On-Chip Networks

Second Edition

Synthesis Lectures on Computer Architecture

Synthesis Lectures on Computer Architecture publishes 50- to 100-page publications on topics pertaining to the science and art of designing, analyzing, selecting and interconnecting hardware components to create computers that meet functional, performance and cost goals. The scope will largely follow the purview of premier computer architecture conferences, such as ISCA, HPCA, MICRO, and ASPLOS.

Analyzing Analytics
Rajesh Bordawekar, Bob Blainey, and Ruchir Puri
2015

Customizable Computing
Yu-Ting Chen, Jason Cong, Michael Gill, Glenn Reinman, and Bingjun Xiao
2015

Die-stacking Architecture
Yuan Xie and Jishen Zhao
2015

Single-Instruction Multiple-Data Execution
Christopher J. Hughes
2015

Power-Efficient Computer Architectures: Recent Advances
Magnus Själander, Margaret Martonosi, and Stefanos Kaxiras
2014

FPGA-Accelerated Simulation of Computer Systems
Hari Angepat, Derek Chiou, Eric S. Chung, and James C. Hoe
2014

A Primer on Hardware Prefetching
Babak Falsafi and Thomas F. Wenisch
2014

On-Chip Photonic Interconnects: A Computer Architect's Perspective
Christopher J. Nitta, Matthew K. Farrens, and Venkatesh Akella
2013

Optimization and Mathematical Modeling in Computer Architecture
Tony Nowatzki, Michael Ferris, Karthikeyan Sankaralingam, Cristian Estan, Nilay Vaish, and David Wood
2013

Security Basics for Computer Architects
Ruby B. Lee
2013

The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines, Second edition
Luiz André Barroso, Jimmy Clidaras, and Urs Hölzle
2013

Shared-Memory Synchronization
Michael L. Scott
2013

Resilient Architecture Design for Voltage Variation
Vijay Janapa Reddi and Meeta Sharma Gupta
2013

Multithreading Architecture
Mario Nemirovsky and Dean M. Tullsen
2013

Performance Analysis and Tuning for General Purpose Graphics Processing Units (GPGPU)
Hyesoon Kim, Richard Vuduc, Sara Baghsorkhi, Jee Choi, and Wen-mei Hwu
2012

Automatic Parallelization: An Overview of Fundamental Compiler Techniques
Samuel P. Midkiff
2012

Phase Change Memory: From Devices to Systems
Moinuddin K. Qureshi, Sudhanva Gurumurthi, and Bipin Rajendran
2011

Multi-Core Cache Hierarchies
Rajeev Balasubramonian, Norman P. Jouppi, and Naveen Muralimanohar
2011

A Primer on Memory Consistency and Cache Coherence
Daniel J. Sorin, Mark D. Hill, and David A. Wood
2011

Dynamic Binary Modification: Tools, Techniques, and Applications
Kim Hazelwood
2011

Quantum Computing for Computer Architects, Second Edition
Tzvetan S. Metodi, Arvin I. Faruque, and Frederic T. Chong
2011

High Performance Datacenter Networks: Architectures, Algorithms, and Opportunities
Dennis Abts and John Kim
2011

Processor Microarchitecture: An Implementation Perspective
Antonio González, Fernando Latorre, and Grigorios Magklis
2010

Transactional Memory, 2nd edition
Tim Harris, James Larus, and Ravi Rajwar
2010

Computer Architecture Performance Evaluation Methods
Lieven Eeckhout
2010

Introduction to Reconfigurable Supercomputing
Marco Lanzagorta, Stephen Bique, and Robert Rosenberg
2009

On-Chip Networks
Natalie Enright Jerger and Li-Shiuan Peh
2009

The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It
Bruce Jacob
2009

Fault Tolerant Computer Architecture
Daniel J. Sorin
2009

The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines
Luiz André Barroso and Urs Hölzle
2009

Computer Architecture Techniques for Power-Efficiency
Stefanos Kaxiras and Margaret Martonosi
2008

Chip Multiprocessor Architecture: Techniques to Improve Throughput and Latency
Kunle Olukotun, Lance Hammond, and James Laudon
2007

Transactional Memory
James R. Larus and Ravi Rajwar
2006

Quantum Computing for Computer Architects
Tzvetan S. Metodi and Frederic T. Chong
2006

On-Chip Networks, Second Edition

Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh

www.morganclaypool.com

ISBN: 9781627059145 paperback
ISBN: 9781627059961 ebook

DOI 10.2200/S00772ED1V01Y201704CAC040

A Publication in the Morgan & Claypool Publishers series
SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE

Lecture #40
Series Editor: Margaret Martonosi, *Princeton University*
Founding Editor Emeritus: Mark D. Hill, *University of Wisconsin, Madison*
Series ISSN
Print 1935-3235 Electronic 1935-3243

On-Chip Networks

Second Edition

Natalie Enright Jerger
University of Toronto

Tushar Krishna
Georgia Institute of Technology

Li-Shiuan Peh
National University of Singapore

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE #40

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

This book targets engineers and researchers familiar with basic computer architecture concepts who are interested in learning about on-chip networks. This work is designed to be a short synthesis of the most critical concepts in on-chip network design. It is a resource for both understanding on-chip network basics and for providing an overview of state of-the-art research in on-chip networks. We believe that an overview that teaches both fundamental concepts and highlights state-of-the-art designs will be of great value to both graduate students and industry engineers. While not an exhaustive text, we hope to illuminate fundamental concepts for the reader as well as identify trends and gaps in on-chip network research.

With the rapid advances in this field, we felt it was timely to update and review the state of the art in this second edition. We introduce two new chapters at the end of the book. We have updated the latest research of the past years throughout the book and also expanded our coverage of fundamental concepts to include several research ideas that have now made their way into products and, in our opinion, should be textbook concepts that all on-chip network practitioners should know. For example, these fundamental concepts include message passing, multicast routing, and bubble flow control schemes.

KEYWORDS

interconnection networks, topology, routing, flow control, deadlock, computer architecture, multiprocessor system on chip

To our families
for their encouragement and patience
through the writing of this book.

Contents

Preface

This book targets engineers and researchers familiar with many basic computer architecture concepts who are interested in learning about on-chip networks. This work is designed to be a short synthesis of the most critical concepts in on-chip network design. We envision this book as a resource for both understanding on-chip network basics and for providing an overview of state-of-the-art research in on-chip networks. We believe that an overview that teaches both fundamental concepts and highlights state-of-the-art designs will be of great value to both graduate students and industry engineers. While not an exhaustive text, we hope to illuminate fundamental concepts for the reader as well as identify trends and gaps in on-chip network research.

With the rapid advances in this field, we felt it was timely to update and review the state of the art in this second edition. We introduce two new chapters at the end of the book, as will be detailed below. Throughout the book, in addition to updating the latest research in the past years, we also expanded our coverage of fundamental concepts to include several research ideas that have now made their way into products and, in our opinion, should be textbook concepts that all on-chip network practitioners should know. For example, these fundamental concepts include message passing, multicast routing, and bubble flow control schemes.

The structure of this book is as follows. Chapter 1 introduces on-chip networks in the context of multi-core architectures and discusses their evolution from simple point-to-point wires and buses for scalability.

Chapter 2 explains how networks fit into the overall system architecture of multi-core designs. Specifically, we examine the set of requirements imposed by cache-coherence protocols in shared memory chip multiprocessors, and contrast that with the requirements in message-passing multi-cores. In addition to examining the system requirements, this chapter also describes the interface between the system and the network.

Once a context for the use of on-chip networks has been provided through a discussion of system architecture, the details of the network are explored. As topology is often a first choice in designing a network, Chapter 3 describes various topology trade-offs for cost and performance. Given a network topology, a routing algorithm must be implemented to determine the path(s) messages travel to be delivered throughout the network fabric; routing algorithms are explained in Chapter 4. Chapter 5 deals with the flow control mechanisms employed in the network; flow control specifies how network resources, namely buffers and links, are allocated to packets as they travel from source to destination. Topology, routing, and flow control all factor into the microarchitecture of the network routers. Details on various microarchitectural trade-offs and design issues are presented in Chapter 6. This chapter includes the design of buffers, switches, and allocators that comprise the router microarchitecture. Although power consumption can

be addressed through innovations in all areas of on-chip networks, we focus our new power discussion in the microarchitecture chapter as this is where many such optimizations are realized.

New Chapter 7 covers the nuts and bolts of modeling and evaluating on-chip networks, from software simulations to RTL design and emulation on FPGA, to architectural models of delay, throughput, area, and power. The chapter also guides the reader on useful metrics for evaluating on-chip networks and ideal theoretical yardsticks for comparing against.

With the plethora of industry and academia on-chip network chips now available, we dedicate a new Chapter 8 to a survey of these. The chapter provides the reader with a sweeping understanding of how the various fundamental concepts presented in the earlier chapters come together, and the implications of the design and implementation of such concepts.

Finally in Chapter 9, we leave the reader with thoughts on key challenges and new areas of exploration that will drive on-chip network research in the years to come. Substantial new research has clearly surfaced, and here we focus on various significant trends that highlight the cross-cutting nature of on-chip network research. Emerging new interconnects and devices substantially change the implementation tradeoffs of on-chip networks, and in turn prompt new designs. Newly important metrics such as resilience, due to increasing variability in the fabrication process, or quality-of-service that is prompted by multiple workloads running simultaneously on many-cores, will add new dimensions and prompt new research ideas across the community.

Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh
May 2017

Acknowledgments

We would like to thank Margaret Martonosi for her feedback and encouragement to create the second edition of this book. We continue to be grateful to Mark Hill for his feedback and support in crafting the previous edition. Additionally, we would like to thank Michael Morgan for the opportunity to contribute once again to this lecture series. Many thanks to Timothy Pinkston and Lizhong Chen for their detailed comments that were invaluable in improving this manuscript. Thanks to Mario Badr, Wenbo Dai, Shehab Elsayed, Karthik Ganesan, Parisa Khadem Hamedani, and Joshua San Miguel of the University of Toronto for proofreading our early drafts. Thanks to Georgia Tech students Hyoukjun Kwon, Ananda Samajdar for feedback on early drafts, and Monodeep Kar for help with literature surveys. Thanks also to the many students and instructors who have used the first edition over the years and provided feedback that lead to this latest edition.

Natalie Enright Jerger, Tushar Krishna, and Li-Shiuan Peh
May 2017

CHAPTER 1

Introduction

Since the introduction of research into multi-core chips in the late 1990s [40, 271, 336], on-chip networks have emerged as an important and growing field of research. As core counts increase, and multi-core processors emerge in diverse domains ranging from high-end servers to smartphones and even Internet of Things (IoT) gateways, there is a corresponding increase in bandwidth demand to facilitate high core utilization and a critical need for scalable on-chip interconnection fabrics. This diversity of application platforms has led to research in on-chip networks spanning a variety of disciplines from computer architecture to computer-aided design, embedded systems, VLSI, and more. Here, we provide a synthesis of critical concepts in on-chip networks to quickly bootstrap students and designers into this exciting field.

1.1 THE ADVENT OF THE MULTI-CORE ERA

The combined pressures from ever-increasing power consumption and the diminishing returns in performance of uniprocessor architectures have led to the advent of multi-core chips. With a growing number of transistors available at each new technology generation, coupled with a reduction in design complexity enabled by the modular design of multi-core chips, this multi-core wave looks set to stay. Recent years have seen every industry chip vendor releasing multi-core products with increasing core counts. This multi-core wave may lead to hundreds and even thousands of cores integrated on a single chip. We have already seen multi-core products targeting HPC with more than 50 cores on-die, and research prototypes with more than 100 cores. Heterogeneity is now common place in many market segments, in terms of the types of components that are integrated on-chip, which further ups the complexity of the on-chip interconnection fabric. Increasingly, besides processor cores, the on-chip fabric has to interconnect embedded memories, accelerators such as DSP modules, video processors, and graphics processors.

1.1.1 COMMUNICATION DEMANDS OF MULTI-CORE ARCHITECTURES

As the number of on-chip cores increases, a scalable low-latency and high-bandwidth communication fabric to connect them becomes critically important [43, 44, 95, 290]. Up to four or eight cores, buses and crossbars are the dominant interconnect. Buses are shared multi-bit physical channels that every core connects to and listens to, while one core can transmit at a time. Buses provide low-latency but poor bandwidth. Crossbars, which are described in more detail later in Chapter 6, are switches providing non-blocking connectivity between any pair of

cores. They have high-bandwidth and fairly low delays, but scale poorly in terms of area and power. As a result, on-chip networks are fast replacing buses and crossbars to emerge as the pervasive communication fabric in many-core chips. Such on-chip networks have routers at every node, connected to neighbors via short local on-chip links; multiple communication flows are multiplexed over these links to provide scalability and high bandwidth. This evolution of interconnection networks as core count increases is clearly illustrated in the choice of a flat crossbar interconnect connecting all eight cores in the Sun Niagara (2005) [199], four packet-switched rings in the 9-core IBM Cell (2005) [173], and five packet-switched meshes in the 64-core Tilera TILE64 (2007) [356].

Multi-core and many-core architectures are now commonplace in a variety of computing domains. These architectures will enable increased levels of server consolidation in data centers [25, 116, 241]. Desktop applications, particularly graphics are already leveraging the multi-core wave [227, 312]. High-bandwidth communication will be required for these throughput-oriented applications. Communication latency can have a significant impact on the performance of multi-threaded workloads; synchronization between threads will require low-overhead communication in order to scale to a large number of cores. In multiprocessor systems-on-chip (MPSoCs), leveraging an on-chip network can help enable design isolation: MPSoCs utilize heterogeneous IP blocks[1] from a variety of vendors; with standard interfaces, these blocks can communicate through an on-chip network in a plug-and-play fashion.

1.2 ON-CHIP VS. OFF-CHIP NETWORKS

While on-chip networks can leverage ideas from prior multi-chassis interconnection networks[2] used in supercomputers [12, 127, 132, 311], clusters of workstations [31], and Internet routers [84], the design requirements facing on-chip networks differ starkly in magnitude; hence, novel designs are critically needed. Fortunately, by moving on-chip, the I/O bottlenecks that faced prior multi-chassis interconnection networks are alleviated substantially: the abundant on-chip wiring supplies bandwidth that is orders of magnitude higher than off-chip I/Os while obviating the inherent delay overheads associated with off-chip I/O transmission.

On the other hand, a number of stringent technology constraints present challenges for on-chip network designs. Specifically, on-chip networks targeting high-performance multi-core processors must supply high bandwidth at ultra-low latencies, with a tight power envelope and area budget. With multi-core and many-core chips, caches and interconnects compete with the cores for the same chip real estate. Integrating a large number of components under tight area and power constraints poses a significant challenge for architects to create a balance between these components.

Innovations in on-chip networks have led to communication latency that is competitive with crossbars leading to widespread adoption. Furthermore, although on-chip networks require

[1] IP blocks are intellectual property in the form of soft macros of reusable logic.
[2] Also referred to as off-chip interconnection networks.

much less power than buses and crossbars, they need to be carefully designed as on-chip network power consumption can be high [43, 352]. For example, up to ~ 30% of chip power is consumed by Intel's 80-core TeraFLOPS network [167, 344] and 36% by the RAW on-chip network [335]. Therefore, it is essential that power constraints be considered. For example, innovations in the Intel Single-chip Cloud Computer [160] lead to power consumption representing only 10% of total chip power. Despite improvements, power continues to be a significant concern moving the field forward. This new edition features a more in-depth discussion of power in Chapter 6.

1.3 NETWORK BASICS: A QUICK PRIMER

In the next few sections, we lay a foundation for terminology and topics covered within this book. Subsequent chapters will explore many of these areas in more depth as well as state-of-the-art research for different components of on-chip network design. Many fundamental concepts are applicable to off-chip networks as well with different sets of design trade-offs and opportunities for innovation in each domain.

Several acronyms have emerged as on-chip network research has gained momentum. Some examples are NoC (network-on-chip), OCIN (on-chip interconnection network) and OCN (on-chip networks), with NoC emerging as the pervasive acronym.

NoC
OCIN
OCN

1.3.1 EVOLUTION TO ON-CHIP NETWORKS

An on-chip network, as a subset of a broader class of interconnection networks, can be viewed as a programmable system that facilitates the transporting of data between nodes.[3] An on-chip network can be viewed as a system because it integrates many components including channels, buffers, switches and control.

With a small number of nodes, dedicated ad hoc wiring can be used to interconnect them. However, the use of dedicated wires is problematic as we increase the number of components on-chip: the amount of wiring required to directly connect every component will become prohibitive.

Designs with low core counts can leverage buses and crossbars. In both traditional multiprocessor systems and newer multi-core architectures, bus-based systems scale to only a modest number of processors. This limited scalability is because bus traffic quickly reaches saturation as more cores are added to the bus, so it is hard to attain high bandwidth. The power required to drive a long bus with many cores tapping onto it is also exorbitant. In addition, a centralized arbiter adds arbitration latency as core counts increase. To address these problems, sophisticated bus designs incorporate segmentation, distributed arbitration, split transactions, and increasingly resemble switched on-chip networks.

Crossbars address the bandwidth problem of buses, and have been used for on-chip interconnects for a small number of nodes. However, crossbars scale poorly for a large number

[3] A node is any component that connects to the network, e.g., core, cache, memory controller, etc.

of cores; they require a large area footprint and consuming high power. For instance, the Sun Niagara 2's flat 8 × 9 crossbar interconnecting all cores and the memory controller has an area footprint close to that of a core. In response, hierarchical crossbars, where cores are clustered into nodes and several levels of smaller crossbars provide the interconnection, are used. For the 16 cores in Sun's Rock architecture, if the same flat crossbar architecture is used, it will require a 17 × 17 crossbar that will take up at least 8× more area than the final hierarchical crossbar design chosen: a 5 × 5 crossbar connecting clusters of four cores each [340]. These sophisticated crossbars resemble multi-hop on-chip networks where each hop comprises small crossbars.

On-chip networks are an attractive alternative to buses and crossbars for several reasons. First and foremost, networks represent a scalable solution to on-chip communication, due to their ability to supply scalable bandwidth at low area and power overheads that correlate sub-linearly with the number of nodes. Second, on-chip networks are very efficient in their use of wiring, multiplexing different communication flows on the same links allowing for high bandwidth. Finally, on-chip networks with regular topologies have local, short interconnects that are fixed in length and can be optimized and built modularly using regular repetitive structures, easing the burden of verification.

1.3.2 ON-CHIP NETWORK BUILDING BLOCKS

The design of an on-chip network can be broken down into its various building blocks: its topology, routing, flow control, router microarchitecture and design, and link architecture. The rest of this book is organized along these building blocks and we will briefly explain each in turn here.

Topology. An on-chip network is composed of channels and router nodes. The network topology determines the physical layout and connections between nodes and channels in the network.

Routing. For a given topology, the routing algorithm determines the path through the network that a message will take to reach its destination. A routing algorithm's ability to balance traffic (or load) has a direct impact on the throughput and performance of the network.

Flow control. Flow control determines how resources are allocated to messages as they travel through the network. The flow control mechanism is responsible for allocating (and de-allocating) buffers and channel bandwidth to waiting packets. In contrast to off-chip networks based on Ethernet technology, most on-chip networks are considered to be lossless by design.

Router microarchitecture. A generic router microarchitecture is comprised of the following components: input buffers, router state, routing logic, allocators, and a crossbar (or switch). Router functionality is often pipelined to improve throughput. Delay through each router in the on-chip network is the primary contributor to communication latency. As a result, significant research effort has been spent reducing router pipeline stages and improving throughput.

Link architecture. Most on-chip network prototypes use conventional full-swing logic and repeated wires. Full-swing wires transition from 0 V (ground) to the supply voltage when

transmitting a 1, and back to ground when transmitting a 0. Repeaters (inverters or buffers) at equal intervals on a long wire are an effective technique to reduce delay, enabling delay to scale linearly with number of repeaters instead of quadratically with length.

1.3.3 PERFORMANCE AND COST

As we discuss different on-chip design points and relevant research, it is important to consider both the performance and the cost of the network. Performance is generally measured in terms of network latency or accepted traffic. For back-of-the-envelope performance calculations, zero-load latency is often used, i.e., the latency experienced by a packet when there are no other packets in the network. Zero-load latency provides a lower bound on average message latency. Zero-load latency is found by taking the average distance (given in terms of network hops) a message will travel times the latency to traverse a single hop.

zero-load latency

In addition to providing ultra-low latency communication, networks must also deliver high throughput. Therefore, performance is also measured by its throughput. A high saturation throughput indicates that the network can accept a large amount of traffic before all packets experience very high latencies, sustaining higher bandwidth. Figure 1.1 presents a latency vs. throughput curve for an on-chip network illustrating the zero-load latency and saturation throughput.

saturation throughput

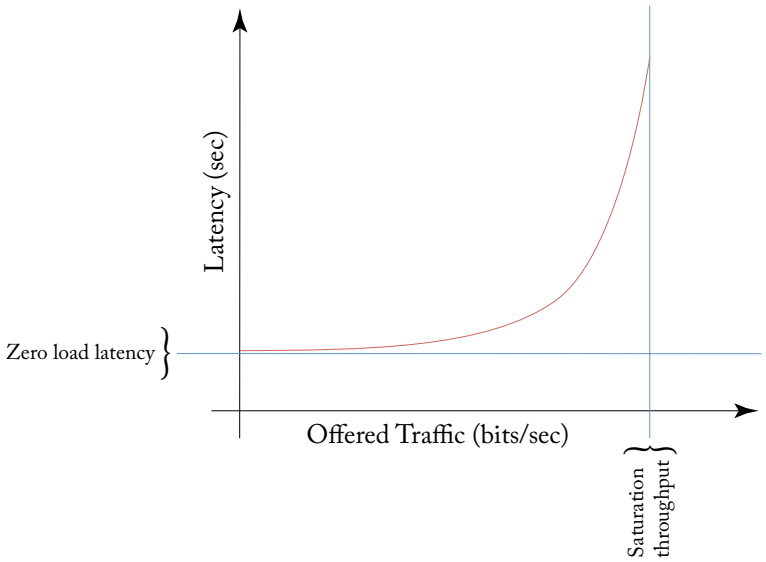


Figure 1.1: Latency vs. throughput for an on-chip network.

The two primary costs associated with an on-chip network are area and power. As mentioned, many-core architectures operate under very tight power budgets. The impact of differ-

ent designs on power and area will be discussed throughout this book, delved in more detail in Chapter 6 on Router Microarchitecture.

1.4 THIS BOOK—SECOND EDITION

This book targets engineers and researchers familiar with many basic computer architecture concepts who are interested in learning about on-chip networks. This work is designed to be a short synthesis of the most critical concepts in on-chip network design. We envision this book as a resource for both understanding on-chip network basics and for providing an overview of state-of-the-art research in on-chip networks. We believe that an overview that teaches both fundamental concepts and highlights state-of-the-art designs will be of great value to both graduate students and industry engineers. While not an exhaustive text, we hope to illuminate fundamental concepts for the reader as well as identify trends and gaps in on-chip network research.

With the rapid advances in this field, we felt it was timely to update and review the state-of-the-art in this second edition. We introduce two new chapters at the end of the book, as will be detailed below. Throughout the book, in addition to updating the latest research in the past years, we also expanded our coverage of fundamental concepts to include several research ideas that have now made their way into products and in our opinion, should be textbook concepts that all on-chip network practitioners should know. For example, these fundamental concepts include message passing, multicast routing and bubble flow control schemes.

The structure of this book is as follows. Chapter 2 explains how networks fit into the overall system architecture of multi-core designs. Specifically, we examine the set of requirements imposed by cache-coherence protocols in shared memory chip multiprocessors, and contrast that with the requirements in message-passing multi-cores. In addition to examining the system requirements, this chapter also describes the interface between the system and the network.

Once a context for the use of on-chip networks has been provided through a discussion of system architecture, the details of the network are explored. As topology is often a first choice in designing a network, Chapter 3 describes various topology trade-offs for cost and performance. Given a network topology, a routing algorithm must be implemented to determine the path(s) messages travel to be delivered throughout the network fabric; routing algorithms are explained in Chapter 4. Chapter 5 deals with the flow control mechanisms employed in the network; flow control specifies how network resources, namely buffers and links, are allocated to packets as they travel from source to destination. Topology, routing, and flow control all factor into the microarchitecture of the network routers. Details on various microarchitectural trade-offs and design issues are presented in Chapter 6. This chapter includes the design of buffers, switches and allocators that comprise the router microarchitecture. Although power consumption can be addressed through innovations in all areas of on-chip networks, we focus our new power discussion in the microarchitecture chapter as this is where many such optimizations are realized.

New Chapter 7 covers the nuts and bolts of modeling and evaluating on-chip networks, from software simulations to RTL design and emulation on FPGA, to architectural models

of delay, throughput, area, and power. The chapter also guides the reader on useful metrics for evaluating on-chip networks and ideal theoretical yardsticks for comparing against.

With the plethora of industry and academia on-chip network chips now available, we dedicate a new Chapter 8 to a survey of these. The chapter provides the reader with a sweeping understanding of how the various fundamental concepts presented in the earlier chapters come together, and the implications of the design and implementation of such concepts.

Finally in Chapter 9, we leave the reader with thoughts on key challenges and new areas of exploration that will drive on-chip network research in the years to come. Substantial new research has clearly surfaced, and here we focus on various significant trends that highlight the cross-cutting nature of on-chip network research. Emerging new interconnects and devices substantially change the implementation tradeoffs of on-chip networks, and in turn prompt new designs. Newly important metrics such as resilience, due to increasing variability in the fabrication process, or quality-of-service that is prompted by multiple workloads running simultaneously on many-cores, will add new dimensions and prompt new research ideas across the community.

CHAPTER 2

Interface with System Architecture

Over the course of the last 15 years, single-processor-core computer chips have given way to multi-core chips. These multi-core and many-core systems have become the primary building blocks of computer systems, marking a major shift in the way we design and engineer these systems.

Achieving future performance gains will rely on removing the communication bottleneck between the processors and the memory components that feed these bandwidth-hungry many-core designs. Increasingly, efficient communication between execution units or cores will become a key factor in improving the performance of many-core chips.

In this chapter, we explore three major types of computer systems where on-chip networks form a critical backbone: shared-memory chip multiprocessors (CMPs) in high end servers and embedded products, message passing systems and multiprocessor SoCs (MPSoCs) in the mobile consumer market. A brief overview of general architectures and their respective communication requirements is presented.

CMP: chip multiprocessor

2.1 SHARED MEMORY NETWORKS IN CHIP MULTIPROCESSORS

Parallel programming is extremely difficult but has become increasingly important [30]. With the emergence of many-core architectures, parallel hardware is now pervasive across a range of commodity systems. The growing prevalence of parallel systems requires an increasing number of parallel applications. Maintaining a globally shared address space alleviates some of the burden placed on the programmers to write high-performance parallel code. This is because it is easier to reason about a global address space than it is for a partitioned one. A partitioned global address space (PGAS) is common in modern SMP designs where the upper address bits choose which socket the memory address is associated with.

SMP: symmetric multiprocessor

In contrast, the message passing paradigm explicitly moves data between nodes and address spaces, so programmers have to explicitly manage communications. A hybrid approach that utilizes message passing (e.g., MPI) between different shared-memory nodes with a partitioned address space is common in massively parallel processing architectures. We focus the bulk of our discussion on shared-memory CMPs since they are widely expected to be the main-

stream multi-core architecture in the next few years. Like SMPs, CMPs typically have a shared global address space; however unlike SMPs, CMPs may exhibit non-uniform memory access latencies. Exceptions to the use of a shared-memory paradigm do exist. For example, the IBM Cell processor uses explicit messaging passing for DMA into local memory. The Intel SCC eschews on-chip cache coherence in favor of a message passing architecture. We will provide a brief overview of message passing systems in Section 2.2.

With the shared-memory model, communication occurs implicitly through the loading and storing of data and the accessing of instructions. As a result, the shared-memory model is an intuitive way to realize this sharing. Logically, all processors access the same shared memory, allowing each to see the most up-to-date data. Practically speaking, memory hierarchies use caches to improve the performance of shared memory systems. These cache hierarchies reduce the latency to access data but complicate the logical, unified view of memory held in the shared memory paradigm. As a result, cache coherence protocols are designed to maintain a coherent view of memory for all processors in the presence of multiple cached copies of data. Caches are designed to be transparent to the programmer. They improve performance by keeping frequently accessed data close to the processor but the programmer bears no responsibilities for managing them. This transparency is also desirable in multiprocessor systems; however, the presence of multiple caches can lead to correctness problems if different versions of the same address can reside in multiple locations at once. Cache coherence protocols are designed to solve this challenge without burdening the programmer. Cache coherence protocols maintain a single-writer, multiple-reader invariant. Cache coherence protocols manage access to shared data such that only one processor can write a cache line at one time. Multiple processors can simultaneously read a cache line without any problem. A full discussion of cache coherence and its many different flavors is outside the scope of this lecture. Interested readers are referred to Sorin et al. [325]. Therefore, it is the cache coherence protocol that governs what communication is necessary in a shared memory multiprocessor.

Figure 2.1 depicts a typical shared memory multiprocessor consisting of 64 nodes. A node contains a processor, private level 1 instruction and data caches and a second level cache that may be private or shared. Beyond the second level of cache, a third level may be incorporated on chip. This third level of cache is most commonly shared by all processors on the chip. The processor to network interface (discussed later in this chapter) and the router serve as the gateway between the local tile and other on-chip components.

Two key characteristics of a shared memory multiprocessor shape its demands on the interconnect: the cache coherence protocol that makes sure nodes receive the correct up-to-date copy of a cache line, and the cache hierarchy.

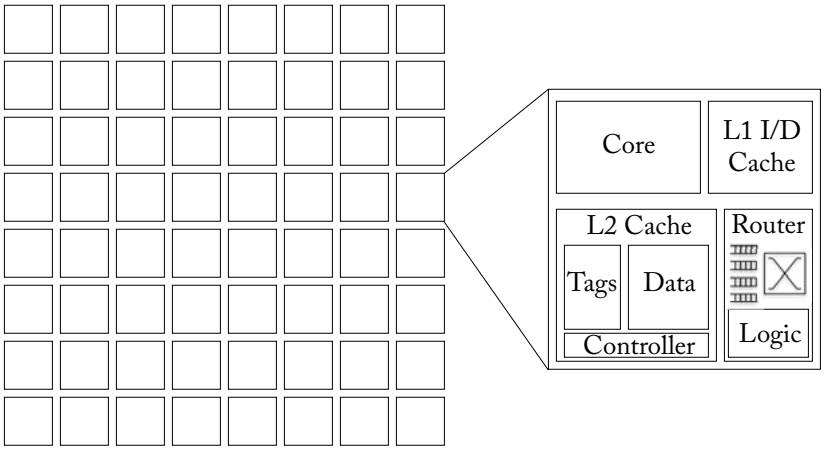


Figure 2.1: Shared memory chip multiprocessor architecture.

2.1.1 IMPACT OF COHERENCE PROTOCOL ON NETWORK PERFORMANCE

Cache coherence protocols typically enforce a single-writer, multiple-reader invariant. Any number of nodes may cache a copy of memory to read from; if a node wishes to write to that memory address, it must ensure that no other nodes are caching that address. The resulting communication requirements for a shared memory multiprocessor consist of data requests, data responses and coherence permissions. Coherence permission needs to be obtained before a node can read or write to a cache block. Depending on the cache coherence protocol, other nodes may be required to respond to a permission request.

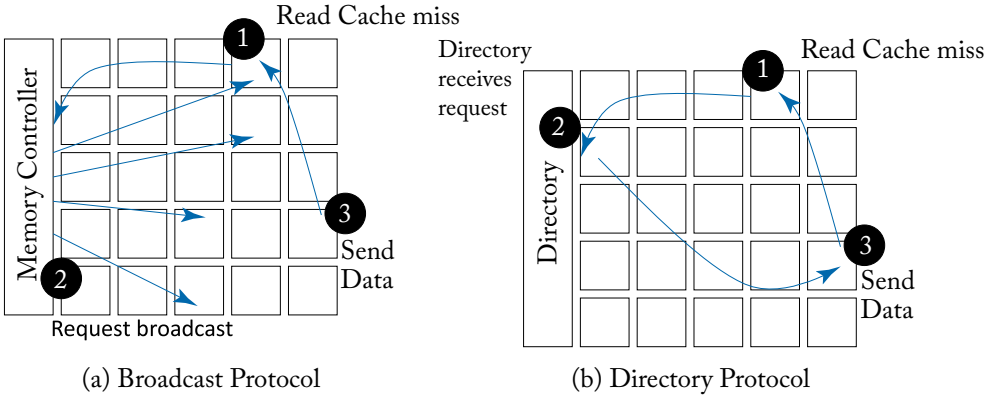
Multiprocessor systems generally rely on one of two different types of coherence protocols: broadcast or directory, as shown in Figure 2.2. Each type of protocol results in different network traffic characteristics. Here we focus on the basics of these coherence protocols, discussing how they impact network requirements. For a more in-depth discussion of coherence, we refer the readers to other texts [85, 152, 325].

broadcast protocol

With a broadcast protocol, coherence requests are sent to all nodes on chip resulting in high bandwidth requirements. Data responses are of a point-to-point nature and do not require any ordering; broadcast systems can rely on two physical networks: one interconnect for ordering and a higher bandwidth, unordered interconnect for data transfers. Alternatively, multiple virtual channels can be used to ensure ordering among coherence traffic; requests and responses can flow through independent virtual channels [166, 200]. Figure 2.2a shows a read request resulting in a cache miss that is (1) sent to an ordering point, (2) broadcast to all cores, and then (3) receives data.

directory protocol

An alternative to a broadcast protocol is a directory protocol. Directory protocols do not



(a) Broadcast Protocol (b) Directory Protocol

Figure 2.2: Coherence protocol network request examples.

rely on any implicit network ordering and can be mapped to an arbitrary topology. Directory protocols rely on point-to-point messages rather than broadcasts; this reduction in coherence messages allows this class of protocols to provide greater scalability. Rather than broadcast to all cores, the directory contains information about which cores have the cache block. A single core receives the read request from the directory in Figure 2.2b resulting in lower bandwidth requirements.

Directories maintain information about the current sharers of a cache line in the system and coherence state information. By maintaining a sharing list, directory protocols eliminate the need to broadcast invalidation requests to the entire system. Addresses are interleaved across directory nodes; each address is assigned a *home node*, which is responsible for ordering and handling all coherence requests to that address. Directory coherence state is maintained in memory; to make directories suitable for on-chip many-core architectures, directory caches are used. Going off-chip to memory for all coherence requests is impractical. By maintaining recently accessed directory information in on-chip directory caches, latency is reduced.

2.1.2 COHERENCE PROTOCOL REQUIREMENTS FOR THE ON-CHIP NETWORK

Cache coherence protocols require several types of messages: unicast, multicast and broadcast. Unicast (one-to-one) traffic is from a single source to a single destination (e.g., from a L2 cache to a memory controller). Multicast (one-to-many) traffic is from a single source to multiple destinations on chip (e.g., cache line invalidation messages from the directory home node to several sharers). Lastly, broadcast traffic (one-to-all) sends a message from a single source to all network destinations on chip.

With a directory protocol, the majority of requests will be unicast (or point-to-point). As a result, this places lower bandwidth demands on the network. Directory-based protocols are often chosen in scalable designs due to the point-to-point nature of communication; however, they are not immune to one-to-many or multicast communication. Directory protocols send out multiple invalidations from a single directory to nodes sharing a cache block.

Broadcast protocols place higher bandwidth demands on the interconnect as all coherence requests are of a one-to-all nature. Broadcast protocols may be required to collect acknowledgment messages from all nodes to ensure proper ordering of requests. Data response messages are point-to-point (unicast) and do not require ordering.

Cache coherent shared memory chip multiprocessors generally require two message sizes. The first message size is for coherence requests and responses without data. These messages consist of a memory address and a coherence command (request or response) and are small in size. In a cache coherence protocol, data is transferred in full cache line chunks. A data message consists of the entire cache block (typically 64 bytes) and the memory address. Both message types also contain additional network-specific data, which will be discussed in subsequent chapters.

2.1.3 PROTOCOL-LEVEL NETWORK DEADLOCK

protocol-level deadlock

In addition to the message types and sizes, shared memory systems require that the network be free from protocol-level deadlock. Figure 2.3 illustrates the potential for protocol level deadlock. If the network becomes flooded with requests that cannot be consumed until the network interface initiates a reply, a cyclic dependence can occur. In this example, if both processors generate a burst of requests that fill the network resources, both processors will be stalled waiting for remote replies before they can consume additional outstanding requests. If replies utilize the same network resources as requests, those replies cannot make forward progress resulting in deadlock.

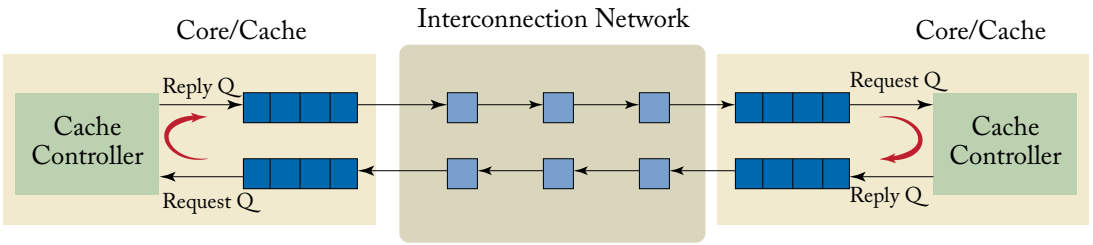


Figure 2.3: Protocol-level deadlock. Figure adapted from [294].

message classes

Protocols can require several different message classes. Each class contains a group of coherence actions that are independent of each other; that is, a request message in one class will not lead to the generation of another request message in the same class, but can trigger a message of a different class. Deadlock can occur when there are resource dependences between messages of different classes [322]. Here we describe three typical classes: requests, interventions, and

responses. Request messages include loads, stores, upgrades, and writebacks. Interventions are messages sent from the directory to request modified data be transferred to a new node. Examples of response messages include invalidation acknowledgments, negative acknowledgments (indicating a request has failed) and data messages.

Multiple virtual channels can be used to prevent protocol-level deadlock. The Alpha 21364 [254] allocates one virtual channel per message class to prevent protocol-level deadlock. By requiring different message classes to use different virtual channels, the cyclic dependence between requests and responses is broken in the network. Virtual channels and techniques to deal with protocol-level deadlock and network deadlock are discussed in Chapter 5.

2.1.4 IMPACT OF CACHE HIERARCHY IMPLEMENTATION ON NETWORK PERFORMANCE

Node design can have a significant impact on the bandwidth requirements for the on-chip network. In this section, we examine the impact of the cache hierarchy and look at how many different entities will share the injection/ejection port to the network. These entities can include multiple levels of cache, directory coherence caches, and memory controllers. Furthermore, how these entities are distributed throughout the chip can have a significant impact on overall network performance.

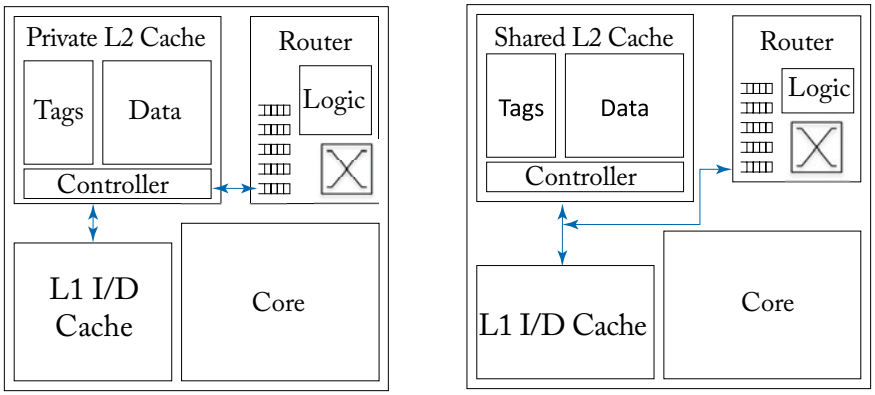
Caches are employed to reduce the memory latency of requests. They also serve as filters for the traffic that needs to be sent into the interconnect. For the purpose of this discussion, we assume a two-level cache hierarchy. Level 1 (L1) caches are split into instruction and data cache and the level 2 (L2) cache is the last level cache and is unified. The trade-offs discussed here could be extrapolated to cache hierarchies incorporating more levels of caching. Current chip multiprocessor research employs either private L2 caches, shared L2 caches, or a hybrid private/shared cache mechanism.

private L2

shared L2

Each of the tiles in Figure 2.1 can contain either a private L2 cache for that tile or a bank of shared cache. With a private L2 cache, an L1 miss is first sent to that processor's local private L2 cache; at the L2, the request could hit, be forwarded to a remote L2 cache that holds its directory, or access off-chip memory. Alternatively, with a shared L2 cache, an L1 miss will be sent to an L2 bank determined by the miss address (not necessarily the local L2 bank), where it could hit in the L2 bank or miss and be sent off-chip to access main memory. Private caches reduce the latency of L2 cache hits on chip and keep frequently accessed data close to the processor. A drawback to private caches is the replication of shared data in several caches on chip. This replication causes on-chip storage to be used less effectively. With each core having a small private L2 cache, interconnect traffic between caches will be reduced, as only L2 cache misses go to the network; however, interconnect traffic bound off chip is likely to increase (as data that do not fit in the private L2 cache will have to be evicted off chip). With a private L2 cache, the on-chip network will interface with just the L2 cache at each node as shown in

Figure 2.4a; the injection and ejection ports of the router are only connected to one component within the tile and do not need to be shared.



(a) Private L2 (b) Shared L2

Figure 2.4: Private and shared caches.

Figure 2.5 provides two walk-through examples of a many-core design configured with private L2 caches. In Figure 2.5a, the load of *A* misses in L1, but hits in the core's private L2 cache, and after step 3, the data are returned to the L1 and the core. However, in Figure 2.5b, the load of *A* misses in the private L2 cache and must be sent to the network interface (4), sent through the network to the memory controller (5), sent off chip, and finally re-traverse the network back to the requestor (6). After step 6, the data are installed in the L2 and forwarded to the L1 and the core. In this scenario, a miss to a private L2 cache requires two network traversals and an off-chip memory access.

Alternatively, the L2 cache can be shared amongst all or some of the cores. Shared caches represent a more effective use of storage as there is no replication of cache lines. However, L2 cache hits incur additional latency to request data from a different tile. Shared caches place more pressure on the interconnection network as L1 misses also go into the network, but more effective use of storage may reduce pressure on the off-chip bandwidth to memory. With shared caches, more requests will travel to remote nodes for data. As shown in Figure 2.4b, the on-chip network must attach to both the L1s and the L2 when the L2 is shared; both levels of cache share the injection and ejection bandwidth of the router.

Figure 2.6 provides two walk-through examples similar to those in Figure 2.5 but with a many-core system configured with a shared L2 cache. In Figure 2.6a, the L1 cache experiences a miss to address *A*. Address *A* maps to a remote bank of the shared L2, so the load request must be sent to the network interface (3) and traverse the network to the appropriate node. The read request arrives at the remote node (4) and is serviced by the L2 bank (5). The data are sent to the network interface (6) and re-traverse the network back to the requestor (7). After step 7,

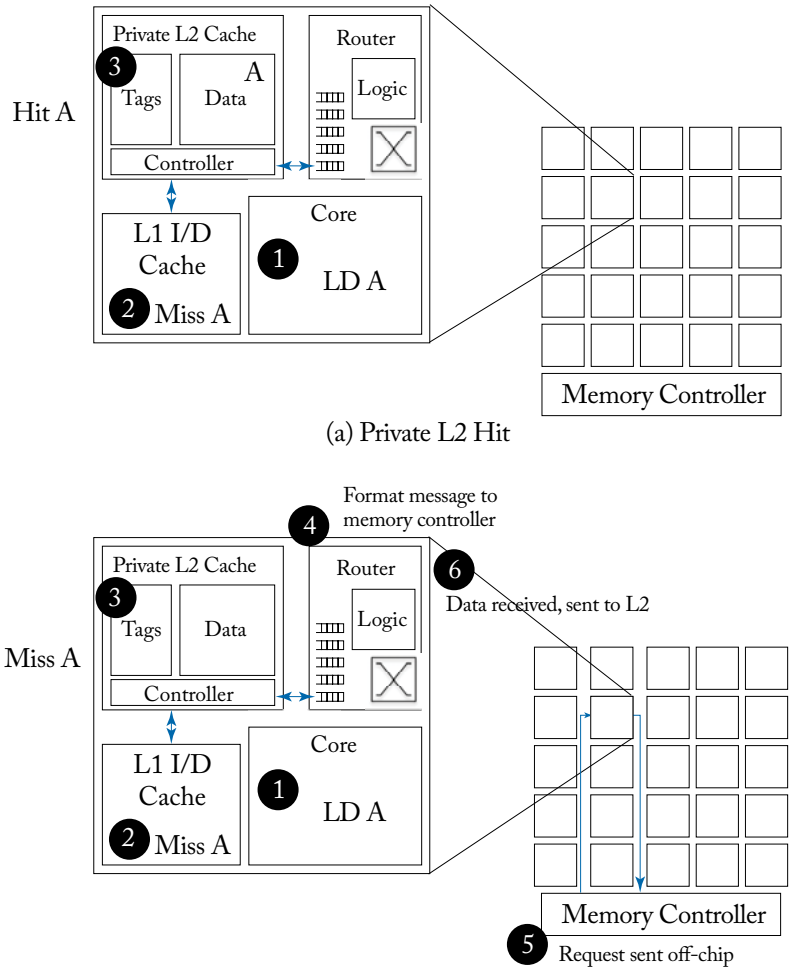


Figure 2.5: Private L2 caches walk-through example.

the data are installed in the local L1 and sent to the core. Here, an L2 hit requires two network traversals when the address maps to a remote cache (e.g., addresses can be mapped by a function $A \bmod N$, where N is the number of L2 banks).

In Figure 2.6b, we give a walk-through example for an L2 miss in a shared configuration. Initially, steps 1-4 are the same as the previous example. However, now the shared L2 bank misses to address A (5). The read request must again be sent to the network interface (6), forwarded through the network to the memory controller and sent off chip (7), returned through the network to the shared L2 bank and installed in the L2 (8) and then sent through the network