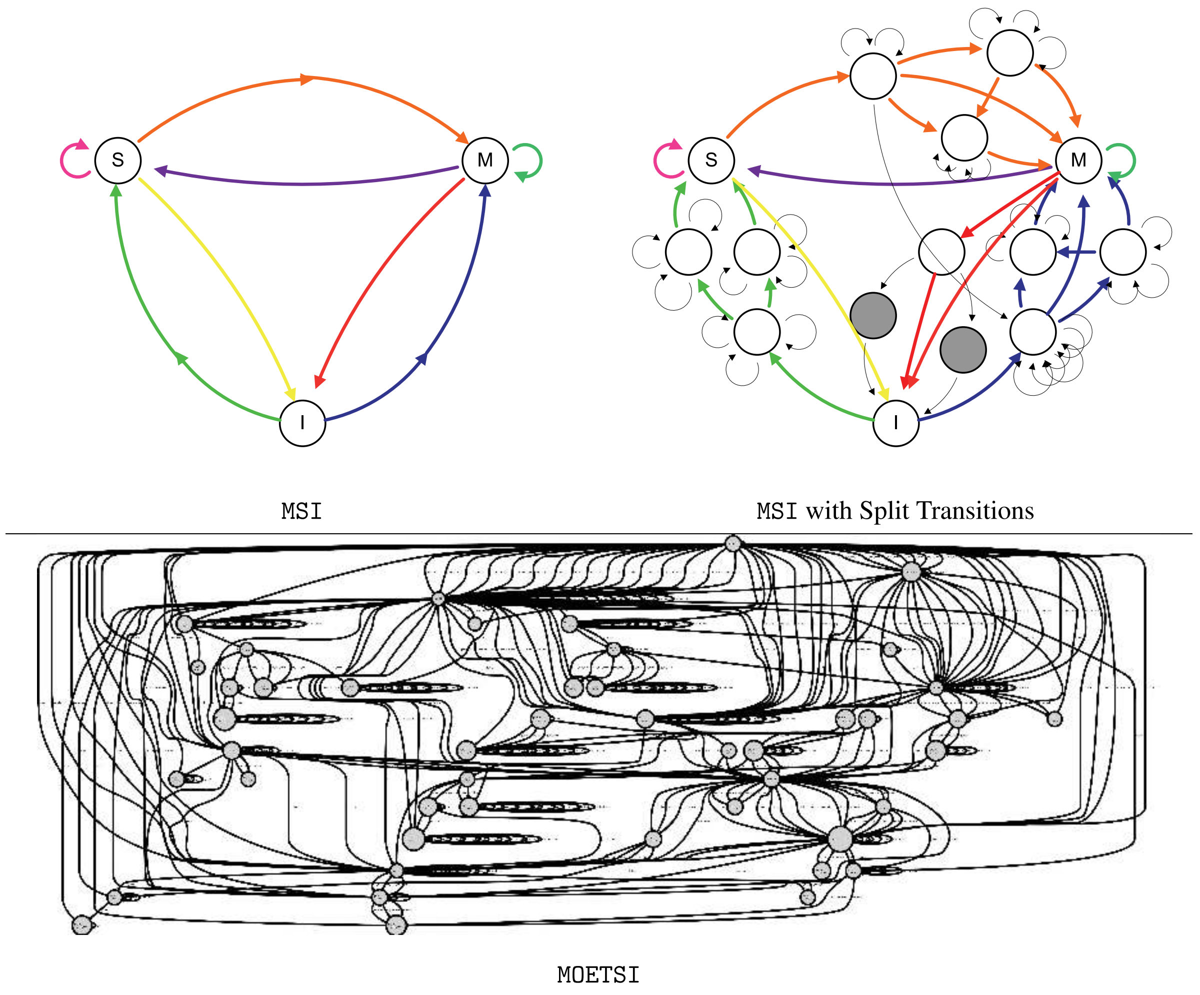


Figure 3.2 **Protocol Finite State Machines. Top: MSI Protocol.** The left figure shows a high-level view of the MSI protocol. The right figure flushes out all of MSI's state transitions, most of which are race transitions. Gray states represent those states which only exist to resolve races (the writeback race). **Bottom: MOETSI** [87] protocol is an example of a protocol more sophisticated (and more complex) than MSI (above)

In-Network Ordering. An interconnect topology which establishes message order may help to reconcile racing requests. For example, buses, and to some extent rings [90], have topological ordering points that place messages in a global order. Once a race has been detected, the established order can be used to resolve the race accordingly. For unordered topologies, like the mesh, order can be built into the topology with techniques such as Timestamp Snooping [91] or In-Network Snoop Ordering (INSO)[92].

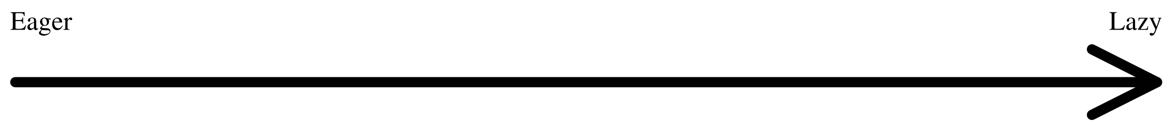
Ordering Points. Alternatively, directories, which are topology-agnostic, can establish order at the directory entry.

A directory that only services one request to block B at a time and delays all other requests to B is called a *blocking* directory (e.g. Wildfire [93]). Races are serialized at the directory by either queueing the request or `NACK`ing the request. A request can only proceed past the directory when the directory becomes unblocked (the prior request has completed). Blocking protocols have fewer race transitions because they detect and resolve races early-on at the directory.

Blocking protocols sacrifice some concurrency by serializing requests at the directory. They also can lead to deadlock if blocking is employed at more than one level of the memory hierarchy. A hierarchical blocking protocol can be made deadlock-free by altering the protocol such that some blocking states become non-blocking states. Marty and Hill call these non-blocking states *safe points* in their two-level hierarchy [14]. Safe points must handle any action from the second-level at any time. The fewer safe points a protocol has, the more it resembles a blocking protocol and the simpler it is. A protocol that has few (or no) safe points is highly concurrent but also suffers from state-space explosion.

Token-Based Ordering. Finally, Token Coherence [94] is an order-less coherence strategy that uses greedy methods to maintain order; the requester that obtains the necessary number of tokens may proceed. In scenarios where two or more requests race for the tokens, requesters could deadlock waiting for each other's tokens. Token Coherence uses time-outs to detect these races and orders them with a *persistent request*. A persistent request preempts other conflicting requests, thereby breaking the deadlock and ordering racing requests. Token Coherence has no pre-preemptive mechanism to prevent or manage races, requiring it to resort to reactive methods (e.g. persistent requests and broadcast) to resolve races. In contrast, Atomic Coherence eagerly resolves races before requests are allowed to become active.

None of the ordering techniques described above eliminate races entirely, and for some, races can lead to livelock, deadlock, or starvation [100]. A request's path consists of a series of transitions that affect the state of the system. We argue that the later in the path that a race is detected, the more system state has been affected, and the harder it is to correctly resolve the race. Buses have straightforward race resolutions because races are detected at issue time (i.e. eagerly). At the other extreme, Token Coherence detects races when a request times-out and recovers with specially-designed mechanisms (i.e. lazily). Figure 3.1 shows where races are resolved in a variety of ordering approaches.



Atomic Protocols	Front-End of Coherence	During Coherence	End of Coherence/Time-Out
Atomic Coherence	Wildfire [93]	SGI-Origin 2000 [95]	Token Coherence [94]
Snoopy w/ Blocking Bus [96, 86]	Ring-Order [90]	SCI [97]	Token Tenure [98]
	Safe Points [14]	Gigaplane [99]	Timestamp Snooping [91]
		INSO [92]	

Table 3.1 **Cache Coherence Complexity Taxonomy.** Where race detection takes place along a racing-request's path. The closer to issue, the more eager the detection and usually the more simplistic the resolution. The further from issue, the lazier the detection and more complexity.

3.2.4 Protocol Verification Methods

Correctly specifying a protocol is difficult. Evidence suggests that verifying a protocol is even more difficult. Verification frustrations have led to specialized verification languages [101] and papers with desperate titles like “So Many States, So Little Time” [102].

Much of the complexity comes from the large size of the finite state machines, which are mostly made of race transitions (as indicated by our analysis in Section 7.4). Race transitions make up a small fraction of all observed transitions in the benchmarks studied. Because of their relative infrequency, race transitions have little impact performance, yet they impose significant design complexity and verification challenges. Race-induced design bugs may be masked away during most executions, but may reveal themselves at inopportune times. Thus, rigorous verification is necessary, but difficult. Verification falls into two categories, formal and informal methods.

Formal Methods. Formal methods have been used to validate real systems [103, 104]. Formal methods require the protocol be translated into some language, often mathematical. Such translation is prone to human error. For instance, the formal verification of the the Alpha EV-6 coherence protocol [101] found many bugs in the protocol, but most of the bugs it found were a result of errors in translating the informal documentation into formal specifications. Human translation errors are a major problem for coherence protocols, for they can occur at *any* point of translation in the process, between conception of the design to final hardware implementation. Finally, although formal verification may be a clean and relatively small representation (on the order of 400 to 1800 to lines of specification for TLA+ [105, 101]), it can also be time consuming; the EV-6 effort took a year to complete. Still, coherence bugs make it into shipping products [106].

Informal Methods. Random traffic generators are an informal method that may stress, but not exhaust, potential race combinations [107, 108]. For large protocols, like the Cray X1 with its 214 million states, random testing would be intractable [109].

Formal and Informal Combinations. Formal methods can be combined with informal methods to overcome the problems of mis-translation between the formal verification language and the verilog implementation, as proposed by Abts, Scott, and Lilja [102]. The authors propose using the formal verification model to drive the execution of the verilog model through the uses of traces.

Perhaps the best way to cull the massive problem of verification is to limit the number of states. This is what motivates our work on Atomic Coherence (Chapter 7).

Chapter 4

Methodology

This chapter presents our methodology. Section 4.1 presents our assumptions about the optical technology. We describe the experimental framework in Section 4.2.

4.1 Optical Assumptions

The area, latency, and power predictions discussed below target the year 2017 and the 16 nm process.

4.1.1 Area

Ring resonators are 4-5 μm in diameter and require 7 μm center-to-center separation. Waveguides are 0.5 μm in diameter and require 5 μm center-to-center separation. These dimensions meet the requirements discussed in Chapter 2.2.2.1 (page 13).

4.1.2 Latency

We assume it takes on the order of one cycle at 10 GHz (100 ps) to bring a ring resonator in-to/out-of resonance. We also assume that detection on a germanium-doped ring resonator can occur within this period. Our assumption lies in the range of other 16 nm estimates, which range from 20 ps to 400 ps [35, 75].

4.1.3 Power

The major contributors to power are the laser and the ring resonators. Laser power comes from a static off-chip source. The laser power must overcome losses due to EO/OE conversion inefficiencies as well as transmission losses in the waveguide. All ring resonators must be electrically or thermally adjusted (or "trimmed") to compensate for fabrication error. A ring is brought into resonance by further adjusting (or "biasing") its index of refraction.

Our laser power model uses a link-loss approach that accumulates losses along the optical path. Because laser power is static, it does not vary with system activity and must be configured for the lossiest path. Ring power is dynamic and does vary with system activity. We present results at minimum and maximum system activity. Our ring power model estimates power using a simple additive approach, by accumulating the power of individual rings.

Device	Value	Description
Ring Resonator:		
Trim	22 μW	Adjustment for fabrication error
Biasing	91 μW	“Biasing” into resonance
Modulation	474 μW	Modulation at 10 Gbit/s
Laser:		
Waveguide		
Single-Mode	1.0 dB/cm	Passing by a bank of rings
Multi-Mode	0.3 dB/cm	Passing between banks of rings
Non-Resonant Ring		
Insertion	0.017 dB	Loss at same λ or neighbor λ ring
Scattering	0.001 dB	Loss at non-neighbor λ ring
Resonant Ring		
Coupling	0.1 dB	Loss at same λ ring
Insertion	0.017 dB	Loss at neighbor λ ring
Scattering	0.001 dB	Loss at non-neighbor λ ring
Distribution		
Coupling	1 dB	Laser coupling loss
Beam Splitter	0.1 dB	Power distribution loss per-split
Ge RCEPD	3.0 dB	Detector $\gamma \rightarrow e^-$ efficiency
Efficiency	5.0 dB	Laser $e^- \rightarrow \gamma$ efficiency
Detected fJ/b	0.15 fJ	Detecting 1k e^-/bit
Splitter Ratio	5%	Power lost at broadcast split

Table 4.1 **Optical Device Power Consumption.**

We calculate power results (Section 5.6, 6.2.2.2, and 7.4.2.4) using the parameters in Table 4.1. The parameters are estimates from researchers at Hewlett-Packard Laboratories and belong to a photonic road-map that is based on prior literature, simulation, and experimentation.

4.2 Simulation

We used different simulators to evaluate our solutions. We designed the Interconnect Control simulator (Section 6) for very fine grain modelling of light propagation. We designed the Atomic Coherence simulator (Section 7) for full-system simulation and abstracted away some of the details of light propagation. Below we discuss commonalities and differences between the simulator setups.

4.2.1 Workloads

4.2.1.1 Descriptions of Workloads

The experiments subject the architectures to a combination of synthetic and realistic workloads that have interesting on-chip communication behavior. Realistic workloads came from scientific (SPLASH-2 [110]) and commercial (Wisconsin Commercial Suite [111]) suites. We also wrote a neural network microbenchmark for Atomic Coherence. The neural network consists of 3 levels, each with $\lfloor n/3 \rfloor$ nodes. All nodes on level i provide data to all nodes on level $i + 1$ in a feed-forward manner. Figure 4.1 shows a neural network.

Information about the workloads can be found in Table 4.2.

4.2.1.2 Running of Workloads

Synthetic workloads create traffic dynamically, while the scientific and commercial workloads use trace files and the neural network uses Pin [112].

Synthetic Workloads Synthetic workloads consist of one-way traffic (requests-only). Our model generates traffic that warms the system. After the system has been warmed, we follow Dally's measurement methodology [113] to measure it in steady state. We mark and measure a number of packets (100K). The run terminates when all of the marked packets have completed.

Trace-Driven Workloads The traces consist of cache misses. The trace-driven approach was chosen to make the many-core simulation tractable, but has the weakness that each trace captures only one of the many possible execution paths. Thus, a trace is always deterministic, making it impossible to study other paths of execution.

The traces came from two simulator infrastructures, COTSON and GEMS [114, 108]. In all cases, we ran similar configurations for the trace-simulator and our target-simulator. Traces of cache misses were collected along with

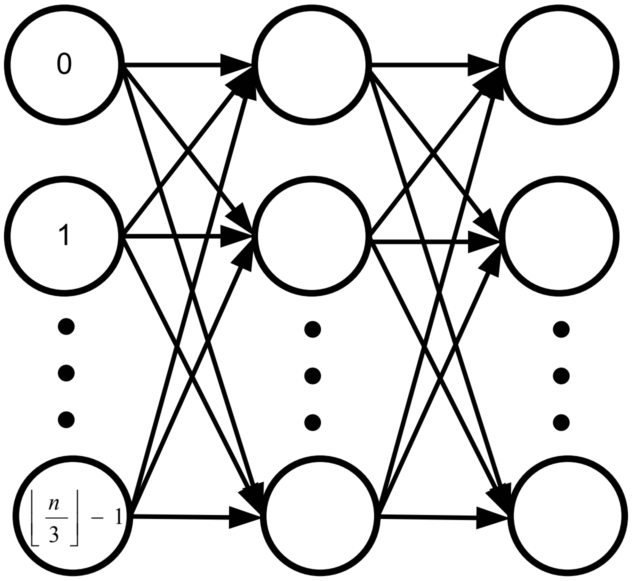


Figure 4.1 **Neural Network Example.** Information propagates in a feed-forward way. Nodes read input from the left, calculate the a dot product, and output the result to the right. Our implementation assumes a node cannot calculate a new value until all of the inputs are ready and all outputs have been read by the upper level (if such a level exists). We stripe the spinning, so all nodes do not begin by spinning on the same node.

load/miss, thread, and timing information. Timing information was used to adequately pad the trace and loosely preserve the original interleaving.

The Interconnect Control chapter (Chapter 6) uses COTSON traces, which contain L2 misses. The simulator in this section *does not* model coherence; each L2 trace-miss simply performs a request-and-response through memory. The Atomic Coherence chapter (Chapter 7) uses GEMS traces, which contain L1 misses. The simulator in this section *does* model coherence; each L1 trace-miss models coherence beginning at the L2. Both simulators avoid cold-start affects with cache-warming.

For each experiment, we warm the model and run for fixed number of misses (or trace entries). We ran for a varying number of misses in COTSON (70K-7M) and approximately a fixed number in GEMs (corresponding to ˜500M instructions).

Pin-Driven Workloads We use Pin for our neural network workload. Pin drives our neural network by running on native hardware and using dynamic instrumentation to supply our simulator.

The neural network has a lot of spinning, so we use transaction counting [111] to its measure performance. The simulator counts each feed-forward pass as a transaction and terminates after the maximum count has been reached (100 passes). We measure performance in number of cycles per pass.

4.2.2 System Setup

We evaluate our designs on a Corona-like many-core [19]. Our arbiter evaluation is true to the original Corona design, while our Atomic Coherence evaluation scales down the design.

We chose Corona because its data interconnect worked well with our proposals and because, as one of the first many-core optical proposals, it is currently the standard of comparison [36, 35, 75].

Below we describe how we modeled chip components and their configurations.

4.2.2.1 Components

Processors. We model simple single-issue in-order processors that are either single-threaded (for Atomic Coherence) or 4-way threaded (for Interconnect Control).

Data Interconnect. All nodes are fully-connected by high-bandwidth DWDM optical channels, as found in the Corona system [19]. Each node is provided with a dedicated single-destination, multiple-source communication channel to each node, called a Multiple Writer, Single Reader (MWSR) interconnect in the Firefly terminology proposed by Pan, et al. [56]. The MWSR interconnect is a common optical interconnect architecture because it deals well with unidirectional wave-pipelined technology [74, 19, 115, 67].

Benchmark	Dynamic	COTSON Traces	GEMS Traces
Synthetic			
Uniform	Uniform Random		
Hot Spot	All clusters to one cluster		
SPLASH-2			
Barnes		64 K particles	512 Bodies
Cholesky		tk29.O	
FFT		16 M points	
FMM		1 M particles	
LU		2048×2048 matrix	
Ocean		2050×2050 grid	2050×2050 grid
Radiosity		roomlarge	
Radix		64 M integers	
Raytrace		balls4	
Volrend		head	
Water-Sp		32K molecules	
Commercial			
Apache			2.0.43, 3200 clients
OLTP			TPC-C v3.0 16 users/proc
			IBM DB2 v7.2 EEE
SPECjbb			3-tier, 1.5 threads, 1.5 warehouses/proc
Other			
Neural Network	3-level, fully connected,		
	feed-forward. 100 passes.		

Table 4.2 **Benchmarks**

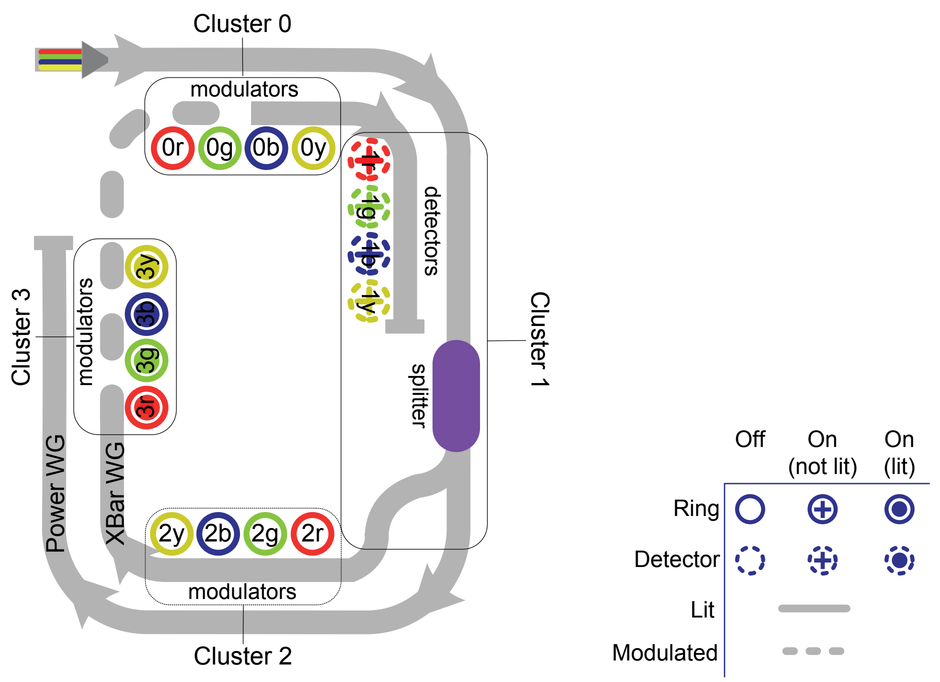


Figure 4.2 **Four Wavelength Corona-like Data Channel.** The home cluster (cluster 1) sources all wavelengths of light (r,g,b,y). The light travels clockwise around the crossbar waveguides. It passes untouched by cluster 2's inactive (off-resonance) modulators. As it passes by cluster 3's active modulators, all wavelengths are modulated to encode data. Eventually, cluster 1's detectors sense the modulation, at which point the waveguide terminates.

Any node may write to a given MWSR channel but only *one* node (the *destination* or *home* node) may read from the channel. Optical data packets traverse the channels in a wave-pipelined, or latchless, manner from a single source to a single destination. Packets are fixed in both temporal and spatial extent. Several short packets may occupy the same channel simultaneously in moving *slots*. Time of flight varies with differing source–destination distance.

When laid out, a channel forms a broken ring that originates at the destination cluster (the channel's *home* cluster), is routed past every other cluster, and eventually terminates back at its origin. Light is sourced at a channel's home by a splitter that provides all wavelengths of light from a power waveguide. Communication is unidirectional, in cyclically increasing order of cluster number.

A cluster sends to another cluster by modulating the light on the destination cluster's channel. Figure 4.2 illustrates the conceptual operation of a four-wavelength channel. Figure 4.3 provides a sample interconnect layout of a 64-node 64-channel system.

Caches. Interconnect Control (Section 6) models neither the caches nor coherence, and relies on the trace to dictate all interconnect traffic. Atomic Coherence (Section 7) models caches and coherence. L1s maintain write-through inclusion with the L2, and blocks are replaced in LRU order.

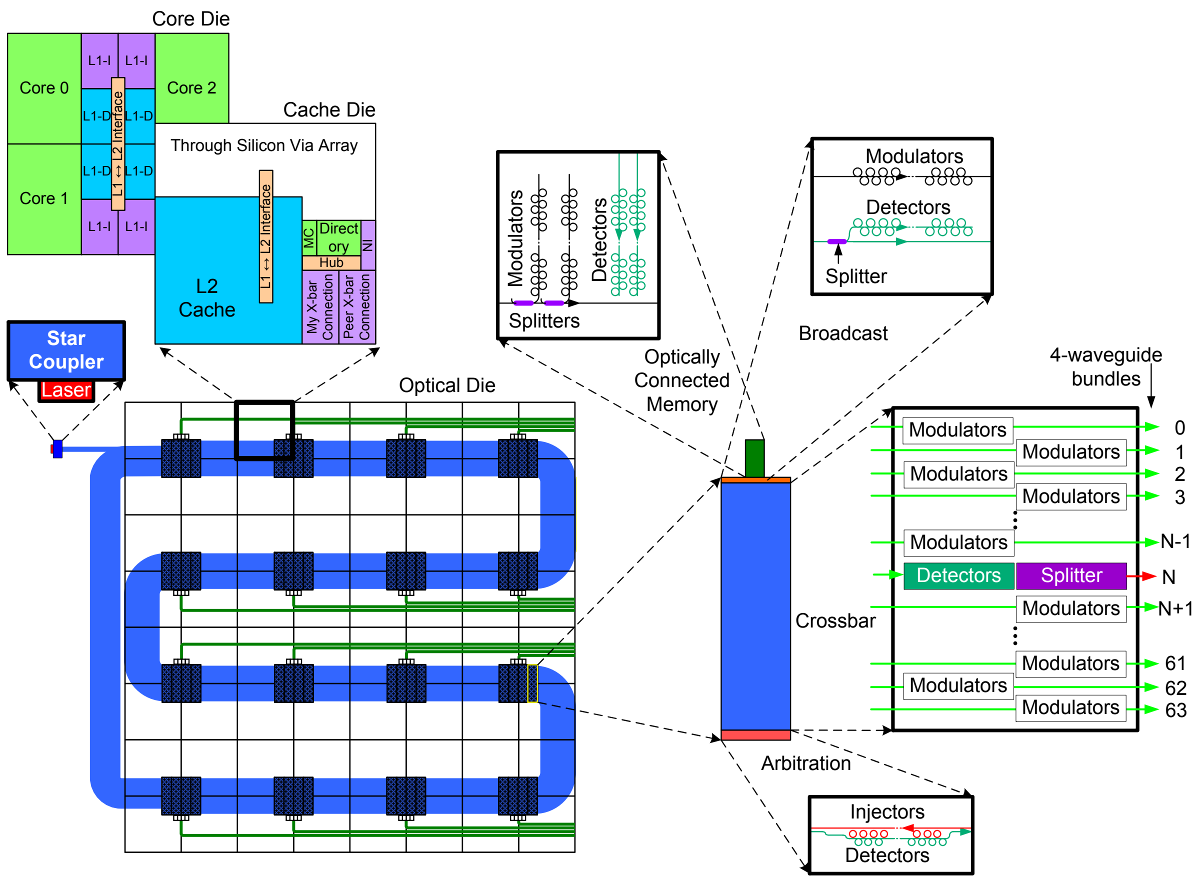


Figure 4.3 **Corona Layout.** Serpentine waveguides and resonator ring detail.

In Atomic coherence, we model variations of the the SGI-Origin 2000 protocol [116]. The SGI-Origin 2000 protocol is insensitive to interconnect ordering, so it is compatible with our interconnects, which have point-to-point ordering but do not exploit it. The SGI-Origin was designed for symmetric multiprocessor systems with the directory at memory. The SGI-Origin protocol is an invalidate protocol that has four stable states: Modified (`M`), Exclusive (`E`), Shared (`S`), and Invalid (`I`). We call this protocol the `MESI` protocol.

When there is a miss to a shared block, the SGI-Origin protocol fetches a copy of the block from memory. This works well when the directory is at memory, as is the case for the original SGI-Origin system. However, when we cache a portion of the directory on the chip, as our processors now allow, we can perform a directory lookup on-chip and possibly source the data on-chip, thereby avoiding memory altogether. Retrieving data on-chip whenever possible requires the addition of states. We add two states – the Owned (`O`) and the Forward (`F`) states. We call this improved protocol the `MOEFSI` protocol.

`O` refers to dirty-shared and `F` refers to clean-shared. These states are responsible for supplying new sharers with data. Whenever there is a read request to a block in either of these states, we assume the sourcing responsibility (the `O/F` designation) is passed to the newest sharer and the sourcer downgrades to `S`. This migration of responsibility to the newest requester fits in well with the Least Recently Used (LRU) replacement schemes at the L2s, keeping the block at the top of the LRU list with every new sharer. We do not allow silent eviction of blocks in the `O` and `F` state, allowing the directory to reliably source the data. An evicted `O` block, unlike an `F` block, is dirty and requires a writeback to memory.

Memory Subsystem. On-chip communication must be balanced with off-chip communication. We again look to photonics to provide high-bandwidth to memory. We assume simple fixed bandwidth and latency of the memory controller and channel.

Memory Model. We support a sequentially consistent memory model, using a directory protocol and an in-order processor that only issues a memory request when it is the oldest in the core.

Clocking. An optical data signal collects skew as is propagates through the interconnect. Like Corona, we use optical global distribution of the clock in order to avoid the need for signal retiming at the destination. A clock distribution waveguide parallels the interconnects' waveguides, with the clock signal traveling clockwise with the data signals. This means that each cluster is offset from the previous cluster by approximately n/t of a clock cycle, where n is the number of nodes on the channel and t is the serpentine's delay.

A cluster's electrical clock is phase locked to the arriving optical clock. Thus, input and output data are in phase with the local clock; this avoids costly retiming except when the serpentine wraps around.

	Details	
Component	Interconnect Control/Corona	Atomic Coherence
Nodes		
Cores	256, 4-threaded 5GHz	128, single-threaded 5 GHz
L1Is/L1Ds	256, 32 KB 4way 2 cycle	64, 32 KB 4 way 2 cycle
L2s	64, 256 KB	16, 2 MB 16 way 10 cycle
Directories	64 (pass-through)	16, 2048 sets 24 way 13 cycle
Memory Controllers	64	16
Interconnects		
On-chip Channels	64, 4 wg 1 cycles/msg/ch $T < 8$ cycles	16, 4 wg 2 cycles/msg/ch $T < 3$ cycles
Off-chip Channels	64, 2 wg 20 ns	16, 2 wg 90 ns

Table 4.3 **System Configurations.**

4.2.2.2 Configuration

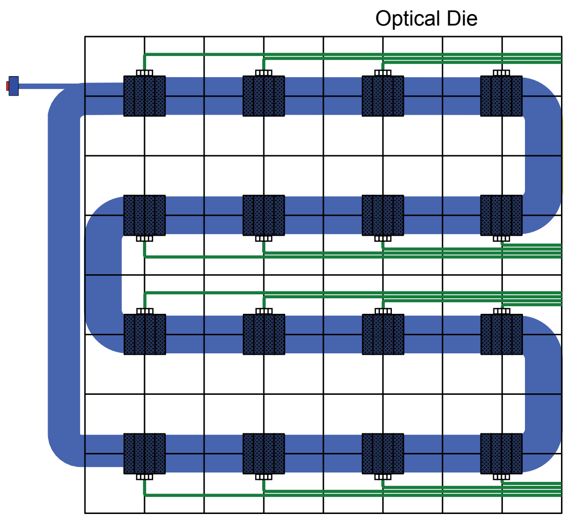
The two simulators are configured differently, as described below.

Interconnect Control. The Interconnect Control studies use an unmodified version of Corona for their evaluation. To stress the interconnect, Corona caches were scaled from 4 MB to 256 KB.

Atomic Coherence. The Atomic Coherence experiments scale the Corona interconnect from 1024 to 128 hardware threads to keep simulation time reasonable, a side effect of modeling coherence. Figure 4.4 and Table 4.2.2.2 show the impact of this scaling on interconnect layout and system configuration. Next we discuss the logic behind our configuration adjustments.

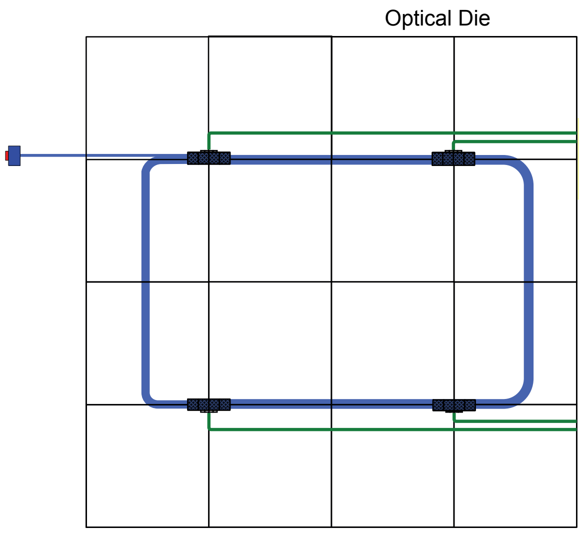
Because Atomic Coherence has fewer threads to model, there should be fewer nodes on the interconnect. We reduce the number of nodes from 64 to 16. So, while Corona requires a snake-shaped interconnect to connect 64 nodes, the interconnect we model can be achieved with a ring-shaped interconnect (see Figure 4.4). The difference in interconnect shape translates to a shorter light-path, allowing light to complete a revolution in less than 3 cycles at 5 GHz, compared to Corona's 8 cycles.

Atomic Coherence models 8 threads per interconnect node, while Corona has 16 threads. We assume this factor of two also affects bandwidth by a factor of two. To reduce the bandwidth by a factor of two, the model supports one transmission every two cycles (compared to Corona's support for one transmission every cycle). The model makes this adjustment by using Corona's 4 waveguide channels, but only clocking them at one-half their original rate (on the leading clock edge instead of both clock edges). Therefore, messages consume 4 waveguides (256 λ) in width and are two cycles long.



(a) Interconnect Control/Corona

(256 data waveguides)



(b) Atomic Coherence Setup

(64 data waveguides)

Figure 4.4 **Data Interconnects Used in Evaluation.**

4.3 Summary

All of our experiments evaluate on Corona-like systems. This chapter provides an overview of the Corona architecture and information about how we modified it. Finally, the chapter provides the necessary details of our simulators, workloads, and methodologies.

Chapter 5

Optical Tokens

The designs we propose are all based on optical tokens, or pulses of lights, that circulate until being atomically seized by an on-resonance ring resonator. In this section we describe and analyze the behavior of such tokens.

5.1 Overview

As we discussed in Section 3.1.1.2, a token symbolizes a resource, either physical or virtual. Tokens are useful in distributed system, where they may be used to guarantee mutual exclusive access to shared resources [117, 94, 118]. We use tokens to provide mutually exclusive access to shared communication channels (Chapter 6) and to guarantee mutually exclusive coherence transactions (Chapter 7).

Our optical tokens are simply pulses of light. The presence of light means the token is free, or available. An on-resonance ring will seize a free token by performing a destructive read on the signal as it passes. Conversely, an off-resonance ring will not seize the token and has no effect on the token's propagation.

Free tokens continually circulate in the optical domain. The latency and power advantages of keeping the token in the optical domain at all times during control come at some cost. Because optical tokens cannot be delayed, they cannot be inspected or modified as they are passing a node without removing the light from the waveguide and converting the signal to an electronic one. If the light is to be placed back on the waveguide, it must be delayed to the next clock cycle. To take advantage of the fast movement of tokens in optics, they should only be removed when a node will in fact use the token. This means that nodes must use local information alone to decide whether or not a given token should be removed.

Finally, tokens may be separated in space, time, and wavelength. Tokens on different wavelengths, but in the same waveguide, do not interfere with each other.

Next we discuss how nodes may seize and release tokens, or from the perspective of the ring resonator, how tokens may be detected and injected.

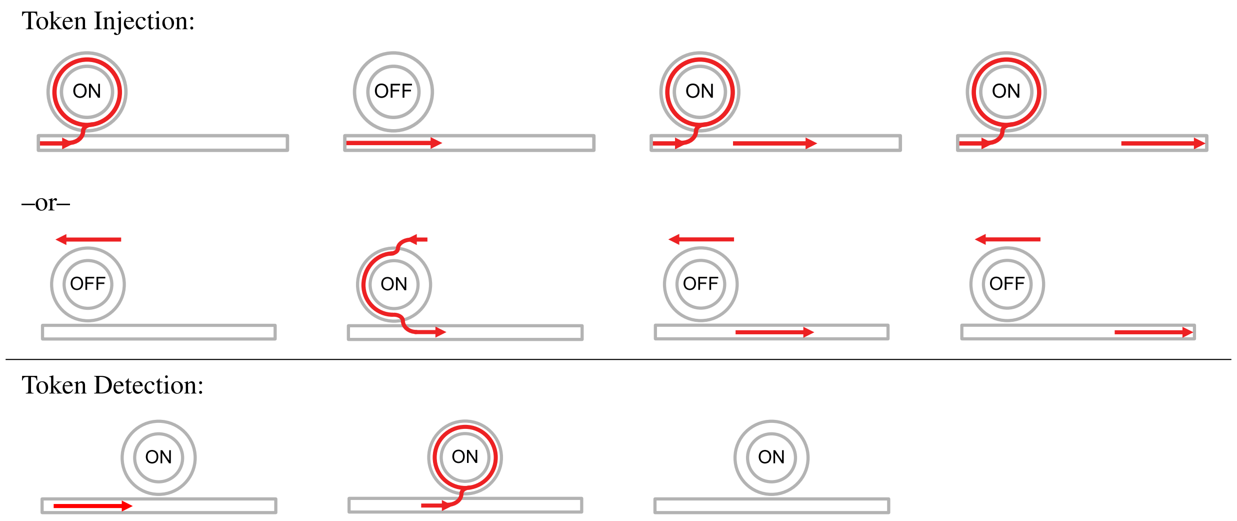


Figure 5.1 **Optical Token Injection and Detection.** Each image is a snapshot in time, progressing from left to right. Although two injectors are presented for completeness, we always implement the second injector design.

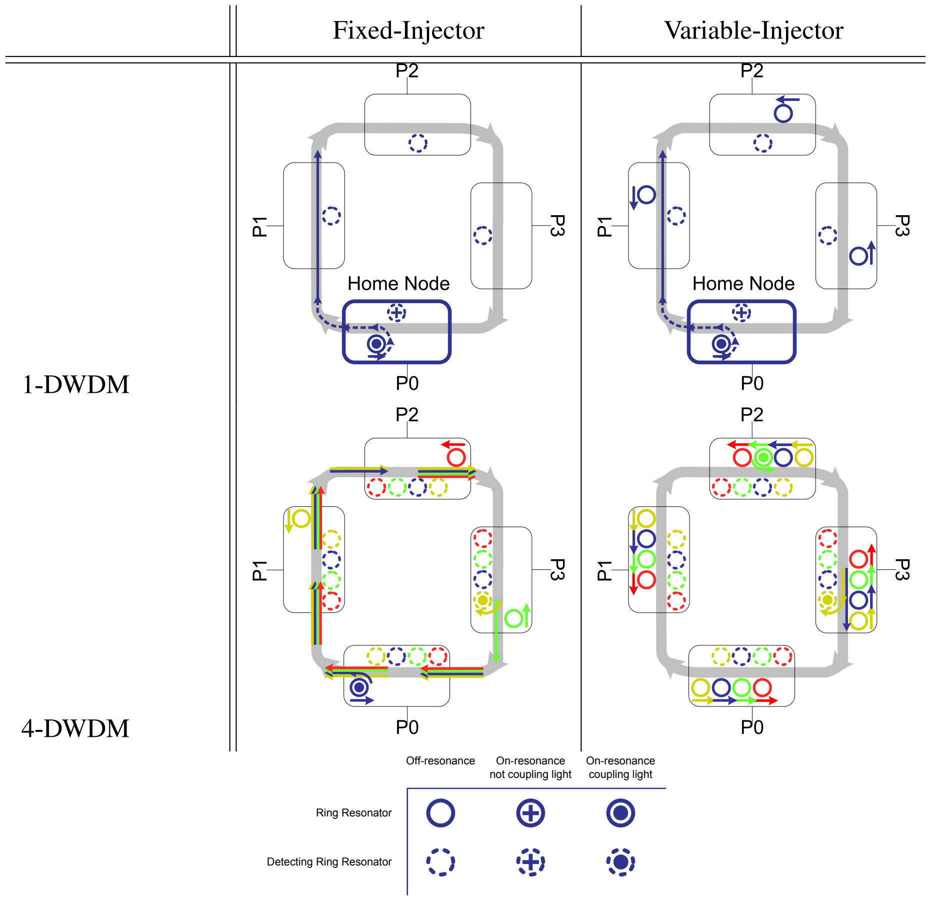


Figure 5.2 **Fixed-Injector And Variable-Injector Token Systems for 1-DWDM and 4-DWDM.**

5.2 Token Inject and Detect

Ring resonators repeatedly inject and detect tokens. Below we discuss the different ways token injection and detection can be achieved. We assume a single-wavelength single-pulse token, though the notion can be expanded in a straightforward manner for application to multiple-wavelength multiple-pulse tokens.

Token Inject. A ring may inject a token into circulation in one of two ways, as shown in the top of Figure 5.1.

The first way is to use a dedicated ring resonator as a gatekeeper to the token (see top row of Figure 5.2). The gatekeeper must be positioned at the most upstream position of a powered waveguide. The ring resonator is always on by default. The ring resonator injects the token by momentarily pulsing its ring resonator off and then back on, allowing a single token's worth of light to pass. This approach has limited applicability, as it only allows for a single fixed node to inject tokens.

The second way is to give every ring access to the power source by moving the power source to a waveguide separate from the token waveguide. With two waveguides, and each ring resonator coupled to both, any ring may inject a token by momentarily pulsing on-and-off its ring resonator. This approach allows more injector flexibility, but requires an extra waveguide and slightly more power. For generality, and because the difference in power is small, we always use this second approach.

Token Detect. A ring may detect a token when the token's path couples to the ring, as shown in the bottom of Figure 5.1.

An on-resonance ring indicates a node's intention of seizing the token; if a token passes it will be seized and detected. The coupled light circulates in the ring resonator, where the photoabsorption properties of germanium converts it into an electrical signal and passes it to the digital die, where it is effectively read as a 1. Eventually the light within the ring dissipates.

5.3 Fixed-Injector and Variable-Injector Systems

Putting together a combination of ring resonators that can inject and detect a token allows us to build a token-passing system.

In this work, we study two token-passing systems, which we call Fixed-Injector and Variable-Injector (Figure 5.2). Both systems circulate tokens on a ring-shaped waveguide. A system's *home* node is responsible for refreshing the token with an OEO repeat (requiring both an injector and a detector). In Fixed-Injector the home node is the sole generator of the tokens, but all nodes may detect the tokens. In Variable-Injector, any node may generate tokens and any node may detect tokens.

The choice between the two system styles depends on the application. If the application only requires a single node to inject tokens, then Fixed-Injector should be used because it saves power (see Section 5.6 page 47 for power results). If the application requires multiple nodes to inject tokens, then Variable-Injector may be better for its versatility. We use both systems in our arbiters (Chapter 6.1.1), but only have use for Variable-Injector in Atomic Coherence (Chapter 7.3).

5.4 Token Clocking and Refresh

Because several ring resonators on a waveguide can potentially detect the token, they must know when to do so such that they reliably detect a whole token. Seizing only a portion of a token makes token semantics unclear: What does it mean to seize a portion of a token? For simplicity, we require the token injectors and detectors to be synchronized on token boundaries. We assume all modulations are clocked in a source synchronous manner, as described in Section 4.2.2.1 (page 37). There, we described how a single optical master-clock waveguide travels in parallel with the other waveguides (e.g. a token waveguide) and collects skew as it passes the nodes. Because the clock travels the same direction and the same distance as the modulated signals, they both have equivalent skew and delay.

Tokens that do not have an origin point and an end point may circulate indefinitely. This creates two problems: clocking and dissipation. The clocking problem occurs when the token passes by the so-called “dateline,” or the point at which the clock-waveguides originate. Figure 5.3 shows the dateline scenario. If the path is not an integral number of wavelength in length, it must be held-back with an OEO buffering until it can be realigned with the clock. If the path is an integral number of clocks in length, then no retiming is necessary. However, even if the path is an integral number of clocks, it still may be necessary to perform an OEO repeat to refresh the token’s power. Acceptable refresh options include refreshing at the dateline or at some fixed point in the token’s path.

5.5 Latency

The latency to pass a token from node to node is the sum of the token’s injection latency (EO), detection latency (OE), and propagation latency. Early spice models indicate we can perform EO and OE in ˜100 ps. The propagation latency is the speed of light in silicon (˜1cm/100ps) multiplied by the distance between injector and detector.

The latency of our token systems scales with the length of the optical path. Since chip area is essentially fixed as core count grows with Moore’s Law, each core in an N-core chip has area proportional to $1/N$ and has length and width proportional to $1/\sqrt{N}$. Our waveguides follow a path that visits all N cores, like the one shown in Figure 5.4. Therefore, the path latency scales in length as $N \times \text{width of one core} = O(\sqrt{N})$. When $N = 64$ on a 576mm^2 chip with a 5GHz clock, flight time is 8 cycles.

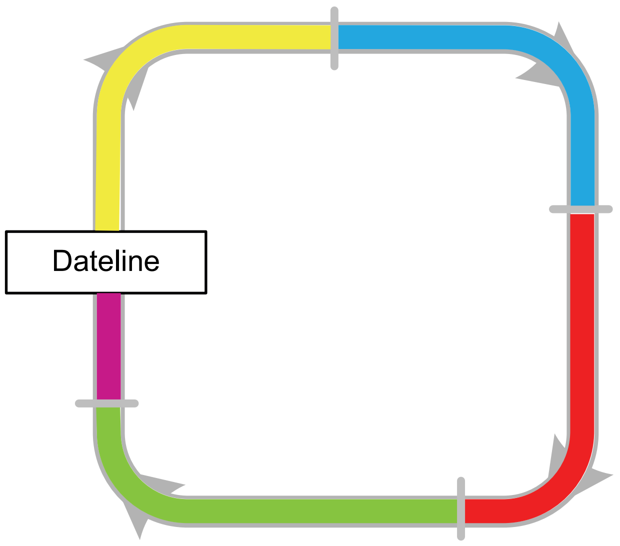


Figure 5.3 **Optical Token Dateline.** Retiming is necessary to compensate for the non-integral message path.

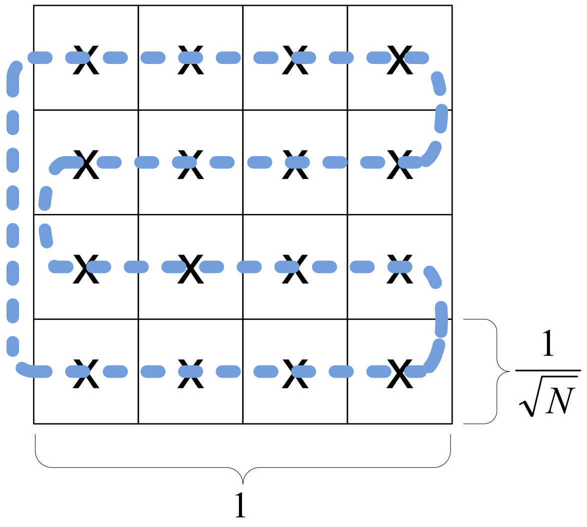


Figure 5.4 **Optical Token Scalability.** The token waveguides snake throughout chip, passing each of the N cores exactly once and having length $O(\sqrt{N})$.

5.6 Power

We study the power consumption of the two token systems, Fixed-Injector and Variable-Injector.

We further separate our power analysis into two parts, laser power and ring power. Ring power is dependent on activity factor, but is otherwise additive with the number of rings. Laser power is static and must be large enough to overcome worst-case losses in the waveguide. Even if a token only travels a portion of the length of the waveguide, it must have the power for its longest communication path.

5.6.1 Laser Power

In this section we study power, starting with a resonator-less waveguide and then adding resonators. We also study the power impact of varying the ring loss constants.

Waveguide with No Rings. Figure 5.5 shows how the waveguide losses grow with distance on a very simple waveguide that has no ring resonators and no curves.

The non-zero power at zero-length is for coupling to the chip, corresponding to a 30% wall-plug efficiency and, once on the chip, for splitting from a main power waveguide.

The chart shows how different degrees of DWDM consume power proportional to the length of the waveguide: 64 wavelengths consume twice as much power as 32 wavelengths, and similarly 32 wavelengths consume twice as much power as 16 wavelengths.

Because losses are measured in decibels (a logarithmic unit), power grows exponentially with length. Adding curvature to the waveguide adds losses, but if the curve has a large enough radius ($> 10\mu$m), then these losses should be negligible. Adding intersections add more significant losses (0.1 dB [119]). Curves and intersections are mostly a non-issue for the optical token systems we study because we use few curves and no intersections.

Waveguide with Rings. Let us add ring-resonators to the straight waveguide above to see how losses compound. A single off-resonance ring has a small effect on the strength of passing light, but multiple of these rings in series have a non-negligible effect.

Ring resonators are organized into banks. One bank contains a ring resonator for every possible wavelength in the system. Thus, with a center-to-center ring pitch of 7 μm, a bank can range in length from 0.0112 to 0.0448 cm for DWDM between 16 and 64. The banks are contained in nodes, which we model every 0.2 cm. We use these measurements (bank and inter-bank lengths) to calculate the single-mode and multi-mode laser power. The waveguide tapers to go from multi- to single- mode and untapers to go from single- to multi- mode. Single-mode is used when passing by banks of rings. It has a small core-size, making it easy to focus light for the high-precision rings. Multi-mode is used when passing between banks of rings. It has a wider core-size, making it less focused, but has lower losses than single-mode (0.3 dB/cm versus 1.0 dB/cm).

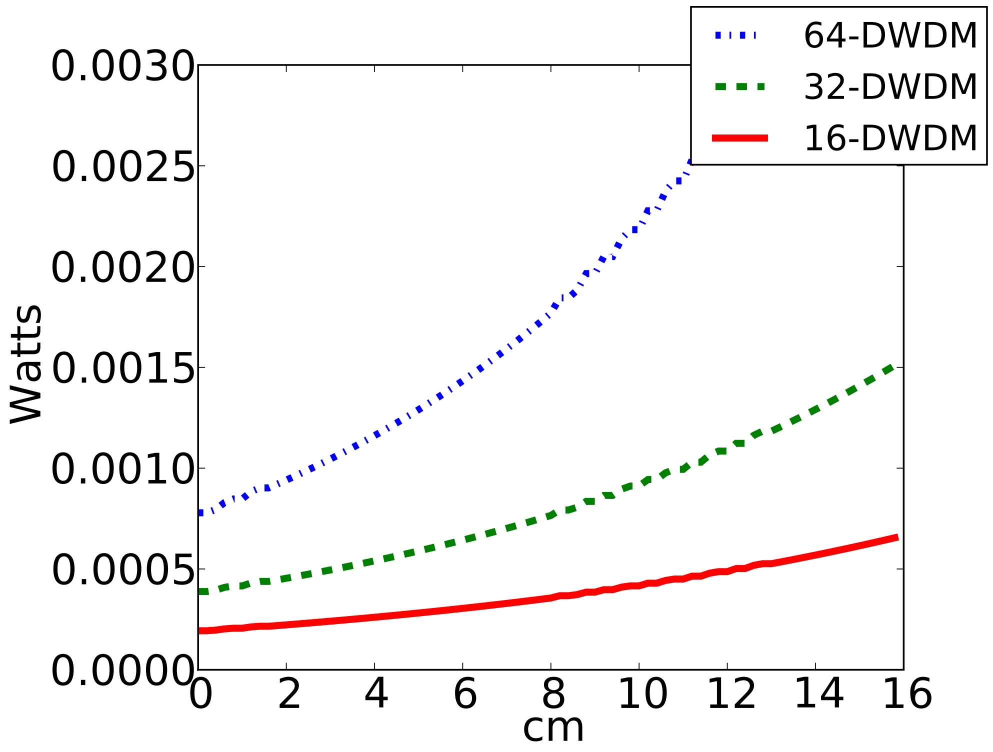


Figure 5.5 **Laser Power as Function of Distance on a Straight Waveguide with No Rings.** The waveguide has no rings and no curvature. N-DWDM means N wavelengths share the waveguide.

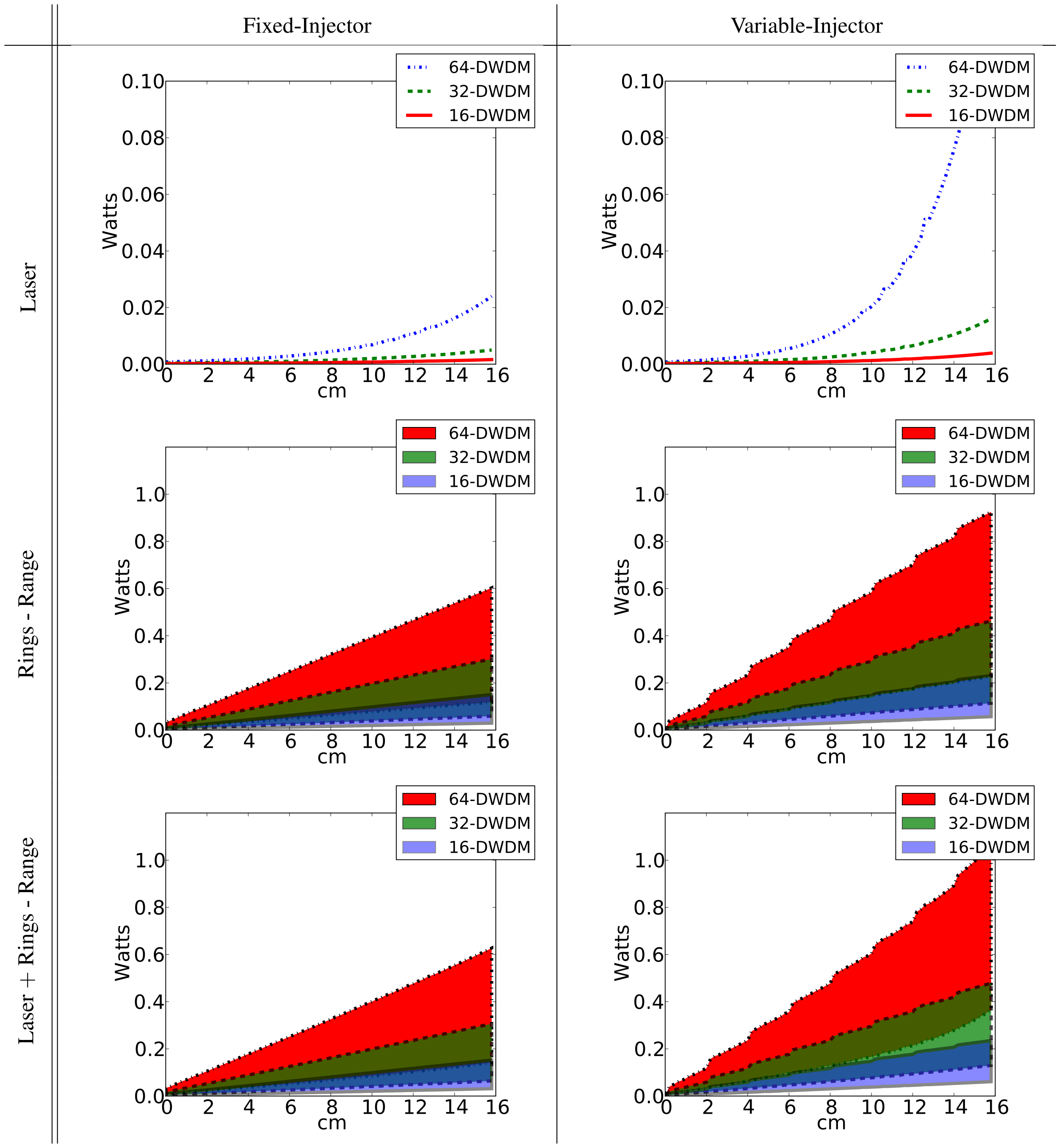


Figure 5.6 **Token Power Results for One Waveguide.** Banks of rings have 0.2 cm center-to-center pitch along a single straight waveguide. The top charts (*Laser*) include ring losses, which are due to passing by off-resonance rings. The middle charts (*Rings - Range*) show the activity-dependent operating range, which depends on tuning the rings. The bottom charts (*Laser + Rings - Range*) sum together the top charts.

Wavelength Matching	Off-Resonance	On-Resonance
$\lambda \in \{\lambda_i\}$	0.017dB	$>$ 10dB(diverting light)
$\lambda \in \{\lambda_{i-1}, \lambda_{i+1}\}$	0.017dB	0.017dB
$\lambda \in \{\lambda_0, \lambda_1 \ldots \lambda_{i-2}, \lambda_{i+2} \ldots \lambda_{n-1}\}$	0.001dB	0.001dB

Table 5.1 **Ring Resonator Losses.** The loss a given λ incurs when it passes by an off- or on- resonance ring at the given wavelength, a neighboring wavelength, or a non-neighboring wavelength. Decibels are from Table 4.1 (in Section 4.2 page 31); 0.017 dB is the insertion loss, 0.001 dB is the scattering loss, and 10 dB is the insertion loss.

The top pair of charts in Figure 5.6 show that the laser power is highly dependent on the losses of off-resonance rings. The higher the DWDM factor, the more off-resonance rings must be passed, and the more power consumed. Also shown is the Variable-Injector design, which requires more (off-resonance) rings than Single-Injector, causing it to consume multitudes more power. The rate of exponential growth in the most sophisticated design, the 64-DWDM Variable-Injector, is of concern, especially if estimates for loss constants grow. Unfortunately, the exact loss of an off-resonance ring is still unknown.

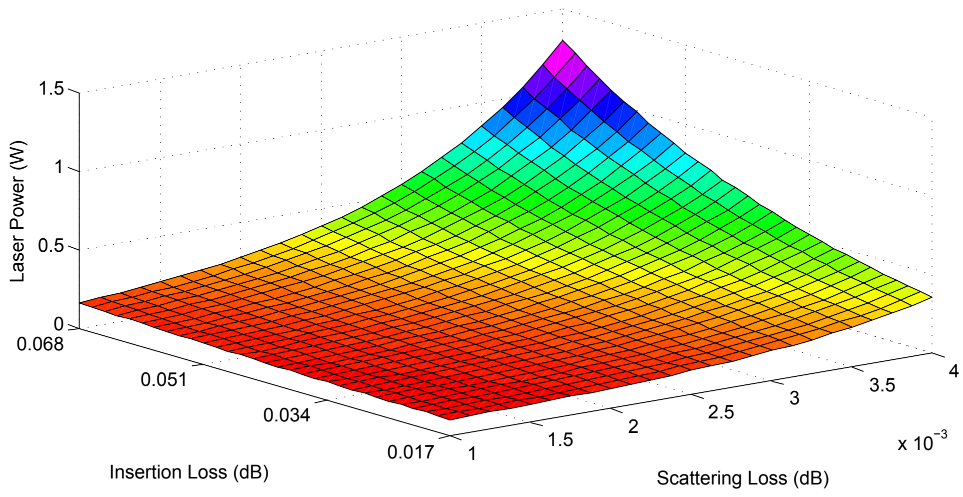
Sensitivity to Ring Losses. Figure 5.7 plots the sensitivity of our systems to different values of off-resonance losses. The different types of losses occur when passing by an off- or on- resonance ring of a near- or far- wavelength. The losses for all combinations of wavelength and resonator state are shown in Table 5.1.

Our token systems are most sensitive to scattering losses because the majority of wavelengths are non-neighboring. Insertion losses cannot be ignored either, as they have the potential to more than double the power required in the range shown. Large combined losses make the systems impractical.

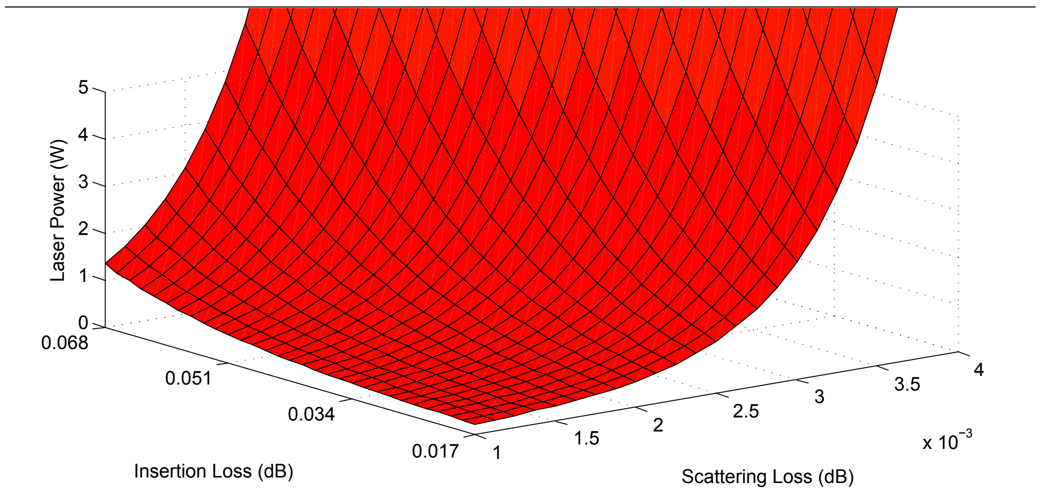
Thus, research into keeping these losses low is necessary for scalability. If the outlook looks bleak, it may be possible break up one monolithic fully-shared waveguide into many waveguides that are partially shared, but cumulatively allow all nodes access to the token. The most obvious way to implement such a system would be to have the token take turns on each waveguide, adding to the time it takes a token to make a visit to every node but possibly lowering the power cost.

5.6.2 Ring Power

Ring resonators require power to adjust their tuning. The middle pair of charts in Figure 5.6 isolates the power of ring resonators in our token systems. It shows the range of power, which is dependent on the activity factor as determined by the number of on-resonance rings (rings trying to detect the token) and the number of modulating rings (rings injecting tokens). As expected, the more rings in the system, the larger the operating range.



Fixed-Injector



Variable-Injector

Figure 5.7 **Laser Power Sensitivity to Scattering and Insertion Losses.** 16 cm 64-DWDM waveguide with 64 nodes. Fixed-Injector has 4160 rings. Variable-Injector has 8192 rings. Our work assumes scattering and insertion losses are .0017 dB and .001 dB (the graph's southern-most corner), resulting in a power budget of 0.11 and 0.24 Watts for Fixed-Injector and Variable-Injector respectively.

5.6.3 Laser and Ring Power

Total power of laser and rings is shown in the bottom pair of Figure 5.6's charts. It is simply the sum of the other charts in the figure. Under our assumptions, the largest contributor to power is the rings. Because Variable-Injector has approximately twice the rings as Fixed-Injector, it also consumes nearly twice the power. Also note that both systems have the possibility to use either low amounts or high amounts of power, depending on the activity factor.

5.7 Comparison To Traditional Token Rings

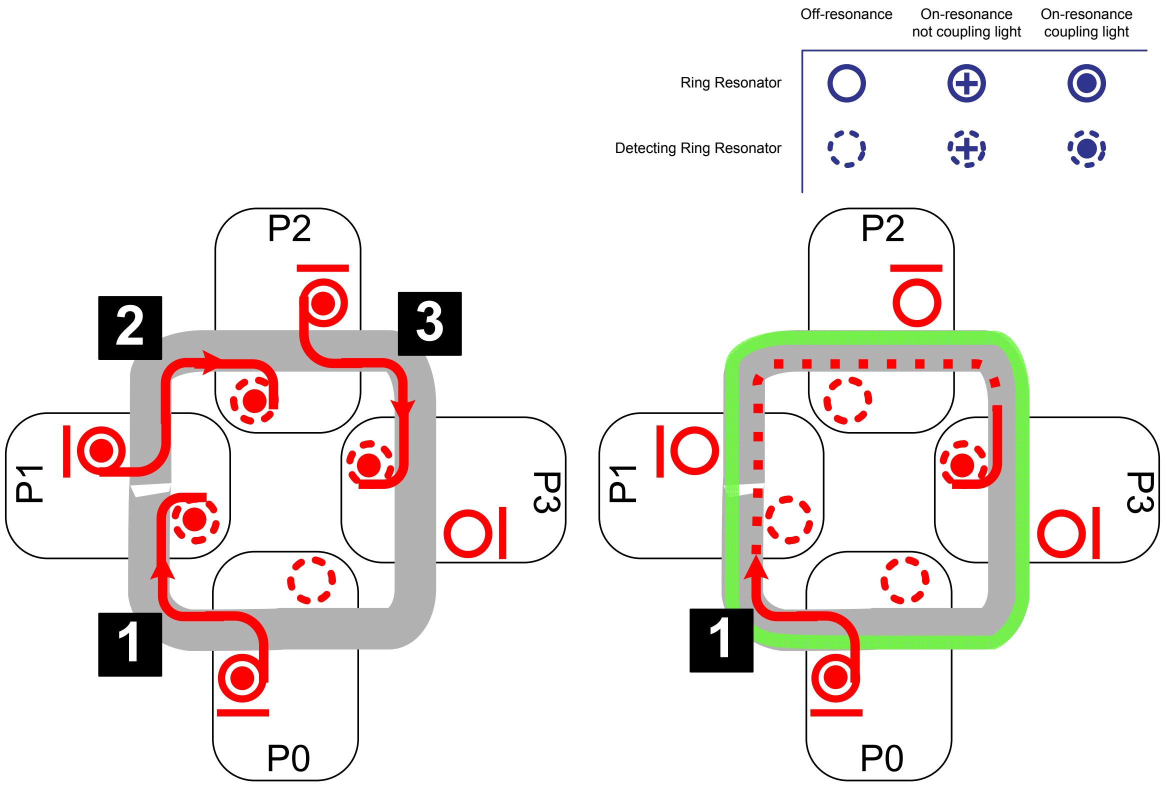
Our optical token systems are very similar to traditional token ring systems; both pass tokens, representing exclusive rights, around a ring. However, we make one key improvement over prior proposals. Optics makes it possible for a token to proceed, without repeat, from one requester to the next. Traditional systems repeat the token at every requester *and* non-requester.

This difference in token-passing approaches affects how performance scales. While traditional token system scalability is concerned mainly with hop count, optical tokens either scale with hop count (if there are many requesters) or path-length (if there are few or no requesters). Comparably equipped optical versions of the two approaches are contrasted in Figure 5.8. At worse, optical tokens should have the same latency as traditional tokens. At best, optical tokens should have much lower latency when tokens are passed directly between requesters at the speed of light.

5.8 Summary

Optical tokens form a theme shared by our many-core arbitration and coherence proposals.

This chapter summarized the basic operation of optical tokens and optical token systems. We proposed two optical systems, Fixed-Injector and Variable-Injector. As the names imply, Fixed-Injector allows only one node to inject tokens while Variable-Injector allows any node to inject tokens. Power results showed the range of both optical systems and indicated that DWDM and ring losses may be a major factor in determining scalability.



Traditional Token Ring Optical Tokens

Figure 5.8 **Traditional Token Ring Vs Optical Tokens.** The scenario uses Variable Injectors. P0 originates the token and P3 requests the token. In the traditional token ring (left), the token must be repeated at each node before finally reaching P3. With optical tokens (right), the token goes directly to node 3 without repeat.

Chapter 6

Photonic Interconnect Control

In Section 3.1 we motivated the need for fast, fair, flexible control of optical interconnects. Here we examine control systems that employ optical tokens in the arbiters of a data interconnect and show simple extensions that allow flow control and fairness.

While we have chosen to take an optical token-based distributed approach to arbitration, other approaches also exist (as discussed in Section 3.1 on page 18). Our choice to use optical tokens is scalable to many cores and has low latency, given we keep the token in the optical domain whenever possible.

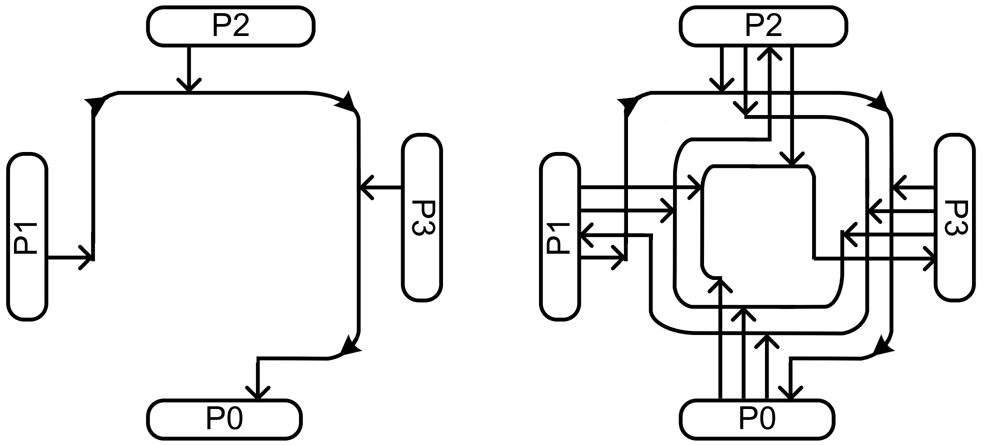
To take advantage of the fast movement of tokens in optics, they should only be removed when a node will in fact use the token. This means that nodes must use local information alone to decide whether or not a given token should be removed. So if, for example, a node is interested in two channels but can only transmit on one, it cannot claim both (by removing their respective tokens) and then release one without affecting (by delaying or consuming) the released token. Our protocols cope with these limitations.

6.1 Photonic Control

We address three aspects of photonic interconnect control: arbitration, flow control, and fairness.

We apply our single-channel arbitration protocols for control of an optical data interconnect that provides a dedicated, single-destination, multiple-source communication channel to each node, called a Multiple Writer, Single Reader (MWSR) interconnect [56]. In addition, our arbiters are general enough to be directly applied to optical broadcast buses [18, 19] and high-radix switches [120].

In an MWSR, *any* node may write to a given channel but only *one* node (the *destination* or *home* node) may read from the channel. Optical data packets traverse the channel in a wave-pipelined, or latchless, manner from a single source to a single destination. Packets are fixed in both temporal and spatial extent. Several short packets may occupy the same channel simultaneously in moving *slots*. Time of flight varies with differing source–destination distance. Figure 6.1 shows the MWSR topology (see Section 3.1.1.1 page 20 for discussion of other optical topologies).



One Channel Four Channels/Fully Connected

Figure 6.1 **Multiple Writer Single Reader Topology.**

6.1.1 Arbitration

In the optical arbitration schemes we describe below, a token represents rights to transmit on a channel, or a portion (slot) of the channel. We call the channel based and slot based protocols Token Channel and Token Slot, respectively and model them after token ring and slotted ring control systems [118, 70].

We let T denote the time of flight, in cycles, that it takes for an optical token to complete a full circuit from injection at its home node to detection at the same home node assuming it is not removed by an intervening node. In other words, T is the number of clocks it takes the token to complete a round trip along the blue path of Figure 5.4 (page 46).

6.1.1.1 Token Channel

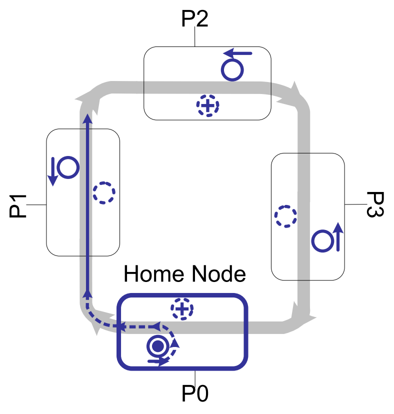
Optical Token Channel is inspired by the 802.5 Token Ring LAN standard [118]. Figure 6.2 shows the operation of Token Channel.

In Token Channel, there is a single token circulated per channel. The token is passed from requester to requester, entirely skipping nonrequesters that lie between. When a node removes and detects a token it has exclusive access to the corresponding data channel and may begin sending (one or more) packets some number of cycles later. The sooner it begins transmitting, the better the utilization of the data channel. In Token Channel, no more than one source can use the channel at any one time: the segment of the data channel from the home node to the token holder carries no data, while the segment from the token holder back to the home node carries light modulated by the token holder. The sender may hold the token (and use the channel) for up to some fixed maximum number $H \geq 1$ of packets. When there is pairwise sharing (e.g. the transpose and bit-transpose traffic pattern [113]), holding can help to send multiple packets in succession. However, for the traffic patterns we present (Uniform and HotSpot), H is less interesting, so we simply use $H = 1$.

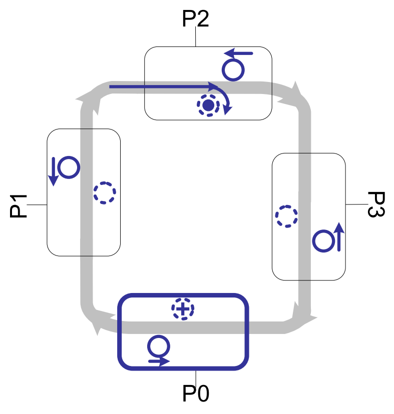
When a source first turns on its detector for an otherwise idle channel, it waits on average $T/2$ cycles for arrival of the token. When a single source wants to send a long sequence of packets on an otherwise idle channel, it transmits H packets at a time and waits the full flight time T for the reinjected token to return. The Token Slot arbiter of the next section reduces these token wait times.

6.1.1.2 Token Slot

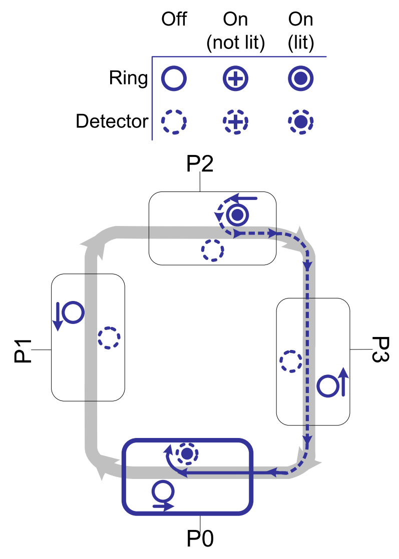
Token Slot is based on a slotted-ring arbitration protocol [70]. It divides the channel into fixed-size, back-to-back slots and circulates tokens in one-to-one correspondence to slots. One packet occupies one slot. A source waits for arrival of a token, removes the token, and modulates the light in the corresponding single-packet slot. The token precedes the slot by some fixed number of cycles which are used to set up the data for transmission. The slot is



P0 injects its token onto the arbitration waveguide. It passes by P1 unperturbed. P2 is requesting the token.

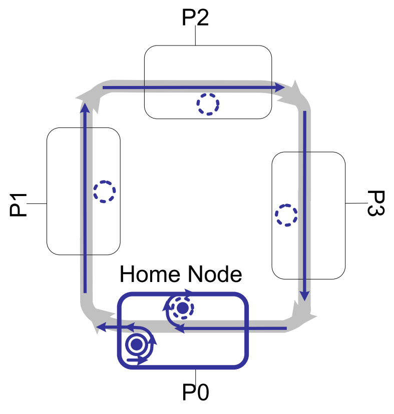


P2 seizes the token.

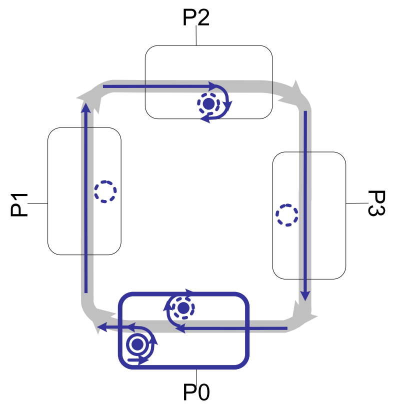


P2 reinjects the token, and the token travels back to the home node (P0).

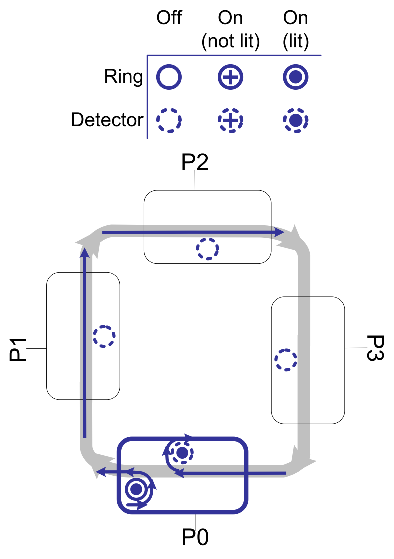
Figure 6.2 **Token Channel.** Arbitration for one data channel of an N data channel interconnect.



The home node (P0) injects tokens in one-to-one correspondence for the four slots.



P2 requests and seizes one of P0's token as it passes by.



The token P2 previously seized is represented by the absence of light traveling by P3.

Figure 6.3 **Token Slot.** Arbitration for one data channel of an N data channel interconnect.

occupied (by modulated light, *i.e.* data) until it gets to the destination, which removes the data, freeing the slot; the destination reinjects a fresh token along with new unmodulated light.

Compared with Token Channel, Token Slot reduces the average source wait time significantly and increases channel bandwidth utilization in our experiments. In the single-source scenario above, the one active source can claim all tokens, using the channel continuously at full bandwidth. In Token Channel, the token holder is the only possible sender (there is only one token in the system), whereas in Token Slot, there are multiple tokens, and there can be multiple sources simultaneously sending at different points on the waveguide.

Unlike Token Channel, which requires a Variable-Injector token system so that each node may have reinjection capabilities on the arbitration wavelength, in Token Slot only the destination needs reinjection capability. Thus, Token Slot may use a Fixed-Injector token system, which requires fewer resources and less power (see Section 5.3 page 44 for more discussion on both token systems).

6.1.2 Flow Control (FC)

Arbitration for the interconnect is only half of the story when it comes to communicating data from source to destination. Flow control is the other half of the story: it guarantees that an outgoing packet has a receive-buffer entry allocated for its arrival at the destination. By managing flow rate, it prevents buffer overflow at the destination, avoiding the complexity and overhead of negative acknowledgements and retransmissions (see Section 3.1.2 page 22 for more discussion of flow control techniques). In this section, we add credit-based flow control to the Token Channel and Token Slot protocols.

The idea is simple: the home node emits credit-filled tokens to communicate the number of available entries to source nodes. A node that removes a credit-bearing token (conveying one or more credits) is guaranteed that there are available entries at the destination.

Token Channel + FC. Token Channel encodes flow control information in the token. The token is enlarged to contain a binary encoding of the number of buffer entries available at the home node. When a node detects the token, it only transmits on the data channel if there are credits available. It then marks its reservation by decrementing the number of credits in the token, and reinjects the diminished token. The token eventually returns home, at which point the home node increments the credits to reflect entries that have become available since the token's last visit. The home emits the enriched token.

Token Slot + FC. Token Slot encodes flow control information simply by emitting a token only when a buffer entry is available. Thus, each token signifies a single credit and, if a node removes a token, it has also reserved an entry at the destination. When the entry is relinquished, the home emits a token.

6.1.3 Fairness

To be useful, a protocol must provide some guarantees of fairness. At a minimum, a fair protocol will not starve any contending user. We show here how to modify both Token Channel and Token Slot to provide fairness while retaining the advantages of fully optical implementations.

In the two protocols of the previous section, the home node emits credits. Nodes close to home have priority over nodes farther downstream in obtaining them. If emitted tokens do not have enough credits to reach distant requesters, these requesters risk starvation. This problem with token protocols is well known; it has been addressed in electronic systems with modifications that cause relatively well served senders to sit on their hands for awhile and give someone else a chance [74]. Here we modify the optical token protocols discussed above, retaining their latency advantages, and providing, by a similar back-off mechanism, fair treatment to under-served senders.

Suppose that the aggregate demand for a channel's bandwidth is less than both its inherent maximum bandwidth and the rate at which the home node frees buffer slots. Simple protocols, for example Token Slot, can fairly satisfy all this demand. The fairness issue surfaces when the aggregate demand by all senders exceeds this maximum achievable service rate. (In practice, full input buffers will throttle the senders so that their aggregate packet insertion rate approaches, but never exceeds, this rate.)

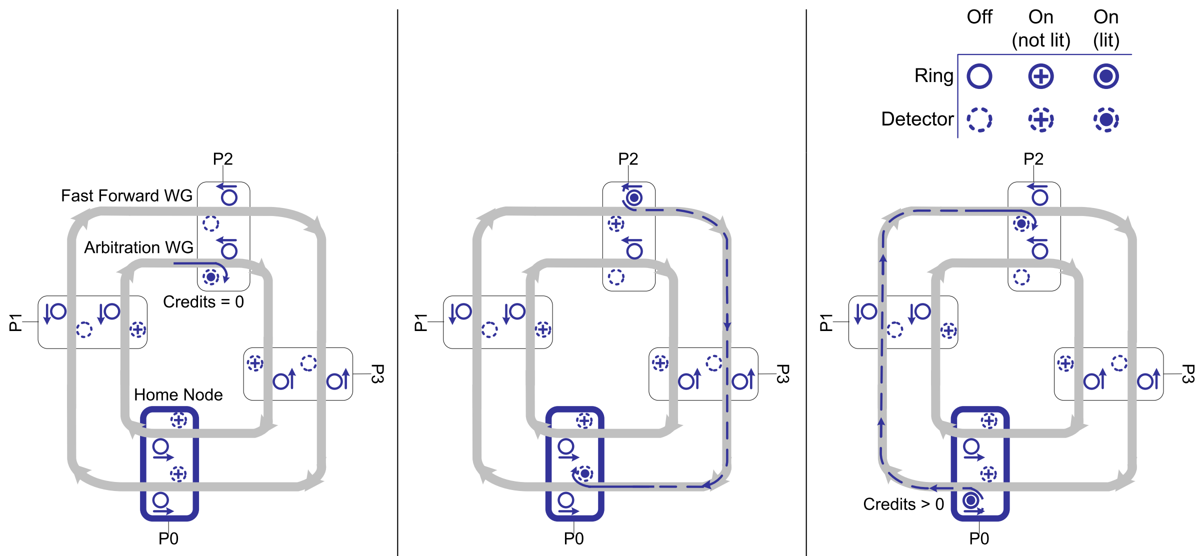
The max-min protocol [121] is a well-accepted notion of what the goal should be in this situation. In max-min, senders that need little service get all they can use. All remaining senders, which get somewhat less service than they might be able to use, get equal service.

Max-min is a worthy goal, but is difficult to achieve exactly in an online, low latency, distributed arbiter. There is no generally accepted metric for measuring the deviation between an achieved service profile and the max-min goal. We instead present the data visually, providing a clear picture of the extent to which we have approximated the max-min goal.

6.1.3.1 Fair Token Channel

Fair protocols such as *i*SLIP [65] implement rotating priority schemes that distribute service in some sort of cyclic manner to competing requesters. We propose to mitigate the unfair behavior of Token Channel with Fast Forward (FF) tokens that implicitly do the same sort of approximately cyclic service allocation when this is needed. When the home node injects an FF token (rather than a regular token), the token travels in its own waveguide, bypassing previously served senders. At some point along the waveguide (as described below), the FF token changes into a regular token and is available for arbitration. By fast forwarding these FF tokens downstream, nodes far from the home node are given higher priority than nodes close to home.

FF tokens are emitted onto a FF waveguide, which is concentric with the arbitration waveguide. The FF waveguide is used to both notify the home node that a node is starving *and* to supply a credit-filled token to that starving node.



P2 wins P0's token from the arbitration waveguide, but does not use the channel because there happens to be no credits left.

P2 fast forwards the token to P0, past any other possible outstanding downstream requesters (*e.g.* P3). P2 also activates its detector on the FF waveguide.

When credits become available again, P0 injects its token onto the FF Waveguide. P2 is expecting the token, as shown by the active detector. It removes the token, sends, and function returns to normal.

Figure 6.4 **Token Channel with Fast Forwarding**

When a potential source node removes the token and finds it empty of credits, it declares itself to be a victim of starvation. It does not reinject the empty token on the arbitration waveguide, but rather puts it on the FF waveguide (see Figure 6.4). Only the home node activates its detector, and so the token quickly returns (in at most T cycles) to the home node, which replenishes it with any newly available credits. Moreover, the arrival of the token at the home node via the FF channel causes the home to send the token back out again on the FF channel. The node that discovered the empty token will have activated its FF channel detector, so the replenished token will take a direct route back to that node. This not only removes the credit bias in favor of nodes near the home node, but has the added bonus of reducing the time needed to refresh an empty token with new credits. Our experiments do show a significant bandwidth improvement compared with simple Token Channel from this effect. Note that the opportunity to remove an empty token propagates cyclically, which is a key to the fairness of Channel FF.

6.1.3.2 Fair Token Slot

The simple Token Slot protocol is inherently unfair: it gives higher priority to nodes closest to the home. To ensure fairness we must make sure that every node having sufficiently high demand is treated nearly equally. Proposed electronic protocols do so using a SAT token that traverses in the direction opposite the token path and moves at most one node per clock [74]. In this section, we provide a low latency protocol, Fair Slot, that achieves this goal in a fully optical implementation. As with Channel-FF, Fair Slot detects starvation (a node that is not receiving adequate service) and then services all under-served nodes, once each, in some sequence before returning to normal function.

Here we describe the implementation of Fair Slot on a single channel (shown in Figure 6.6). In an MWSR system, each channel independently implements this algorithm.

The essential idea is to switch between simple Token Slot and an alternative that allocates credits to senders in a cyclic manner. During periods of little load, we use the simple protocol. When higher load causes some senders to receive less service than they need, the protocol switches to providing fixed quanta of credits and bandwidth to the under-served senders, once each cyclically, before returning to the simple protocol.

In Fair Slot, senders can be in one of three possible states: satisfied, hungry, and suspended. A sender must traverse the state diagram from satisfied to hungry to suspended cyclically (as shown in Figure 6.5). The entire channel goes through alternating phases of plenty and famine, the famine state being triggered by notification to the home node that there is at least one hungry sender. The presence of a hungry sender is communicated back to the home node using an optical `NOR` (any hungry node removes all the light from the hunger waveguide) allowing the home node to detect whether or not there is any hungry sender. When a hungry sender is detected by the home node, the channel enters the famine mode. The channel state of famine *vs.* plenty is communicated to the nodes using a separate broadcast waveguide so that all nodes can read it with no delay or reinjection.

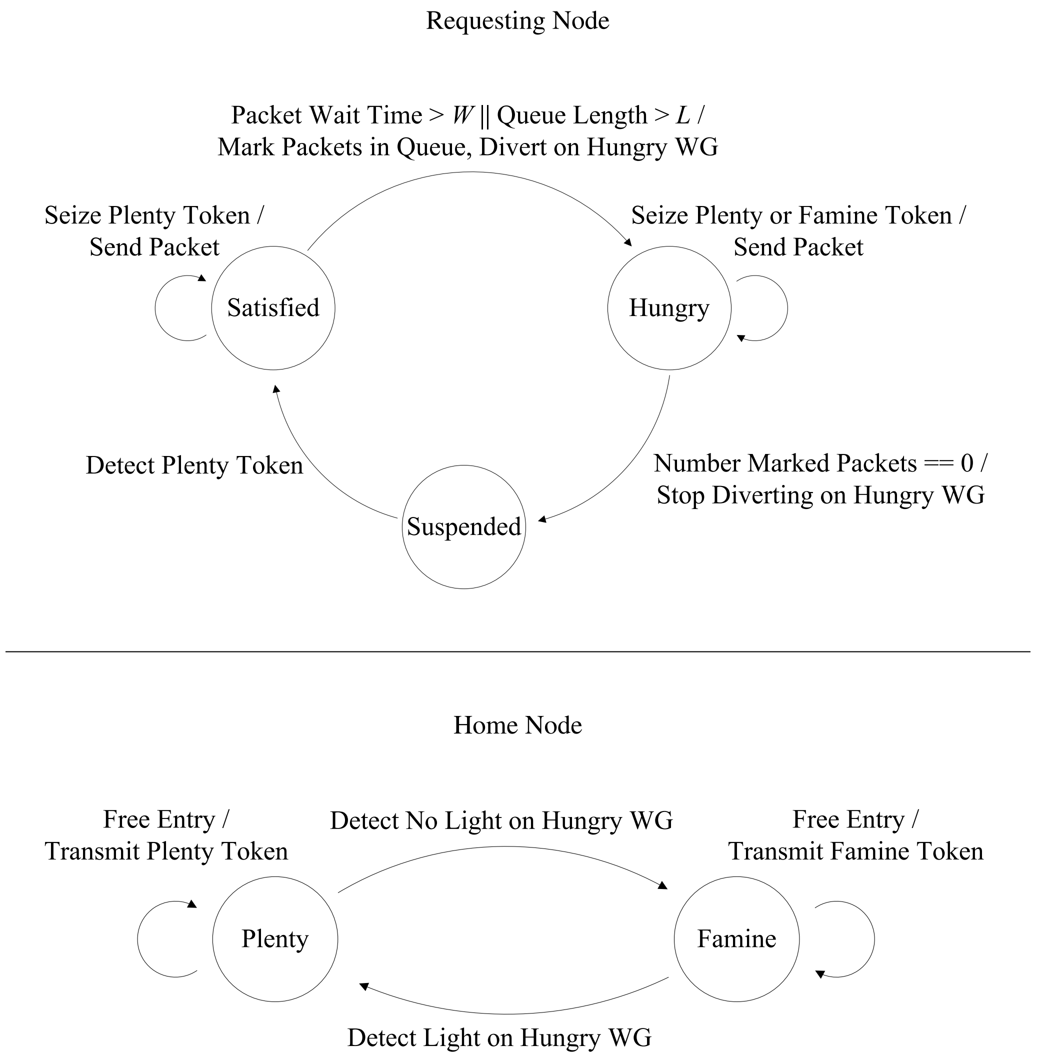
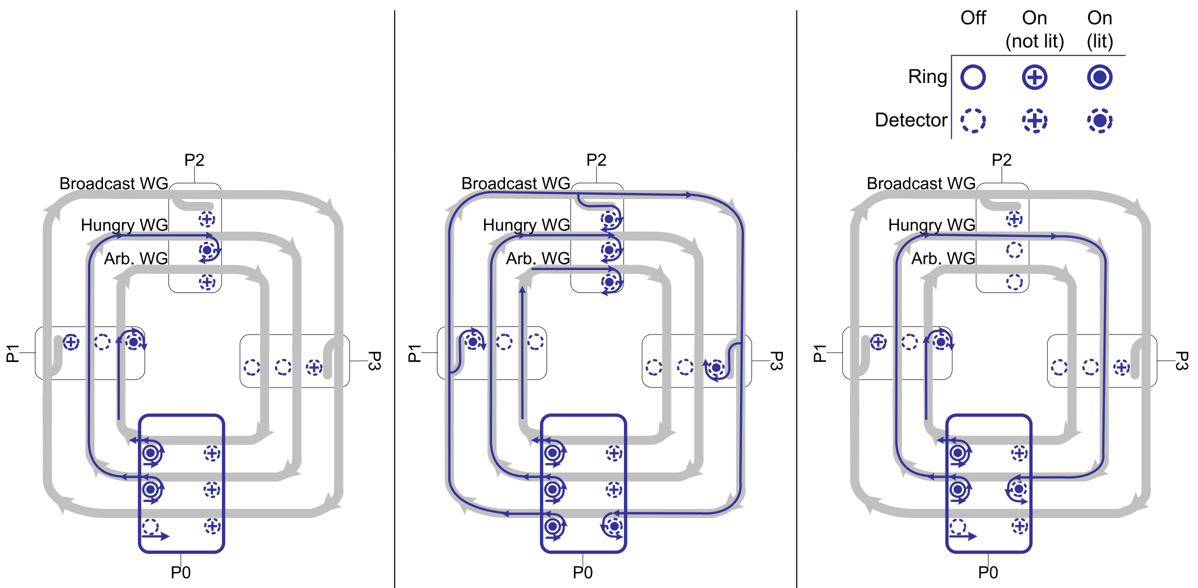

Requesting Node
Packet Wait Time > *W* || Queue Length > *L* / Mark Packets in Queue, Divert on Hungry WG
Seize Plenty Token / Send Packet
Seize Plenty or Famine Token / Send Packet
Satisfied
Hungry
Detect Plenty Token
Number Marked Packets == 0 / Stop Diverting on Hungry WG
Suspended
Home Node
Free Entry / Transmit Plenty Token
Detect No Light on Hungry WG
Free Entry / Transmit Famine Token
Plenty
Famine
Detect Light on Hungry WG

Figure 6.5 **Fair Slot Finite State Machine.**



The system is in Plenty mode. P1 is blocking P2 from winning P0's tokens. P2 diverts on the Hungry WG.

P0 detects no light on the Hungry WG and broadcasts on the Broadcast WG, putting the system into Famine mode. Node P1 passes on all famine tokens, allowing P2 to seize a token.

P2 becomes satisfied and releases the Hungry WG. P0 detects light on the Hungry WG, and stops broadcasting, returning the system to Plenty mode. Execution returns to normal.

Figure 6.6 **Fair Slot Implementation.**

Arrival of the famine signal changes the local (per node) channel status from plenty to famine. We call a token a famine token or a plenty token according to the accompanying channel state. Only a hungry node may grab a famine token. Other nodes (both satisfied and suspended) let them pass by. Any node can take a plenty token.

A satisfied node becomes hungry according to a local criterion. This can be the presence of an old packet (a packet is old if it has been in the system longer than some threshold) or when its input queue length exceeds some threshold. In either case, there will be an upper bound, L, on the number of packets in the queue of a node that becomes hungry.

When a node enters hunger for a given channel, it marks all packets in its input queue for that channel. A hungry node grabs any arriving tokens until these marked packets are flushed. Any packets that arrive after the onset of hunger can join the queue, but they aren't sent during this period of hunger. Once it sends the last of the marked packets, the node de-asserts hunger and goes into suspended state.

A node in suspended state passes on all famine tokens. When the first plenty token arrives, the node transitions to satisfied; it can also grab the token if it wishes. On the next clock it can transition to hungry if it meets the criterion.

No node can remain forever hungry. To see this, suppose node h becomes hungry. It asserts its hunger and this signal must reach the home node, which transitions to famine mode if not already in it. By the current famine we shall mean the epoch of consecutive cycles, including the clock cycle at which h's hunger signal arrives at home, over which the home node emits famine tokens. The famine tokens of the current famine may be taken by other hungry nodes upstream of h. No one of these will take more than L of these tokens in the current famine, as any node that takes L must have flushed its buffer (keeping newly arrived packets in the queue) and hence have gone into suspension. Node h will therefore eventually receive enough packets to flush the marked packets from it input queue, suspend, and de-assert its hunger.

No node can remain forever hungry. To see this, suppose node h becomes hungry. It asserts its hunger and this signal must reach the home node, which transitions to famine mode if not already in it. By the current famine we shall mean the epoch of consecutive cycles, including the clock cycle at which h's hunger signal arrives at home, over which the home node emits famine tokens. The famine tokens of the current famine may be taken by other hungry nodes upstream of h. No one of these will take more than L of these tokens in the current famine, as any node that takes L must have flushed its buffer (keeping newly arrived packets in the queue) and hence have gone into suspension. Node h will therefore eventually receive enough packets to flush the marked packets from it input queue, suspend, and de-assert its hunger.

No node returns to hungry state before all nodes that were hungry when it became hungry leave hunger. To see this, note that the transition from suspended to satisfied requires the arrival of the broadcast that the system has transitioned to the plenty state. The home node generates this signal only when the or-ed hunger signal goes low. This cannot happen until all nodes that were hungry when the given node became hungry have suspended and de-asserted it.

When the last node goes from hungry to suspended and de-asserts hunger, there is a delay of up to T cycles before home gets the word. The famine tokens generated by home will be wasted as there are no hungry senders left to take them. This has a small impact on utilization.

6.1.4 Multiple Channel Control

We evaluate the proposed optical arbiters for a $N = 64$-node, optical MWSR interconnect. The MWSR requires 64 arbiters to arbitrate for the 64 channels. In systems using DWDM, each data channel would be assigned a unique wavelength, and all could share the same arbitration waveguides. Examples of fully connected MWSR arbiters are shown in Figure 6.7. Here we consider some engineering and scalability issues that arise when arbitrating for several data channels in parallel.

Virtual output queues (VOQs) allow sources with requests destined for differing destinations to make these requests independently [122]. All our experiments employ VOQs. A node may *nominate* a subset of its non-empty VOQs for arbitration. A node that nominates and wins all N tokens may not be permitted (for power reasons) to simultaneously transmit on all N. Each token beyond the limit is an extra win. In Token Channel, each extra win sacrifices the immediate transmission opportunity; the requester can hold the extra tokens until it can transmit, or can reinject them; in either case this causes a data channel bubble. In Token Slot, each extra token won causes the corresponding slot to go unused. To prevent this, the number of nominated requests may be limited to m. Bounding $m < n$ reduces the likelihood of over-winning. However, over-winning is only harmful at high loads; at low-loads, the bandwidth would have likely been wasted anyway.

For the system we evaluate m is limited by the number of buffer entries (8 requests, 8 responses). This means that even at high load, a node will have at most 16 nominations. Our experiments indicate that optimal performance is achieved when we permit a node to make up to $m = 16$ nominations, but set its transmissions quota to two. A statistical analysis of Token Channel that assumed a random distribution of N tokens across all slots and a random distribution of a node's N' requests came to the same conclusion: In a 64 node Token Channel system ($N = 64$), if a node makes 16 nominations ($N' = 16$) it can probabilistically expect to win two; if it nominates 8 ($N' = 8$) it is can expect to win one.

6.2 Evaluation

6.2.1 Experimental Setup

We model Corona's MWSR 5 GHz interconnect. It has single cycle, cache-line-sized packets and optical path length T = 8 cycles. More discussion the experimental setup can be found in Section 4.2.2 page 4.2.2.

We provided each node with a modest number of output buffers (8 request, 8 response) and input buffers (16 shared). Having few buffers saves on energy but also puts increased pressure on timely flow control. Based on our

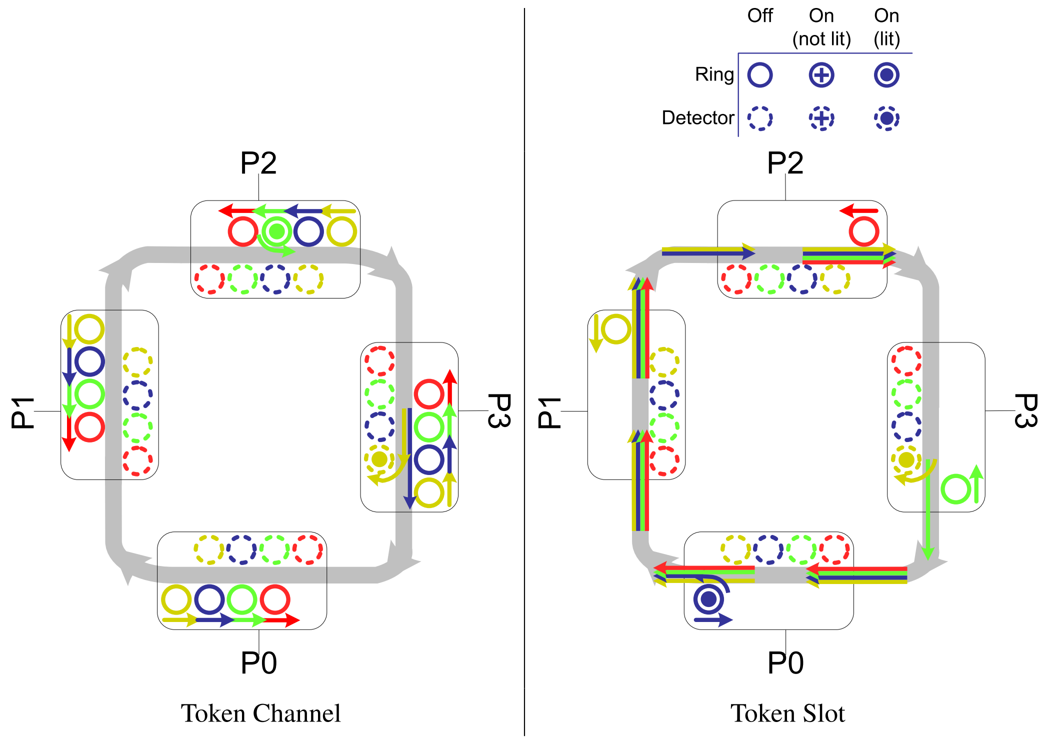


Figure 6.7 **Arbiters for 4-channel MWSR.** Four wavelengths arbitrate for four data channels (not shown).

discussion of nominations in the last section, we set the maximum number of concurrent nominations to 16 (the maximum output buffer size) and the number of simultaneous transmissions to two.

In order to test the proposed optical arbiters under a variety of traffic conditions, we use the 1024-threaded versions of the SPLASH-2 workloads [110] as well as synthetic workloads (Uniform and Hotspot). For the synthetic workloads, we report the aggregate achieved packet delivery rate, the latency, and the delivery rate seen by the least serviced sender, all as a function of the offered load. An offered load of 1 means that the interconnect is at capacity for the given workload. Values over 1 indicate that demand exceeds capacity.

More discussion of our workloads and setup can be found in Chapter 4.

6.2.2 Results

6.2.2.1 Performance

We measured performance attained with five different arbiters, The first is called Baseline and is an optical channel-token protocol that delays the token by a half-cycle at each node, regardless of whether or not the node wants the channel. It is motivated by earlier work of Ha and Pinkston [72] and Kodi and Louri [73], who considered VCSEL-based technology with an electronic hop at each node. The others are our Token Channel, Token Channel w/ FF, as used in Corona[19]), Token Slot, and Fair Slot arbiters.

As shown in Figure 6.8(a) & (b), the Baseline proposal has low bandwidth potential and high latency solely because the tokens must be repeated at every node. The baseline only achieves 32% utilization in our Hotspot simulations, because our token can have at most 16 credits and the full round trip for a fully utilized token is at least $16 + (N - 16)/2 + 8 = 48$ cycles. Token Channel does little better on Hotspot; it saturates under a higher load but its bandwidth eventually drops off to the level of the baseline. The bandwidth drop seen in Hotspot is actually a result of credits becoming scarce and a zero-credit token being delayed by losing requesters. In fact, Token Channel behaves exactly like baseline at high loads because every node requests the token.

Fast forwarding tokens, intended to improve the fairness of Token Channel, here improves performance, because the FF waveguide quickly returns zero-credit tokens to the home node. The average token round trip drops from 48 cycles to 26 cycles under heavy load. Both variants of Token Channel, however, use a single token, repeatedly delayed in flight by senders, which causes its on-board credit count to be updated infrequently and to therefore carry increasingly stale information.

The data of Figure 6.8 indicate that Token Slot is the best protocol available. Its implementation is relatively straightforward and it achieves nearly the best possible throughput for both test cases across the range from very light to heaviest achievable offered load. But Token Slot is unfair, as Figure 6.8(f) shows: the least-served sender starves once the aggregate demand exceeds the channel capacity. Fair Slot, which removes this unfairness, is able to achieve 90% channel utilization: 10% of available tokens (bandwidth) go unused when the system transitions modes. In HotSpot at full load, every node always has a packet available for the destination. For Uniform, Fair Slot achieves

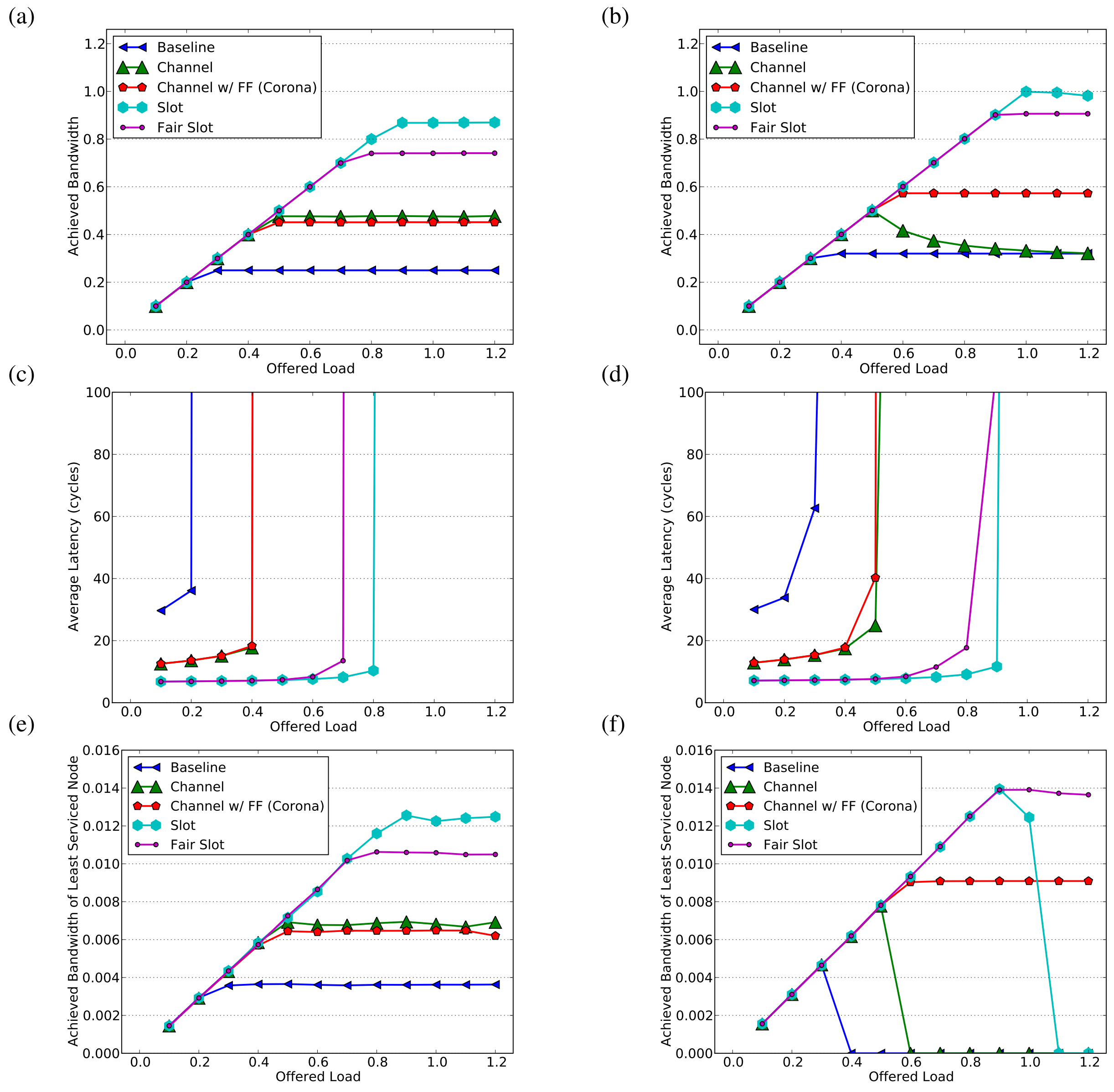


Figure 6.8 **Bandwidth (a & b), Average Latency(c & d), and Worst Serviced Node (e & f)** for Uniform random traffic (left) and HotSpot traffic (right).

74% utilization at maximum load: only a fraction of senders have a packet available for the destination due to small shared input queues.

Figures 6.8(e) and 6.8(f) give a closer look at how well Channel with FF and Fair Slot ensure fairness. The service given to the least-serviced sender indicates whether or not there is a starvation problem. At low loads, fairness questions do not arise – all demand is satisfied. For Uniform, all protocols remain fair even at high load. The combination of input queue size and credit availability ensure that starvation can not happen with uniform traffic. In

Hotspot, at high loads, the simple protocols become unfair. Token Slot is efficient and fair below network saturation, but falls over beyond that point. Fair Slot and Channel w/ FF avoid starvation at all measured loads. They are fair: the bandwidth seen by the least served sender is close to an equal share of the aggregate bandwidths shown in Figures 6.8(a) and (b).

In order to confirm that Token Channel w/ FF and Fair Slot achieves excellent arbitration latency and channel utilization with excellent fairness, we tried a more complex test case.

We generated single-channel traffic in which half of senders have random, small demands ($[0, 1/N]$) and half have random, heavy demand ($[1/N, 15/N]$) such that the total demand is 4 times the channel's bandwidth. We then assign the loads to nodes in ascending order, descending order, or random order. Figures 6.9 and 6.10 show achieved bandwidth for Token Slot and Token Channel, respectively. While the "unfair" protocols have problems with starvation (higher numbered nodes are not getting any service), the "fair" protocols do not. In the Fair protocols, the low-demand senders get all the bandwidth they need. The high demand senders are treated nearly equally, except that the two closest to the home node get more service than they deserve. But none starve, and the allocation is quite close to a max-min allocation of the aggregate achieved bandwidth. The figures also show the max-min allocation of the full channel bandwidth; the difference between the achieved service and the max-min service is caused by less than full channel utilization and the extra service for the two closest senders.

Finally, Figure 6.11 shows results from running 1024-threaded instances of SPLASH-2 benchmarks. Some of the workloads, such as Barnes, use the interconnect little and are indifferent to the arbitration policy. However, the rest show that arbitration is a major determinant of performance. Fair Slot performs the best, outperforming Token Channel w/ FF by as much as 60%.

We have presented performance results for a 64-node arbiter. Because Corona's data interconnect and our arbiters have very similar architectures, we believe the arbiter can scale with the Corona interconnect. However, at some point, the propagation time of a transmission becomes intolerable and a hierarchical MWSR, with arbitration at each level, will be necessary.

6.2.2.2 Power

Figure 6.12 uses the power methodology from Section 4.1.3 to show average power consumption of our 64-node arbiters.

Configurations require different number and types of waveguides, as shown in the figure's table. More waveguides are required to support features such as flow control and fairness. Fixed and Variable Injectors are described in Section 5.3. Broadcast waveguides distribute power through a tree structure, incurring splitter costs at each branch. Roughly speaking, the more waveguides a system requires, the more power it consumes.

As the chart shows, ring resonator power dominates all configurations, while laser power stays below one watt. Baseline and Channel have the same layout and components, but Baseline's ring resonators are always on-resonance,

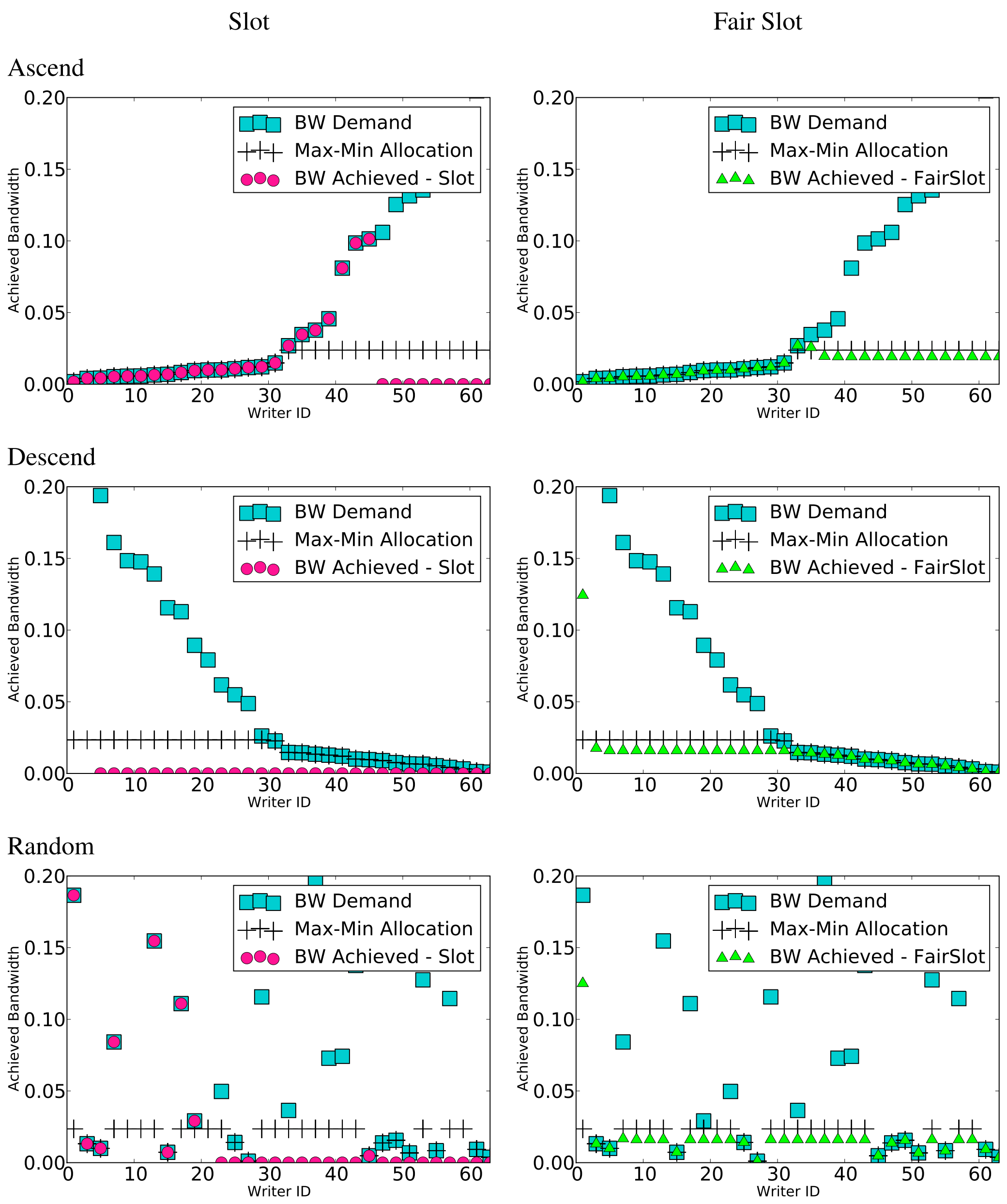


Figure 6.9 **Max-Min Results for Slot Arbiters.** Single-Channel bandwidth allocation for Token Slot and Fair Slot. Workloads use ascending demand assignment (top), descending demand assignment (middle), and random assignment (bottom). Note: Excessive demands are shown at the chart's limits and only every other node is plotted (for clarity).

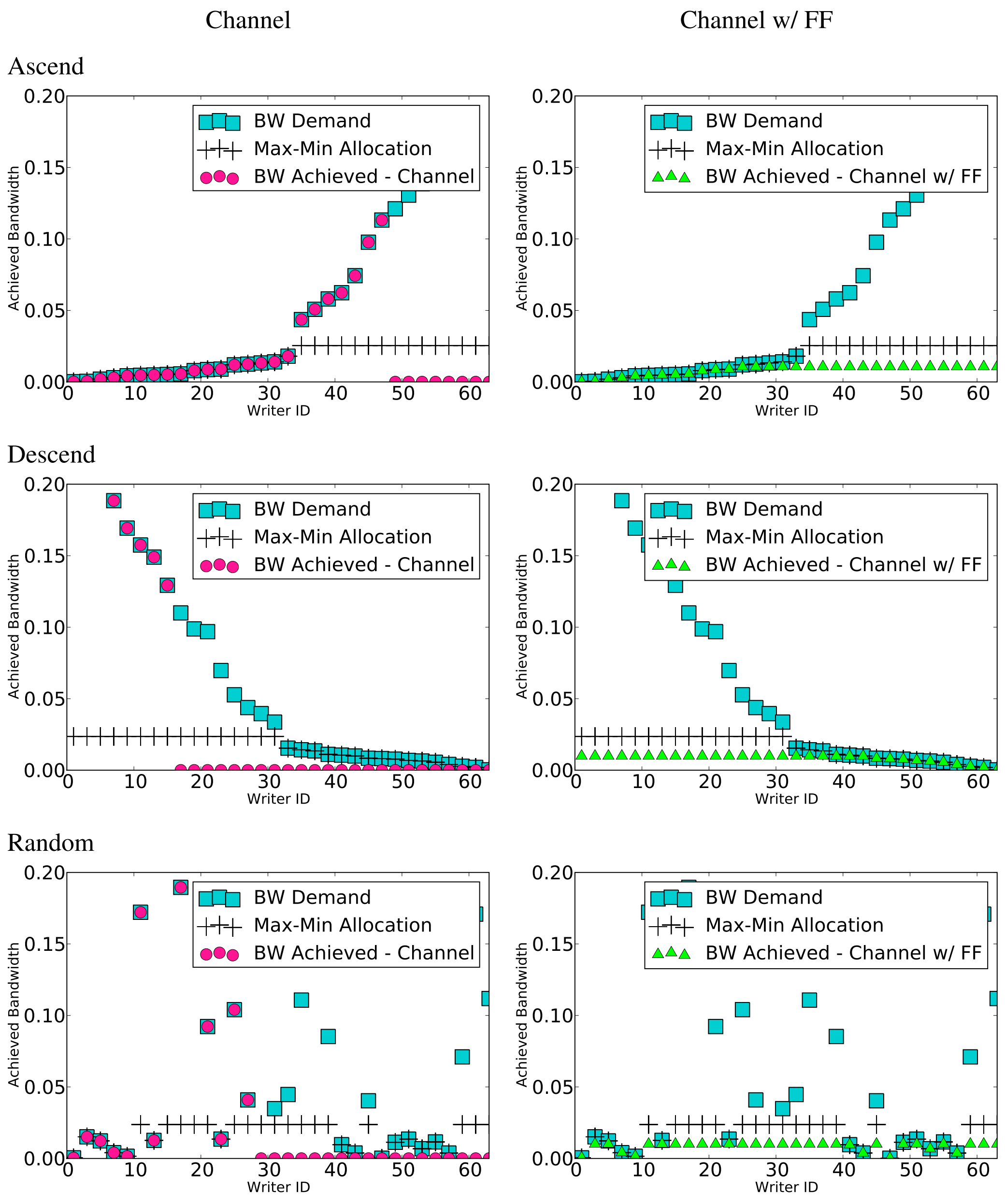


Figure 6.10 **Max-Min Results for Channel Arbiters.** Single-Channel bandwidth allocation for Token Channel and Token Channel w/ FF. Workloads use ascending demand assignment (top), descending demand assignment (middle), and random assignment (bottom). Note: Excessive demands are shown at the chart's limits and only every other node is plotted (for clarity).

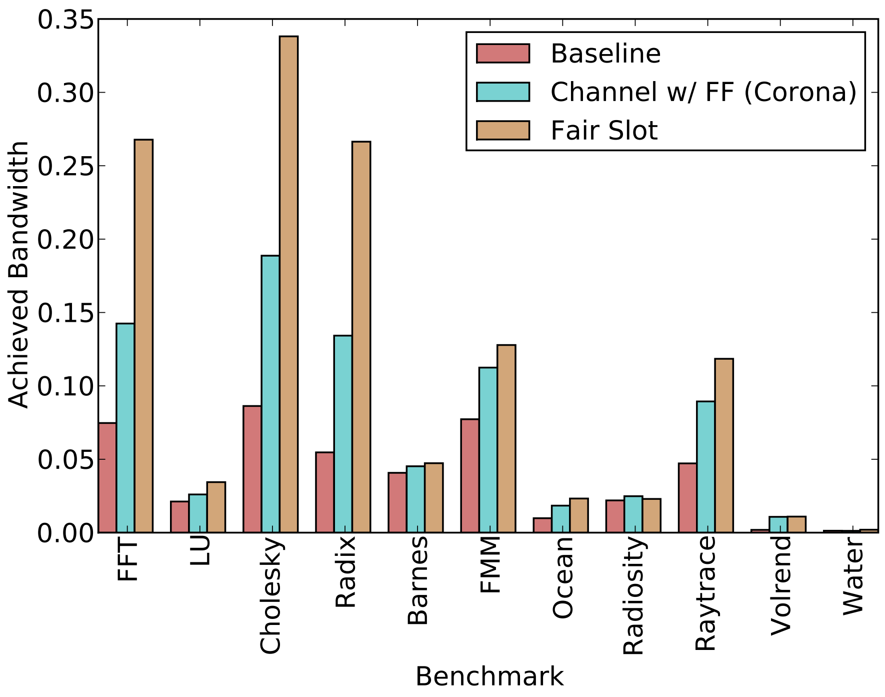
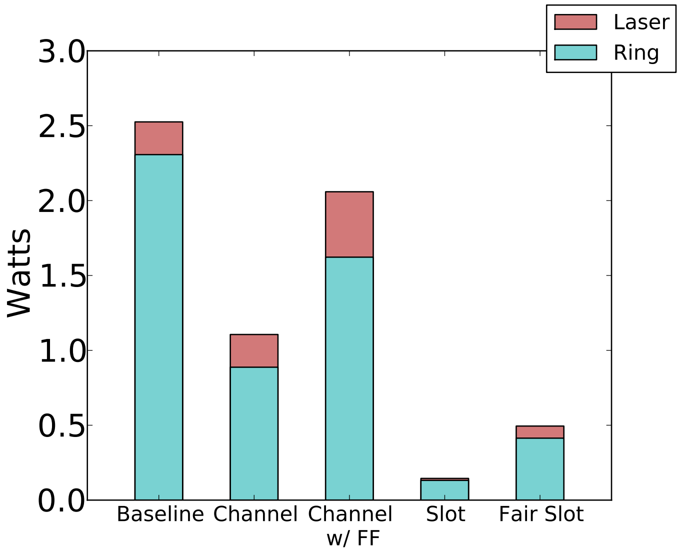


Figure 6.11 **SPLASH-2 Workloads: Achieved Bandwidth.**



	#	and waveguide type
Baseline	4	Variable Injector
Channel	4	Variable Injector
Channel w/ FF	8	Variable Injector (split between tokens and FF)
Slot	1	Fixed Injector
Fair Slot	$2+1$	Fixed Injector + Broadcast

Figure 6.12 **Arbiter Power.**

causing it to consume more power. Slot consumes the least amount of power, around one-quarter of a watt. Slot also requires very little laser power, relatively speaking, because it uses Fixed Injectors which result in low losses from off-resonance rings. Finally, the fairness schemes always consume more power than their baseline counterparts.

6.3 Discussion

6.3.1 Timing Assumptions

In this work, we assume that the token can be detected and (if necessary) reinjected one tick (100 picoseconds) after it is removed. We intentionally chose an aggressive target to push the limits of arbitration. Early SPICE models of the analog circuitry indicate that the target is feasible.

However, if a ring resonator cannot react quickly to a token's reception, then the performance of Token Channel and Token Slot may suffer. Below we discuss how each arbiter is affected. We also propose solutions that essentially divide a fast arbiter into two (or more) slow sub-arbiters.

Token Channel Critical Delay: Token Reinjection. In Token Channel, after sending one or more packets, the sender reinjects a token into the arbitration waveguide in parallel with the last clock edge of its most recent transmission. If the sender cannot reinject the token at this time, so that the token lags the end of the data packets, then a data channel "bubble" results, wasting data bandwidth.

If injecting on the trailing clock edge is too soon, a possible fix is to narrow the data channel, increasing the serialization delay of a data packet so as to cover the time needed to read, process, and reinject the token. Adding more data channels could then compensate for this narrowing.

Token Slot Critical Delay: Single-Token Detection. We assume a requester can detect a token and deactivate its detector before the next token arrives. If the ring resonator cannot respond to the token before the next token arrives, then one or more following tokens may be inadvertently claimed. Arbitration pipelining [63] can hide this effect. For example, if the ring resonators have a k-cycle latency, then for each data channel, k arbiters have tokens every k^{th} slot can be employed. Another solution would be to use one arbiter, but only sample it every $1/k$ cycles.

Effect of Multiple Sub-Arbiters. When there are multiple arbiters that arbitrate for the same destination, a nomination must choose to try for one, or more, of the arbiters. Below, we call each of these arbiters a sub-arbiter.

If a nomination only selects one sub-arbiter, the choice of sub-arbiter does not matter and the bandwidth achieved is comparable to a non-pipelined version. Let us assume an arbiter for a data channel capable of bandwidth B is divided into k sub-arbiters, each capable of arbitrating for B/k of the bandwidth. When the cumulative demand for a data channel is less than B/k, then no channel can be saturated and the choice of sub-arbiter does not matter. However, when the demand is greater than B/k, choice does matter. Choosing a single sub-arbiter at random is a good policy,

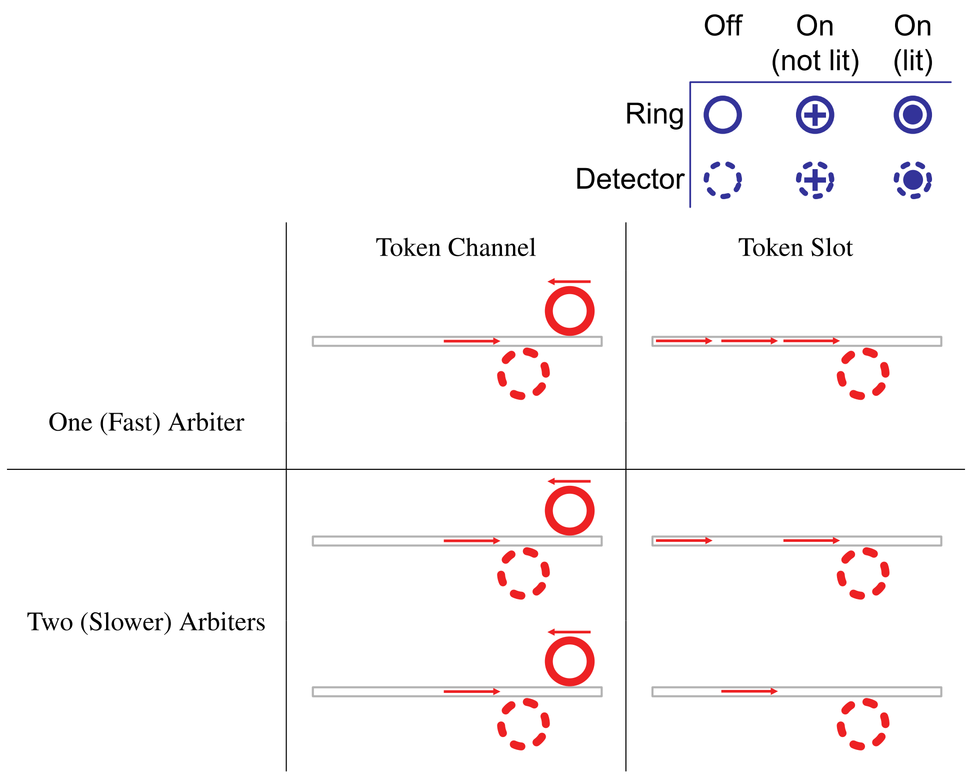


Figure 6.13 **Layout of Multiple Arbiters.** The figure shows how to compensate for slow detection at a ring by transforming a single arbiter into two arbiters for Token Channel and Token Slot.

although it can probabilistically lead to times when data channels are imbalanced (and even oversaturated). However, all sub-arbiters are equally likely to be imbalanced, overtime leading to even data channel usage.

Binding a nomination to one sub-arbiter may prolong wait-time if a token is missed on an alternative sub-arbiter. For Token Channel, the Poisson arrivals closely mirror that of the unpipelined version, but data transmissions take twice as long. For Token Slot, the inverse is true: the Poisson arrivals have double the inter-arrival times of the unpipelined version, but the data transmissions take the same amount of time. The net effect of both is felt in the communication latency and in the hold-times of critical buffer resources.

Binding a nomination to several sub-arbiters at once can reduce latency, but at the cost of lost bandwidth. In general, Token Channel, with its sparse distribution of tokens across the arbitration channel, is less likely than Token Slot to simultaneously win multiple tokens. The density of Token Slot's tokens (and the likelihood of winning multiple tokens) depends on how many tokens are in circulation (i.e. the data channel's load).

6.3.2 Supporting Prioritization

In Section 6.1.3 we discussed how to enforce fair sharing of the interconnect. The inverse of this problem is how to deliberately enforce unfair sharing, or priorities.

LAN token ring systems support prioritization by allowing nodes and tokens to have eight different priority levels. A node can only use a token if the node's priority level is greater than or equal to the token's priority level.

Our Token Channel with Fast Forwarding and Fair Slot proposals could be extended to to support prioritization levels. Multiple priority levels can be represented through multiple fast-forward waveguides (Token Channel) or multiple pairs of Broadcast/Hungry Waveguides (Token Slot). The nodes could use these extra waveguides to communicate priorities to the home node, which can take appropriate action. Both proposals require extra optical resources and some extra delay for communicating priority needs to/from the home node.

6.4 Summary

This chapter proposes and evaluates nearly all-optical arbitration solutions. Furthermore, it shows how the solutions may be extended to challenging problems related to flow-control and fairness that arise when power comes from a single laser source.

Chapter 7

Atomic Coherence

7.1 Introduction

This chapter advocates Atomic Coherence, a framework that simplifies cache coherence protocol specification, design, and verification by decoupling race detection and race resolution from the protocol's operation. Atomic Coherence strives to provide complexity-effective high-performance coherence, filling the gap between today's fast-and-complex designs and yesterday's slow-and-simple designs (as discussed in Section 3.2 page 23).

Atomic Coherence simplifies protocols by requiring conflicting coherence requests to the same addresses to be serialized with a mutex before the request is allowed to issue to the coherence subsystem. With Atomic Coherence, complex coherence transactions follow a predictable race-free path that is easy to design, analyze, and verify. Avoiding races significantly reduces the size of the protocol's state-space.

Atomic Coherence also has the potential to improve performance with little added complexity. Because requests are guaranteed not to race, it is far simpler to add optimizations to the protocol without significantly impacting complexity. This includes, for example, allowing the cache holding a mutex to directly interact with other caches without first visiting the directory or allowing speculation in the protocol. We extend a directory protocol along these lines and find that without races, such extensions are trivial to implement.

Furthermore, the Atomic Coherence idea is applicable to any protocol prone to races on any interconnect, including any combination of the following: snoopy/directory protocols, invalidate/update protocols, strict/weak consistency, and ordered/unordered interconnects. We only study Atomic Coherence in a single context, but we discuss the impact it has on other contexts, namely weak memory models and general interconnects (see Section 7.5 and Section 8.2.2; pages 97 and 103 respectively).

Finally, as we discuss below, an Atomic Coherence implementation can have a non-negligible impact on bottom line performance. Our implementation exploits the unique property of optical tokens to minimize the impact.

7.2 Atomic Coherence

Below we discuss the basics of Atomic Coherence, as well as its complexity and performance implications.

7.2.1 Atomic Coherence Definition and Discussion

We define Atomic Coherence in the following way:

> We say a system implements *Atomic Coherence* if, at any time, there is at most one coherence request per block performing coherence outside of its requesting node.

Practically, this definition means that a coherence request cannot leave the requesting node until the requester has acquired the corresponding mutex, thereby preventing other requests from becoming concurrently active. When the coherence request has finished performing coherence, it is safe to release the mutex. We explore two policies for releasing the mutex, reflecting two views of coherence:

- `Atomic CResp` Policy: Release the mutex when all control responses received.
- `Atomic DResp` Policy: Release the mutex when all control *and* data responses received.

`Atomic DResp` is the simplest policy, from a race perspective, because the mutex prevents racing requests from interacting. However, this simplicity comes at the cost of considerable increase in mutex hold time (especially for long-latency DRAM fetches), and results in additional pressure on finite mutex resources. `Atomic CResp` is only slightly more complex but eases pressure: independent requests can race, but a racing request will only encounter a pending block for which all control activity has quiesced and, other than waiting for data, is otherwise coherent. `CResp` races are few and easy to specify in a finite state machine. They are also easy to resolve through either `NACK`ing, queueing, or absorbing the race. For our implementation of `MOEFSI`, this slight addition of complexity reintroduces 7 of the original 79 races back into the protocol. Meanwhile, because the mutex can be released before data has arrived, `CResp` eases mutex resource pressure and improves performance.

If a coherence request does not require data (e.g. Upgrade), then `Atomic CResp` and `DResp` are identical in protecting the request from concurrent requests. However, `Atomic CResp` differs in that, if the request requires a data response (e.g. transitioning from `I`), then there is a vulnerability window in which a race can occur. `Atomic DResp` allows no such races. From `Atomic CResp`, the vulnerability window depends on response ordering and only occurs when the data response is the last response; it spans the time between the last control response and the data response. We can limit the races to one per request if our protocol requires every coherence request to generate a control response, as is common procedure for most protocols. For example, if coherence request A releases the mutex before it has acquired its data, the next coherence request (B) to acquire the mutex may generate a race request affecting A. A will recognize B's request as a later ordered event and will delay the request until its data has arrived. When A eventually handles B's request, A sends B a control response. Because B cannot release the mutex until it gets this control response, A is safe from any other races. This example sets up a two-node data forwarding chain between A and B. We can extend this idea to build longer chains, where each node in the chain is awaiting data. Chains can be built of reads, writes, or a combination of reads and writes. The possible races are protocol specific and depend on the type of chaining support. In all cases, a control response, like the one used in the previous example, can be used to set the end of a chain and prevent any further races. Subsequent requests cannot join the chain and must wait until the control has completed and the mutex released.

The only prior coherence protocols that we are aware of that satisfy the strictest `Atomic DResp` version of Atomic Coherence are ones that rely on a shared, blocking bus [123, 86, 96], where successfully arbitrating for and holding the bus for the entire coherence transaction are analogous to acquiring and retaining the mutex in our `Atomic DResp` protocols. `Atomic CResp` protocols correspond to two-transaction split busses, where control is completed in the first bus transaction and data in the second. Other protocols, such as the Wildfire design [93] and Safe Points [14], which block at the directory, are not strictly atomic, since conflicting requests are allowed to leave requesting nodes and race to the directory (see Section 3.2.3 page 24 for descriptions of these protocols). Such protocols do gain many of the complexity benefits of strictly atomic protocols, since races can only occur before messages reach the directory, and are precluded thereafter. However, in the general case, ordering requests at the directory restricts implementations to protocols that always visit the directory first, before performing any other coherence operations. This precludes not only snoopy coherence implementations, but also speculative optimizations that directly probe some subset of caches in parallel with or prior to reaching the directory (e.g. multicast snooping [81]).

7.2.2 Enforcing Atomicity with Mutexes

From the definition of Atomic Coherence, we can define atomic protocols as being made up of two substrates: an *Atomic substrate* and a *Coherence substrate*. The Atomic substrate enforces the invariant that only one coherence request to a memory block may in progress at a time. The Coherence substrate performs the coherence, enforcing contracts for block permissions and data. The substrates are independent from each other.

We use hardware mutexes in the Atomic substrate. A miss that has seized its mutex may leave its node to obtain coherence permissions in the Coherence substrate. The mutex is released after the original request and all of the activity it generates has been serviced. Releasing the mutex only after all coherence activity has quiesced guarantees that there will be no coherence interference with subsequent conflicting requests.

The most general and straightforward way for the protocol to determine when it is safe to release the mutex is by requiring a central location (such as the requester or directory) to collect an acknowledgement for every message. Sometimes acknowledgements are not necessary if the interconnect happens to have ordering or timing guarantees [124, 125, 91], or if the mutex release acts as an implicit acknowledgement (discussed more in Section 7.2.4.2 page 82).

The classic textbook-example of a race is the writeback race from the SGI-Origin 2000 protocol. This race occurs when two requests cross each other - a writeback request (e.g. `PutX`) and an exclusive intervention request (e.g. `IntervEx`). An example of this race is shown in Figure 7.1(a). Races like this are often considered corner-cases of the protocol design and are individually resolved with custom solutions. SGI-Origin's solution elegantly resolves the

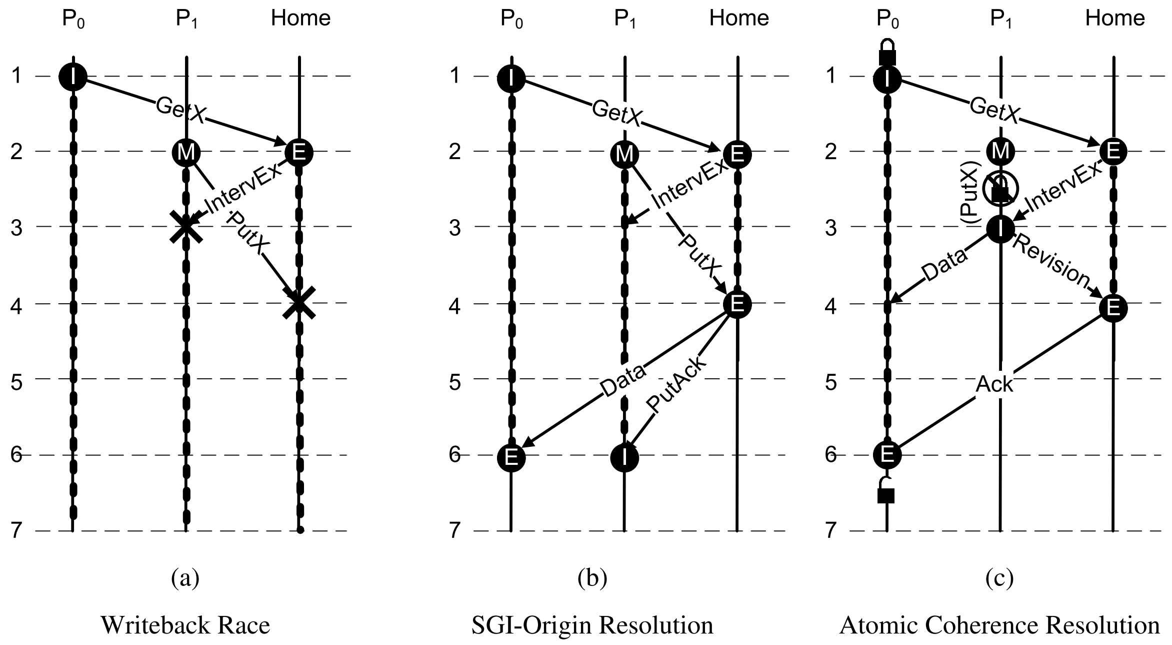


Figure 7.1 **(a) SGI-Origin Writeback Race.** P_1 performs a writeback (PutX) and expects an acknowledgement (PutAck) message, but instead receives an exclusive intervention (IntervEx) from P_0's unrelated request. Meanwhile, the home, which is in a split-state while it transfers ownership, receives a PutX from P_1 when it is expecting a Revision response from P_1. (The Revision contains data that is written back to memory.) The PutX cannot be dropped, because it is the only valid copy. Nor can it be NACKd until the competing GetX completes, because that would cause the memory and caches to be incoherent with each other. **(b) SGI-Origin Resolution.** P_1 ignores the InterventionEx it receives, because it has a writeback outstanding that the protocol cannot drop. The home simultaneously handles both requests, by dually treating the PutX request like a Revision response. **(c) Atomic Coherence Resolution.** P_0 seizes the mutex first and proceeds with its GetX. P_1 later has a PutX request, but cannot issue it because P_0 holds the mutex. P_1 satisfies the intervention from the M state in accordance with the S GI-Origin specifications. A final ACK handshake is necessary between the Home and P_0 to ensure th at all coherence activity has completed before the mutex is released.

race without consuming extra resources (Figure 7.1(b)). However, many races are not resolved so elegantly; races that are uncovered after the initial protocol design are especially troublesome. Figure 7.1(c) shows how our mutex-based Atomic Coherence avoids this race entirely.

7.2.3 Complexity Implications of Atomic Coherence

Atomic Coherence does away with all race transitions in the Coherence substrate, leaving only stable transitions and split transitions[1]. Once a request has been granted issuing-rights, it will only receive events that will further it towards its destination stable state (e.g. responses and acknowledgements). Guaranteeing forward progress eliminates the possibility of pathological livelock and starvation scenarios. More importantly, as we show in Section 7.4.2 (page 89), Atomic Coherence simplifies the protocol's logic and reduces the number of total transitions that must be exercised during verification.

7.2.4 Performance Implications of Atomic Coherence

Atomic Coherence can negatively affect performance in two ways. First, the act of acquiring the mutex before issuing a request is added to the request's critical path. Second, Atomic Coherence may serialize coherence requests when it would have been perfectly coherent to service them in parallel (e.g. parallel reads). However, we can counteract this latency by adding performance enhancing techniques to the protocol with very little complexity cost. Below, we discuss the performance pitfalls and opportunities of Atomic Coherence.

7.2.4.1 Performance Pitfalls

Performance pitfalls that arise from the need to quickly seize a mutex and effects of serialization.

Mutex Resources. Because atomicity is on the critical path, a free mutex should be seized as quickly as possible. Options include spatially locating a mutex near its likely requester, prefetching a mutex, and, as we show, circulating a mutex by requesters. In this work, we circulate the mutexes via optical tokens. Our optical tokens allow for an available mutex to make a full pass by every potential requester in under a nanosecond.

Furthermore, a limitation in the number of physical mutexes available may pose additional performance stress. The obvious solution is to hash addresses to blocks or employ counting bloom filters [126], but false collisions must be kept to a minimum. If we assume a stable system, an approximation of the average mutex occupancy (number of mutexes simultaneously held) can be derived from Little's Law [127]. According to Little's Law, the average number of mutexes in the system is equal to the product of the average arrival (or seize) rate of mutex requests and the average latency of a mutex hold. Thus, if the goal is to keep the number of mutexes required to a minimum, it is

[1]Recall from Section 3.2.2 (page 24) the three different types of transitions: stable transitions occur between the stable states (e.g. {`M,S,I`}), split transitions occur when intermediate steps must be completed on the way to a new stable state, and race transitions occur when an unexpected event arrives in the middle of a split transition.

best to have small mutex-hold times (i.e. have fast coherence transitions) or low mutex-arrival rates (i.e. low L2 miss rates). Finally, if hashing is employed, the rate of arrival as well as the average occupancy are reduced by the rate of collision-occurrences.

Serialization. Many protocols allow parallel reads to proceed simultaneously, as long as there is not an intervening write. Conceptually, this seems safe, as long as the reads are free of side effects. However, coherence read requests often do have side effects, such as updates to sharing lists or LRU state. As long as these metadata updates can be correctly handled, it would be safe to allow concurrent read requests in the Coherence subsystem. As described, Atomic Coherence serializes these reads, which may negatively affect performance by inhibiting concurrency. The size of the serialization delay depends on how much of the serialization delay cannot be hidden, as diagrammed in Figure 7.2's scenarios.

Our results (Figure 7.10) show that when read serialization is non-negligible, it can sometimes be mitigated with a straightforward push protocol optimization (see Section 7.4). Nonetheless, for completeness, let us discuss how the atomicity notion can be relaxed to allow concurrent read requests. Let us assume an invalidate protocol which maintains the invariant that there can either be one writer and no readers - or - no writer and any number of readers. Also, let us replace the atomic mutex with two specialized mutexes - a read mutex and a write mutex - that may be atomically operated on together. A read is safe to proceed if there are no writes in progress. Thus, a read must set the read-mutex and check that the write mutex is not held (no outstanding writes). A write is safe to proceed if there are neither reads nor writes in progress. Thus, a write must set both mutexes (no outstanding writes or reads).

Finally, the Atomic substrate should prevent any request from starving for service. Deadlock is caused by circular dependencies. Any subsequent coherence action that is caused by the current request must be decoupled from it to prevent resource cycles from forming. For example, a demand miss may initiate a writeback, and the mutex acquisition for that writeback must be decoupled from the mutex release for the current request, otherwise the two can form a resource chain that can become part of a system-wide cycle. Such causally-related actions can be decoupled by using a writeback buffer that allows the mutex acquisition for the writeback action to occur after the mutex release for the demand request, hence preventing the formation of a cycle.

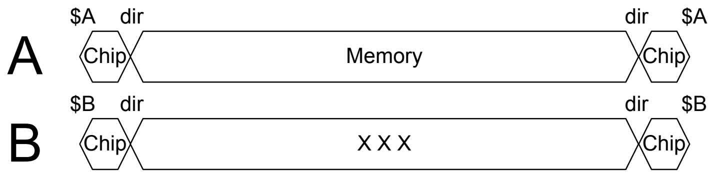
Other service issues, like livelock and fairness, we leave to future work, but believe the Atomic substrate can provide fair service by adapting techniques from prior work [33, 128, 129]. For now, we enforce a simple fairness policy restricting a cache to only one coherence request per mutex acquisition before passing the mutex on.

7.2.4.2 Performance Opportunities

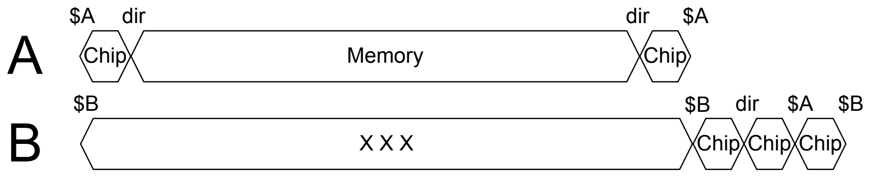
Below we describe how Atomic Coherence can help performance in two ways. First we describe how a simple race-free protocol can be straightforwardly transformed into a race-free sophisticated protocol. Then we describe how the release of a mutex can eliminate `ACK` messages, possibly easing pressure on strained interconnects.

$-to-Memory Miss

Conventional Coherence:



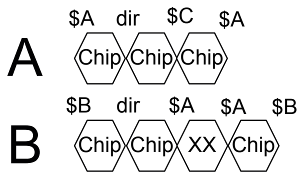
Atomic Coherence:



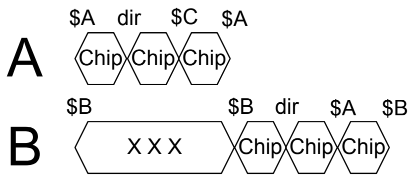
Atomic Coherence overlaps a relatively large amount of B's stall time with A's memory time.

$-to-$ Miss

Conventional Coherence:



Atomic Coherence:



Atomic Coherence exposes a relatively large amount of B's stall time, because the work cannot be overlapped. The conventional coherence protocol overlaps with with read-chain forwarding [97].

Figure 7.2 **Parallel and Serial Miss Timing Diagrams.** Qualitative timing diagrams for two scenarios where read misses to block A and block B are issued simultaneously. Each diagram shows where the time is spent: communicating On-Chip (Chip), communicating to Memory (Memory), or Stalled (X); and the hop locations: L2 Cache ($) or Directory (dir).

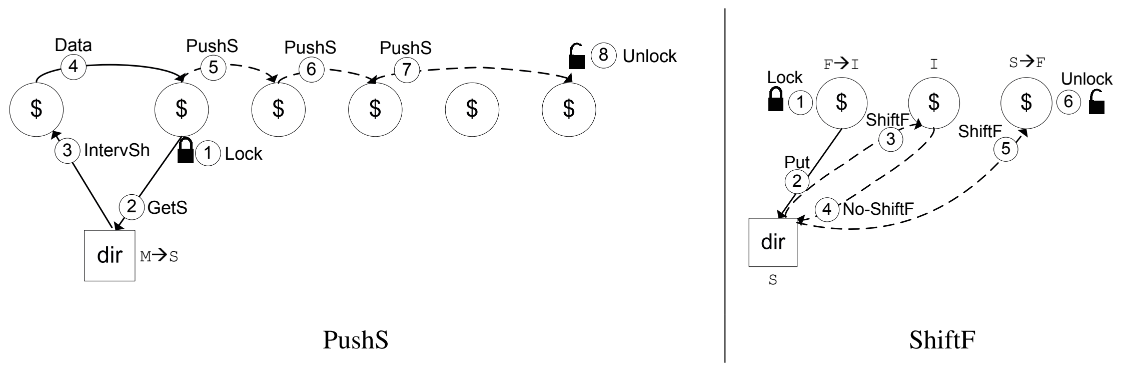


Figure 7.3 **PushS and ShiftF Overview. PushS:** When a block goes from Modified to Shared at the directory, the old sharing list is used to "push" (prefetch) the block into caches (pushes shown with dotted lines). Note: The mutex is seized by the requester but released by a non-requester. **ShiftF:** When an `O/F` block is evicted, the directory serially sends `ShiftF` messages until it finds an `S` block that it can shift the `F` state t (e.g. `S` $\rightarrow$ `F`). Note: The mutex is seized by the requester but released by a non-requester.

The optimizations described below are trivial to implement in atomic protocols but would be non-trivial to in non-atomic protocols.

Push and Shift Protocols. Our push and shift protocols exploit the property that, while a processor holds a mutex, it may perform coherence in any way it sees fit, as long as it leaves the system in a coherent state when it releases the mutex. Data may be *push*ed into caches that do not currently hold it. Similarly, permissions (like the `O` and `F`) state may be *shifted* between sharers.

Our push protocol pushes data into caches so that they can avoid future misses. It works as follows (shown in Figure 7.3): when a block transitions to exclusive at the directory, we do not erase the sharer list. Instead, we hold the sharing list in place for future reference. When the block is later read and leaves the exclusive state, the old sharing list becomes our prediction of new sharers as well as the new sharing list. The pushed value is sent in a message, visiting each predicted cache in series, similar in spirit to Piranha's Cruise Missile Invalidate [130]. The data is then prefetched into each processor's cache, and the last cache visited releases the mutex. For performance reasons, our implementation eagerly passes the missile. A separate missile trails the initial missile, waiting for the L2 to process the push before continuing. Only when the last node in the list has received *both* the missile and the trailing missile, may the mutex may be released. Note that different nodes seize and release the mutex: the initiator of the miss seizes the mutex while the last missile site releases the mutex.

Our ShiftF protocol tries to keep a shared block in a state that sources to other on-chip caches (e.g. `O/F`). A block that is in one of these *sourcing* states elides a memory reference, saving both time and energy. Our shift protocol works in the following way: when an `F` or `O` block is evicted, the directory tries to find another sharer to take on the `F` state (if `O`, the data is cleaned to `F` with a memory writeback). While holding the mutex, the directory goes through the sharing list, serially checking if a sharer has the block in a non-sourcing state (i.e. `S`). If in `S`, the block transitions to `F`, its LRU state updated, and the mutex released. If in `I` (invalid) (because of a silent eviction), the sharer informs the directory it is no longer a sharer and the directory tries the next sharer. If the block is no longer cached, the directory entry also becomes invalid and the mutex released. Note that shifting has many benefits: it keeps an on-chip sourcer, updates the directory's sharer information (which may be stale due to silent evictions), and cleans the data.

Both PushS and ShiftF are unattractive in conventional non-atomic implementations, since they require multiple steps and potentially many serial visits to multiple nodes, making them very vulnerable to races and latent protocol bugs. However, in our atomic substrate they are trivial to implement and, as we show in Section 7.4, provide compelling performance benefits.

Implicit Acknowledgements. When a coherence message is sent, sometimes the cache coherence protocol requires an acknowledgement. Acknowledgements are a way of guaranteeing that the message has been processed by the recipient and are commonly used for writebacks and invalidations.

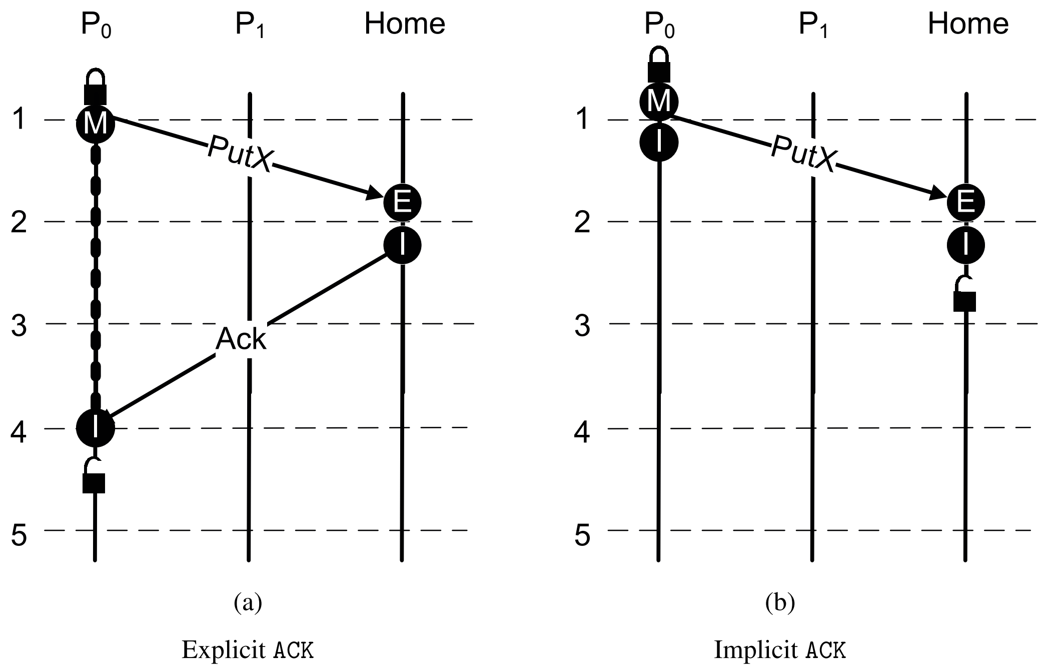


Figure 7.4 **Implicit Acknowledgements in Atomic Coherence.** (a) Atomic Coherence with Explicit ACKs. The ACK is a remnant of our race-prone SGI-Origin 2000 protocol and was originally required to detect follow-on races. (b) Atomic Coherence with implicit ACKs. Atomic Coherence guarantees the writeback's race-free completion, so P_0 requires no ACK and the Home can safely release the mutex that was seized by P_0.

In some cases, Atomic Coherence can eliminate acknowledgements in the Coherence substrate by relying on the Atomic substrate to supply the `ACK`, via a mutex release.

For example, some protocols may only finalize a writeback after it has been acknowledged by the directory. We could naively build an atomic protocol that performs writebacks in this way: a node seizes the mutex before initiation of the writeback and releases the mutex after receiving the directory's `ACK`. (We call this `ACK` message an explicit `ACK`.) Clearly, this approach is coherently safe, but can be optimized with implicit an acknowledgement.

Atomic Coherence can optimize the writeback transaction by removing the `ACK` message and having the directory, instead of the cache, release the mutex. This is possible because Atomic Coherence guarantees the writeback's race-free success and so the `ACK` is unnecessary; the mutex may simply be released by the directory. In this way, the mutex release acts as an *implicit* `ACK`. Figure 7.4 contrasts explicit and implicit writeback acknowledgements.

The elimination of explicit `ACK`s can benefit the system in many ways. Fewer messages holding fewer resources allows better throughput at lower latency. Releasing the mutex before the last acknowledgement can help lower wait-times of pending mutex acquisitions. Also, because the Atomic substrate comes at some cost to bandwidth, eliminating `ACK`s can help recoup some of this cost. Each implicit `ACK` consumes only a mutexes-worth of bandwidth (1 bit for our experiments), versus many bits in a regular control message. Finally, implicit `ACK`s simplify a protocol's finite state machines by removing the split states associated with a cache waiting for a directory's writeback acknowledgement.

7.3 A Photonic Atomic Substrate

Atomic Coherence works best if mutex arbitration occurs quickly and fairly. We have chosen an optical embodiment, because it offers low-latency fair cross-chip mutex communication.

We use optical tokens with Variable-Injectors, as described in Section 5.3. Variable injectors permit any node to seize or release the light/token/mutex. If the ring resonator does not divert the light, it must try again on the next revolution of the mutex. The retry is effectively a local-`NACK` and thus consumes no interconnect bandwidth.

Requesters must know when to turn on their resonators to divert a mutex. We assume that each mutex is assigned a spatial "slot" on a wavelength in a waveguide and that this assignment is known globally. For instance, mutex m can be found by cache c at slot i on wavelength j in waveguide k. Slots are separated by clock edges, and nodes turn on and off their ring resonators to align with the clock's edges. Each nodes is responsible for refreshing a fixed subset of the mutexes with OEO repeats.

The resource cost of having an optical mutex for every cached address, much less every memory address, would be prohibitive. We reduce this number by hashing several addresses to a single mutex. We use the lowest order bits above the block offset, because they have been shown to have high levels of entropy [131].

When the request completes and it comes time to release the mutex, the node must return the mutex to its assigned slot in the waveguide. It does so by re-injecting the token on its respective slot. The ring-shaped token waveguide guarantees fair round-robin ordering of requesters.

7.4 Evaluation

7.4.1 Experimental Setup

We model a 128-core scaled-down version of Corona (as described in Section 4.2.2 page 33). Cores are concentrated into 16 interconnect nodes (8 cores share an interconnect node). We discuss the setup more in Section 4.2 (page 31).

Atomic Substrate. It takes less than 600 nanoseconds (or < 3 clocks at 5 GHz) for light to pass by all nodes. This allows a waveguide to store 2-and-a-fraction mutexes across 64 wavelengths in transit. We allow for the dateline (or alternatively, the home node) to also buffer 1-and-a-fraction mutexes, allowing for a total 4 mutexes to be associated with each wavelength (and 256 mutexes associated with each waveguide).

Unless otherwise noted, we configured the Atomic substrate with 1024 mutexes across 4 waveguides. The four mutex waveguides are in addition to the 64 data waveguides. Each miss address maps to a mutex by `XOR`ing address bits$[6:16]$ (the bits above the block offset) with a mask made up of bits$[17:21]$. We call this hash function `XOR5`. Four mutexes share each wavelength, with approximately two in circulation and approximately two buffered for repeat. Thus, each mutex has a four cycle period, spending ˜2 cycles in a waveguide and ˜2 cycles in a buffer. Our choices for mutex number, hashing function, and mutex frequency came from the sensitivity analysis performed in Section 7.4.2.5

Coherence Substrate. The directory tries to find on-chip sources of data by sending interventions to caches it believes may have a block in the `M`, `O`, `E`, or `F` state. To reliably source data, caches that have the block in one of these four states may not silently evict the block. Caches may only silently evict Shared (`S`) blocks. Our atomic protocols lower the cost of these writebacks by employing implicit `ACK`ing whenever a cache evicts an `M`, `O`, `E`, or `F` block.

We begin with an implementation of an SGI-Origin-like protocol (`MESI`) and add the `O` and `F` state to it to obtain our baseline (`MOEFSI`). (The `O` and `F` [132] states refer to dirty-shared and clean-shared). We apply our Atomic-to-Data-Response (`Atomic DResp`) and Atomic-to-Coherence-Response (`Atomic CResp`) to this protocol to demonstrate the impact of mutex acquisition and request serialization has on bottom line performance. Next, we optimize the performance of our protocols by adding support for shifting the `F` state (`ShiftF`) and pushing the `S` state (`PushS`). Finally, we study the impact of removing the explicit acknowledgements through implicit `ACKs`.

Workloads. Our experimental traces come from a mix of scientific (SPLASH-2) and commercial (Wisconsin Commercial Workload Suit) workloads. We also model a feed-forward neural network as a "best-case" microbenchmark for our atomic push mechanism, `PushS`. Our `PushS` missile exploits the neural network's sharing behavior, which alternates between single-writer and many-reader. The missile pushes shared data to all nodes in a network level when any node reads the pushed value. We believe the same sort of push mechanism could be used in other programs that have

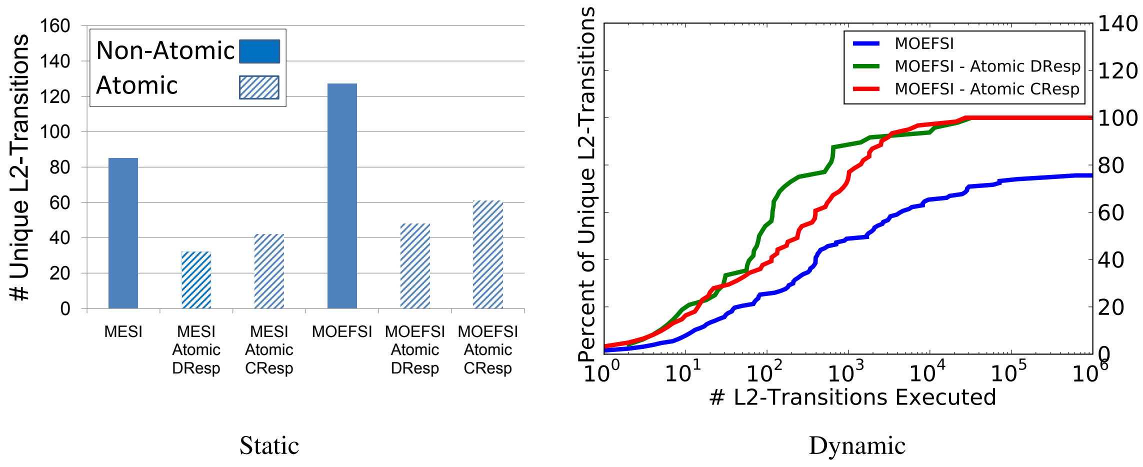


Figure 7.5 **Complexity Analysis for Atomic Coherence.** Static counting of L2 Transitions (left) and cumulative coverage of L2 Transitions using a directed random tester (right). (Note: log-scale on x-axis).

similar sharing patterns or benefit from prefetching dirty data into caches (e.g. MapReduce for Shared Memory [133]). More discussion of our workloads and setup can be found in (Section 4.2.1.1).

Each experiment was run in server-consolidation mode with eight 16-threaded instances, except for the neural network, which was run as a single 128 thread instance. Threads were assigned to cores in a semi-random fashion, to both simulate real-world scheduling and to stress coherence. Inter-core sharing occurs at the L1 (2-cores) and the L2 (8-cores)

7.4.2 Results

Below we study the relationship between protocol complexity and performance. In Section 7.4.2.1 we show how Atomic Coherence allows for simple protocols, while in Section 7.4.2.2 we show how Atomic Coherence allows for high-performing protocols. Complexity and performance analysis are combined in Section 7.4.2.3. Finally, we show how Atomic Coherence performance is sensitive to resource allocation (Section 7.4.2.5) and how our simple atomic optimizations perform (Section 7.4.2.6).

7.4.2.1 Complexity

The primary advantage of Atomic Coherence is a reduction in protocol complexity. Complexity is difficult to quantify using a single metric, so this section characterizes the protocols presented earlier both statically, in terms of the overall number of unique transitions in each protocol, as well as dynamically, by reporting coverage achieved by a random protocol tester.

Figure 7.5(left) shows the percent of unique transitions included in each of the five protocols studied. We only count the transitions triggered at the L2 by events from the interconnect (interventions, responses, etc.) because this is where the protocol's complexity lies. The two non-atomic protocols (`MESI` and `MOEFSI`) contain 85 and 127 unique transitions, respectively. This difference conveys the increase in complexity of supporting two new states and the related races. The figure also shows that, if we enforce coherence atomicity through to the completion of the data (`Atomic DResp`) or the coherence (`Atomic CResp`), our transition counts drop by at least 50%. The dropped transitions existed solely to handle protocol races. `Atomic CResp` has slightly more transitions than `Atomic DResp`, because `Atomic CResp` must queue the next conflicting request (if any) while it is waiting for its data. When the data arrives, it satisfies its own request and then forwards the data to the subsequent requester. This case can only occur while waiting for data, involves at most a single subsequent request (for our protocols), and adds only a slight amount of complexity to the protocol, while significantly reducing the mutex hold time for misses to main memory.

These results illustrate the dramatic simplification that is enabled by decoupling race resolution from the protocol and provide evidence for our claim that atomic protocols should be substantially easier to design, debug, and validate.

Races are rare in program execution. To exercise races more fully, we collected coverage results using a directed random tester similar to the one provided with GEMS [108]. The tester generates pseudo-random memory references at each node in a targeted mix of reads and writes that are designed to trigger both independent and conflicting memory references in the coherence subsystem. The tester also inserts random delays at two points in the interconnect (at the port to the data interconnect and the port to the Atomic substrate) in order to synthetically reorder various messages.

The purpose of the tester is to excite as many unique transitions as quickly as possible in order to quickly uncover protocol bugs and allow rapid testing and turnaround for protocol fixes. We are not claiming that random testing alone is sufficient for validation of a real design, but it is a very useful and widely-deployed tool for flushing out design errors, particularly early in the design process. An efficient random tester that provides fast and early coverage of as many unique transitions as possible will substantially improve the productivity of the protocol designer.

The results in Figure 7.5(right) clearly illustrate the complexity advantages of our atomic protocols. Well within the first million transitions, `Atomic DResp` and `CResp` have reached 100% coverage of all unique transitions, while the non-atomic `MOEFSI` protocol begins leveling off asymptotically at 75% coverage. This result suggests two conclusions: (1) turnaround time to uncover, fix, and test design bugs is reduced substantially, since all protocol transitions are exercised early and often; and (2) the rare and likely problematic corner cases lurking in the remaining unexercised 25% of transitions in the non-atomic protocol are completely eliminated, reducing the unpleasant consequences of encountering them late in the design cycle.

Of course, tweaking the random tester to synthetically force decreasingly likely protocol races to occur sooner could probably improve the cumulative distribution for non-atomic protocols, but this activity requires designer effort as well, since such tweaks will likely have to be specific to each protocol and interconnect topology. In contrast, our very simple, generic, first attempt at a random tester (˜200 lines of C code) produces excellent coverage in a very timely manner, as long as it is exercising one of our atomic protocols.

These results illustrate the dramatic simplification that is enabled by decoupling race resolution from the protocol, and provide evidence for our claim that atomic protocols should be substantially easier to to design, debug, and validate.

Our experiments with the random tester also produced three interesting anecdotal observations: first, the tester uncovered multiple latent, obscure bugs in our non-atomic MOEFSI protocol; second, the tester failed to uncover any bugs in our atomic protocols, which were correct in their first iteration; third, the tester uncovered opportunities for simplifying our atomic protocols, since it failed to exercise transitions that, upon closer examination, we found to be impossible and subsequently removed from the protocol. These observations lend credence to our claims that atomic protocols are substantially easier to design and verify.

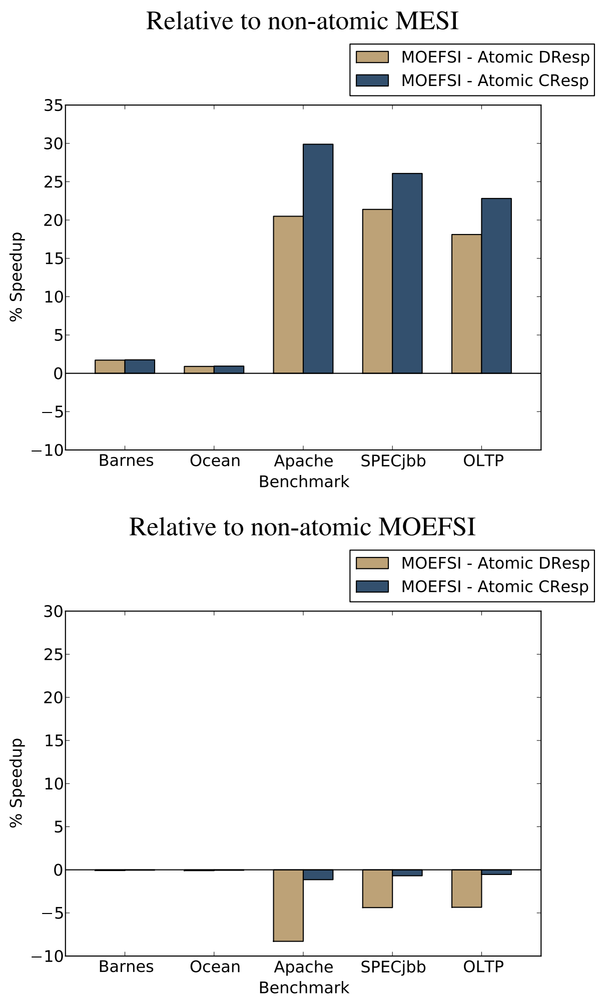
7.4.2.2 Performance

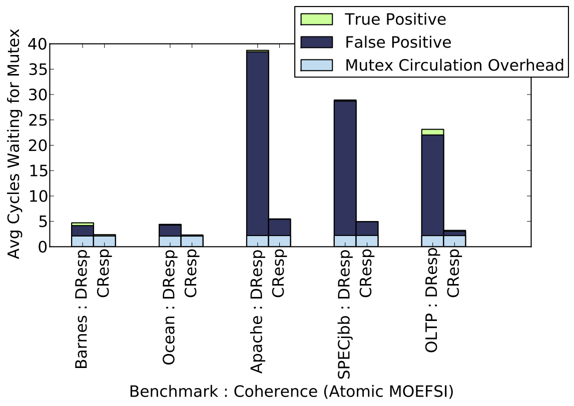
As discussed in Section 7.2, Atomic Coherence poses both potential performance pitfalls as well as opportunities. In this section, we present results that gauge the impact of these pitfalls.

Atomic protocols can increase cache miss latency due to their inherent serialization overhead (acquiring the mutex before initiating the transaction), as well as unnecessary serialization (false conflicts due to a finite set of mutexes or serialization of safe read-read conflicts). Figure 7.6(a) shows Atomic Coherence's performance relative to non-atomic `MESI` (top) and `MOEFSI` (bottom). To summarize: relative to non-atomic `MESI`, Atomic `MOEFSI` achieves 14% better performance with significantly less complexity. Relative to non-atomic `MOEFSI`, atomic `MOEFSI` causes a minor performance degradation: less than 2% average slowdown. We believe this is a small price to pay for dramatic protocol simplification.

We found that Barnes and Ocean, when run with in-order cores, are bound by computation and are mostly insensitive to the coherence protocol (and also insensitive to Atomic Coherence). For the commercial workloads, `Atomic CResp` performs better than `Atomic DResp`, because it benefits from early release of the mutex. Releasing the mutex when main memory is the only step remaining in the request-process eases pressure on the mutex pool.

Figure 7.6(b) shows the sources of mutex latency. When a mutex is free, it takes on average 2 cycles (and at most 4 cycles) for the mutex to circulate and reach the node, as shown by the bottom (light blue) bar in the stack. When a mutex is unavailable, it is because either (1) an independent request to the *same* block is active (true positive) or (2) an independent request to a *different* block is active (false positive). True positives are rare, which enforces observations that races to a block are also rare [94]. False positives are a major contributor to `Atomic DResp` and result from the pressure caused by memory misses holding mutexes for long periods of time. The minor complexity benefit of `Atomic DResp` is difficult to justify given this issue. Later, we show how `Atomic DResp` also is more sensitive to its resource budget, generally requiring more mutexes or efficient hashing to achieve satisfactory performance. Thus, given these tradeoffs, `Atomic CResp` more reasonable and complexity-effective policy of the two.





(a) Speedup

(b) Cycle Overhead for Mutex Acquisition on L2 Miss

Figure 7.6 **Performance and Overhead for Atomic Coherence.** Configurations use 1024 mutexes across four waveguides.

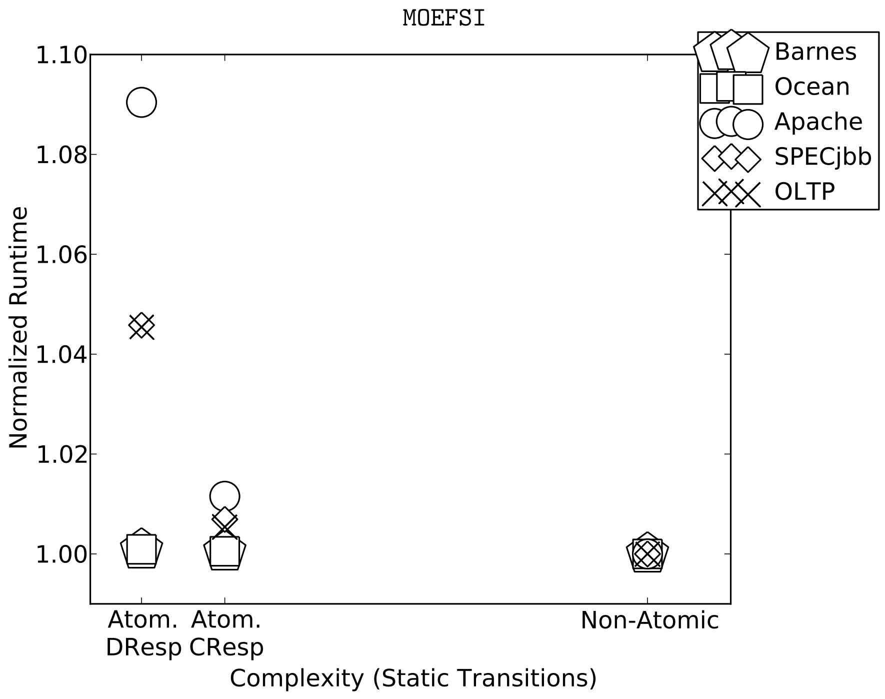


Figure 7.7 **Complexity Vs Normalized Runtime.** Complexity is a count of static L2 state transitions. Runtime is normalized to non-atomic MOEFSI. Points nearest to the origin have high performance and low complexity.

7.4.2.3 Complexity-Effective Performance

Figure 7.7 puts complexity and performance into perspective, showing the relation between the two. `Atomic DResp` is the simplest protocol, but takes a performance hit when running the commercial workloads. Adding a few more state-transitions, `Atomic CResp` improves performance on these workloads and comes very close to the performance of the substantially more complicated non-atomic `MOEFSI` protocol.

7.4.2.4 Power

Using a link-loss approach for laser power and an additive approach for ring power, as discussed in Section 4.1.3, an average loaded atomic waveguide with variable injectors will consume 52 milliwatts of power. Thus, our four waveguide systems consume 208 milliwatts. Each waveguide has 64 wavelength, is 5 cm in length, and supports 16 nodes clocked at 5 GHz.

7.4.2.5 Sensitivity Analysis

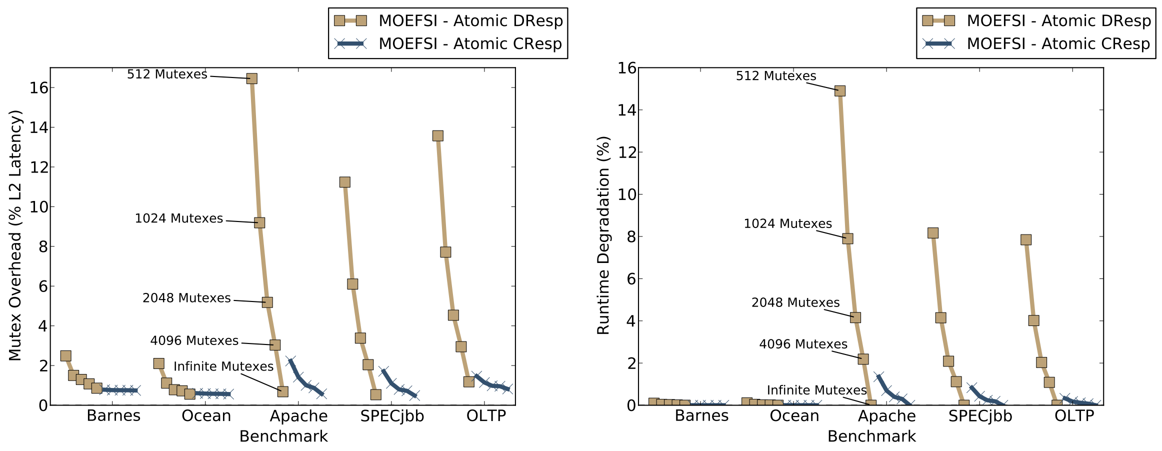
The performance of Atomic Coherence is highly dependent on the speed of mutex acquisition. Here we study three variables that affect acquisition time: number of mutexes, latency of mutexes, and the hashing of mutexes to optical tokens. We study each variable in isolation.

Mutex Number. The mutex overhead is largely dependent on the size of the mutex pool (to avoid false positives) and the circulation time of a free mutex (to promptly seize free mutexes). Figure 7.8(a) shows how adding more mutexes, between 512–4096 (or 2–16 waveguides), decreases mutex overhead (left) and improves performance (right). `Atomic DResp` is highly sensitive to the pool size, resulting in a larger range of impact. The effect is largely due to queueing delay, as responses contend for a limited set of mutex resources. `Atomic CResp` is less sensitive with less than 2% overhead in the majority of data points and is the preferred design point in terms of complexity-effectiveness.

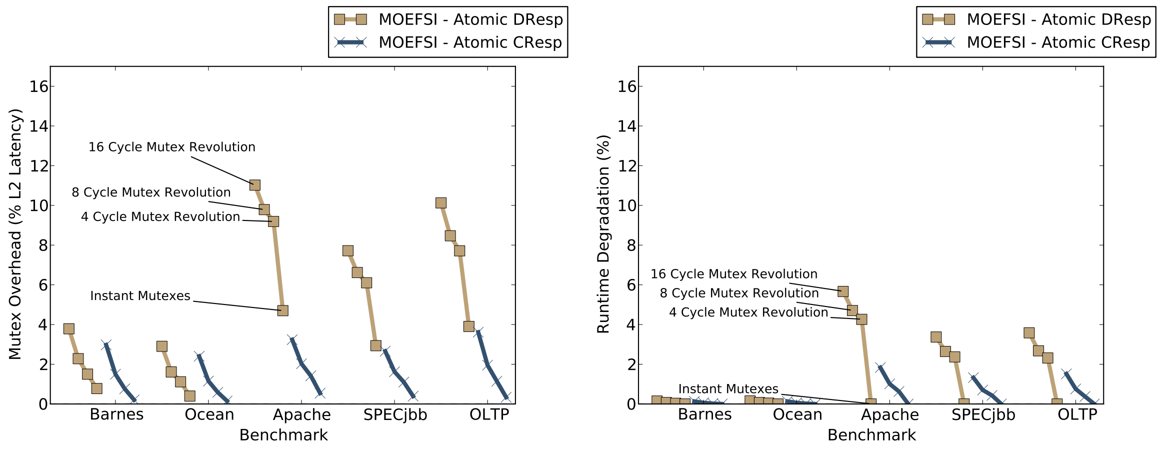
Mutex Latency. Varying the latency of a mutex's revolution period also affects the system, as shown in Figure 7.8(b). The minimum circulation time of a mutex is set by light's propagation in the topology. The circulation time can be arbitrarily extended by adding buffers to the dateline, or another fixed point. We varied the circulation time from 0 (instant) to 16 cycles. For the points we looked at, varying this latency has a smaller impact than varying mutexes, but remains non-negligible.

Hash Function. Avoiding hashing collisions to unrelated addresses is important for performance. We studied three hashing functions: Direct Mapped, `XOR5`, and H3. Direct Mapped hashing simply calculates a mutex by extracting the bits just above the block offset. `XOR5` is a variation on direct mapped. It exclusive-or's 5 of the higher-order bits into

(a) Mutex # Sensitivity : Pool = $[512, 1024, 2048, 4096, Infinite]$ Mutexes



(b) Mutex Latency Sensitivity : Circulation = $[16, 8, 4, Instant]$ Cycles



(c) Hash Function Sensitivity : Function = [Direct, XOR5, H3]

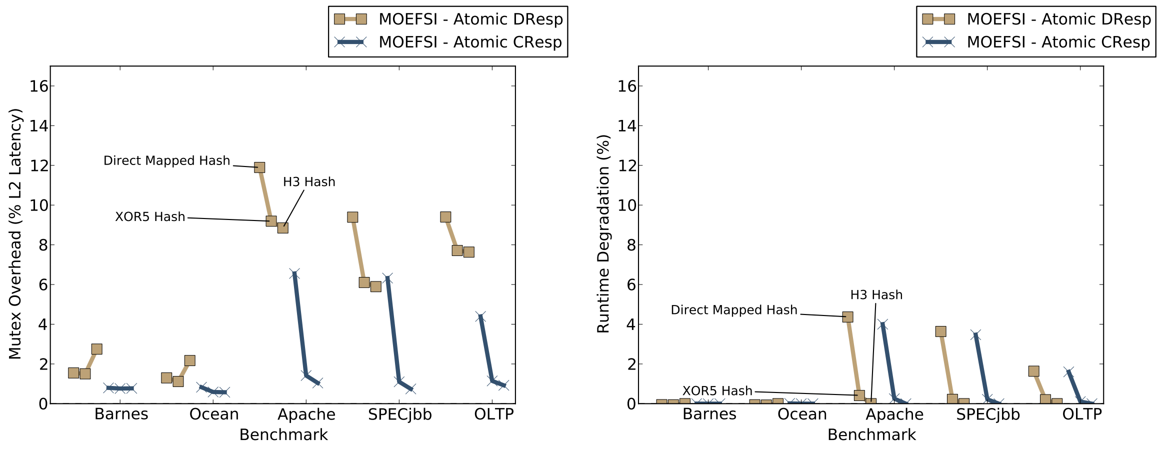
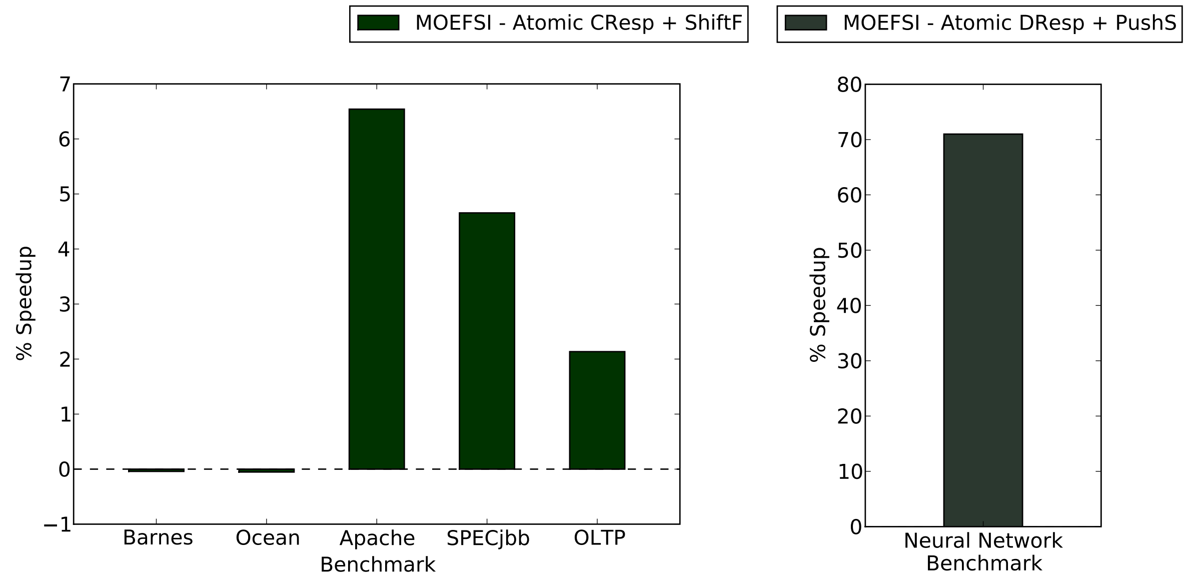


Figure 7.8 **Sensitivity Study for Atomic Coherence.** How the number of mutexes, circulation latency, and hash function affects the portion of L2 miss time spent waiting to seize a mutex (left) and the system runtime degradation (right). Runtime degradation is relative to the best performing configuration (right-most series).



(a) ShiftF

ΔL2 Transitions = -5

(b) PushS

ΔL2 Transitions = 13

Figure 7.9 **Atomic Coherence Optimizations' Speedup.** Speedup is relative to non-atomic `MOEFSI`.

the upper bits of the direct mapping index. Finally, H3 is a universal hash functions that provides uncorrelated and uniformly distributed hashing [134] but has high hardware cost.

Our findings, shown in Figure 7.8(c), indicate the quality of the hash function does matter. `XOR5` performs comparably to the more complex $H3$.

7.4.2.6 Performance Optimization Results

In Section 7.2.4.2 we discussed straightforward atomic implementations of `PushS`, `ShiftF`, and implicit acknowledgements. Here we evaluate their performance.

ShiftF. Figure 7.9(a) shows how `ShiftF` improves performance up to ˜6%, mainly by preserving an on-chip sourcer of data via the `F` state, improving performance for subsequent would-be misses.

Our implementation slightly reduces complexity, too. When we added `ShiftF` to `Atomic CResp`, we removed 5 transitions. Given that a miss occurs to on-chip data that is coherent with memory, our original `Atomic CResp` could source a miss' data from from memory *or* from an on-chip sharer, depending on whether there was `O/F` sharer or not. With `ShiftF`, such a miss may $\hat{o}$nly source from an on-chip sharer because the protocol guarantees an `O/F` sharer. This effect removes some of the possible responses a miss is able to see, also shrinking the state-event space.

PushS. PushS improves neural network performance by anticipating read misses to a block and pushing the S-state data into prospective caches. We found that our scientific and commercial workloads did not benefit from this optimization, because they did not have cyclic patterns of many-readers and a single writer. We wrote a neural network micro-benchmark that exhibits this behavior and is able to capture the potential of PushS, improving performance over non-atomic MOEFSI by 70%, as shown in Figure 7.9(b). This pattern is traditionally expensive to implement and is one reason why programmers use locks sparingly. Our PushS protocol addresses this cost with a simple high-performance implementation.

Implicit Acknowledgements. All of our atomic protocols implement implicit acknowledgements to mitigate the large number of ACKs required by non-atomic protocols. Figure 7.10 provides information about the resulting ACK reduction, in terms of traffic makeup and bandwidth utilization. Atomic Coherence goes a long way to eliminating these ACKs in Apache, SPECjbb, and OLTP. In these workloads, implicit ACKs save almost 20% of the messages. For the interconnect we evaluated on, which sizes control and data messages equally, this would translate to 20% savings in bandwidth (top figure). For an interconnect that is able to support different flit sizes for different message types (bottom figure), the implicit ACKs save less than 2% of the bandwidth.

Bandwidth pressure can directly affect performance by causing messages to wait for bandwidth service. The pressure can also have second-order effects on buffering resources, such as MSHRs, since spending more time in the interconnect translates to longer occupancies in the MSHRs. If our configuration was bandwidth or MSHR constrained, this savings could have translated into better performance for Atomic Coherence.

7.5 Discussion

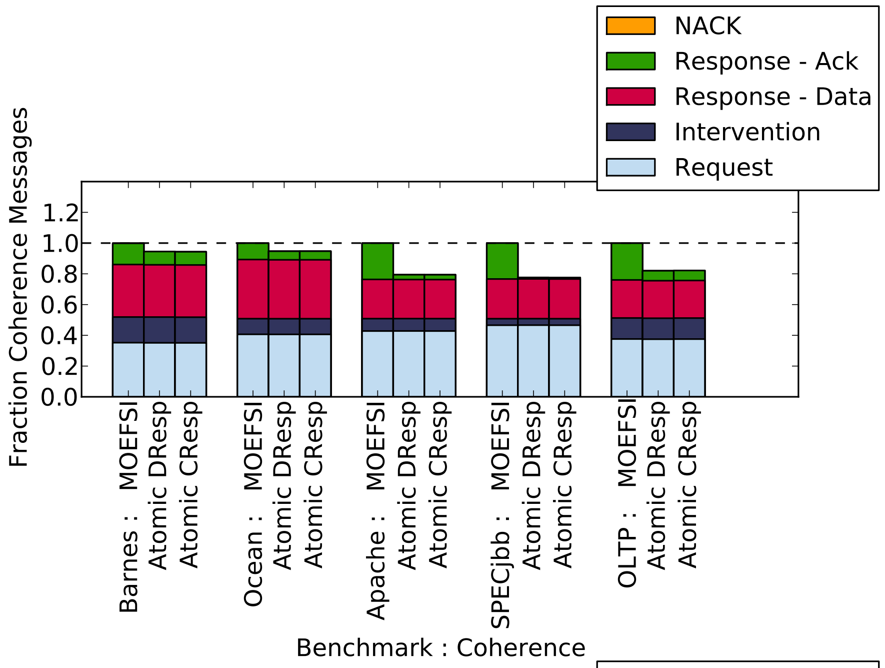
7.5.1 Other Opportunities for Aggressive Coherence

Our PushS and ShiftF protocols speculate for performance by pushing data into caches for future reference or shifting ownership responsibility for future on-chip misses. Non-atomic protocols, for a long time, have been also using speculation for the same goal [135, 136, 137, 138, 139, 81]. However, prior proposals have required additional coherence events and protocol states, which beget additional split and race transitions in the protocol, contributing to state-space explosion and creating verification headaches.

We believe Atomic Coherence can substantially simplify speculative protocols by dampening the state-space explosion. It could be used either in part, protecting speculative activity from interfering with non-speculative activity, or in full, acting as an umbrella to protect the entire protocol.

7.5.2 Relation to Token Coherence

In both Atomic Coherence and Token Coherence [94], tokens are used for cache coherence purposes. Atomic Coherence employs tokens to resolve races. Token Coherence, on the other hand, employs tokens to obtain coherency



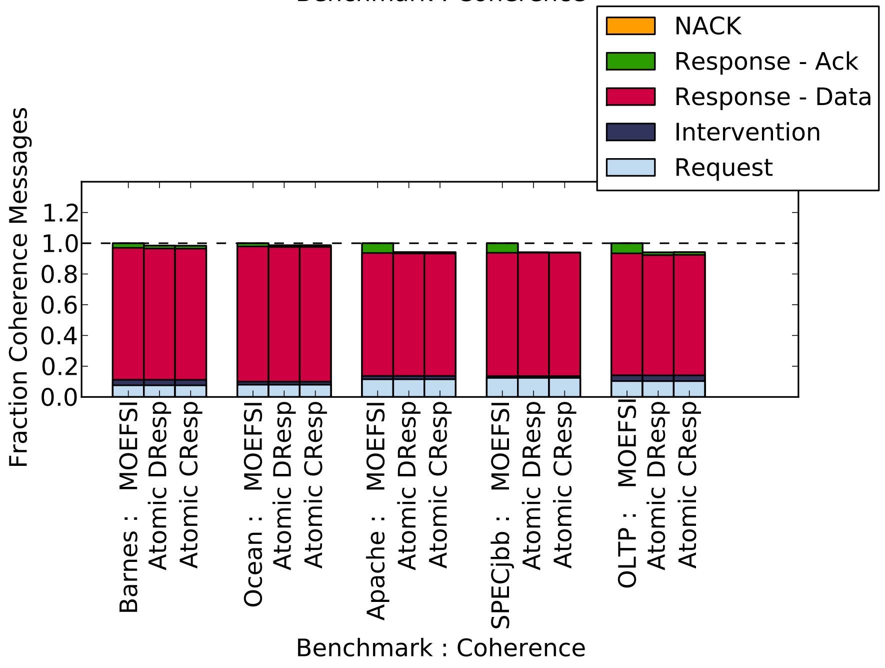


Figure 7.10 **Coherence Traffic Breakdown for `MOEFSI`.** The top chart shows breakdown by message-count and the bottom chart shows breakdown by bandwidth (not counting internal flit fragmentation). All results are normalized to non-atomic `MOEFSI`. "Response - Other" represents all non-`ACK` responses (includes data).

permissions. To these ends, Atomic Coherence releases the token after the block transitions to a stable state, while Token Coherence holds tokens for the lifetime of the block. Also, Token Coherence has no pre-preemptive mechanism to prevent or manage races, requiring it to resort to reactive methods (e.g. persistent requests and broadcast) to resolve races.

Thus, the two ideas are somewhat orthogonal: Atomic Coherence works with the front-side (races) of a coherence transaction while Token Coherence works with the back-side of the coherence transaction (maintaining and transferring state permissions). The two approaches could theoretically be combined to provide a race-free token-based protocol.

7.5.3 Data-Race-Free

Data-race-free models segregate memory operations into data and synchronization [140]. The models guarantee some type of memory consistency, usually sequential consistency [141], as long as races only occur to synchronization variables. If races occur to data, the memory operation's result is undefined. Guaranteeing consistency under agreed circumstances (i.e. synchronization variables), instead of all circumstances, may help many-cores provide scalable coherency and consistency [142].

As described, our proposal detects races to all types of memory operations. If we have an Atomic Coherence protocol that is also sequentially consistent (like the one we evaluated), supporting data-race freedom is trivial. It simply means that only synchronization variables must use the Atomic substrate; data operations may bypass it. Also, because synchronization should occur with less frequency than data, a data-race-free implementation should require fewer optical resources.

7.5.4 Memory Consistency Implications

Atomic Coherence may impact the function and performance of the consistency model by perturbing order and atomicity.

Ordering The processor supports the memory model by enforcing the order between loads and stores (and any combination of the two). For strict models without speculation, re-ordering is disallowed. For weaker memory models, re-ordering is allowed (often for the sake of gaining performance through memory level parallelism). Whatever the model, Atomic Coherence works below the processor, in the caches, and does not affect the ordering decisions carried out by the processor.

Atomic Coherence does affect the order that is carried out at the interconnect, by adding stalls that might not have been present in a non-atomic coherence protocol. These stalls are not unique to Atomic Coherence and are part of most any interconnect that has queueing.

Atomicity As the name implies, Atomic Coherence has atomicity implications which can affect the coherence protocol's support of the memory consistency model. Atomic Coherence causes consistency to err on the side of strictness. A relaxed consistency model may allow a processor to read the value of a write before all other processors have seen the value. In an invalidate protocol, Atomic Coherence would prevent such relaxations, because the store that held the mutex would invalidate all blocks and would use the mutex to protect other nodes from re-reading the block until the block was system-wide coherent. The mutex protects the block until all nodes have received the pushed value.

If desired, some non-atomicity could be built into Atomic Coherence. Write atomicity says that the effect of a node's write is seen by the entire system at the same time. Atomic Coherence can purposefully violate write atomicity. While a store miss holds a mutex, it could push the new value into caches. If it did this before it invalidated all copies of the block, store atomicity would be violated; some nodes could read the old value while other nodes could read the new value. It is unclear if this relaxation benefits performance, or if it is worth the effort. Traditionally, relaxing store atomicity has been partially done out of convenience. For Atomic Coherence, undoing atomicity is not convenient and requires some non-trivial effort.

7.6 Summary

In this chapter we presented Atomic Coherence. Atomic Coherence does away with race transitions in the Coherence substrate, leaving only stable transitions and split transitions. Whereas split states are currently the Achilles heel of coherence protocol designers, with Atomic Coherence split-states are race-free and their transitions are predictable.

We evaluated a version of Atomic Coherence that uses optical tokens to guarantee mutual exclusion. Our results show that Atomic Coherence dramatically simplifies the task of implementing coherence protocols, eliminating all race transitions and enabling a sophisticated and aggressive protocol that is quantitatively simpler than the original baseline protocol. Our experiments also explored the design space and concluded, for our target system, Atomic Coherence could be implemented with modest overhead.

Chapter 8

Summary and Reflections

Many-core communication systems must provide performance to meet growing computational needs. They must also provide functionality to manage massive concurrency, so that programmers may reason about many-core operation. This dissertation contributes many-core concurrency solutions for performance and function.

Below, I summarize these contributions (Section 8.1) and reflect on some of my findings (Section 8.2).

8.1 Summary

This dissertation is grounded in the philosophy that nanophotonics is a revolutionary technology and should be exploited to benefit all facets of many-core communication, not just raw data transmission. With this in mind, we offer optical tokens as a way of providing low-cost mutual exclusion. We show the versatility of optical tokens by solving two many-core problems, one old and one new.

First, we applied optical tokens to a new problem, the arbitration of MWSR nanophotonic interconnects. For a long time, electrical arbiters have struggled with providing electrical crossbars the timely decisions they require. With the emergence optical crossbars, the decision latency has become even more critical. This thesis contributes a set of nanophotonic arbiters that pair well with nanophotonic interconnects, allowing for fair and full channel utilization. It also contributes an optical flow control solution. Flow control credits can quickly reach any requester, allowing for efficient distribution of credits and fast turn around of credits. The fairness solution aims for a max-min allocation of bandwidth and comes very close for adversarial traffic.

Next, we applied optical tokens to an old problem, the complexity-vs-performance trade-off of cache coherence protocols. State-of-the-art cache coherence protocols are fast, but they are also complex. Our results show that fast protocols do not have to be complex. Complexity arises from rare protocol races, which are an inevitable result of concurrency. Our Atomic Coherence proposal simplifies protocols by using optical tokens to mitigate races in the coherence subsystem. The tokens act as mutexes, protecting misses during their quest for data and permissions. All conflicting requests that try to seize a taken mutex will fail. We show that Atomic Coherence can simplify a protocol dramatically while having little effect on bottom-line performance. Furthermore, we showed how simple protocol optimization are simple and can compensate for the minor performance losses.

8.2 Reflections

8.2.1 Speculation Freedom

Our proposals show just two examples of how low-latency access to globally shared information can benefit many-cores. Our proposals depend on the high-performance of optics, which allowed us to study speculation-free solutions.

Avoiding speculation has been a recent trend used within processing cores to increase power-performance efficiency. Avoiding speculation in the communication fabric can also benefit power-performance efficiency, but has not been extensively studied by our community. This work studies non-speculative communication: our arbiters do not have to be speculatively pipelined for performance and our coherence protocols can issue requests without speculating about conflicts.

I believe we can extend the idea of speculation-free communication to other areas for simpler, lower power systems. Aside from arbitration and coherence, other areas could include memory consistency, transactional memory, or locking [143, 135]. Current implementations in these areas use expensive forms of speculation that have hefty roll-back costs. Furthermore, costs is incurred not only during the rare mispeculation, but also when there is no mispeculation and periodic bookkeeping is required (e.g. checkpointing). As systems grow larger, the associated costs of checkpointing demands can also grow.

We believe our optical token proposal is a good candidate to eliminate or defray speculation in all parts of the communication subsystem. Traditional technologies used in new ways, or in old ways, could also work. For example, simply returning to blocking cache coherence protocols may work well given that a single light-weight core may be less sensitive to communication latency and may be able to tolerate coherence blocking.

Speculation-free systems are often simpler systems. Simple systems have the benefit that they are easier to design and re-design. When a customer wants a slight variation on a product, a supplier can easily alter a simple design. From a business perspective, this helps create faster cycles and offers the customer more variety.

8.2.2 Applicability of Contributions to Other Technologies

In this work, we applied optical tokens to systems that already used optics for data communication. Below we consider how system adjustments might play out.

Optical Tokens in Otherwise Electrical Systems. If we employ optical tokens in otherwise purely electrical systems, we have to consider a different set of constraints.

The optical arbiters may be overkill for electrical meshes, which do not require global arbitration and can manage local arbitration very well. More benefit may be had in electrical crossbars, especially crossbars in high-radix router chips, where allocators cannot keep up. If the crossbar can switch fast enough, optical arbitration may be of assistance.

Atomic Coherence with an optical atomic substrate and an electrical coherence substrate should function correctly but would probably need performance tuning. It is safe to say that an electrical mesh will have longer communication latencies than an optical interconnect. Longer communication latencies mean coherence transitions take more time and mutexes are held longer. In Section 7.2.4.1, we used Little's Law to explain how long coherence delays necessitate more mutex resources. Thus, to keep the resource requirements reasonable, mutexes should be released as soon as possible, but atomicity should not be compromised. If protecting all of shared memory all of the time is too prohibitive, it may make sense to only protect a subset of addresses or a subset of the protocol's finite state machine.

Principles in Purely Electrical Systems. We showed a promising optical implementation of Atomic Coherence. We believe Atomic Coherence may also be successful in purely electrical systems. Whereas transmitting mutexes across the chip is fast and cheap for optics, it is slow and expensive for electronics. Nor is it practical for all mutexes to continually circulate in an electrical interconnect; mutexes will need to spend most of their time stored in on-chip memory. Thus, we need the mutexes to exploit locality. Ideally, we would like all mutex requests to be satisfied locally, in a matter of a few cycles.

Fortunately, architects understand locality well. I believe the mutexes can adopt and adapt already existing locality techniques and tools to achieve better performance. Two especially useful tools would be mutex migration and coarse grain mutexes. Mutex migration could move mutexes to where they may be of most use. The idea is similar to cache-block migration [139]. Coarse grain mutexes could build on ideas from coarse grain coherence, so that a single "super"-mutex could manage one, or many, mutexes [144].

LIST OF REFERENCES

[1] K. Asanovic, R. Bodik, B. C. Catanzaro, J. J. Gebis, P. Husbands, K. Keutzer, D. A. Patterson, W. L. Plishker, J. Shalf, S. W. Williams, and K. A. Yelick, "The Landscape of Parallel Computing Research: A View from Berkeley," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-183, Dec 2006. [Online]. Available: http://www.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-183.html

[2] G. E. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, April 1965.

[3] "Intel news release: Dual core era begins," 2005. [Online]. Available: http://www.intel.com/pressroom/archive/releases/2005/20050418comp.htm

[4] M. Kistler, M. Perrone, and F. Petrini, "Cell Multiprocessor Communication Network: Built for Speed," *IEEE Micro*, vol. 26, no. 3, pp. 10–23, 2006.

[5] U. Nawathe, M. Hassan, L. Warriner, K. Yen, B. Upputuri, D. Greenhill, A. Kumar, and H. Park, "An 8-Core 64-Thread 64b Power-Efficient SPARC SoC," in *ISSCC*, Feb 2007, pp. 108–109.

[6] Tilera Corporation, "TILE64 Processor Family," URL: http://www.tilera.com/products/processors.php.

[7] J. D. Davis, J. Laudon, and K. Olukotun, "Maximizing cmp throughput with mediocre cores," in *PACT '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 51–62.

[8] J. Laudon and L. Spracklen, "The coming wave of multithreaded chip multiprocessors," *Int. J. Parallel Program.*, vol. 35, no. 3, pp. 299–330, 2007.

[9] C. E. Leiserson, Z. S. Abuhamdeh, D. C. Douglas, C. R. Feynman, M. N. Ganmukhi, J. V. Hill, D. Hillis, B. C. Kuszmaul, M. A. St. Pierre, D. S. Wells, M. C. Wong, S.-W. Yang, and R. Zak, "The network architecture of the Connection Machine CM-5 (extended abstract)," in *SPAA '92*. New York, NY, USA: ACM, 1992, pp. 272–285.

[10] W. T.-Y. Hsu and P.-C. Yew, "An effective synchronization network for hot-spot accesses," *ACM Trans. Comput. Syst.*, vol. 10, no. 3, pp. 167–189, 1992.

[11] C. J. Beckmann and C. D. Polychronopoulos, "Fast barrier synchronization hardware," in *Supercomputing '90*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1990, pp. 180–189.

[12] S. L. Scott, "Synchronization and communication in the t3e multiprocessor," in *ASPLOS '96*. New York, NY, USA: ACM, 1996, pp. 26–36.

[13] W. J. Dally and B. Towles, "Route packets, not wires: on-chip inteconnection networks," in *DAC '01*. New York, NY, USA: ACM, 2001, pp. 684–689.

[14] M. R. Marty and M. D. Hill, "Virtual hierarchies to support server consolidation," in *ISCA '07*. New York, NY, USA: ACM, 2007, pp. 46–56.

[15] T. Lovett and R. Clapp, "Sting: a cc-numa computer system for the commercial marketplace," in *ISCA '96*. New York, NY, USA: ACM, 1996, pp. 308–317.

[16] S. Williams, "Panel Discussion: What's Next After CMOS?" in *Hot Chips-19*, August 2007.

[17] A. Louri and A. Kodi, "SYMNET: An Optical Interconnection Network for Scalable High-Performance Symmetric Multiprocessors," *Applied Optics*, vol. 42, no. 17, 2003.

[18] N. Kirman, M. Kirman, R. Dokania, J. Martinez, A. Apsel, M. Watkins, and D. Albonesi, "Leveraging Optical Technology in Future Bus-based Chip Multiprocessors," *MICRO-39*, vol. 9, no. 13, pp. 492–503, 2006.

[19] D. Vantrease, R. Schreiber, M. Monchiero, M. McLaren, N. P. Jouppi, M. Fiorentino, A. Davis, N. Binkert, R. G. Beausoleil, and J. H. Ahn, "Corona: System Implications of Emerging Nanophotonic Technology," *ISCA-35*, 2008.

[20] D. Miller, "Rationale and Challenges for Optical Interconnects to Electronic Chips," *Proceedings of the IEEE*, vol. 88, no. 6, pp. 728–749, June 2000.

[21] D. Sylvester and K. Keutzer, "Impact of small process geometries on microarchitectures in systems on a chip," *Proceedings of the IEEE*, vol. 89, no. 4, pp. 467–489, 2001.

[22] J. Davis, R. Venkatesan, A. Kaloyeros, M. Beylansky, S. Souri, K. Banerjee, K. Saraswat, A. Rahman, R. Reif, and J. Meindl, "Interconnect limits on gigascale integration (GSI) in the 21st century," *Proceedings of the IEEE*, vol. 89, no. 3, pp. 305–324, 2001.

[23] B. Amrutur and M. Horowitz, "Speed and Power Scaling of SRAM's," *IEEE Journal of Solid-State Circuits*, vol. 35, no. 2, pp. 175–185, 2000.

[24] B. Cherkauer and E. Friedman, "A unified design methodology for CMOS tapered buffers," *IEEE Transactions on Very Large Scale Integration(VLSI) Systems*, vol. 3, no. 1, pp. 99–111, 1995.

[25] M. Haurylau, G. Chen, H. Chen, J. Zhang, N. Nelson, D. Albonesi, E. Friedman, and P. Fauchet, "On-chip optical interconnect roadmap: Challenges and critical directions," *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 12, no. 6, p. 1699, 2006.

[26] G. Chen, H. Chen, M. Haurylau, N. Nelson, D. Albonesi, P. Fauchet, and E. Friedman, "Predictions of CMOS compatible on-chip optical interconnect," *INTEGRATION, the VLSI journal*, vol. 40, no. 4, pp. 434–446, 2007.

[27] S. Scott, "The Impact of Optics on HPC System Interconnects," in *Hot Interconnects*, August 2009.

[28] Q. Xu, D. Fattal, and R. Beausoleil, "Silicon microring resonators with 1.5-μm radius," *Optics Express*, vol. 16, no. 6, pp. 4309–4315, 2008.

[29] Q. Xu, S. Manipatruni, B. Schmidt, J. Shakya, and M. Lipson, "12.5 Gbit/s carrier-injection-based silicon micro-ring silicon modulators," *Optics Express*, vol. 15, no. 2, pp. 430–436, 2007.

[30] J. Xue, A. Garg, B. Ciftcioglu, S. Wang, I. Savidis, J. Hu, M. Jain, M. Huang, H. Wu, E. G. Friedman, G. W. Wicks, and D. Moore, "An Intra-Chip Free-Space Optical Interconnect," *3rd Workshop on Chip Multiprocessor Memory Systems and Interconnects. Held in Conjunction with ISCA-36*, 2009.

[31] M. Lipson, "Guiding, modulating, and emitting light on silicon-challenges and opportunities," *Journal of Lightwave Technology*, vol. 23, no. 12, p. 4222, 2005.

[32] W. Bogaerts, R. Baets, P. Dumon, V. Wiaux, S. Beckx, D. Taillaert, B. Luyssaert, J. Campenhout, P. Bienstman, and D. Thourhout, "Nanophotonic waveguides in silicon-on-insulator fabricated with CMOS technology," *Journal of Lightwave Technology*, vol. 23, no. 1, p. 401, 2005.

[33] D. Vantrease, N. Binkert, R. Schreiber, and M. H. Lipasti, "Light speed arbitration and flow control for nanophotonic interconnects," in *MICRO-42*. New York, NY, USA: ACM, 2009, pp. 304–315.

[34] Y. Chang, C. Wang, and L. Coldren, "High-efficiency, high-speed VCSELs with 35 Gbit/s error-free operation," *Electron. Lett*, vol. 43, no. 19, pp. 1022–1023, 2007.

[35] M. Cianchetti, J. Kerekes, and D. Albonesi, "Phastlane: a rapid transit optical routing network," in *ISCA-36*. ACM New York, NY, USA, 2009.

[36] N. Kırman and J. Martınez, "A Power-efficient All-optical On-chip Interconnect Using Wavelength-based Oblivious Routing," *ASPLOS-XV*, 2010.

[37] ITRS, "International technology roadmap for semiconductors," 2007, http://public.itrs.net.

[38] B. Kim and V. Stojanović, "Equalized interconnects for on-chip networks: Modeling and optimization framework," in *Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design*. IEEE Press, 2007, pp. 552–559.

[39] D. Miller, "Device Requirements for Optical Interconnects to Silicon Chips," *Proceedings of the IEEE*, vol. 97, no. 7, 2009.

[40] B. R. Koch, A. W. Fang, O. Cohen, and J. E. Bowers, "Mode-locked silicon evanescent lasers," *Optics Express*, vol. 15, no. 18, p. 11225, Sep 2007.

[41] B. Guha, B. Kyotoku, and M. Lipson, "CMOS-compatible athermal silicon microring resonators," *Opt. Express*, vol. 18, pp. 3487–3493, 2010.

[42] S. Manipatruni, R. Dokania, B. Schmidt, N. Sherwood-Droz, C. Poitras, A. Apsel, and M. Lipson, "Wide temperature range operation of micrometer-scale silicon electro-optic modulators," *Optics Letters*, vol. 33, no. 19, pp. 2185–2187, 2008.

[43] S. Xiao, M. Khan, H. Shen, and M. Qi, "Multiple-channel silicon micro-resonator based filters for WDM applications," *Optics Express*, vol. 15, no. 12, pp. 7489–7498, 2007.

[44] B. Small, B. Lee, K. Bergman, Q. Xu, and M. Lipson, "Multiple-wavelength integrated photonic networks based on microring resonator devices," *Journal of Optical Networking*, vol. 6, no. 2, pp. 112–120, 2007.

[45] Y. Vlasov and S. McNab, "Losses in single-mode silicon-on-insulator strip waveguides and bends," *Opt. Quantum Electron*, vol. 26, p. 977, 1994.

[46] K. Lee, D. Lim, L. Kimerling, J. Shin, and F. Cerrina, "Fabrication of ultralow-loss Si/SiO_2 waveguides by roughness reduction," *Optics letters*, vol. 26, no. 23, pp. 1888–1890, 2001.

[47] W. Bogaerts, P. Dumon, D. Thourhout, and R. Baets, "Low-loss, low-cross-talk crossings for silicon-on-insulator nanophotonic waveguides," *Optics Letters*, vol. 32, no. 19, pp. 2801–2803, 2007.

[48] R. Nagarajan, M. Kato, J. Pleumeekers, P. Evans, D. Lambert, A. Chen, V. Dominic, A. Mathur, P. Chavarkar, M. Missey, A. Dentai, S. Hurtt, J. Bäck, R. Muthiah, S. Murthy, R. Salvatore, C. Joyner, J. Rossi, R. Schneider, M. Ziari, H.-S. Tsai, J. Bostak, M. Kauffman, S. Pennypacker, T. Butrie, M. Reffle, D. Mehuys, M. Mitchell, A. Nilsson, S. Grubb, F. Kish, and D. Welch, "Large-scale photonic integrated circuits for long-haul transmission and switching," *J. Opt. Netw.*, vol. 6, no. 2, pp. 102–111, 2007. [Online]. Available: http://jon.osa.org/abstract.cfm?URI=JON-6-2-102

[49] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. Loh, D. McCaule, P. Morrow, D. Nelson, D. Pantuso *et al.*, "Die stacking (3d) microarchitecture," in *Proceedings of the 39th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 2006, pp. 469–479.

[50] A. Joshi, C. Batten, Y.-J. Kwon, S. Beamer, I. Shamim, K. Asanovic, and V. Stojanovic, "Silicon-photonic clos networks for global on-chip communication," in *NOCS '09*. Washington, DC, USA: IEEE Computer Society, 2009, pp. 124–133.

[51] C. Batten, A. Joshi, J. Orcutt, A. Khilo, B. Moss, C. Holzwarth, M. Popovic, H. Li, H. Smith, J. Hoyt, F. Kartner, R. Ram, V. Stojanovic, and K. Asanovic, "Building manycore processor-to-dram networks with monolithic silicon photonics," in *HOTI '08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 21–30.

[52] R. Chang, N. Talwalkar, C. Yue, and S. Wong, "Near speed-of-light signaling over on-chip electrical interconnects," *IEEE Journal of Solid-State Circuits*, vol. 38, no. 5, pp. 834–838, 2003.

[53] B. Beckmann, "Managing wire delay in chip multiprocessor caches," Ph.D. dissertation, University of Wisconsin - Madison, 2006.

[54] M. Chang, J. Cong, A. Kaplan, M. Naik, G. Reinman, E. Socher, and S. Tam, "CMP Network-on-Chip Overlaid with Multi-Band RF-Interconnect," *HPCA '08*, February 2008.

[55] J. Kim, W. J. Dally, B. Towles, and A. K. Gupta, "Microarchitecture of a high-radix router," in *ISCA '05*. Washington, DC, USA: IEEE Computer Society, 2005, pp. 420–431.

[56] Y. Pan, P. Kumar, J. Kim, G. Memik, Y. Zhang, and A. Choudhary, "Firefly: illuminating future network-on-chip with nanophotonics," *SIGARCH Comput. Archit. News*, vol. 37, no. 3, pp. 429–440, 2009.

[57] A. Shacham, B. G. Lee, A. Biberman, K. Bergman, and L. P. Carloni, "Photonic NoC for DMA Communications in Chip Multiprocessors," in *Proceedings of Hot Interconnects*, Aug 2007, pp. 29–35.

[58] N. Kirman and J. F. Martínez, "A power-efficient all-optical on-chip interconnect using wavelength-based oblivious routing," *SIGPLAN Not.*, vol. 45, no. 3, pp. 15–28, 2010.

[59] X. Zheng, P. Koka, H. Schwetman, J. Lexau, R. Ho, J. Cunningham, and A. Krishnamoorthy, "Silicon photonic WDM point-to-point network for multi-chip processor interconnects," in *Group IV Photonics, 2008 5th IEEE International Conference on*, 2008, pp. 380–382.

[60] N. McKeown, A. Mekkittikul, V. Anantharam, and J. Walrand, "Achieving 100% throughput in an input-queued switch," *IEEE Transactions on Communications*, vol. 47, no. 8, pp. 1260–1267, 1999.

[61] A. S. Tanenbaum and J. R. Goodman, *Structured Computer Organization*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[62] N. McKeown, M. Izzard, A. Mekkittikul, W. Ellersick, and M. Horowitz, "Tiny Tera: A Packet Switch Core," *IEEE Micro*, vol. 17, no. 1, pp. 26–33, 1997.

[63] S. S. Mukherjee, F. Silla, P. Bannon, J. Emer, S. Lang, and D. Webb, "A Comparative Study of Arbitration Algorithms for the Alpha 21364 Pipelined Router," *SIGARCH Computer Architecture News*, vol. 30, no. 5, pp. 223–234, 2002.

[64] C. Minkenberg, F. Abel, P. Muller, R. Krishnamurthy, M. Gusat, P. Dill, I. Iliadis, R. Luijten, B. R. Hemenway, R. Grzybowski, and E. Schiattarella, "Designing a Crossbar Scheduler for HPC Applications," *IEEE Micro*, vol. 26, no. 3, pp. 58–71, 2006.

[65] N. McKeown, "The iSLIP scheduling algorithm for input-queued switches," *IEEE/ACM Trans. Netw.*, vol. 7, no. 2, pp. 188–201, 1999.

[66] C. Minkenberg, F. Abel, P. Muller, R. Krishnamurthy, M. Gusat, P. Dill, I. Iliadis, R. Luijten, B. R. Hemenway, R. Grzybowski, and E. Schiattarella, "Designing a crossbar scheduler for hpc applications," *IEEE Micro*, vol. 26, no. 3, pp. 58–71, 2006.

[67] P. Krishnamurthy, M. Franklin, and R. Chamberlain, "Dynamic reconfiguration of an optical interconnect," in *ANSS '03*. Washington, DC, USA: IEEE Computer Society, 2003, p. 89.

[68] C. Qiao and R. G. Melhem, "Time-division optical communications in multiprocessor arrays," in *SC '91*. New York, NY, USA: ACM, 1991, pp. 644–653.

[69] W. Farmer and E. Newhall, "An Experimental Distributed Switching System to Handle Bursty Computer Traffic," *ACM Symposium on Problems in the Optimization of Data Communications Systems*, pp. 4–33, 1969.

[70] J. Pierce, "How far can data loops go?" *IEEE Transactions on Communications*, vol. 20, no. 3, pp. 527–530, Jun 1972.

[71] E. Hafner, Z. Nenadal, M. Tschanz, R. Div, H. Ltd, and S. Berne, "A Digital Loop Communication System," *IEEE Communications*, vol. 22, no. 6, pp. 877–881, 1974.

[72] J.-H. Ha and T. M. Pinkston, "A new token-based channel access protocol for wavelength division multiplexed multiprocessor interconnects," *Journal of Parallel Distributed Computing*, vol. 60, no. 2, pp. 169–188, 2000.

[73] A. Kodi and A. Louri, "Design of a high-speed optical interconnect for scalable shared memory multiprocessors," in *High Performance Interconnects, 2004. Proceedings. 12th Annual IEEE Symposium on*, 2004, pp. 92–97.

[74] M. Marsan, A. Bianco, E. Leonardi, A. Morabito, and F. Neri, "All-optical WDM multi-rings with differentiated QoS," *Communications Magazine, IEEE*, vol. 37, no. 2, pp. 58–66, 1999.

[75] Y. Pan, J. Kim, and G. Memik, "Flexishare: Energy-efficient nanophotonic crossbar architecture through channel sharing," in *HPCA-16*, 2010.

[76] SUN Microsystems, "Developing Scalable Applications for Throughput Computing," *Technical White Paper*, 2005.

[77] H. T. Kung, T. Blackwell, and A. Chapman, "Credit-based flow control for atm networks: credit update protocol, adaptive credit allocation and statistical multiplexing," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 101–114, 1994.

[78] I. Cidon and Y. Ofek, "MetaRing: A Full-Duplex Ring with Fairness and Spatial Reuse," *IEEE Transactions on Communications*, vol. 41, no. 1, pp. 110–120, 1993.

[79] R. Falconer and J. Adams, "Orwell: A Protocol for an Integrated Services Local Network," *British Telecom Technology Journal*, vol. 3, no. 4, pp. 27–35, 1985.

[80] K. Imai, T. Ito, H. Kasahara, and N. Morita, "ATMR: Asynchronous Transfer Mode Ring Protocol," *Computer Networks and ISDN Systems*, vol. 26, no. 6-8, pp. 785–798, 1994.

[81] 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 *ISCA '99*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 294–304.

[82] J. Zebchuk, V. Srinivasan, M. K. Qureshi, and A. Moshovos, "A tagless coherence directory," in *MICRO 42*. New York, NY, USA: ACM, 2009, pp. 423–434.

[83] A. Agarwal, R. Simoni, J. Hennessy, and M. Horowitz, "An evaluation of directory schemes for cache coherence," in *ISCA '88*. Los Alamitos, CA, USA: IEEE Computer Society Press, 1988, pp. 280–298.

[84] A. Gupta *et al.*, "Reducing memory and traffic requirements for scalable directory-based cache coherence schemes," in *ICPP*, Urbana-Champaign, IL, August 1990.

[85] J. H. Choi and K. H. Park, "Segment directory enhancing the limited directory cache coherence schemes," in *IPPS '99/SPDP '99*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 258–267.

[86] J. Archibald and J.-L. Baer, "Cache coherence protocols: evaluation using a multiprocessor simulation model," *ACM Trans. Comput. Syst.*, vol. 4, no. 4, pp. 273–298, 1986.

[87] K. Lepak, "Exploring, defining, and exploiting recent store value locality," Ph.D. dissertation, University of Wisconsin - Madison, 2003.

[88] H.-m. D. Toong, S. O. Strommen, and E. R. Goodrich, II, "A general multi-microprocessor interconnection mechanism for non-numeric processing," in *CAW '80*. New York, NY, USA: ACM, 1980, pp. 115–123.

[89] D. Abts, Y. Chen, and D. J. Lilja, "Heuristics for complexity-effective verification of a cache coherence protocol implementation," Protocol Implementation., Laboratory for Advanced Research in Computing Technology and Compilers, Tech. Rep., 2003.

[90] M. R. Marty and M. D. Hill, "Coherence ordering for ring-based chip multiprocessors," in *MICRO '06*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 309–320.

[91] M. M. K. Martin, D. J. Sorin, A. Ailamaki, A. R. Alameldeen, R. M. Dickson, C. J. Mauer, K. E. Moore, M. Plakal, M. D. Hill, and D. A. Wood, "Timestamp snooping: an approach for extending smps," in *ASPLOS '00*. New York, NY, USA: ACM, 2000, pp. 25–36.

[92] N. Agarwal, L.-S. Peh, and N. Jha, "In-network snoop ordering: Snoopy coherence on unordered interconnects," in *HPCA '09*, 2009.

[93] E. Hagersten and M. Koster, "Wildfire: A scalable path for smps," in *HPCA '99*. Washington, DC, USA: IEEE Computer Society, 1999, p. 172.

[94] M. M. K. Martin, M. D. Hill, and D. A. Wood, "Token coherence: decoupling performance and correctness," in *ISCA '03*. New York, NY, USA: ACM, 2003, pp. 182–193.

[95] J. Laudon and D. Lenoski, "The sgi origin: a ccnuma highly scalable server," in *ISCA '97*. New York, NY, USA: ACM, 1997, pp. 241–251.

[96] J. R. Goodman, “Using cache memory to reduce processor-memory traffic,” in *ISCA ’83*. New York, NY, USA: ACM, 1983, pp. 124–131.

[97] D. B. Gustavson, “The scalable coherent interface and related standards projects,” *IEEE Micro*, vol. 12, no. 1, pp. 10–22, 1992.

[98] A. Raghavan, C. Blundell, and M. M. K. Martin, “Token tenure: Patching token counting using directory-based cache coherence,” in *MICRO ’08*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 47–58.

[99] A. Singhal, D. Broniarczyk, F. Cerauskis, J. Price, L. Yuan, C. Cheng, D. Doblar, S. Fosth, N. Agarwal, K. Harvey, E. Hagersten, and B. Liencres, “Gigaplane: A high performance bus for large smps,” *Hot Interconnects IV*, pp. 41–42, 1996.

[100] D. Culler, J. P. Singh, and A. Gupta, *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann, August 1998.

[101] R. Joshi, L. Lamport, J. Matthews, S. Tasiran, M. Tuttle, and Y. Yu, “Checking cache-coherence protocols with tla+,” *Form. Methods Syst. Des.*, vol. 22, no. 2, pp. 125–131, 2003.

[102] D. Abts, S. Scott, and D. J. Lilja, “So many states, so little time: Verifying memory coherence in the cray x1,” in *IPDPS ’03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*. Washington, DC, USA: IEEE Computer Society, 2003, p. 11.2.

[103] Á. Eiríksson, “Integrating formal verification methods with a conventional project design flow,” in *Design Automation Conference*. IEEE Computer Society, 1996, pp. 666–671.

[104] F. Pong, A. Nowatzyk, G. Aybay, and M. Dubois, “Verifying distributed directory-based cahce coherence protocols: S3.mp, a case study,” in *Euro-Par ’95*. London, UK: Springer-Verlag, 1995, pp. 287–300.

[105] M. Marty, J. Bingham, M. Hill, A. Hu, M. Martin, and D. Wood, “Improving multiple-CMP systems using token coherence,” in *HPCA ’05*. Citeseer, 2005.

[106] “Intel core2 extreme processor x6800 and intel core2 duo desktop processor e6000 and e4000 sequence,” p. 37, 2008. [Online]. Available: http://www.intel.com/design/processor/specupdt/313279.htm

[107] D. Wood, G. Gibson, and R. Katz, “Verifying a multiprocessor cache controller using random test generation,” *IEEE Design & Test of Computers*, vol. 7, no. 4, pp. 13–25, 1990.

[108] M. M. K. Martin, D. J. Sorin, B. M. Beckmann, M. R. Marty, M. Xu, A. R. Alameldeen, K. E. Moore, M. D. Hill, and D. A. Wood, “Multifacet’s general execution-driven multiprocessor simulator (gems) toolset,” *SIGARCH Comput. Archit. News*, vol. 33, no. 4, pp. 92–99, 2005.

[109] D. Abts, Y. Chen, and D. Lilja, “Heuristics for complexity-effective verification of a cache coherence protocol implementation,” *Laboratory for Advanced Research in Computing Technology and Compilers Technical Report No. ARCTiC*, pp. 03–04, 2003.

[110] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The SPLASH-2 Programs: Characterization and Methodological Considerations,” in *ISCA*, Jun 1995, pp. 24–36.

[111] A. R. Alameldeen, M. M. K. Martin, C. J. Mauer, K. E. Moore, M. Xu, M. D. Hill, D. A. Wood, and D. J. Sorin, “Simulating a $2m commercial server on a $2k pc,” *IEEE Computer*, vol. 36, no. 2, pp. 50–57, 2003.

[112] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: building customized program analysis tools with dynamic instrumentation," in *PLDI '05*. New York, NY, USA: ACM, 2005, pp. 190–200.

[113] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kauffman, 2004, iSBN 0122007514.

[114] A. Falcon, P. Faraboschi, and D. Ortega, "Combining Simulation and Virtualization through Dynamic Sampling," in *ISPASS*, Apr 2007.

[115] A. Kodi and A. Louri, "Performance adaptive power-aware reconfigurable optical interconnects for high-performance computing (hpc) systems," in *SC '07*. New York, NY, USA: ACM, 2007, pp. 1–12.

[116] J. Laudon and D. Lenoski, "The SGI Origin: A ccNUMA Highly Scalable Server," *SIGARCH Computer Architecture News*, vol. 25, no. 2, pp. 241–251, 1997.

[117] F. Mueller, "Prioritized Token-Based Mutual Exclusion for Distributed Systems," in *Proceedings of the Third Workshop on Object-Oriented Real-Time Systems (OORTS)*. IEEE Computer Society, 1997, p. 72.

[118] ANSI/IEEE, "Local Area Networks: Token Ring Access Method and Physical Layer Specifications, Std 802.5," Tech. Rep., 1989.

[119] F. Poon, "Silicon cross-connect filters using microring resonator coupled multimode-interference-based waveguide crossings," *Opt. Express*, vol. 16, pp. 8649–8657, 2008.

[120] J. H. Ahn, N. Binkert, A. Davis, M. McLaren, and R. S. Schreiber, "Hyperx: topology, routing, and packaging of efficient large-scale networks," in *SC '09*. New York, NY, USA: ACM, 2009, pp. 1–11.

[121] D. Bertsekas and R. Gallager, *Data Networks*. Prentice Hall, 1992.

[122] Y. Tamir and G. L. Frazier, "High-performance Multi-Queue Buffers for VLSI Communications Switches," *SIGARCH Computer Architecture News*, vol. 16, no. 2, pp. 343–354, 1988.

[123] D. D. Corso and H. Kirrman, *Microcomputer Buses and Links*. Orlando, FL, USA: Academic Press, Inc., 1986.

[124] K. Gharachorloo, M. Sharma, S. Steely, and S. Van Doren, "Architecture and design of alphaserver gs320," in *ASPLOS-IX*. New York, NY, USA: ACM, 2000, pp. 13–24.

[125] C. Williams, P. F. Reynolds, and B. R. de Supinski, "Delta coherence protocols," *IEEE Concurrency*, vol. 8, no. 3, pp. 23–29, 2000.

[126] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[127] J. Little, "A proof for the queuing formula: L= λ W," *Operations Research*, vol. 9, no. 3, pp. 383–387, 1961.

[128] K. J. Nesbit, N. Aggarwal, J. Laudon, and J. E. Smith, "Fair queuing memory systems," in *MICRO '06*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 208–222.

[129] A. Kägi, D. Burger, and J. R. Goodman, "Efficient synchronization: let them eat qolb," *SIGARCH Computer Architure News*, vol. 25, no. 2, pp. 170–180, 1997.

[130] L. Barroso, K. Gharachoroloo, R. McNamara, A. Nowatzyk, S. Qadeer, B. Sano, S. Smith, R. Stets, and B. Verghese, "Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing," *ISCA-27*.

[131] L. Yen, "Signatures in transactional memory systems," Ph.D. dissertation, University of Wisconsin - Madison, 2009.

[132] H. Hum and J. Goodman, "Forward state for use in cache coherency in a multiprocessor system," Dec 2002, US Patent App. 10/325,069.

[133] C. Ranger, R. Raghuraman, A. Penmetsa, G. Bradski, and C. Kozyrakis, "Evaluating mapreduce for multi-core and multiprocessor systems," in *HPCA '07*. Washington, DC, USA: IEEE Computer Society, 2007, pp. 13–24.

[134] J. L. Carter and M. N. Wegman, "Universal classes of hash functions (extended abstract)," in *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*. New York, NY, USA: ACM, 1977, pp. 106–112.

[135] R. Rajwar and J. R. Goodman, "Speculative lock elision: enabling highly concurrent multithreaded execution," in *MICRO '01*. Washington, DC, USA: IEEE Computer Society, 2001, pp. 294–305.

[136] J. Huh, J. Chang, D. Burger, and G. S. Sohi, "Coherence decoupling: making use of incoherence," in *ASPLOS-XI*. New York, NY, USA: ACM, 2004, pp. 97–106.

[137] A.-C. Lai and B. Falsafi, "Memory sharing predictor: the key to a speculative coherent dsm," in *ISCA '99*. Washington, DC, USA: IEEE Computer Society, 1999, pp. 172–183.

[138] B. M. Beckmann, M. R. Marty, and D. A. Wood, "Asr: Adaptive selective replication for cmp caches," in *MICRO-39*. Washington, DC, USA: IEEE Computer Society, 2006, pp. 443–454.

[139] N. Eisley, L.-S. Peh, and L. Shang, "Leveraging on-chip networks for data cache migration in chip multiprocessors," in *PACT '08*. New York, NY, USA: ACM, 2008, pp. 197–207.

[140] S. V. Adve and M. D. Hill, "Weak ordering—a new definition," in *ISCA '90*. New York, NY, USA: ACM, 1990, pp. 2–14.

[141] H.-J. Boehm and S. V. Adve, "Foundations of the c++ concurrency memory model," in *PLDI '08*. New York, NY, USA: ACM, 2008, pp. 68–78.

[142] B. Lucia, L. Ceze, K. Strauss, S. Qadeer, and H. Boehm, "Conflict Exceptions: Simplifying Concurrent Language Semantics with Precise Hardware Exceptions for Data-Races," 2010.

[143] L. Hammond, V. Wong, M. Chen, B. D. Carlstrom, J. D. Davis, B. Hertzberg, M. K. Prabhu, H. Wijaya, C. Kozyrakis, and K. Olukotun, "Transactional memory coherence and consistency," *SIGARCH Comput. Archit. News*, vol. 32, no. 2, p. 102, 2004.

[144] J. Cantin, M. Lipasti, and J. Smith, "Improving Multiprocessor Performance with Coarse-Grain Coherence Tracking," *ACM SIGARCH Computer Architecture News*, vol. 33, no. 2, pp. 246–257, 2005.