On-Chip Networks

On-Chip Networks

Natalie Enright Jerger and Li-Shiuan Peh

www.morganclaypool.com

ISBN: 9781598295849 paperback
ISBN: 9781598295856 ebook

DOI 10.2200/S00209ED1V01Y200907CAC008

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

Lecture #8
Series Editor: Mark Hill, *University of Wisconsin, Madison*

Series ISSN
Synthesis Lectures on Computer Architecture
Print 1935-3235 Electronic 1935-3243

On-Chip Networks

Natalie Enright Jerger
University of Toronto

Li-Shiuan Peh
Princeton University

SYNTHESIS LECTURES ON COMPUTER ARCHITECTURE #8

MORGAN & CLAYPOOL PUBLISHERS

ABSTRACT

With the ability to integrate a large number of cores on a single chip, research into on-chip networks to facilitate communication becomes increasingly important. On-chip networks seek to provide a scalable and high-bandwidth communication substrate for multi-core and many-core architectures. High bandwidth and low latency within the on-chip network must be achieved while fitting within tight area and power budgets. In this lecture, we examine various fundamental aspects of on-chip network design and provide the reader with an overview of the current state-of-the-art research in this field.

KEYWORDS

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

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

Chris

Chin-Kai, Kea-Roy, and Keari

Contents

Acknowledgments

First we would like to thank Mark Hill for the feedback and support he has provided from the inception of this project through the final product. Additionally, we would like to thank Michael Morgan for the opportunity to contribute to this lecture series. Many thanks to Dennis Abts and Timothy Pinkston for their detailed comments that were invaluable in improving this manuscript. Thanks to Konstantinos Aisopos, Emmanouil Koukoumidis, Abhishek Bhattarcharjee of Princeton University and Danyao Wang of the University of Toronto for proofreading our early drafts. James Balfour, Amit Kumar and Partha Kundu provided helpful insight on the microarchitectural implementation section.

Natalie Enright Jerger and Li-Shiuan Peh
June 2009

CHAPTER 1
Introduction

Since the introduction of research into multi-core chips more than a decade ago [23, 165, 203], on-chip networks have emerged as an important and growing field of research. As core counts increase, there is a corresponding increase in bandwidth demand to facilitate high core utilization and a critical need for scalable interconnection fabrics such as on-chip networks. On-chip networks will be prevalent in computing domains ranging from high-end servers to embedded system-on-chip (SoC) devices. 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, in both general-purpose computing chips as well as application-specific SoCs. 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. In addition to the integration of many general-purpose cores on a single chip, increasing transistor counts will lead to greater system integration for multiprocessor systems-on-chip (MPSoCs). MPSoCs will leverage a wide array of components, including processing cores, embedded memory and accelerators such as DSP modules and video processors.

MPSoC: Multiprocessor systems-on-chip

1.1.1 COMMUNICATION DEMANDS OF MULTI-CORE ARCHITECTURES

As the number of on-chip cores increases, a scalable and high-bandwidth communication fabric to connect them becomes critically important [167, 57, 59]. As a result, packet-switched 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 wiring, while multiplexing multiple communication flows over these interconnects 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) [120], four packet-switched rings in the 9-core IBM Cell (2005) [108], and five packet-switched meshes in the 64-core Tilera TILE64 (2007) [212].

Multi-core and many-core architectures will be commonplace in a variety of computing domains. These architectures will enable increased levels of server consolidation in data centers [68, 143, 13]. Desktop applications, particularly graphics can leverage the multi-core wave [190, 137]. 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 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 [72, 189, 4, 76], clusters of workstations [18] and Internet routers [44], 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. 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. 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 $8x$ more area than the final hierarchical crossbar design chosen: A 5x5 crossbar connecting clusters of four cores each [204].

In order for widespread adoption of on-chip networks, the communication latency of a network implementation must be competitive with crossbars. Furthermore, although on-chip networks require much less power than buses and crossbars, they need to be carefully designed as on-chip network power consumption can be high [211, 27]. For example, up to $\sim 30\%$ of chip power is consumed by Intel's 80-core TeraFLOPS network [103, 207] and 36% by the RAW on-chip network [202]. Therefore, it is essential that power constraints are evaluated as this field matures.

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

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 network). As there is no widely-agreed upon difference between them, throughout this book we avoid the use of acronyms and generically refer to all kinds of on-chip networks.

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, which are considered the simplest variants of on-chip networks. In both traditional multiprocessor systems and newer multi-core architectures, bus-based systems scale only to 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 of cores; requiring a large area footprint and consuming high power. In response, hierarchical crossbars, where cores are clustered into nodes and several levels of smaller crossbars provide the interconnection, are used. 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

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

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 lecture 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[4]. Resources can be allocated to packets in their entirety (done in store-and-forward and virtual cut-through flow control); however, this requires very large buffer resources making it impractical on chip. Most commonly, on-chip networks handle flow control at the flit level[5]. Buffers and channel bandwidth are allocated on the smaller granularity of flits rather than whole packets; as a result, routers can be designed with smaller buffers.

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. Thus far, all on-chip network prototypes have used conventional full-swing logic and pipelined wires. Pipelined wires uses repeaters to improve signal reach. While full-swing repeatered links dominate current designs, there are exciting ongoing research and opportunities for alternative link architectures. We discuss this research in Chapter 7.

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

[4]A packet is a subdivision of a message.
[5]A flit is a flow control unit, a subdivision of a packet.

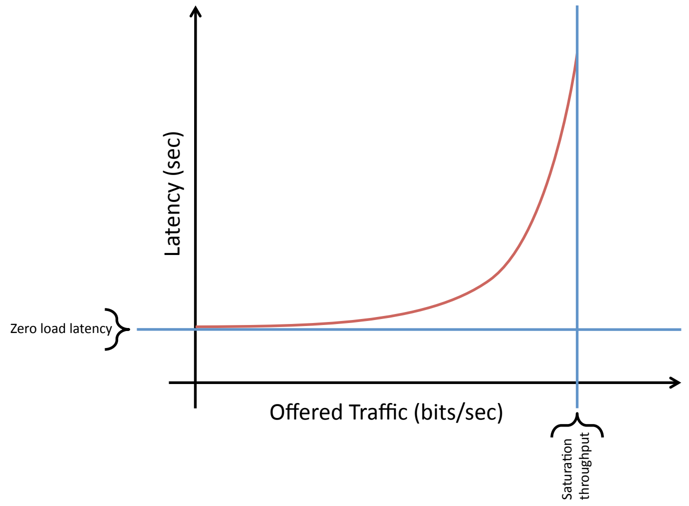


Figure 1.1: Latency vs Throughput for an on-chip network.

In addition to providing ultra-low latency communication, network's 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 versus throughput curve for an on-chip network illustrating the zero-load latency and saturation throughput. saturation throughput

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 different designs on power and area will be discussed throughout this book.

1.4 COMMERCIAL ON-CHIP NETWORK CHIPS

With on-chip networks being a nascent research area, there are only a handful of chips that have been designed incorporating sophisticated on-chip networks [202, 187, 94, 190, 199, 184, 130, 154, 93, 212, 208][6]. In this book, we select four commercial chips as case studies for illustrating the diversity in on-chip network chip designs. A brief summary of the chips is given here, followed by a discussion of their specific design choices in network interface design, topology, routing, flow control and router microarchitecture in subsequent chapters.

IBM Cell. The Cell architecture [94, 85] is a joint effort between IBM, Sony and Toshiba to design a power-efficient family of chips targeting game systems, but that are general enough for other domains as well. Cell is a product that is in most game consoles in the market today. It is a 90nm, 221mm^2 chip that can run at frequencies above 4GHz. It consists of one IBM 64-

[6]This is a non-exhaustive list of academic and industrial chips, to the best of the authors' knowledge.

bit Power Architecture core and eight Synergistic Processing Elements (SPEs) that are single-instruction-multiple-data (SIMD)-based. These nine nodes are interconnected with an on-chip Element Interconnect Bus (EIB) that overlays bus access semantics on four ring networks, with a maximum bisection bandwidth of over 300GBytes/s[7].

bisection bandwidth

Intel TeraFLOPS. The TeraFLOPS processor [207, 208, 95] is a research prototype chip that is targeted at exploring future processor designs with high core counts. It is a 65nm, $275mm^2$ chip with 80 tiles running at a targeted frequency of 5GHz. Each tile has a processing engine (PE) connected to an on-chip network router. The PE consists of two single-precision floating-point multiply-accumulator (FPMAC) units; 3KB of single-cycle instruction memory (IMEM), and 2KB of data memory (DMEM). The router interface block (RIB) interfaces the PE with the router, and any PE can send or receive instruction and data packets to or from any other PE. Two FPMACs in each PE providing 20 GigaFLOPS of aggregate performance, partnered with a maximum bisection bandwidth of 320 GBytes/s in the on-chip network enable the chip to realize a sustained performance of 10^{12} floating-point operations per second (1.0 TeraFLOPS) while dissipating less than 100W.

Tilera TILE64 and TILE64Pro. The TILE64 and TILE64Pro architectures [25, 7, 212] are products from Tilera targeted towards high-performance embedded applications such as networking and real-time video processing. These chips support a shared memory space across the 64 tiles (TILE64Pro has additional support for cache coherence termed Dynamic Distributed Cache), with each tile consisting of L1 and L2 caches and a 3-issue VLIW processor core connected to the four mesh networks. The TILE64 chip at 90nm, 750MHz, has 5MB of on-chip cache and on-chip networks that provide a maximum bisection bandwidth of 2Tb/s with each tile dissipating less than 300mW.

ST Microelectronics STNoC. The STNoC [43, 42] is a prototype architecture and methodology targeted to replace the widely-used STBus in MPSoCs. It is targeted towards the unique demands of MPSoCs on the on-chip network fabric: automated synthesis of network design, compatibility with heterogeneous, diverse IP blocks from other vendors and prior knowledge of traffic demands. It has been applied to the Morpheus chip targeting 90nm, with eight nodes on the on-chip network connecting an ARM9 core, an array of ALUs, a reconfigurable array, embedded FPGA, and memory controllers [124].

1.5 THIS BOOK

The 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 as of Summer 2009. 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

[7]Bisection bandwidth is the bandwidth across the cut down the middle of the network.

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.

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 the requirements of standardized MPSoC on-chip protocols and their effects on the network design trade-offs. 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. Finally in Chapter 7, 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.

CHAPTER 2

Interface with System Architecture

Over the course of this decade, 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 two 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, 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 [17]. With the emergence of many-core architectures, parallel hardware will be present in commodity systems. The growing prevalence of parallel systems requires an increasing number of parallel applications. Maintaining a 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 mainstream multi-core architecture in the next few years. 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.

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

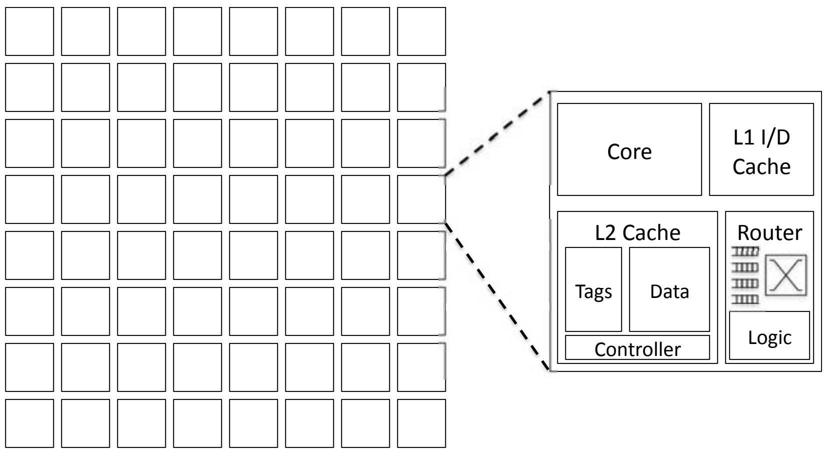


Figure 2.1: Shared Memory Chip Multiprocessor Architecture.

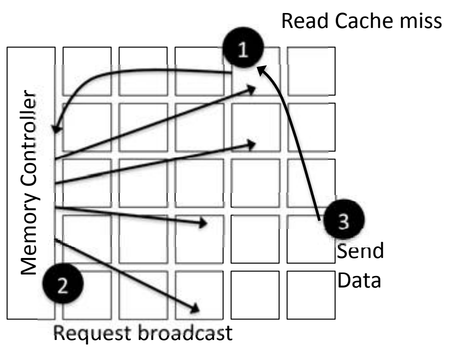
way to realize this sharing. Logically, all processors access the same shared memory, allowing each to see the most up-to-date data. Practically speaking, memory hierarchies use caches to improve the performance of shared memory systems. These cache hierarchies reduce the latency to access data but complicate 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. 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. 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.

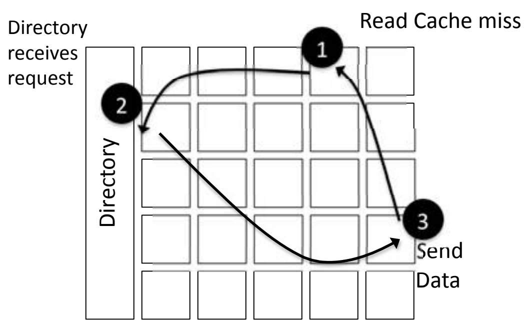
Two key characteristics of a shared memory multiprocessor shape its demands on the interconnect; the cache coherence protocol that makes sure nodes receive the correct up-to-date copy of a cache line, and the cache hierarchy.

2.1.1 IMPACT OF COHERENCE PROTOCOL ON NETWORK PERFORMANCE

Many cache coherence protocols 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



(a) Broadcast Protocol



(b) Directory Protocol

Figure 2.2: Coherence protocol network request examples.

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, each of which 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 [45, 88].

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 [121, 102]. A read request resulting in a cache miss that is sent to an ordering point (1), broadcast to all cores (2) then receives data (3) is shown in Figure 2.2a.

directory protocol

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

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

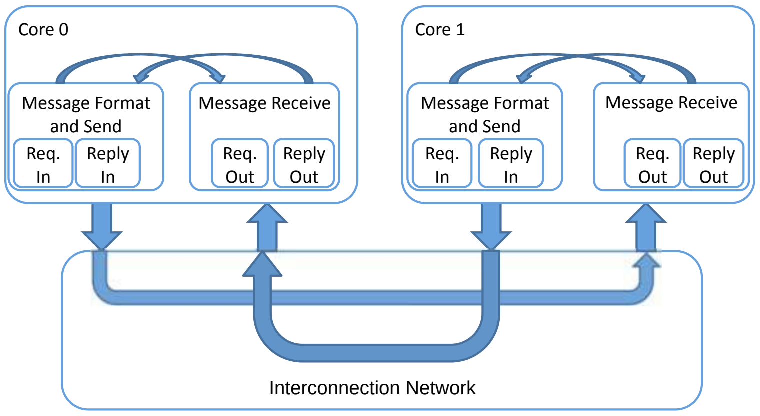


Figure 2.3: Protocol-level deadlock.

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 [195]. 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 will fail) and data messages are also present.

Multiple virtual channels can be used to prevent protocol-level deadlock. Virtual channels and techniques to deal with protocol-level deadlock and network deadlock will be discussed in Chapter 5. The Alpha 21364 [151] 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.

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 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 placed on 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.

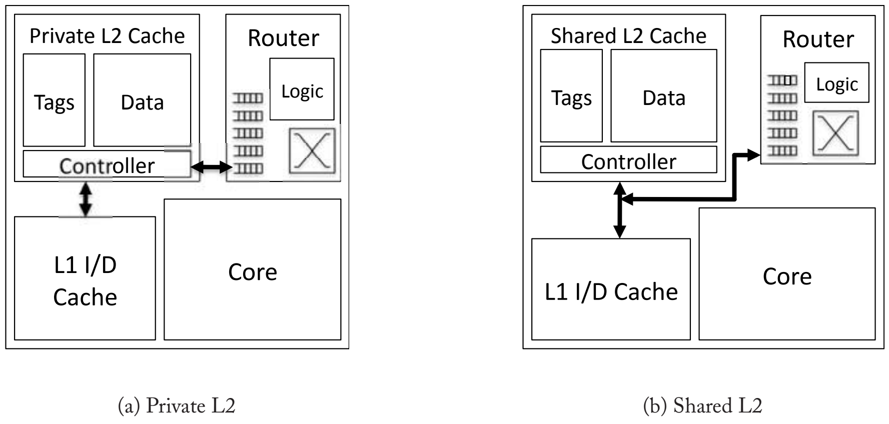


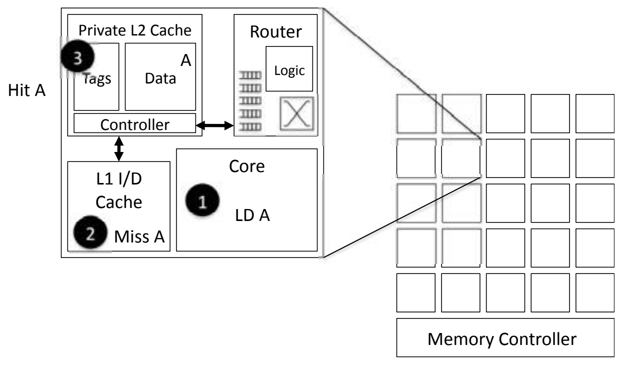
Figure 2.4: Private and Shared Caches.

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.

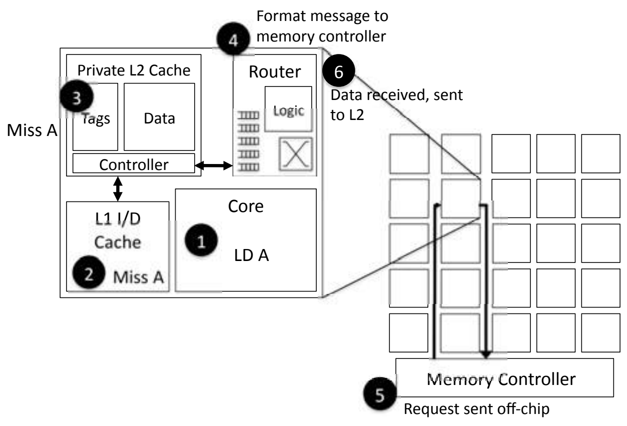
private L2

shared L2

Figure 2.5 provides two walk-through examples of a many-core 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.



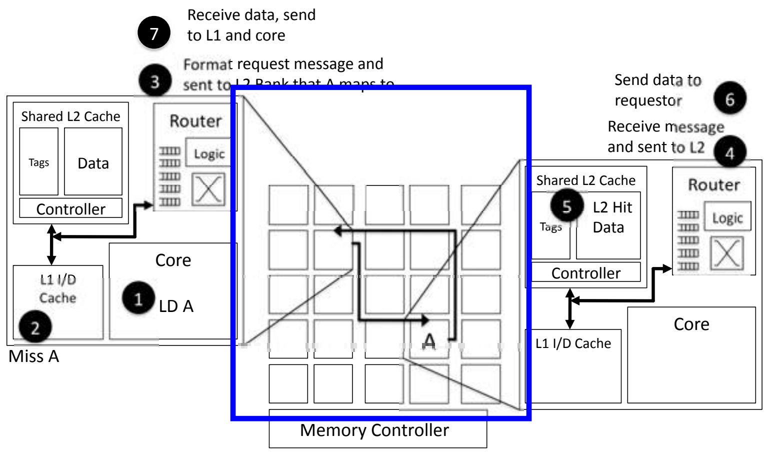
(a) Private L2 Hit



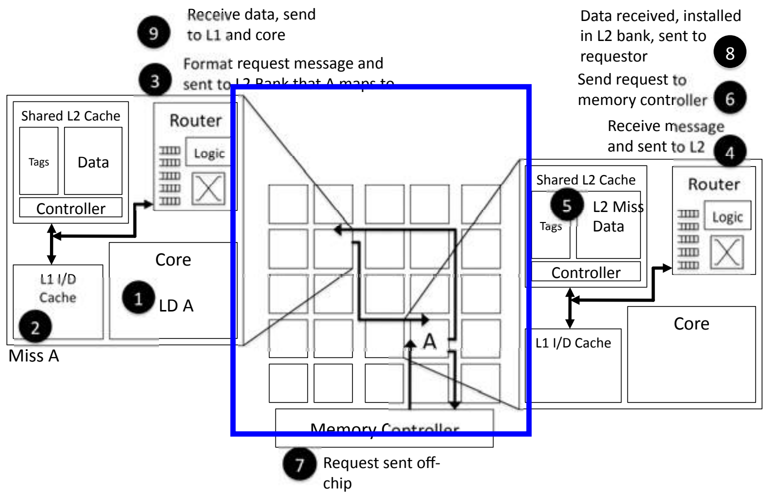
(b) Private L2 Miss

Figure 2.5: Private L2 caches walk-through example.

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 onto the network, but through 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.



(a) Shared L2 Hit


9 Receive data, send to L1 and core
3 Format request message and sent to L2 Bank that A maps to
Data received, installed in L2 bank, sent to requestor 8
Send request to memory controller 6
Receive message and sent to L2 4
Shared L2 Cache
Tags
Data
Controller
Router
Logic
L1 I/D Cache
Core
1 LD A
2
Miss A
A
5 L2 Miss Data
Memory Controller
7 Request sent off-chip

(b) Shared L2 Miss

Figure 2.6: Shared L2 cache walk-through example.

Figure 2.6 provides two similar walk-through examples to those in Figure 2.5 but with a many-core 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, 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 (addresses can be mapped by a function $A \bmod N$, where N is the number of L2 banks).

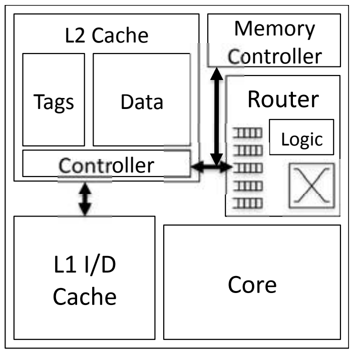
In Figure 2.6b, we give a walk-through example for an L2 miss in a shared configuration. Initially, steps 1-4 are the same as the previous example. However, now the shared L2 bank misses to address A (5). The read request must be again sent to the network interface (6), forwarded through the network 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 back to the requestor (9). Once the data have been received (9), it can be installed in the private L1 and sent to the core. This shared miss scenario requires 4 network traversals to satisfy the read request.

2.1.5 HOME NODE AND MEMORY CONTROLLER DESIGN ISSUES

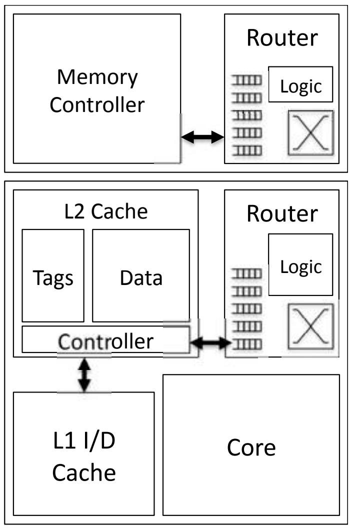
directory
home node

With a directory protocol, each address statically maps to a *home node*. The directory information resides at the home node which is responsible for ordering requests to all addresses that map to this *home node*. The directory either supplies the data from off-chip, either from memory or from another socket, or sends intervention messages on-chip to acquire data and/or permissions for the coherence request. For a shared L2 cache, the *home node* with directory information is the cache bank that the address maps to. From the example in Figure 2.6, the directory is located at the tile marked A for address A. If remote L1 cache copies need to be invalidated (for a write request to A), the directory will send those requests through the network. With a private L2 cache configuration, there does not need to be a one-to-one correspondence between the number of *home nodes* and the number of tiles. Every tile can house a portion of the directory (n *home nodes*), there can be a single centralized directory, or there can be a number of *home nodes* in between 1 and n. Broadcast protocols such as the Opteron protocol [41] require an ordering point similar to a home node from which to initiate the broadcast request.

In a two-level cache hierarchy, L2 cache misses must be satisfied by main memory. These requests travel through the on-chip network to the memory controllers. Memory-intensive workloads can place heavy demands on the memory controllers making memory controllers hot-spots for network traffic. Memory controllers can be co-located with processor and cache tiles. With such an arrangement, the memory controllers will share a network injection/ejection port with the cache(s), as depicted in Figure 2.7a. Policies to arbitrate between the memory controller and the local cache for injection bandwidth are needed in this design. Alternatively, memory controllers can be placed as individual nodes on the interconnection network; with this design, memory controllers do not have



(a) Memory Controller co-located with core and caches



(b) Memory Controller connected to own router

Figure 2.7: Memory Controllers.

to share injection/ejection bandwidth to/from the network with cache traffic (shown in Figure 2.7b. Traffic is more isolated in this scenario; the memory controller has access to the full amount of injection bandwidth.

2.1.6 MISS AND TRANSACTION STATUS HOLDING REGISTERS

A processor-to-network interface is responsible for formatting network messages to handle cache misses (due to a load or store), cache line permission upgrades and cache line evictions. Figure 2.8 depicts a possible organization for the processor to network interface. When a cache miss occurs, a miss status handling register (MSHR) is allocated and initialized. For example, on a read request, the MSHR will be initialized to a read pending state and the message format block will create a network message. The message will be formatted to contain the destination address (in the case of a directory protocol, this will be the location of the home node as determined by the memory address), the address of the cache line requested and the message request type (e.g. Read). Below the message format and send block, we show several possible message formats that may be generated depending on the type of request. When a reply message comes from the network, the MSHR will match the reply to one of the outstanding requests and complete the cache miss actions. The message receive block is also responsible for receiving request messages from the directory or another processor tile

miss status handling register (MSHR)

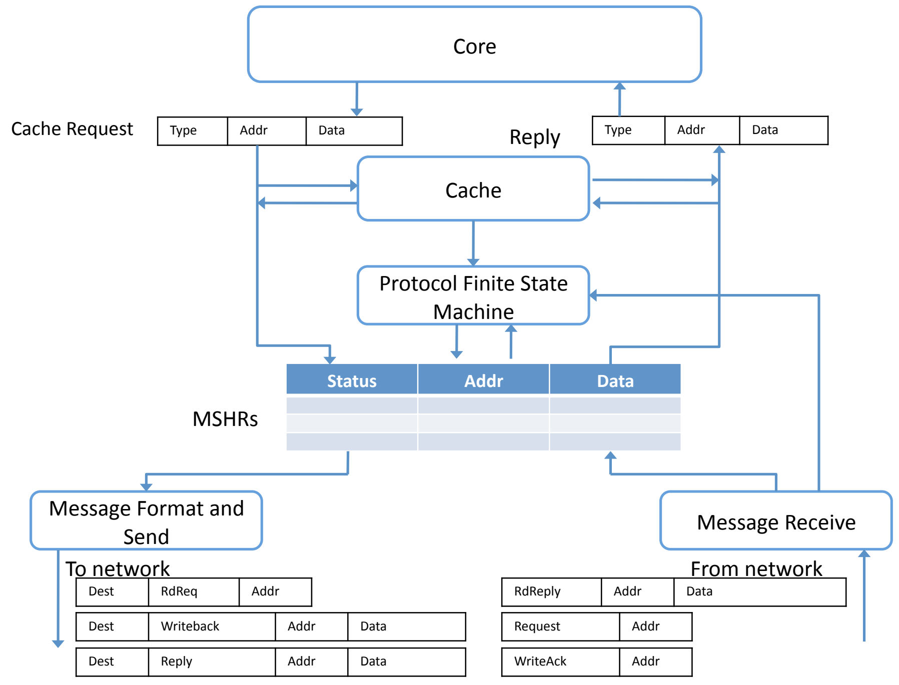


Figure 2.8: Processor-to-Network Interface (adapted from Dally and Towles [47]).

to initiate cache-to-cache transfers; the protocol finite state machine will take proper actions and format a reply message to send back into the network. Messages received from the network may also have several different formats that must be properly handled by the message receive block and the protocol finite state machine.

The memory-to-network interface (shown in Figure 2.9) is responsible for receiving memory request messages from processors (caches) and initiating replies. Different types and sizes of messages will be received from the network and sent back into the network as shown about the message format and send block and the message receive block. At the memory side, transaction status handling registers (TSHRs) handle outstanding memory requests. If memory controllers are guaranteed to service requests in order, the TSHRs could be simplified to a FIFO queue. However, as memory controllers often reorder memory requests to improve utilization, a more complicated interface is required. Once a memory request has been completed, the message format and send block is responsible for formatting a message to be injected into the network and sent back to the original requester. A network interface employing MSHRs and TSHRs is similar to the design utilized by the SGI Origin [129].

transaction status handling register (TSHR)

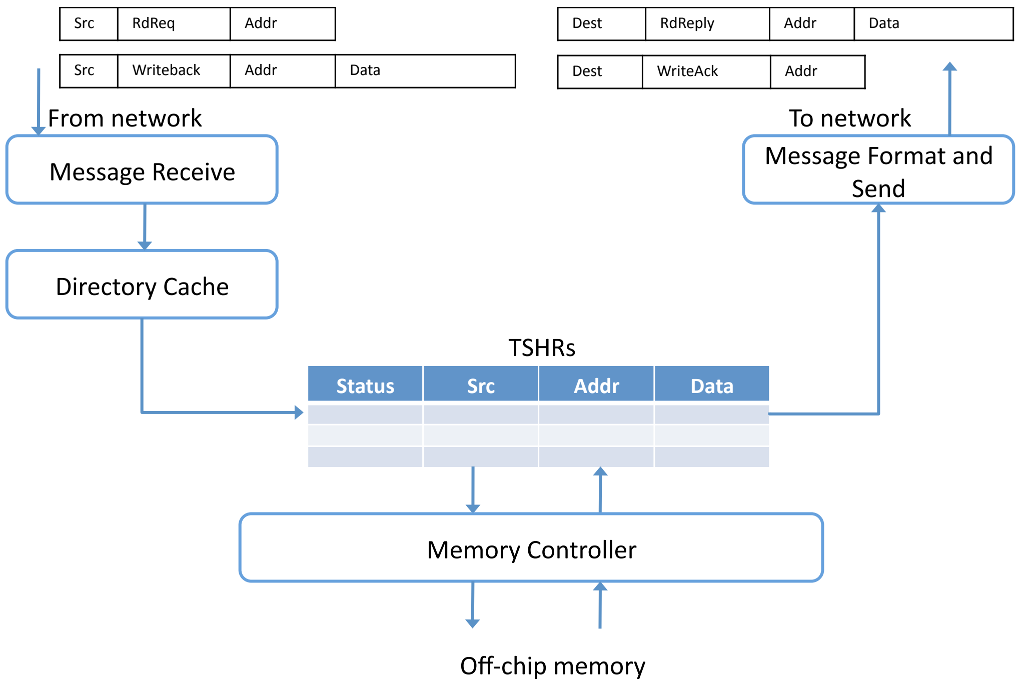


Figure 2.9: Memory-to-Network Interface (adapted from Dally and Towles [47]).

2.2 SYNTHESIZED NOCS IN MPSOCS

IP blocks

Systems-on-a-chip (SoCs) typically refer to chips that are tailored to a specific application or domain area, which are designed quickly through the composition of IP blocks (processor cores, memories, memory controllers, I/O interfaces, fixed-function units, etc.) purchased from a variety of vendors. Examples of SoCs include cell phones, mobile multimedia players, and home entertainment devices. The on-chip interconnect integrating all the various IP blocks on a SoC is usually referred to as a Network-on-a-chip (NoC), though NoC has also been used as an acronym widely to encompass all on-chip networks.

While the fundamental concepts of topology, routing, flow control and router microarchitecture are applicable to on-chip networks in both CMPs and MPSoCs, MPSoCs possess key distinguishing characteristics that critically impact NoC design.

These characteristics are as follows:

- Since the SoC is targeted towards a specific domain, the intended applications are known a priori. Therefore, the on-chip network design can be tailored for the specific applications at design time.

- To ensure fast time-to-market and low costs, NoC design is automated through synthesis flows. The emphasis is on using IP blocks rather than custom-designed hardware.
- There is a standard interface that the NoC has to adhere to, in order to ensure interoperability and composability with IP blocks from other vendors.
- Area and power constraints are typically orders of magnitude tighter; thankfully, delay targets and bandwidth needs are also significantly lower than that in CMPs, as targeted chips tend to be in the mobile consumer market.

We will discuss each of the above attributes next, and highlight in subsequent chapters when these attributes lead to different design decisions in the various dimensions of on-chip network design.

2.2.1 THE ROLE OF APPLICATION CHARACTERIZATION IN NOC DESIGN

While an understanding of application traffic behavior and communications requirements is a necessary first step in the design of any on-chip network, it has several unique characteristics in MPSoCs. First, the system integrator tends to have very clear knowledge of the set of applications that will run atop the MPSoC. This task is often eased with the knowledge that no other applications need to be supported. For instance, a set-top box for Digital TV uses a closed embedded chip where new applications will not be run after deployment. Second, the time to market is especially tight, with a design cycle on the order of months rather than years, which is common in the CMP market. This leads to a MPSoC needing quick integration with IP blocks obtained from third party vendors. The holy grail in NoC design in MPSoCs is to allow designers to feed in NoC traffic characterization to a design tool, which will then automatically explore the vast design space, choose a specific NoC design, then generate a fully synthesizable NoC that can be fed into regular back-end CAD design tools that will synthesize, generate and place-and-route a netlist for final fabrication.

Given that NoC synthesis is still very much in the research stage, there exists a variety of approaches that have been proposed for inputting NoC traffic information. Task or core graphs such as that of a video decoder (VOPD) in Figure 2.10 [104, 30], are most commonly used in research, where each core or IP block that needs to communicate will be depicted with a circle, and the peak or average communication bandwidth between cores is characterized and marked on links between cores. For example, the code running on the ARM core sends data at an average bandwidth of 16Gbps to the *idct* IP block and the *padding* IP block. Spreadsheets with latency and average bandwidth requirements between cores have also been proposed as an input interface feeding into subsequent synthesis tool chains [78]. Here, we are assuming that the applications have already been mapped onto specific cores or IP blocks prior to the start of NoC synthesis process. task/core graph

Traffic demands can be derived through higher-level simulations of software atop the hardware platform, such as the MPARM simulator [26] that allows designers to plug in SystemC hardware models of diverse processor cores (e.g. ARM, PowerPC, MIPS cores), private and shared memory modules, hardware interrupt devices, and an interconnection fabric that conforms to the

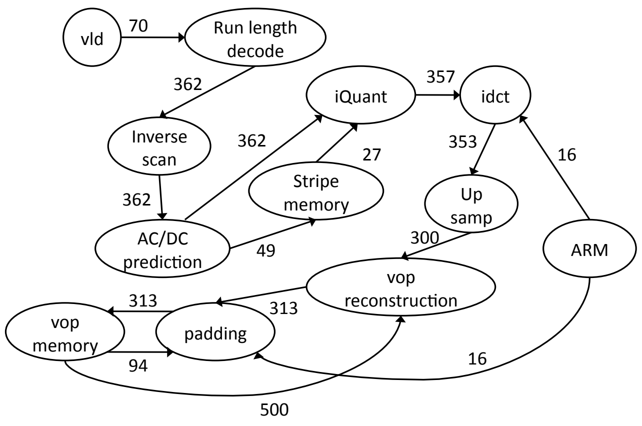


Figure 2.10: VOPD task graph example from [104, 30].

AMBA, STBus or XPipes [104] interface standards (see Section 2.2.4), and simulate to profile and characterize traffic.

2.2.2 DESIGN REQUIREMENTS FOR ON-CHIP NETWORK

The delay, area and power constraints of MPSoCs on NoCs are fairly different from those of CMPs on the on-chip networks, largely because of the different market segment and product characteristics. With MPSoCs typically targeting the low-end mobile embedded market, clock speeds do not tend to be as aggressive as those of CMPs, typically on the order of MHz rather than GHz. This means that pipelining of NoC routers may not be necessary (see Chapter 6 for details on router pipelining). Moreover, with standardized interfaces adding significant delay, NoC routers are usually less latency-sensitive than CMP routers. The area and power budgets for MPSoCs are also at least an order of magnitude lower than those of CMPs, with CMP server chips typically targeting >100W power envelopes while MPSoC chips tend to be just several watts.

The tight time to market, however, makes automatic composition and generation of NoCs critical requirements. This is made very challenging by the heterogeneous nature of cores which typically requires irregular topologies and tailored microarchitectures.

2.2.3 NOC SYNTHESIS

The most distinguishing requirement of MPSoCs vs. CMPs is the need for a NoC synthesis tool that can automatically generate a synthesizable NoC. Such a tool takes as input the locations of IP blocks on chip and their communication demands (captured in say, a core graph), explores the design space of alternative NoCs, selects a specific design, then generates and instantiates the netlist in synthesizable RTL or SystemC, so that it can be seamlessly fed into back-end CAD tools such as Cadence's Encounter [101].

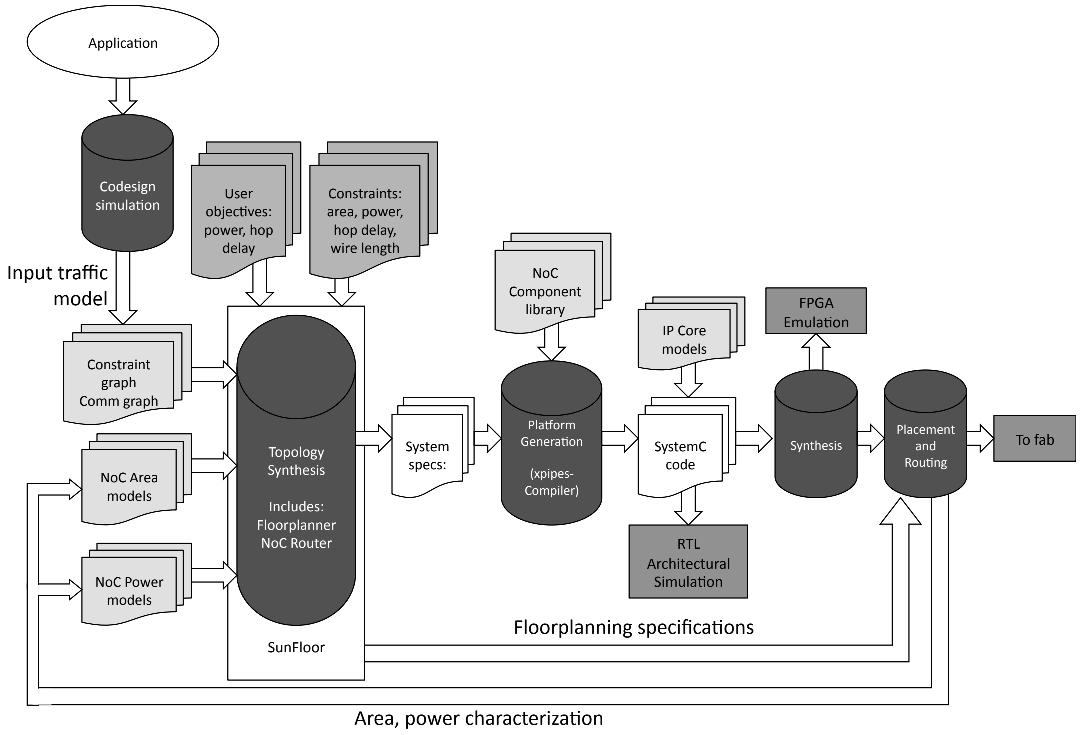


Figure 2.11: Synthesis Flow from [183].

Figure 2.11 illustrates a synthesis flow [183]. This synthesis requires accurate area and power models describing various components of the NoC design; these can be refined using a feedback loop as the tool iteratively modifies the design. A library of soft macros is used to produce the SystemC code for the NoC design. Finally, in the back-end flow, a netlist is created and synthesized for fabrication.

Figure 2.12 shows the NoC design that is generated by the SUNMAP [157] synthesis toolchain when fed with the VOPD task graph shown earlier. An irregular topology with routers of different sizes and links that were tuned to the specific bandwidth were arrived at through an automatic synthesis algorithm that optimized for power. The floorplanning of the NoC took into account the area footprint of the IP blocks as well as the instantiated NoC routers. More details about the synthesis algorithm are discussed in Section 3.4.

Such a tool requires several key components. First, it needs a library of soft macros covering all NoC building blocks needed for the supported design space. For instance, in SUNMAP, these are parameterized hardware models in SystemC of the various router microarchitecture components

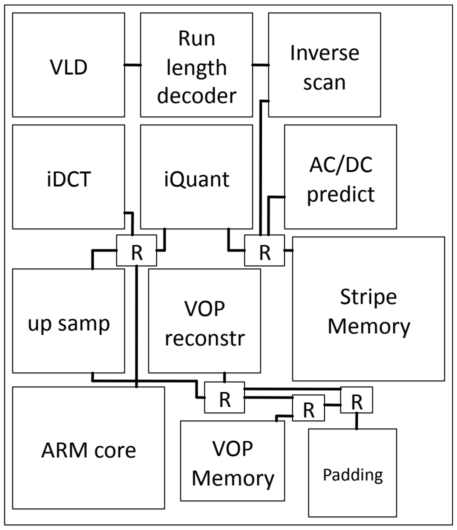


Figure 2.12: A custom design for a video object plane decoder (VOPD) (from[30]).

(crossbars, buffers, arbiters and allocators) and links. These soft macros can then be automatically composed and instantiated by the NoC synthesis tool.

Second, the tool needs delay, area and power models that can provide rapid estimates for comparing alternative designs and selecting a suitable NoC design. Here, a variety of models have been used, from very abstract metrics such as hop count, to architectural power models that take as input architectural parameters such as the topology, number of buffers, to more rigorous RTL-based power estimates using back-end synthesis pre-place and route tools. SUNMAP uses hop count as a proxy for delay and RTL-based power estimates based on linear regression applied to a huge number of router implementations of different microarchitectural parameters.

Third, a floorplanner tool specific to NoCs will be needed, to determine locations of each router and the lengths of the links, as well as layout of various components within a router. Here, the tool can either take IP block placement as fixed and propose a relative NoC floorplan, or make multiple iterative passes co-optimizing the core and NoC floorplan. Area estimates of individual routers aid in this process. The SUNMAP tool uses a floorplanning algorithm [6] that minimizes both area and wire lengths.

2.2.4 NOC NETWORK INTERFACE STANDARDS

NoCs have to adhere to standardized protocols so that they can plug-and-play with IP blocks that were also designed to interface with the same standard. Such standardized protocols define the rules for all signaling between the IP blocks and the communication fabric, while permitting configuration of specific instances. Several widely used standards for on-chip communications in SoCs today are

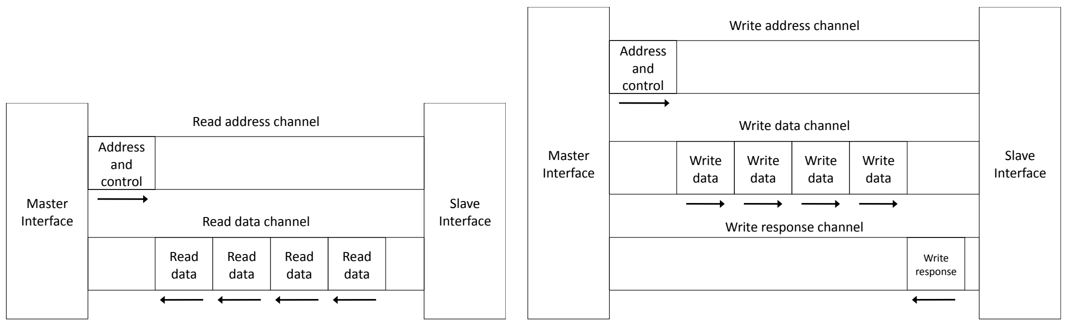
ARM's AMBA [16], ST Microelectronics' STBus [147], and Sonics' OCP [196]. Here, we will discuss features that are common across these standards, using ARM's AMBA AXI as the specific protocol for illustration.

Bus-based transaction semantics. First, as current SoCs predominantly use buses as the on-chip interconnects, these standard interfaces have bus-based semantics where nodes connected to the interconnect are defined as masters or slaves, and communicate via transactions. Masters start a transaction by issuing requests; slaves then receive and subsequently process the request. The transaction is completed when the slave responds to the original request. This request-response transaction model matches those used in buses, making it easier to design network interface wrappers around IP blocks that were originally designed to interface with buses. For instance, a processor core will be a master that initiates a new transaction through issuing a write request to a memory module, while the memory module will be the slave that executes the write request and responds with an acknowledgment response.

Every transaction in the AMBA AXI protocol sends address and control information on the address channel, while data are sent on the data channel in bursts. Writes have an additional response channel. These channels are illustrated for AXI reads and writes in Figure 2.13. The sizes of these channels can range from 8 to 1024 bits, with a particular size instantiated for each design. So an NoC that interfaces using the AXI protocol has to have these three channels. A write from the master node will lead to its network interface encapsulating and translating the address in the address channel to the slave's node destination address in the message header, and the data in the write data channel encoded as the body of the message. The message will then be broken down into packets and injected into the injection port of the attached router. At the destination, the packets will be assembled into a message, and the address and control information extracted from the header and fed into the AXI write address channel, while the data are obtained from the body and fed into the AXI write data channel. Upon receipt of the last flit of the message, the network interface will then compose a write response message and send it back to the master node, feeding into the AXI write response channel at the master.

Out-of-order transactions. Many of the latest versions of these standards relax the strict ordering of bus-based semantics so point-to-point interconnect fabrics such as crossbars and NoCs can be plugged in, while retaining backwards compatibility to buses, such as OCP 3.0 [180], AXI [14], STNoC [146].

For instance, AXI relaxes the ordering between requests and responses, so responses need not return in the same order as that of requests. Figure 2.14 illustrates this feature of AXI which allows multiple requests to be outstanding and slaves to be operating at different speeds. This allows multiple address and data buses to be used, as well as split-transaction buses (where a transaction does not hold on to the bus throughout, but instead, requests and responses of the same transaction separately arbitrate for the bus), and ultimately, NoCs. In NoCs, packets sent between different pairs of nodes can arrive in different order from the sending order, depending on the distance between



(a) Read (b) Write

Figure 2.13: AXI Read and Write Channels [16].

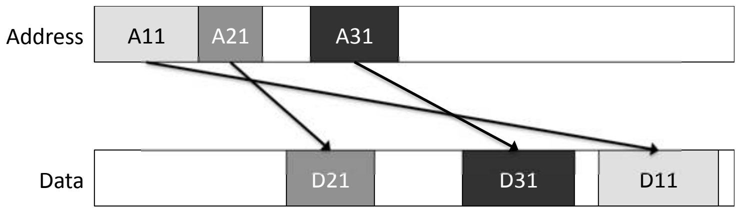


Figure 2.14: The AXI Protocol allows messages to complete out of order: D21 returns data prior to D11 even though A11 occurred prior to A21.

the nodes and the actual congestion level. A global ordering between all nodes is difficult to enforce. So out-of-order communication standards are necessary for NoC deployment.

Here, we use Arm's AXI as an illustrating example, hoping to provide readers with brief background material on the different design issues of NoCs in MPSoCs. We refer interested readers to Pasricha and Dutt [172] that goes through all existing on-chip bus-based protocols in detail.

2.3 BIBLIOGRAPHIC NOTES

We begin each bibliographic notes section with a look at our four case studies followed by state-of-the-art research proposals for on-chip networks and seminal work in off-chip networks.

2.3.1 CASE STUDIES

IBM Cell. The EIB of the IBM Cell interfaces with 12 elements on-chip: the PPE, the eight SPEs, the memory controller, and two I/O interfaces in and out of the chip. The IBM Cell uses explicit message passing as opposed to a shared-memory paradigm. It is designed to preserve DMA over split-transaction bus semantics, so snoopy coherent transfers can be supported atop the four unidirectional rings. In addition to the four rings, these 12 elements interface to an address-and-command bus that handles bus requests and coherence requests. The rings are accessed in a bus-like manner, with a sending phase where the source element initiates a transaction (e.g. issues a DMA), a command phase through the address-and-command bus where the destination element is informed about this impending transaction, then the data phase where access to rings is arbitrated and if access is granted, data are actually sent from source to destination. Finally, the receiving phase moves data from the NIC (called the Bus Interface Controller (BIC)) to the actual local or main memory or I/O.

Intel TeraFLOPS. The TeraFLOPS network supports message passing, through send and receive instructions in the ISA. Any tile can send/receive to any other tile, and send/receive instructions have latencies of 2 cycles (within a tile's pipeline), and 5 cycles of router pipeline along with a cycle of link propagation for each hop. The 2-cycle latency of send/receive instructions is the same as that of local load/store instructions. In addition, there are sleep/wakeup instructions to allow software to put entire router ports to sleep for power management. These instructions trigger sleep/wakeup bits in the packet header to be set, which can turn on/off 10 sleep regions in each router as they traverse the network.

Tilera TILE64 and TILE64Pro. The iMesh has sophisticated network interfaces supporting both shared memory and message passing paradigms. The UDN (user dynamic network) supports user-level messaging, so threads can communicate through message passing in addition to the cache coherent shared memory. Upon message arrivals, user-level interrupts are issued for fast notification. Message queues can be virtualized onto off-chip DRAM in case of buffer overflows in the NIC. The caches and memory controllers are connected to the MDN (memory dynamic network) and TDN (tile dynamic network), with inter-tile cache transfers going through the TDN and responses going through the MDN, providing system-level deadlock freedom through two separate physical networks. The coherence protocol supported is not detailed, but TILE64Pro is marketed as having much higher performing coherence protocols. The Tilera designs provide support for communication via explicit message passing or through implicit user-level cache-coherence shared-memory.

ST Microelectronics STNoC. The STNoC encapsulates its support for communication and synchronization primitives and low-level platform services within what it calls the Interconnect Processing Unit (IPU). Examples of communication primitives are send, receive, read and write, while synchronization primitives are test-and-set and compare-and-swap. The aim is to have a library of different IPUs that support specific primitives so MPSoC designers can select the ones that are compatible with their IP blocks. For instance, IP blocks that interface with the old STBus require read-modify-write primitives that will be mapped to appropriate IPUs. Currently STNoC supports two widely used SoC bus standards fully: the STBus and AMBA AXI [15], and plans to add IPUs for other programming models and standards.

2.3.2 BRIEF STATE-OF-THE-ART SURVEY

Other System Architectures. In addition to the emergence of on-chip networks for cache-coherent chip multiprocessors, tiled microprocessors also leverage on-chip networks for scalar operand networks. A tiled microprocessor distributes functional units across multiple tiles; these designs mitigate problems of wire delay present in large superscalar architectures. Instructions are scheduled across available tiles. Architectures such as TRIPS [187], RAW [202] and Wavescalar [199] use operand networks to communicate register values between producing and consuming instructions. The result of an instruction is communicated to consumer tiles which can then wake up and fire instructions waiting on the new data. While we focus primarily on shared-memory CMPs here, several interconnection networks rely on message passing, including on-chip [202, 212, 95] and off-chip designs [72, 52, 123, 132].

Interaction of on-chip caches and on-chip network. Intel's Larrabee architecture [190] features a shared L2 cache design; each core has fast access to its own L2 subset and utilizes a ring network to communicate with remote L2 caches. The dynamic non-uniform cache architecture (NUCA) [112] utilizes an on-chip network among banks of a large shared cache to move data quickly to the processor. TRIPS employs both a scalar operand network and an on-chip cache network [80].

To maintain a coherent view of memory, all nodes must observe memory requests in the same order. This ordering is typically achieved through implicit ordering intrinsic to the interconnect (e.g. bus, ring) or through the use of an ordering point (e.g. AMD Opteron [41]). Inherent ordering properties within the topology can also be beneficial for chip multiprocessor. Work by Marty and Hill [142] exploits the partial ordering of a ring to ease the implementation of coherence solutions. Cache coherence protocols that leverage on-chip network properties and are embedded in-network have been explored as a way to further optimize communication energy and performance [64, 67, 9].

NoC Synthesis. Synthesis toolchains have been developed to design and optimize various on-chip network designs in the MPSoC domain. Æthereal [77] and Nostrum [128] provide high level models that allow for iterative design refinement. Cycle-accurate simulators such as MPARM [26] can be used to do design space exploration; emulation on an FPGA is another option that is orders of magnitude faster [74, 140]. Traffic models can be extracted based on average traffic flowing between cores [91, 96] to facilitate customized network design; however, care must be taken as traffic can vary widely depending on inputs. Once the network design has been defined, libraries of network components can be used to instantiate RTL [46, 198, 77, 131]. More detailed information on NoC synthesis can be found in Benini and De Micheli [29].

CHAPTER 3
Topology

The on-chip network topology determines the physical layout and connections between nodes and channels in the network. The effect of a topology on overall network cost-performance is profound. A topology determines the number of hops (or routers) a message must traverse as well as the interconnect lengths between hops, thus influencing network latency significantly. As traversing routers and links incurs energy, a topology's effect on hop count also directly affects network energy consumption. Furthermore, the topology dictates the total number of alternate paths between nodes, affecting how well the network can spread out traffic and hence support bandwidth requirements.

The implementation complexity cost of a topology depends on two factors: the number of links at each node (node degree) and the ease of laying out a topology on a chip (wire lengths and the number of metal layers required).

In this chapter, we will first describe several metrics that are very useful for developing back-of-the-envelope intuition when comparing topologies. Next, we will describe several commonly-used topologies in on-chip networks, and compare them using these metrics.

3.1 METRICS FOR COMPARING TOPOLOGIES

Since the first decision designers have to make when building an on-chip network is, frequently, the choice of the topology, it is useful to have a means for quick comparisons of different topologies before the other aspects of a network (such as its routing, flow control and microarchitecture) are even determined. Here, we describe several abstract metrics that come in handy when comparing different topologies at the early stages of design. Bisection bandwidth is a metric that is often used in the discussion of cost of off-chip networks. Bisection bandwidth is the bandwidth across a cut down the middle of the network. Bisection bandwidth can be used as a proxy for cost. It represents the amount of global wiring that will be necessary to implement the network. As a metric, bisection bandwidth is less useful for on-chip networks as global on-chip wiring is considered abundant; therefore, we do not delve into bisection calculations as an abstract metric. Figure 3.1 shows three commonly-used on-chip topologies. These topologies are used to illustrate the following metrics:

Degree. The degree of a topology refers to the number of links at each node. For instance, for the topologies in Figure 3.1, a ring topology has a degree of 2 since there are two links at each node, while a torus has a degree of 4 as each node has 4 links connecting it to 4 neighboring nodes. Note that in the mesh network, not all switches have a uniform degree. Degree is useful as an abstract metric of a network's cost, as a higher degree requires more ports at routers, which increases implementation complexity. degree

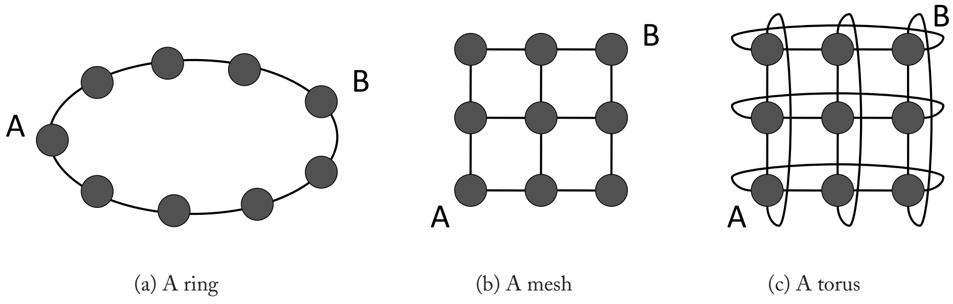


Figure 3.1: Common on-chip network topologies.

Hop count. The number of hops a message takes from source to destination, or the number of links it traverses, defines hop count. This is a very simple and useful proxy for network latency, since every node and link incurs some propagation delay, even when there is no contention. The maximum hop count is given by the diameter of the network; the diameter of the network is the largest, minimal hop count over all source-destination node pairs in the network. In addition to the maximum hop count, average hop count is very useful for as a proxy for network latency. The average minimum hop count is given by the average minimum hop count over all possible source-destination pairs in the network.

maximum hop count

average hop count

For the same number of nodes, and assuming uniform random traffic where every node has an equal probability of sending to every other node, a ring (Figure 3.1a) will lead to higher hop count than a mesh (Figure 3.1b) or a torus [54] (Figure 3.1c). For instance, in the figure shown, assuming bidirectional links and shortest-path routing, the maximum hop count of the ring is four, that of a mesh is also four, while a torus improves the hop count to two. Looking at average hop count, we see that the torus again has the lowest average hop count ($1\frac{1}{3}$). The mesh has a higher average hop count of $1\frac{7}{9}$. Finally, the ring has the worst average hop count of the three topologies in Figure 3.1 with an average of $2\frac{2}{9}$. The formulas for deriving these values will be presented in Section 3.2.

uniform random traffic

Maximum channel load. This metric is useful as a proxy for estimating the maximum bandwidth the network can support, or the maximum number of bits per second (bps) that can be injected by every node into the network before it saturates. Intuitively, it involves first determining which link or channel in the network will be the most congested given a particular traffic pattern, as this link will limit the overall network bandwidth. Next, the load on this channel is estimated. Since at this early stage of design, we do not yet know the specifics of the links we are using (how many actual interconnects form each channel, and each interconnects' bandwidth in bps), we need a relative way of measuring load. Here, we define it as being relative to the injection bandwidth. So, when we say the load on a channel is 2, it means that the channel is loaded with twice the injection bandwidth. So, if we inject a flit every cycle at every node into the network, two flits will wish to traverse this

maximum channel load

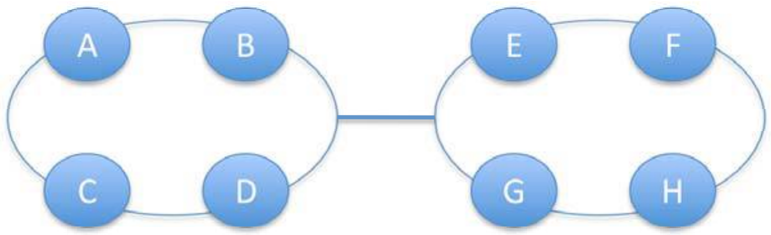


Figure 3.2: Channel load example with 2 rings connected via a single channel.

specific channel every cycle. If the bottleneck channel can handle only one flit per cycle, it constrains the maximum bandwidth of the network to half the link bandwidth, i.e. at most, a flit can be injected every other cycle. Thus, the higher the maximum channel load, the lower the network bandwidth.

Channel load can be calculated in a variety of ways, typically using probabilistic analysis. If routing and flow control are not yet determined, channel load can still be calculated assuming ideal routing (the routing protocol distributes traffic amongst all possible shortest paths evenly) and ideal flow control (the flow control protocol uses every cycle of the link whenever there is traffic destined for that link).

Here, we will illustrate this with a simple example, but the rest of the chapter will just show formulas for the maximum channel load of various common on-chip network topologies rather than walking through their derivations. Figure 3.2 shows an example network topology with two rings connected with a single channel. First, we assume uniform random traffic where every node has an equal probability of sending to every other node in the network including itself. To calculate maximum channel load, we need to first identify the bottleneck channel. Here, it is the single channel between the rings, shown in bold. We will assume it is a bidirectional link. With ideal routing, half of every node's injected traffic will remain within its ring, while the other half will be crossing the bottleneck channel. For instance, for every packet injected by node A, there is 1/8 probability of it going to either B, C, D, E, F, G, H or itself. When the packet is destined for A, B, C, D, it does not traverse the bottleneck channel; when it is destined for E, F, G, H, it does. Therefore, 1/2 of the injection bandwidth of A crosses the channel. So does 1/2 of the injection bandwidth of the other nodes. Hence, the channel load on this bottleneck channel, or the maximum channel load, is $8 * 1/2 = 4$. As a result, the network saturates at a bandwidth that is 1/4 the injection bandwidth. Adding more nodes to both rings will further increase the channel load, and thus decrease the bandwidth.

Path diversity. A topology that provides multiple shortest paths ($|R_{src-dst}| > 1$, where R represents the path diversity) between a given source and destination pair has greater path diversity than a topology where there is only a single path between a source and destination pair ($|R_{src-dst}| = 1$). Path diversity within the topology gives the routing algorithm more flexibility to load-balance traffic and to route around faults in the network (as discussed in Chapter 4). The ring in Figure 3.1a provides no path diversity ($|R| = 1$), because there is only one shortest path between pairs of nodes.

path diversity

If a packet travels clock-wise between A and B (in Figure 3.1a), it traverses four hops; if the packet goes counter-clockwise, it traverses five hops. So more paths can be supplied only at the expense of a greater distance travelled. With an even number of nodes in a ring, two nodes that are half-way around the ring from each other will have a path diversity of two due to two minimal paths. On the other hand, the mesh and torus in Figure 3.1b and c provide a wider selection of distinct paths between source and destination pairs. In Figure 3.1b, the mesh supplies six distinct paths between A and B, all at the shortest distance of four hops.

bus

crossbar

We focus here on switched topologies. However, there are two other types of networks that have been used to connect a small number of components: buses and crossbars. A bus connects a set of components with a single, shared channel. Each message on the bus can be observed by all components on the bus; it is an effective broadcast medium. Buses have limited scalability due to saturation of the shared channel as additional components are added. A crossbar is a switch that directly connects n inputs to m outputs without any intermediate stages. Using a crossbar creates a fully connected network with a single hop between all possible source and destination pairs. Crossbars will be discussed further as components of router microarchitectures in Chapter 6.

direct network

indirect network

A network topology can be classified as either direct or indirect. With a direct topology , each terminal node (e.g. a processor core or cache in a chip multiprocessor) is associated with a router, so all routers are sources and destinations of traffic. In a direct topology, nodes can source and sink traffic, as well as switch through traffic from other nodes. In an indirect topology, routers are distinct from terminal nodes; only terminal nodes are sources and destinations of traffic, intermediate nodes simply switch traffic to and from terminal nodes. In a direct network, packets are forwarded directly between terminal nodes. With an indirect network, packets are switched indirectly through a series of intermediate switch nodes between the source and the destination. To date, most designs of on-chip networks have used direct networks. Co-locating switches with terminal nodes is often most suitable in area-constrained environments such as on-chip networks.

3.2 DIRECT TOPOLOGIES: RINGS, MESHES AND TORI

k-ary n-cubes

Mesh and torus networks can be described as k-ary n-cubes , where k is the number of nodes along each dimension, and n is the number of dimensions. For instance, a 4×4 mesh or torus is a 4-ary 2-cube with 16 nodes, a 8×8 mesh or torus is a 8-ary 2-cube with 64 nodes, while a $4 \times 4 \times 4$ mesh or torus is a 4-ary 3-cube with 64 nodes. This notation assumes the same number of nodes on each dimension, so total number of nodes is k^n. Practically speaking, most on-chip networks utilize 2-dimensional topologies that map well to the planar substrate as otherwise, more metal layers will be needed; this is not the case for off-chip networks where cables between chassis provide three-dimensional connectivity. In each dimension, k nodes are connected with channels to their nearest neighbors. Rings fall into the torus family of network topologies as k-ary 1-cubes.

edge symmetry

With a torus, all nodes have the same degree; however, with a mesh, nodes along the edge of the network have a lower degree than nodes in the center of the network. A torus is also edge-symmetric (a mesh is not), this property helps the torus network balance traffic across channels.

Due to the absence of edge-symmetry, a mesh network experiences significantly higher demand for center channels than for edge channels.

Next, we examine the values for the torus and the mesh in terms of the abstract metrics given in Section 3.1. A torus network requires two channels in each dimension or $2n$. So for a 2-D torus, the switch degree would be four and for a 3-D torus, the switch degree would be six. The degree is the same for a mesh, although some ports on the edge of the network will go unused. The average hop count for a torus network is found by averaging the minimum distance between all possible node pairs. This gives

$$H_{avg} = \begin{cases} \frac{nk}{4} & k\ even \\ n(\frac{k}{4} - \frac{1}{4k}) & k\ odd \end{cases} .$$

Without the wrap-around links of a torus, the average minimum hop count for a mesh is slightly higher and is given by

$$H_{avg} = \begin{cases} \frac{nk}{3} & k\ even \\ n(\frac{k}{3} - \frac{1}{3k}) & k\ odd \end{cases} .$$

The channel load across the bisection of a torus under uniform random traffic with an even k is $k/8$. For a mesh, the channel load increases to $k/4$.

Both mesh and torus networks provide path diversity for routing messages. To route a message from src to dst, where src and dst are separated by Δsrc and Δdst hops, there are several ways of choosing how to take these hops. If we restrict the path diversity to only minimum paths, the number of possible paths is given by[1]

$$|R_{src-dst}| = \begin{pmatrix} \Delta src + \Delta dst \\ \Delta src \end{pmatrix} .$$

As the number of dimensions increase, so does the path diversity.

3.3 INDIRECT TOPOLOGIES: BUTTERFLIES, CLOS NETWORKS AND FAT TREES

A butterfly network is an example of an indirect topology. Butterfly networks can be described as k-ary n-flies. Such a network would consist of k^n terminal nodes (e.g. cores, memory), and comprises n stages of k^{n-1} $k \times k$ intermediate switch nodes. In other words, k is the degree of the switches, and n is the number of stages of switches. Figure 3.3 illustrates a 2-ary 3-fly network; source and destination nodes are shown as logically separate in this figure with source nodes on the left and destination nodes on the right.

k-ary n-fly

Next, we analyse the butterfly using the metrics given in Section 3.1. The degree of each intermediate switch in a butterfly network is given as $2k$. Unlike the mesh or the torus where the

[1]Assuming all paths take the same direction in each dimension.

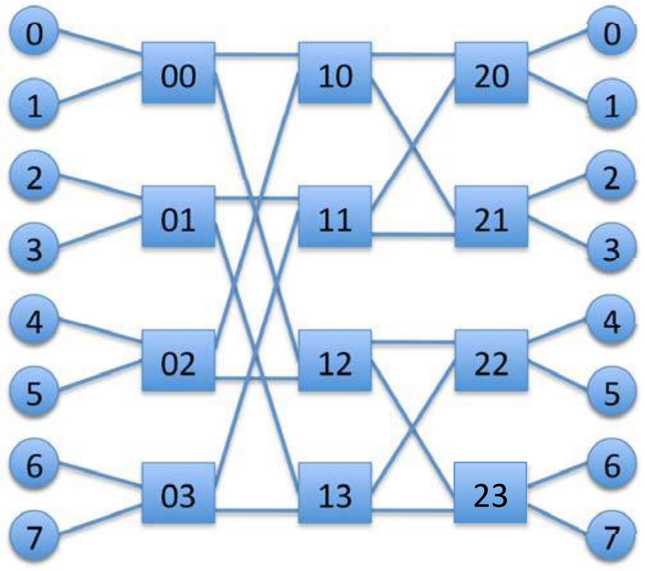


Figure 3.3: A 2-ary 3-fly butterfly network.

hop count varies based on source-destination pair, every source-destination pair in a butterfly network experiences the same hop count given by $n - 1$ (assuming that the source and destination nodes are also switches). With uniformly distributed traffic, the maximum channel load for the butterfly is 1. Other traffic patterns that require significant traffic to be sent from one half of the network to the other half will increase the maximum channel load.

The primary disadvantages of a butterfly network are the lack of path diversity and the inability of these networks to exploit locality. With no path diversity, a butterfly network performs poorly in the face of unbalanced traffic patterns such as when each node in one half of the network sends messages to a node in the other half.

Clos

A symmetric Clos network is a three-stage[2] network characterized by the triple, (m, n, r) where m is the number of middle stage switches, n is the number of input/output ports on each input/output switch (first and last stage switches), and r is the number of first/last stage switches. When $m > 2n - 1$, a Clos network is strictly non-blocking, i.e. any input port can connect to any unique output port, like a crossbar. A Clos network consists of $r \times n$ nodes. A 3-stage Clos has a hop count of four for all source destination pairs. A Clos network does not use identical switches at each stage. The degree of the first and last stage switches is given by $n + m$ while the degree of the middle

[2]A Clos network with a larger number of odd stages can be built by recursively replacing the middle switches with a 3-stage Clos.

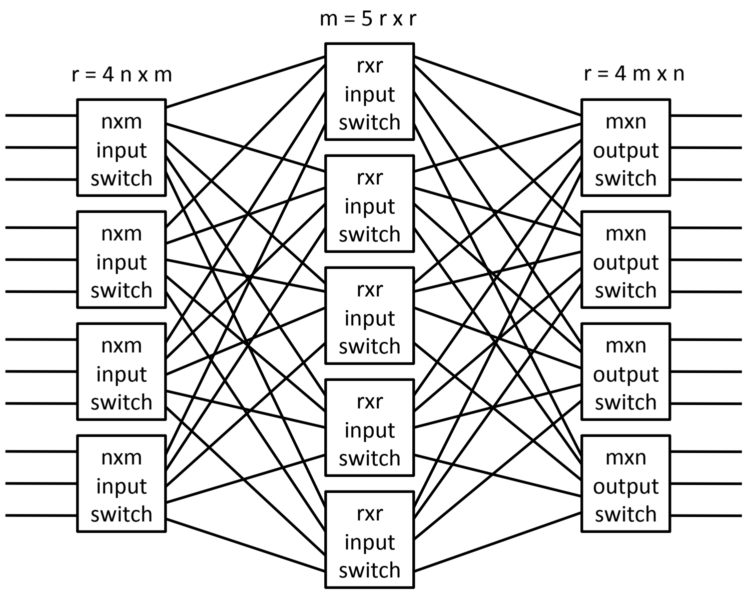


Figure 3.4: An $(m = 5, n = 3, r = 4)$ symmetric Clos network with $r = 4n \times m$ input-stage switches, $m = 5r \times r$ middle-stage switches, and $r = 4m \times n$ output-stage switches. Crossbars form all switches.

switches is $2r$. With m middle stages, a Clos network provides path diversity of $|R_{src-dst}| = m$. A disadvantage of a Clos network is its inability to exploit locality between source and destination pairs. Figure 3.4 depicts a 3-stage Clos network characterized by the triple (5, 3, 4).

A Clos network can be folded along the middle set of switches so that the input and output switches are shared. In Figure 3.5b, a 5-stage folded Clos network characterized by the triple (2, 2, 4) is depicted. The center stage is realized with another 3-stage Clos formed using (2, 2, 2) Clos network. This Clos network is folded along the top row of switches.

A fat tree [133] is logically a binary tree network in which wiring resources increase for stages closer to the root node (Figure 3.5a). A fat tree can be constructed from a folded Clos network, as shown in Figure 3.5b giving path diversity over the tree network in Figure 3.5a. The Clos is folded back on itself at the root, logically giving a 5-stage Clos network. In a fat tree, messages are routed up the tree until a common ancestor is reached and then routed down to the destination; this allows the fat tree to take advantage of locality between communicating nodes. Each switch in the fat tree has a logical degree of four, though the links in higher-level nodes are much wider than those in the lower levels. fat tree

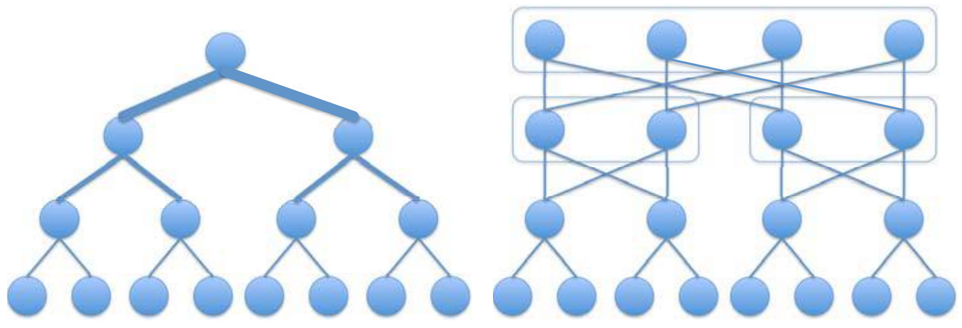


Figure 3.5: A fat tree network.

3.4 IRREGULAR TOPOLOGIES

MPSoC design may leverage a wide variety of heterogeneous IP blocks; as a result of the heterogeneity, regular topologies such as a mesh or a torus described above may not be appropriate. With these heterogeneous cores, a customized topology will often be more power efficient and deliver better performance than a standard topology.

Often, communication requirements of MPSoCs are known a priori. Based on these structured communication patterns, an application characterization graph can be constructed to capture the point-to-point communication requirements of the IP blocks. To begin constructing the required topology, the number of components, their size and their required connectivity as dictated by the communication patterns must be determined.

An example of a customized topology for a video object plane decoder is shown in Figure 3.6. The MPSoC is composed of 12 heterogeneous IP blocks. In Figure 3.6a, the design is mapped to a 3×4 mesh topology requiring 12 routers (R). When specific application characteristics are taken into account (e.g. not every block needs to communicate directly with every other block), a custom topology is created (Figure 3.6b). This irregular topology reduces the number of switches from 12 to 5; by reducing the number of switches and the links in the topology, significant power and area savings are achieved. Some blocks can be directly connected without the need for a switch, such as the VLD and run length decoder units. Finally, the degree of the switches has changed; the mesh in Figure 3.6a requires a switch with 5 input/output ports (although ports can be trimmed on edge nodes). The 5 input/output ports represent the four cardinal directions: north, south, east and west plus an Injection/Ejection port. All of these ports require both input and output connections leading to 5×5 crossbars. With a customized topology, not all blocks need both input and output ports; the largest switch in Figure 3.6b is a 3×3 switch. Not every connection between links coming into and out of a router is necessary in the customized topology resulting in smaller switches; connectivity has been limited because full connectivity is not needed by this specific application.

3.4.1 SPLITTING AND MERGING

Two types of techniques have been explored for customizing a topology: splitting and merging. With splitting, a large crossbar connecting all nodes is first created and then iteratively split into multiple small switches to accommodate a set of design constraints. Alternatively, a network with a larger number of switches such as a mesh or torus can be used as a starting point. From this starting point, switches are merged together to reduce area and power.

splitting

Splitting. One technique for arriving at a customized network topology is to begin with a large fully-connected switch (crossbar). Such a large crossbar will likely violate the design constraints and must iteratively be split into smaller switches until design constraints are satisfied [92]. When a switch is split into two smaller switches creating a partition, the bandwidth provided between the two switches must satisfy the volume of communication that must now flow between partitions. Nodes can be moved between partitions to optimize the volume of communication between switches.

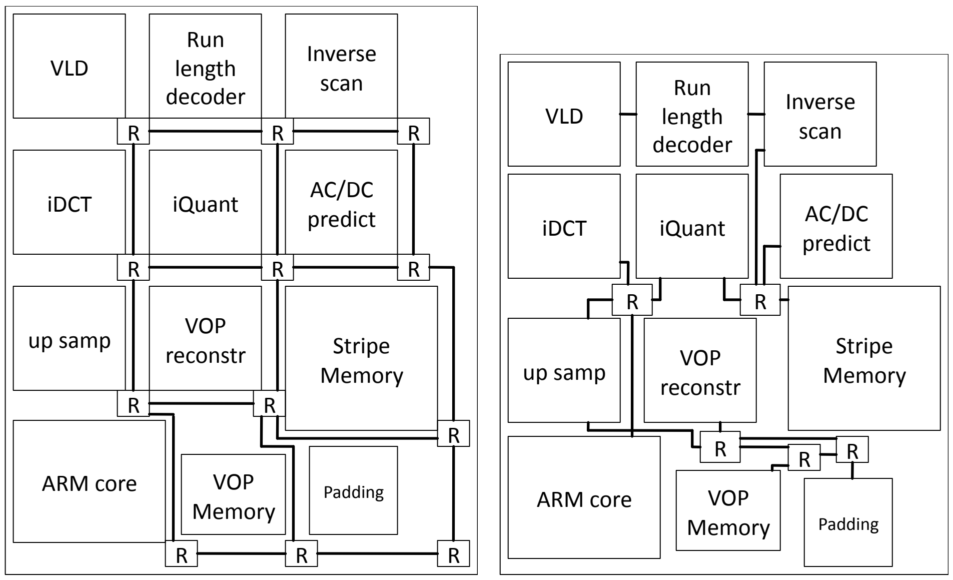
merging

Merging. An alternative to iteratively splitting larger switches into smaller switches is to begin with large number of switches and merge them [197, 179]. By merging adjacent routers in the topology, power and area costs can be reduced. In this type of design flow, floorplanning of the various MPSoC components is done as the first step. Floorplanning can be done based on the application characterization graph, e.g. nodes that communicate heavily should be placed in close proximity during floor planning. Next, routers are placed at each of the channel intersection points, where three or more channels merge or diverge. The last step merges adjacent routers if they are close together and if merging will not violate bandwidth or performance constraints and given that there is some benefit from such merging (e.g. power is reduced).

3.4.2 TOPOLOGY SYNTHESIS ALGORITHM EXAMPLE

Topology synthesis and mapping is an NP-hard problem [179]; as a result, several heuristics have been proposed to find the best topology in an efficient manner. In this section, we provide an example of one such application-specific topology synthesis algorithm from Murali et al. [157]. This algorithm is an example of a splitting algorithm; they begin with an application communication graph showing the bandwidth required between the various application tasks as shown in Figure 3.7a.

The algorithm synthesizes a number of different topologies, starting with a topology where all IP cores are connected through one large switch to the other extreme where each core has its own switch. For each switch count, the algorithm tunes the operating frequency and the link width. For a given switch count i, the input graph (Figure 3.7a) is partitioned into i min-cut partitions. Figure 3.7b shows a min-cut partition for $i = 3$. The min-cut partition is performed so that the edges of the graph that cross partitions have lower weights than the edges within partitions. Additionally, the number of nodes assigned to each partition remains nearly the same. Such a min-cut partition will ensure that traffic flows with high bandwidth will use the same switch for communication.



(a) Mesh (b) Custom

Figure 3.6: A regular (mesh) topology and a custom topology for a video object plane decoder (VOPD) (from [30]).

Once the min-cut partitions have been determined, routes must be restricted to avoid deadlocks. Next, physical links between switches must be established and paths must be found for all traffic flows through the switches. Once the size of the switches and their connectivity is determined, the design can be evaluated to see if power consumption of the switches and hop count objectives have been met. Finally, a floorplanner is used to determine the area and wire lengths of a synthesized design.

3.5 LAYOUT AND IMPLEMENTATION

While rings have poorer performance (latency, throughput, energy and reliability) when compared to higher-dimensional networks, they have lower implementation overhead. A ring has a node degree of two while a mesh or torus has a node degree of four, where node degree refers to the number of links in and out of a node. A higher node degree requires more links and higher port counts at routers. All three topologies, featured in Figure 3.1, are two-dimensional planar topologies that map readily to a single metal layer. Moreover, while the ring and mesh physical layouts correspond to their logical layout in Figure 3.1, the torus has to physically arranged in a folded form to equalize

folded torus

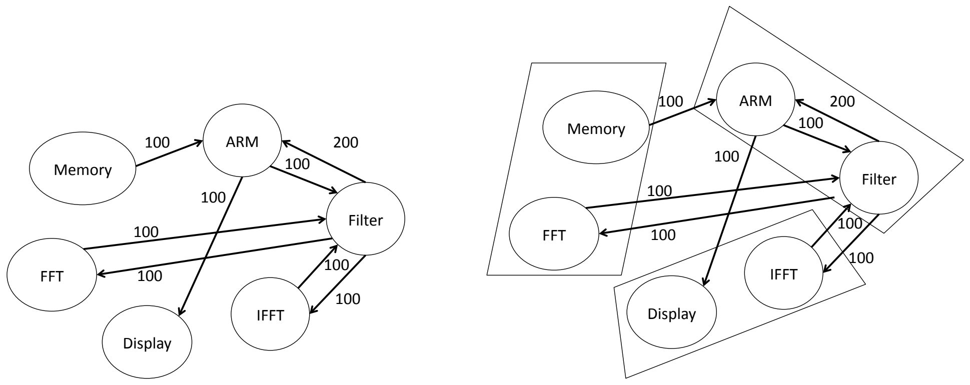


Figure 3.7: Topology Synthesis Algorithm example.

wire lengths (see Figure 3.8) instead of employing long wrap-around links between edge nodes. A torus illustrates the importance of considering implementation details in comparing alternative topologies. While a torus has lower hop count (which leads to lower delay and energy) compared to a mesh, wire lengths in a folded torus are twice that in a mesh of the same size, so per-hop latency and energy are actually higher. Furthermore, a torus requires twice the number of links which must be factored into the wiring budget. If the available wire tracks along the bisection is fixed, a torus will be restricted to narrower links than a mesh, thus lowering per-link bandwidth, and increasing transmission delay.

3.6 CONCENTRATORS

Up to this point, we have assumed a one-to-one correspondence between network nodes and terminal nodes. However, this need not be the case. Frequently, multiple cores that do not require the bandwidth of a single network node will share a node by using concentrators. Figure 3.9 shows a mesh where there are nodes shared by four tiles (cores, caches, etc). The use of concentration reduces the number of routers needed in the network, thereby reducing the hop count and the number of network routers. By reducing hop count and the number of routers, concentration helps networks scale to larger sizes. Using concentration, this 3×3 mesh connects 36 nodes with only 9 routers. A 6×6 mesh using 36 routers would be required to connect the same number of nodes without concentration. While saving area and hop count, concentration can increase network complexity. The concentrator must implement a policy for sharing injection bandwidth. This policy can dynamically share bandwidth or statically partition bandwidth so that each node gets $\frac{1}{c}$ the injection bandwidth,

concentration

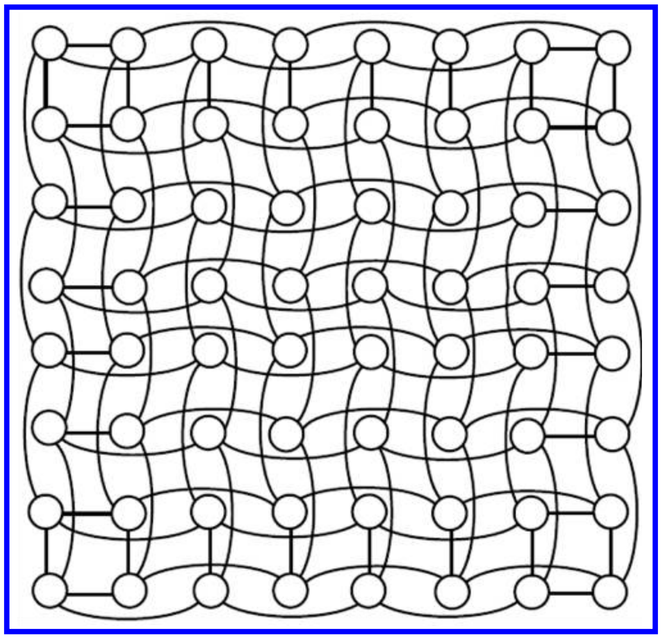


Figure 3.8: Layout of a 8x8 folded torus.

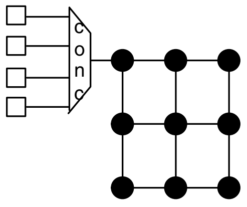


Figure 3.9: A mesh where four nodes are sharing bandwidth through a concentrator.

where c is the concentration factor. Furthermore, another drawback of using concentration is that during periods of bursty communication, the injection port bandwidth can become a bottleneck.

3.7 IMPLICATION OF ABSTRACT METRICS ON ON-CHIP IMPLEMENTATION

We introduced various abstract metrics at the beginning of this chapter, and used them to compare and contrast various common topologies. Here, we will discuss the implications of these simple metrics on on-chip network implementation, explaining why they are good proxies for on-chip network delay, area and power, while highlighting common pitfalls in the use of these metrics.

Node degree is useful as a proxy for router complexity, as higher degree implies greater port count. Adding a port in a router leads to additional input buffer queue(s), additional requestors to the allocators, as well as additional ports to the crossbar switch, all major contributors to a router's critical path delay, area footprint, and power. While router complexity is definitely increased for topologies with higher node degree, link complexity does not correlate directly with node degree. This is because link complexity depends on the link width, as link area and power overheads correlate more closely with the number of wires than the number of ports. So if the same number of wires is divided amongst a 2-port router and a 3-port router, link complexity will be roughly equal.

Hop count is a metric that is widely used as a proxy for overall network latency and power. Intuitively, it makes sense, since flits typically have to stop at each hop, going through the router pipeline followed by the link delay. However, hop count does not always correlate with network latency in practice, as it depends heavily on the router pipeline length and the link propagation delay. For instance, a network with only two hops, router pipeline depths of 5 cycles, and long inter-router distances requiring 4 cycles for link traversal, will have an actual network latency of 18 cycles. Conversely, a network with three hops where each router has a single-cycle pipeline and the link delay is a single cycle, will have a total network latency of only six cycles. If both networks have the same clock frequency, the latter network with the higher hop count will instead be faster. Unfortunately, factors such as router pipeline depth are typically not known until later in the design cycle.

With topologies typically trading off node degree and hop count, i.e. a topology may have low node degree but high average hop count (e.g. a ring), while another may have high node degree but low average hop count (e.g. a mesh), comparisons between topologies become trickier. Implementation details have to be factored in before an astute choice can be made.

Maximum channel load is another metric that is useful as a proxy of network performance and throughput. Here, it is a good proxy for network saturation throughput and maximum power. The higher the maximum channel load on a topology, the greater the congestion in the network caused by the topology and routing protocol, and thus, the lower the overall realizable throughput. Clearly, the specific traffic pattern affects maximum channel load substantially and representative traffic patterns should be used in estimating maximum channel load and throughput. Since it is a good proxy for saturation, it is also very useful for estimating peak power, as dynamic power is highest with peak switching activity and utilization in the network.

3.8 BIBLIOGRAPHIC NOTES

3.8.1 CASE STUDIES

IBM Cell. Its main interconnect consists of four unidirectional rings, two in each direction. As each ring is 16 bytes wide, runs at 1.6GHz, and can support 3 concurrent transfers, total network bisection bandwidth is 307.2GB/s [10, 119]. However, the bus access semantics and the ring topology can lead to a worst-case throughput of 50% with adversarial traffic patterns.

Tilera TILE64. The iMesh consists of five 8 × 8 meshes, each channel consisting of two 32-bit unidirectional links, leveraging the wealth of wiring available on-chip. Tilera designers explained that their choice of meshes over tori avoided the longer link lengths of a folded torus [212]. The edges of the meshes are fed with off-chip traffic from the four memory controllers and high-speed I/Os. Traffic is statically divided across the five meshes, with one for handling each of user-level messaging traffic, I/O traffic, memory traffic, inter-tile traffic and compiler-scheduled traffic. At the chip frequency of 1GHz, iMesh provides a bisection bandwidth of 320GB/s. Actual realizable throughput depends on the traffic and how it is managed and balanced by the flow control and routing protocols.

Intel TeraFLOPS. The TeraFLOPS chip consists of an 8 × 10 mesh, with each channel composed of two 38-bit unidirectional links. It runs at an aggressive clock of 5GHz on a 65nm process. This design gives it a bisection bandwidth of 380GB/s or 320GB/s of actual data bisection bandwidth, since 32 out of the 38 bits of a flit are data bits; the remaining 6 bits are used for sideband. Again, depending on the traffic, routing and flow control, realizable throughput will be a fraction of that.

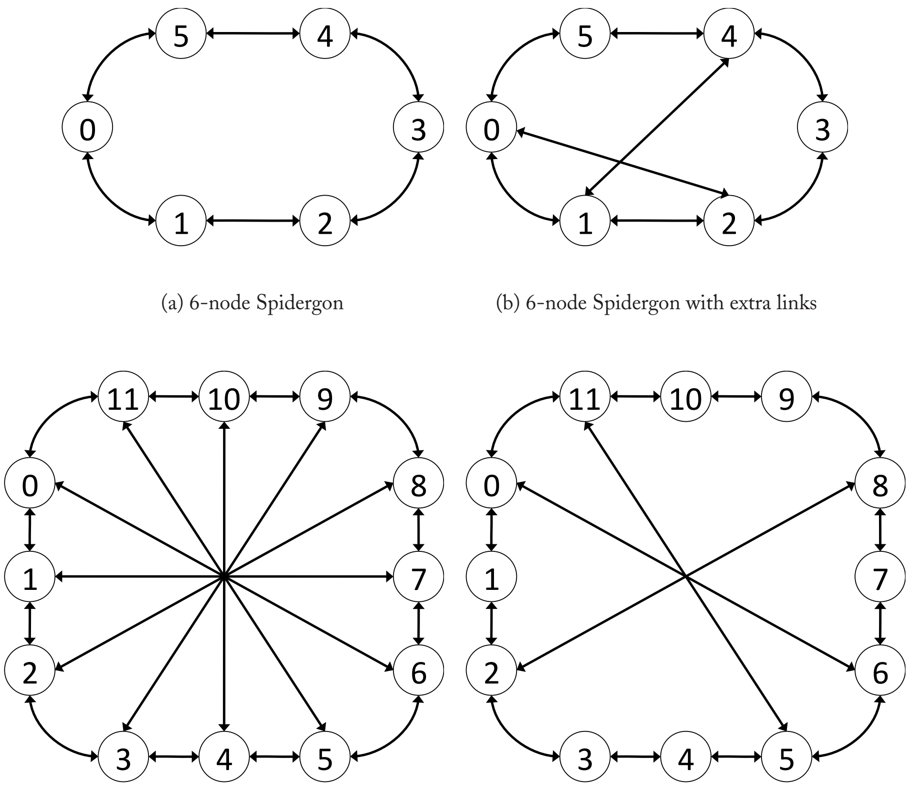
ST Microelectronics STNoC. The STNoC proposes a novel pseudo-regular topology, the Spidergon, that can be readily tailored depending on the actual application traffic characteristics, which are known a priori. Figure 3.10 sketches several variants of spidergons. Figure 3.10a shows a 6-node spidergon that can have more links added to cater to higher bandwidth needs (Figure 3.10b). Figure 3.10c shows a maximally-connected 12-node spidergon, where most links can be trimmed off when they are not needed (Figure 3.10d).

The pseudo-regularity in STNoC permits the use of identical router nodes across the entire range of spidergon topologies: routers with a degree of three, which simplifies design. Compared to a mesh that is trimmed and hence not symmetrical in X and Y dimensions, the spidergon's hop count scales steadily with node count rather than erratically (A 12 × 12 mesh has higher average hop count than a 12 × 10 mesh but a 12 × 14 mesh's average hop count is smaller than a 12 × 12), making it easier for a synthesis algorithm to arrive at the optimal topology. It also makes it possible to design a general routing algorithm across the topology family as discussed in Chapter 4. A regular layout algorithm is also possible, as Figure 3.11 illustrates.

3.8.2 BRIEF STATE-OF-THE-ART SURVEY

Significant research exists into all of the topologies discussed in this chapter. Much of this research has been conducted with respect to off-chip networks [47, 63, 178], but fundamental concepts apply to on-chip networks as well.

The majority of on-chip network proposals gravitate towards either ring or mesh topologies. For example, the IBM Cell processor, the first product with an on-chip network, used a ring topology, largely for its design simplicity, its ordering properties and low power consumption. Four rings are used to boost the bandwidth and to help alleviate the latency (halving the average hop count). Similarly, the proposed Intel Larrabee [190] also adopted the two-ring topology. Another simple,



(a) 6-node Spidergon

(b) 6-node Spidergon with extra links

(c) 12-node Spidergon with all links

(d) 12-node Spidergon with links removed

Figure 3.10: Various Spidergon Topologies.

regular topology, the mesh, has also been adopted. The MIT Raw chip, the first chip with an on-chip network has 4 meshes.

In Balfour and Dally [21], a comparison of various on-chip network topologies including a mesh, concentrated mesh, torus and fat tree, is presented. Furthermore, cost (including power and area) and performance are considered in this design space exploration. Balfour and Dally also suggest potential benefits for employing multiple parallel networks (all of the same topologies) to improve network throughput. Multiple meshes have been used in the MIT Raw chip and its follow-on commercialization into the Tilera TILE64 chip[212]. Each parallel network is a mesh topology and different types of traffic are routed over the distinct networks. Multiple rings have been used in the IBM Cell [176] and the proposed Intel Larrabee [190].

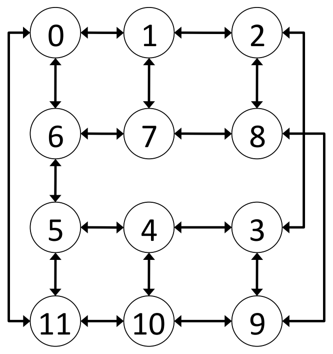


Figure 3.11: 12 Node Full Spidergon Layout for the logical Spidergon depicted in Figure 3.10c.

Novel topologies have also been proposed for on-chip networks, focusing on the unique properties of on-chip networks such as the availability of large number of wiring tracks as well as the irregularity of MPSoC's traffic demands. Examples include the flattened butterfly [113], the dragonfly [114], a hierarchical star [130] and the spidergon [43]. The flattened butterfly has been proposed to make butterfly networks suitable for on-chip use [113]. With the proposed 4×4 flattened butterfly, each destination can be reached with a maximum of two hops. However, minimal routing can do a poor job of balancing the traffic load, so non-minimal paths may be selected, thereby, increasing the hop count. A potential disadvantage to the flattened butterfly is the presence of long links. Hierarchical combinations of topologies, as well as the addition of express links between non-adjacent nodes have also been proposed [49, 83, 58], and tailored to the MPSoC domain where there is prior knowledge of NoC bandwidth demands and connectivity [164, 91].

In the MPSoC domain, a variety of topologies have been explored. SPIN [11] proposes using a fat tree network. Both Æthereal [78] and xpipes [104] leverage irregular topologies customized for specific application demands. The Nostrum design relies on a mesh [148]. Bolotin et al. [33] propose trimming unnecessary links from a mesh and leveraging non-uniform link bandwidth within the mesh.

CHAPTER 4

Routing

After determining the network topology, the routing algorithm is used to decide what path a message will take through the network to reach its destination. The goal of the routing algorithm is to distribute traffic evenly among the paths supplied by the network topology, so as to avoid hotspots and minimize contention, thus improving network latency and throughput. All of these performance goals must be achieved while adhering to tight constraints on implementation complexity: routing circuitry can stretch critical path delay and add to a router's area footprint. While energy overhead of routing circuitry is typically low, the specific route chosen affects hop count directly, and thus substantially affects energy consumption.

4.1 TYPES OF ROUTING ALGORITHMS

In this section, we briefly discuss various classes of routing algorithms. Routing algorithms are generally divided into three classes: deterministic, oblivious and adaptive.

While numerous routing algorithms have been proposed, the most commonly used routing algorithm in on-chip networks is dimension-ordered routing (DOR) due to its simplicity. Dimension-ordered routing is an example of a deterministic routing algorithm, in which all messages from node A to B will always traverse the same path. With DOR, a message traverses the network dimension-by-dimension, reaching the ordinate matching its destination before switching to the next dimension. In a 2-dimensional topology such as the mesh in Figure 4.1, X-Y dimension-ordered routing sends packets along the X-dimension first, followed by the Y-dimension. A packet travelling from (0,0) to (2,3) will first traverse 2 hops along the X-dimension, arriving at (2,0), before traversing 3 hops along the Y-dimension to its destination.

dimension-order routing (DOR)

deterministic routing

Another class of routing algorithms are oblivious ones, where messages traverse different paths from A to B, but the path is selected without regard to network congestion. For instance, a router could randomly choose among alternative paths prior to sending a message. Figure 4.1 shows an example where messages from (0,0) to (2,3) can be randomly sent along either the Y-X route or the X-Y route. Deterministic routing is a subset of oblivious routing.

oblivious routing

A more sophisticated routing algorithm can be adaptive, in which the path a message takes from A to B depends on network traffic situation. For instance, a message can be initially following the X-Y route and see congestion at (1,0)'s east outgoing link. Due to this congestion, the message will instead choose to take the north outgoing link towards the destination (see Figure 4.1).

adaptive routing

Routing algorithms can also be classified as minimal and non-minimal. Minimal routing algorithms select only paths that require the smallest number of hops between the source and the destination. Non-minimal routing algorithms allow paths to be selected that may increase the

minimal routing

non-minimal routing

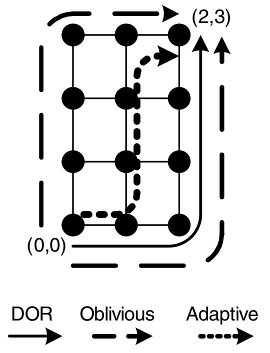


Figure 4.1: *DOR* illustrates an X-Y route from (0,0) to (2,3) in a mesh, while *Oblivious* shows two alternative routes (X-Y and Y-X) between the same source-destination pair that can be chosen obliviously prior to message transmission. *Adaptive* shows a possible adaptive route that branches away from the X-Y route if congestion is encountered at (1,0).

number of hops between the source and destination. In the absence of congestion, non-minimal routing increases latency and also power consumption as additional routers and links are traversed by a message. With congestion, the selection of a non-minimal route that avoids congested links, may result in lower latency for packets.

Before we get into details on specific deterministic, oblivious and adaptive routing algorithms, we will discuss the potential for deadlock that can occur with a routing algorithm.

4.2 DEADLOCK AVOIDANCE

In selecting or designing a routing algorithm, not only must its effect on delay, energy, throughput and reliability be taken into account, most applications also require the network to guarantee deadlock freedom. A deadlock occurs when a cycle exists among the paths of multiple messages. Figure 4.2 shows four gridlocked (deadlocked) messages waiting for links that are currently held by other messages, preventing any message from making forward progress. The packet entering router A from the South input port is waiting to leave through the East output port, but another packet is holding onto that exact link while waiting at router B to leave via the South output port, which is again held by another packet that is waiting at router C to leave via the West output port and so on.

routing deadlock

Deadlock freedom can be ensured in the routing algorithm by preventing cycles among the routes generated by the algorithm, or in the flow control protocol by preventing router buffers from being acquired and held in a cyclic manner [63, 178]. A network that uses a deadlock-prone routing algorithm requires a flow control protocol that ensures deadlock freedom, as discussed in Section 5.6.

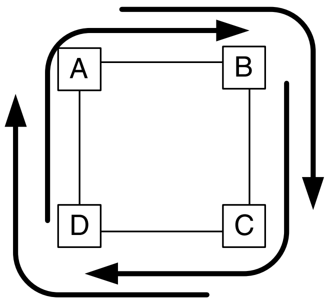


Figure 4.2: A classic network deadlock where four packets cannot make forward progress as they are waiting for links that other packets are holding on to.

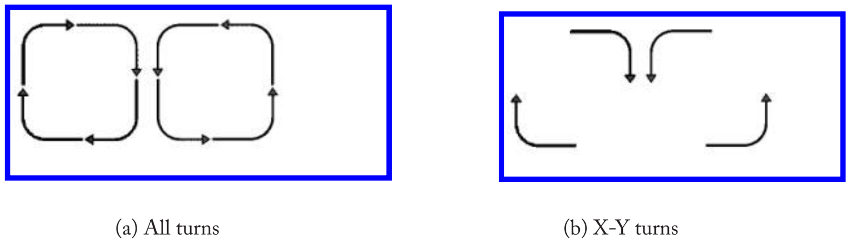


Figure 4.3: Possible routing turns for a 2D Mesh.

4.3 DETERMINISTIC DIMENSION-ORDERED ROUTING

A routing algorithm can be described by which turns are permitted. Figure 4.3a illustrates all possible turns in a 2D mesh network while Figure 4.3b illustrates the more limited set of permissible turns allowed by DOR X-Y routing. Allowing all turns results in cyclic resource dependencies, which can lead to network deadlock. To prevent these cyclic dependencies, turns may be disallowed. As you can see, no cycle is present in Figure 4.3b. Specifically, a message traveling east or west is allowed to turn north or south; however, messages traveling north and south are permitted no turns. Two of the four turns in Figure 4.2 will not be permitted, so a cycle is not possible.

Alternatively, Y-X routing can be used where messages traveling north or south are allowed to turn east or west but once a message is traveling East or West, no further turns are permitted. Depending on the network dimensions, i.e. whether there are more nodes along X or Y, one of these routing algorithms will balance load better with uniform random traffic since channel load is higher along the dimension with fewer nodes.

Dimension order routing is both simple and deadlock-free; however, it eliminates path diversity in a mesh network and thus lowers throughput. With dimension order routing, exactly one path exists between every source and destination pair. Without path diversity, the routing algorithm is unable to route around faults in the network or avoid areas of congestion. As a result of routing restrictions, dimension order routing does a poor job of load balancing the network.

4.4 OBLIVIOUS ROUTING

Using an oblivious routing algorithm, routing paths are chosen without regard to the state of the network. By not using information about the state of the network, these routing algorithms can be kept simple.

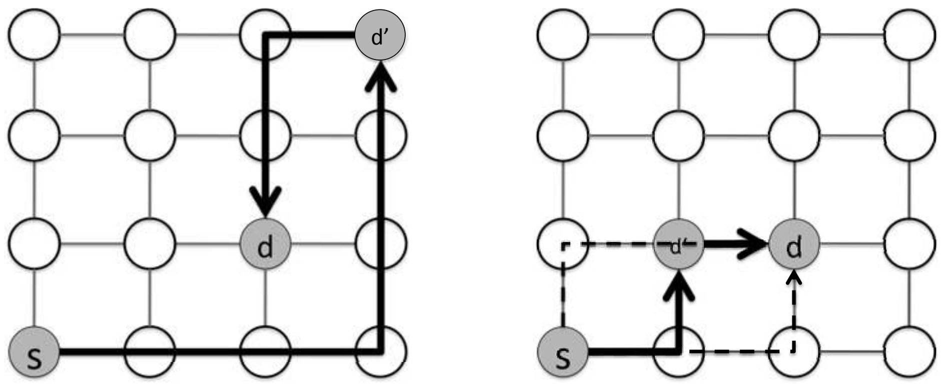
Valiant's routing algorithm

Valiant's randomized routing algorithm [205] is one example of an oblivious routing algorithm. To route a packet from source *s* to destination *d* using Valiant's algorithm, an intermediate destination *d'* is randomly selected. The packet is first routed from *s* to *d'* and then from *d'* to *d*. By routing first to a randomly selected intermediate destination before routing to the final destination, Valiant's algorithm is able to load balance traffic across the network; the randomization causes any traffic pattern to appear to be uniform random. Load balancing with Valiant's algorithm comes at the expense of locality; for example, by routing to an intermediate destination, the locality of near neighbor traffic on a mesh is destroyed. Hop count is increased, which in turn increases the average packet latency and the average energy consumed by the packet in the network.

Valiant's routing algorithm can be restricted to support only minimal routes [158], by restricting routing choices to only the shortest paths in order to preserve locality. In a k-ary n-cube topology, the intermediate node *d'* must lie within the minimal quadrant; the smallest n-dimensional sub-network with *s* and *d* as corner nodes bounding this quadrant.

With both Valiant's randomized routing and minimal adaptive routing, dimension order routing can be used to route from *s* to *d'* and from *d'* to *d*. If DOR is used, not all paths will be exploited but better load balancing is achieved than deterministic routing from *s* directly to *d*. Figure 4.4 illustrates a routing path selected using Valiant's algorithm and minimal oblivious routing. In Figure 4.4a, Valiant's algorithm randomly selects an intermediate destination *d'*. The random selection can destroy locality and significantly increase hop count; here, the hop count is increased from three hops to nine hops. To preserve locality, minimal oblivious routing can be employed as in Figure 4.4b. Now, *d'* can only be selected to lie within the minimal quadrant formed by *s* and *d*, preserving the minimum hop count of three. One possible selection is highlighted (two other paths are possible for this source-destination pair as shown with dashed lines).

Valiant's routing algorithm and minimal oblivious routing are deadlock free when used in conjunction with X-Y routing. An example of an oblivious routing algorithm that is not deadlock free is one that randomly chooses between X-Y or Y-X routes. The oblivious algorithm that randomly



(a) Valiant's Routing Algorithm (b) Minimal Oblivious Routing Algorithm

Figure 4.4: Oblivious Routing Examples.

chooses between X-Y or Y-X routes is not deadlock-free because all four turns from Figure 4.2 are possible leading to potential cycles in the link acquisition graph.

4.5 ADAPTIVE ROUTING

A more sophisticated routing algorithm can be adaptive, i.e. the path a message takes from A to B depends on the network traffic situation. For instance, a message can be going along the X-Y route, see congestion at (1,0)'s east outgoing link and instead choose to take the north outgoing link towards the destination (see Figure 4.1).

Local or global information can be leveraged to make adaptive routing decisions. Adaptive routing algorithms often rely on local router information such as queue occupancy and queuing delay to gauge congestion and select links [50]. The backpressure mechanisms used by flow control (discussed in the next chapter) allow congestion information to propagate from the congestion site back through the network.

Figure 4.5 shows all possible (minimal) routes that a message can take from Node (0,0) to Node (2,3). There are nine possible paths. An adaptive routing algorithm that leverages only minimal paths could exploit a large degree of path diversity to provide load balancing and fault tolerance.

Adaptive routing can be restricted to taking minimal routes between the source and the destination. An alternative option is to employ misrouting, which allows a packet to be routed in a non-productive direction resulting in non-minimal paths. When misrouting is permitted, livelock becomes a concern. Without mechanisms to guarantee forward progress, livelock can occur as a packet is continuously misrouted so as to never reach its destination. We can combat this problem

misrouting

livelock

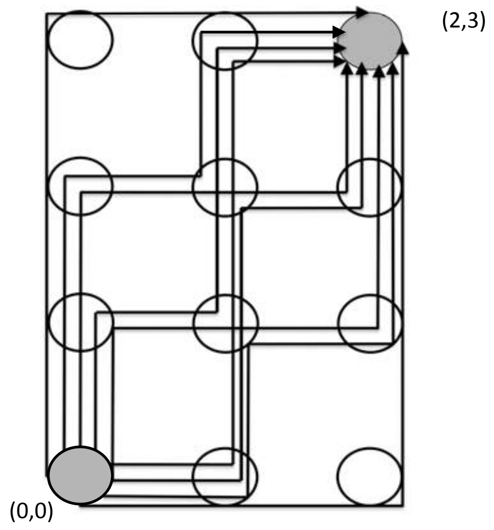


Figure 4.5: Adaptive Routing Example.

by allowing a maximum number of misroutes per packet and giving higher priority to packets than have been misrouted a large number of times. Misrouting increases the hop count but may reduce end-to-end packet latency by avoiding congestion (queueing delay).

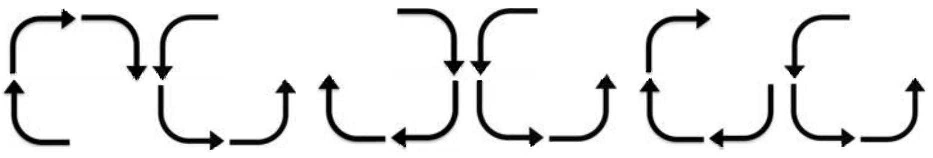
With a fully-adaptive routing algorithm, deadlock can become a problem. For example, the adaptive route shown in Figure 4.1 is a superset of oblivious routing and is subject to potential deadlock. Planar-adaptive routing [38] limits the resources needed to handle deadlock by restricting adaptivity to only two dimensions at a time. Duato has proposed flow control techniques that allow full routing adaptivity while ensuring freedom from deadlock [62]. Deadlock-free flow control will be discussed in Chapter 5.

Another challenge with adaptive routing is preserving inter-message ordering as may be needed by the coherence protocol. If messages must arrive at the destination in the same order that the source issued them, adaptive routing can be problematic. Mechanisms to re-order messages at the destination can be employed or messages of a given class can be restricted in their routing to prevent re-ordering.

4.5.1 ADAPTIVE TURN MODEL ROUTING

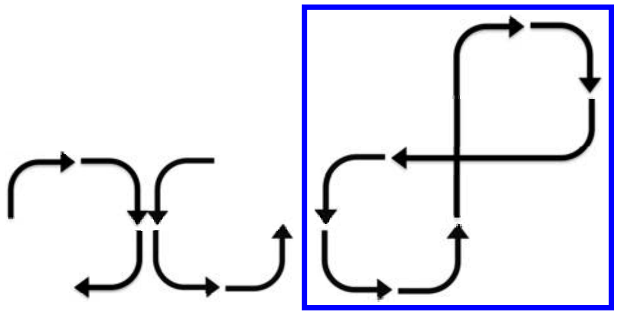
turn model routing

While we introduced turn model routing earlier in Section 4.3, discussing how dimension order X-Y routing eliminates two out of four turns (Figure 4.3), here, we explain how turn model can be more broadly applied to derive deadlock-free adaptive routing algorithms. Adaptive turn model routing eliminates the minimum set of turns needed to achieve deadlock freedom while retaining some path diversity and potential for adaptivity.



(a) West First Turns (b) North Last Turns (c) Negative First Turns

Figure 4.6: Turn Model Routing.



(a) Illegal Turn Model Routing (b) Resulting Deadlock Cycle

Figure 4.7: Turn Model Deadlock.

With dimension order routing only four possible turns are permitted of the eight turns available in a two dimensional mesh. Turn model routing [75] increases the flexibility of the algorithm by allowing six out of eight turns. Only one turn from each cycle is eliminated.

In Figure 4.6, three possible routing algorithms are illustrated. Starting with all possible turns (shown in Figure 4.6a), the north to west turn is eliminated; after this elimination is made, the three routing algorithms shown in Figure 4.6 can be derived. In Figure 4.6a, the west-first algorithm is shown; in addition to eliminating the North to West turn, the South to West turn is eliminated. In other words, a message must first travel in the West direction before traveling in any other direction. The North-Last algorithm (Figure 4.6b) eliminates both the North to West and the North to East turns. Once a message has turned North, no further turns are permitted; hence, the North turn must be made last. Finally, Figure 4.6c removes turns from North to West and East to South to create the Negative-First algorithm. A message travels in the negative directions (west and south) first before it is permitted to travel in positive directions (east and north). All three of these turn model routing algorithms are deadlock-free. Figure 4.7 illustrates a possible turn elimination that is invalid; the

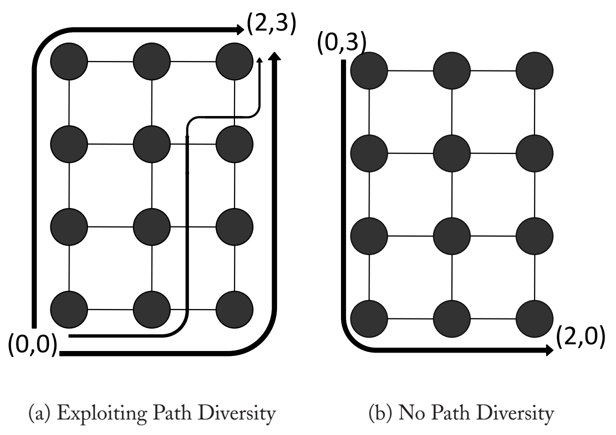


Figure 4.8: Negative-First Routing example.

elimination of North to West combined with the elimination of West to North can lead to deadlock. A deadlock cycle is depicted in Figure 4.7b that can result from a set of messages using the turns specified in Figure 4.7a.

Odd-even turn model routing [39] proposes eliminating a set of two turns depending on whether the current node is in an odd or even column. For example, when a packet is traversing a node in an even column[1], turns from East to North and from North to West are prohibited. For packets traversing an odd column node, turns from East to South and from South to West are prohibited. With this set of restrictions, the odd-even turn model is deadlock free provided 180° turns are disallowed. The odd-even turn model provides better adaptivity than other turn model algorithms such as West-First. With West-First, destinations to the West of the source, have no flexibility; with odd-even routing, there is flexibility depending on the allowable turns for a given column.

In Figure 4.8, we apply the Negative-First turn model routing to two different source destination pairs. In Figure 4.8a, three possible routes are shown between (0,0) and (2,3) (more are possible); turns from North to East and from East to North are permitted allowing for significant flexibility. However, in Figure 4.8b, there is only one path allowed by the algorithm to route from (0,3) to (2,0). The routing algorithm does not allow the message to turn from East to South. Negative routes must be completed first, resulting in no path diversity for this source-destination pair. As illustrated by this example, turn model routing provide more flexibility and adaptivity than dimension-order routing but it is still somewhat restrictive.

[1]A column is even if the dimension-0 coordinate of the column is even.

Table 4.1: Routing Algorithm and Implementation Options .

Routing Algorithm	Source Routing	Combinational	Node Table
Deterministic			
DOR	Yes	Yes	Yes
Oblivious			
Valiant's	Yes	Yes	Yes
Minimal	Yes	Yes	Yes
Adaptive	No	Yes	Yes

4.6 IMPLEMENTATION

In this section, we discuss various implementation options for routing algorithms. Routing algorithms can be implemented using look-up tables at either the source nodes or within each router. Combinational circuitry can be used as an alternative to table-based routing. Implementations have various trade-offs, and not all routing algorithms can be achieved with each implementation. Table 4.1 shows examples for how routing algorithms in each of the three different classes can be implemented.

4.6.1 SOURCE ROUTING

Routing algorithms can be implemented in several ways. First, the route can be embedded in the packet header at the source, known as source routing. For instance, the X-Y route in Figure 4.1 can be encoded as $< E, E, N, N, N, Eject >$, while the Y-X route can be encoded as $< N, N, N, E, E, Eject >$. At each hop, the router will read the leftmost direction off the route header, send the packet towards the specified outgoing link, and strip off the portion of the header corresponding to the current hop.

There are a few benefits to source routing. First, by selecting the entire route at the source, latency is saved at each hop in the network since the route does not need to be computed or looked up. The per-router routing hardware is also saved; no combinational routing logic or routing tables are needed once the packet has received its route from the source node. Second, source routing tables can be reconfigured to deal with faults and can support irregular topologies. Multiple routes per source-destination pair can be stored in the table (as shown in Table 4.2) and selected randomly for each packet to improve load balancing.

The disadvantages of source routing include the bit overheads required to store the routing table at the network interface of each source and to store the entire routing path in each packet; these paths are of arbitrary length and can grow large depending on network size. For a 5-port switch, each routing step is encoded by a 3-bit binary number. Just as the packet must be able to handle arbitrary length routing paths, the source table must also be designed to efficiently store different length paths. Additionally, by choosing the entire route at the source node, source based routing is unable to take advantage of dynamic network conditions to avoid congestion. However, as mentioned, multiple

Table 4.2: Source routing table at Node (0,0) for the 2x3 mesh in Figure 4.1 .

Destination	Route 0	Route 1
00	X	X
10	EX	EX
20	EEX	EEX
01	NX	NX
11	NEX	ENX
21	NEEX	ENEX
02	NNX	NNX
12	ENNX	NENX
22	EENNX	NNEEX
03	NNNX	NNNX
13	NENNX	ENNNX
23	EENNNX	NNNEEX

routes can be stored in the table and selected either randomly or with a given probability to improve the load distribution in the network.

4.6.2 NODE TABLE-BASED ROUTING

More sophisticated algorithms are realized using routing tables at each hop which store the outgoing link a packet should take to reach a particular destination. By accessing routing information at each hop (rather than all at the source), adaptive algorithms can be implemented and per-hop network congestion information can be leveraged in making decisions.

Table 4.3 shows the routing table for the west-first turn model routing algorithm on a 3-ary 2-mesh. Each node's table would consist of the row corresponding to its node identifier, with up to two possible outgoing links for each destination. Ejection is indicated by an *X*. By implementing a turn model routing algorithm in the per-node tables, some adaptivity can be achieved.

Compared to source routing, node-based routing requires smaller routing tables at each node. Each routing table needs to store only the routing information to select the next hop for each destination rather than the entire path. When multiple outputs are included per destination, node-based routing supports some adaptivity. Local information about congestion or faults can be used to bias the route selection to the non-congested link or to a non-faulty path. Node-based routing tables can also be programmable. By allowing the routing tables to be changed, the routing algorithm is better able to tolerate faults in the network.

The most significant downside to node routing tables is the increase in packet delay. Source routing requires a single look-up to acquire the entire routing path for a packet. With node-based routing, the latency of a look-up must be expended at each hop in the network.

Table 4.3: Table-based routing for a 3x3 mesh with West-First Turn Model Algorithm .

	To								
From	00	01	02	10	11	12	20	21	22
00	X -	N -	N -	E -	E N	E N	E -	N E	N E
01	S -	X -	N -	E S	E -	E N	E S	E -	E N
02	S -	S -	X -	E S	E S	E -	E S	E S	E -
10	W -	W -	W -	X -	N -	N -	E -	E N	E N
11	W -	W -	W -	S -	X -	N -	E S	E -	N E
12	W -	W -	W -	S -	S -	X -	E S	E S	E -
20	W -	W -	W -	W -	W -	W -	X -	N -	N -
21	W -	W -	W -	W -	W -	W -	S -	X -	N -
22	W -	W -	W -	W -	W -	W -	S -	S -	X -

4.6.3 COMBINATIONAL CIRCUITS

Alternatively, the message can encode the ordinates of the destination and use comparators at each router to determine whether to accept (eject) or forward the message. Simple routing algorithms are typically implemented as combinational circuits within the router due to the low overhead.

With source routing, the packet must contain space to carry all the bits needed to specify the entire path. Routing using combinational circuits requires only that the packet carry the destination identifier. The circuits required to implement the routing algorithm can be quite simple and executed with very low latency. An example circuit to compute the next hop based on current buffer occupancies in a 2-D mesh is shown in Figure 4.9. Alternatively, the route selection could implement dimension order routing rather than make a selection based on queue lengths.

By implementing the routing decision in combinational circuits, the algorithm is specific to one topology and one routing algorithm. The generality and configurability of table-based strategies are sacrificed. Despite the speed and simplicity of using a circuit to compute the next hop in the routing path, this computation adds latency to the packet traversal when compared to source-based routing. As with node-routing, the next output must be determined at each hop in the network. As will be discussed in Chapter 6, this routing computation can add a pipeline stage to the router traversal.

4.6.4 ADAPTIVE ROUTING

Adaptive routing algorithms need mechanisms to track network congestion levels, and update the route. Route adjustments can be implemented by modifying the header, by employing combinational circuitry that accepts as input these congestion signals, or by updating entries in a routing table. Many congestion sensitive mechanisms have been proposed, with the simplest being tapping into information that is already captured and used by the flow control protocol, such as buffer occupancy or credits [193, 50].

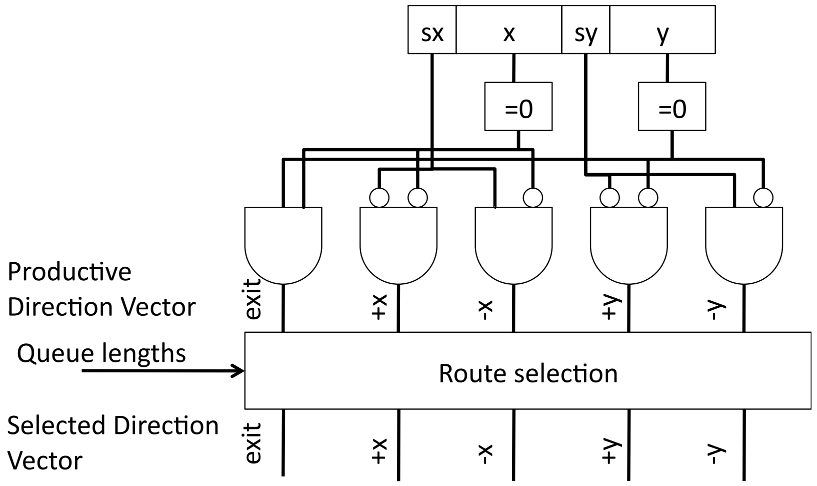


Figure 4.9: Combinational Routing Circuit for 2-D Mesh.

The primary benefit of increasing the information available to the routing circuitry is adaptivity. By improving the routing decision based on network conditions, the network can achieve higher bandwidth and reduce the congestion latency experienced by packets.

The disadvantage of such an approach is complexity. Additional circuitry is required for congestion-based routing decisions; this circuitry can increase the latency of a routing decision and the area of the router. Although the leveraging of information already available at the router is often done to make routing decisions, increasing the sophistication of the routing decision may require that additional information be communicated from adjacent routers. This additional communication could increase the network area and energy.

4.7 ROUTING ON IRREGULAR TOPOLOGIES

The discussion of routing algorithms in this chapter has assumed a regular topology such as a torus or a mesh. In the previous chapter, the potential for power and performance benefits of using irregular topologies for MPSoCs composed of heterogeneous nodes was explored. Irregular topologies can require special considerations in the development of a routing algorithm. Common routing implementations for irregular networks rely on source table routing or node-table routing [104, 34, 69]. Care must be taken when specifying routes so that deadlock is not induced. Turn model routing may not be feasible if certain connectivity is removed by the presence of oversized cores in a mesh network, for example.

4.8 BIBLIOGRAPHIC NOTES

4.8.1 CASE STUDIES

IBM Cell. Routing in the EIB is simple: just a choice of left or right on each of the four unidirectional rings. The ring arbiter will prevent allocations of transfers that are going more than halfway around the ring, i.e. only shortest path routes are permitted.

Intel TeraFLOPS. The network uses source-based routing, with each hop encoded as a 3-bit field corresponding to the 5 possible output ports a packet can take at each router. The packet format supports up to 10 hops (30 bits for route information), with a chained header bit that when set, indicates that routing information for more hops can be accessed in the packet data. Source routing enables the use of many possible oblivious or deterministic routing algorithms or even adaptive routing algorithms that chooses a route based on network congestion information at injection point. This enables the TeraFLOPS to tailor routes to specific applications.

Tilera TILE64. The four dynamic networks (UDN, IDN, MDN and TDN) use the dimension-ordered routing algorithm, with the destination address encoded in X-Y ordinates in the header. The static network (STN) allows the routing decision to be pre-set. This is achieved through circuit switching: a setup packet first reserves a specific route, the subsequent message then follows this route to the destination.

ST Microelectronics STNoC. The STNoC can be routed using regular routing algorithms that are identical at each node, leveraging the ring-like topology of the spidergon. For instance, the Across-First routing algorithm sends packets along the shortest paths, using the long across links that connect non-adjacent nodes in STNoC only when that gives the shortest paths, and only as the first hop. For instance, in Figure 4.10, when going from Node 0 to 4, packets will be routed from Node 0 to 3 on the long across link, then from 3 to 4 on the short link, leading to a 2-hop route. Note though that here, clearly, link length differs significantly and needs to be taken into account. Despite a low hop count, long link traversals cycles may increase the packet latency for Across-First routing. The Across-First routing algorithm is not deadlock-free, relying on the flow control protocol to ensure deadlock freedom instead.

STNoC routing is implemented through source routing, encoding just the across link turn and the destination ejection, since local links between adjacent rings are default routes. The Across-First algorithm can be implemented within the network interface controller either using routing tables or combinational logic.

4.8.2 BRIEF STATE-OF-THE-ART SURVEY

In this section, we provide a brief overview of current routing algorithm and implementation research in on-chip networks.

Algorithms. Many proposed on-chip network designs utilize dimension order routing for its simplicity and deadlock freedom, such as the MIT Raw [202], Intel TeraFLOPS, and Austin TRIPS chips [80]; however, other routing techniques have been proposed.

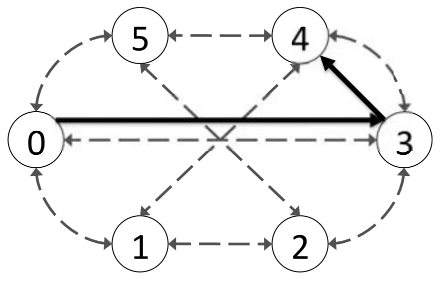


Figure 4.10: A 6 node Spidergon with the highlighted path from Node 0 to Node 4 using the Across-First routing algorithm.

Various oblivious routing algorithms [194, 191, 5] have been explored to push bandwidth without the added complexity of adaptive routing. Adaptive routing algorithms have also been investigated [117, 193, 79, 39], with various papers focusing on the implementation overhead of adaptive routing given the tight design constraints of on-chip networks [79]. Routing algorithms that dynamically switch between adaptive and deterministic have been proposed [97]. Æthereal employs source-based routing and relies on the turn model for deadlock freedom [78]. Flattened butterfly [113] is an example of an on-chip network that uses non-minimal routing to improve load balancing in the network; non-minimal routes are also employed in a bufferless on-chip network to prevent packets from being dropped [150]. Deflective routing [160] attempts to route packets to a free virtual channel along the minimal path but will misroute when this is not possible. Customized routing can be specified for application-specific designs with well understood communication patterns [98, 156].

Implementation. Various table-based implementations of routing algorithms have been explored for on-chip networks. Node table-based routing is proposed for regular topologies [186, 70], while table-based routing for irregular topologies have also been explored [69, 34, 104, 168].

This chapter has focused its discussion of routing algorithms on unicast routing, that is routing a packet from a single source to a single destination. Recent research has also explored the need to support multicast routing [66, 186, 2]. Both [66] and [186] leverage table-based routing implementations to achieve multicast routing.

CHAPTER 5
Flow Control

Flow control governs the allocation of network buffers and links. It determines when buffers and links are assigned to messages, the granularity at which they are allocated, and how these resources are shared among the many messages using the network. A good flow control protocol lowers the latency experienced by messages at low loads by not imposing high overhead in resource allocation, and drives up network throughput by enabling effective sharing of buffers and links across messages. In determining the rate at which packets access buffers (or skip buffer access altogether) and traverse links, flow control is instrumental in determining network energy and power consumption. The implementation complexity of a flow control protocol includes the complexity of the router microarchitecture as well as the wiring overhead required for communicating resource information between routers.

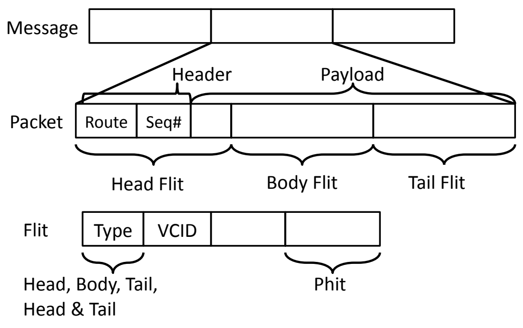
5.1 MESSAGES, PACKETS, FLITS AND PHITS

When a message is injected into the network, it is first segmented into packets, which are then divided into fixed-length flits, short for flow control units. For instance, a 128-byte cache line sent from a sharer to a requester will be injected as a message, and if the maximum packet size is larger than 128 bytes, the entire message will be encoded as a single packet. The packet will consist of a head flit that contains the destination address, body flits and a tail flit that indicates the end of a packet. Flits can be further broken down into phits, which are physical units and correspond to the physical channel width. The breakdown of messages to packets and packets to flits is depicted in Figure 5.1a. Head, body and tail flits all contain parts of the cache line and the cache coherence command. For instance, if flit size is 128 bits, the 128-byte packet will consist of 8 flits: 1 head, 6 body and 1 tail, ignoring the extra bits needed to encode the destination and other information needed by the flow control protocol. In short, a message is the logical unit of communication above the network, and a packet is the physical unit that makes sense to the network. A packet contains destination information while a flit may not, thus all flits of a packet must take the same route.

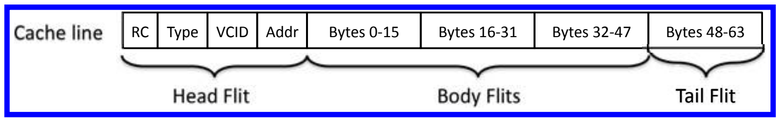
message
packet
flit

head, body and tail flits
phits

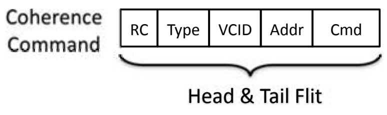
Due to the abundance of on-chip wiring resources, channels tend to be wider in on-chip networks, so messages are likely to consist of a single packet. In off-chip networks, channel widths are limited by pin bandwidth; this limitation causes flits to be broken down into smaller chunks called phits. To date, in on-chip networks, flits are composed of a single phit and are the smallest subdivision of a message due to wide on-chip channels. Additionally, as illustrated in Figure 5.1, many messages will in fact be single-flit packets. For example, a coherence command need only carry the command and the memory address which can fit in a 16-byte wide flit.



(a) Message Composition



(b) Cache Line Packet



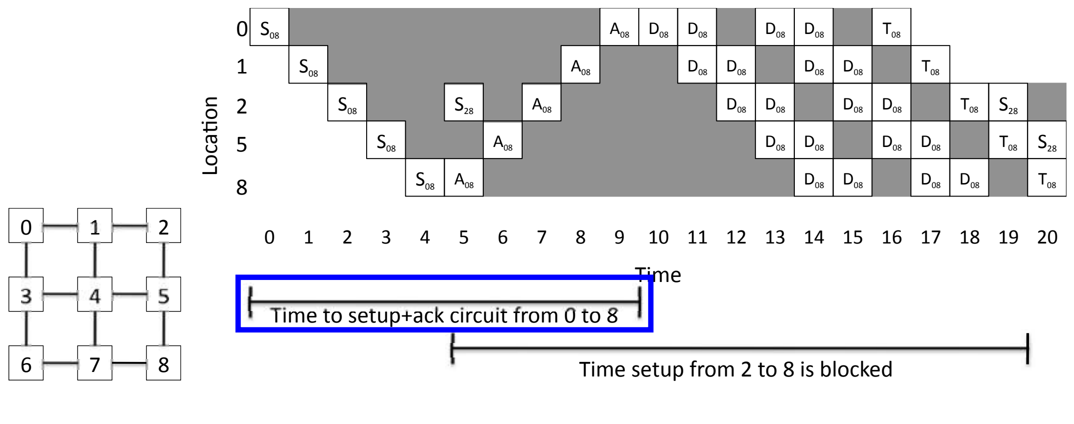
(c) Coherence Command Packet

Figure 5.1: Composition of Message, Packets, Flits: Assuming 16-byte wide flits and 64-byte cache lines, a cache line packet will be composed of 5 flits and a coherence command will be a single-flit packet. The sequence number (Seq#) is used to match incoming replies with outstanding requests, or to ensure ordering and detect lost packets.

Flow control techniques are classified by the granularity at which resource allocation occurs. We will discuss techniques that operate on message, packet and flit granularities in the next sections with a table summarizing the granularity of each technique at the end.

5.2 MESSAGE-BASED FLOW CONTROL

We start with circuit-switching, a technique that operates at the message-level, which is the coarsest granularity, and then refine these techniques to finer granularities.



Location
Time
Time to setup+ack circuit from 0 to 8
Time setup from 2 to 8 is blocked

(a) Network (b) Time-Space Diagram

Figure 5.2: Circuit-switching example from Core 0 to Core 8, with Core 2 being stalled. S: Setup flit, A: Acknowledgement flit, D: Data message, T: Tail (deallocation) flit. Each D represents a message; multiple messages can be sent on a single circuit before it is deallocated. In cycles 12 and 16, the source node has no data to send.

5.2.1 CIRCUIT SWITCHING

Circuit switching pre-allocates resources (links) across multiple hops to the entire message. A probe (a small setup message) is sent into the network and reserves the links needed to transmit the entire message (or multiple messages) from the source to the destination. Once the probe reaches the destination (having successfully allocated the necessary links), an acknowledgement message will be transmitted back to the source. When the source receives the acknowledgement message, it will release the message which can then travel quickly through the network. Once the message has completed its traversal, the resources are deallocated. After the setup phase, per-hop latency to acquire resources is avoided. With sufficiently large messages, this latency reduction can amortize the cost of the original setup phase. In addition to possible latency benefits, circuit switching is also bufferless. As links are pre-reserved, buffers are not needed at each hop to hold packets that are waiting for allocation, thus saving on power. While latency can be reduced, circuit switching suffers from poor bandwidth utilization. The links are idle between setup and the actual message transfer and other messages seeking to use those resources are blocked.

Figure 5.2 illustrates an example of how circuit-switching flow control works. Dimension order X-Y routing is assumed with the network shown in Figure 5.2a. As time proceeds from left to right (Figure 5.2b), the setup flit, *S* constructs a circuit from Core 0 to Core 8 by traversing the selected route through the network. At time 4, the setup flit has reached the destination and begins sending an acknowledgement flit, *A* back to Core 0. At time 5, Core 2 wants to initiate a

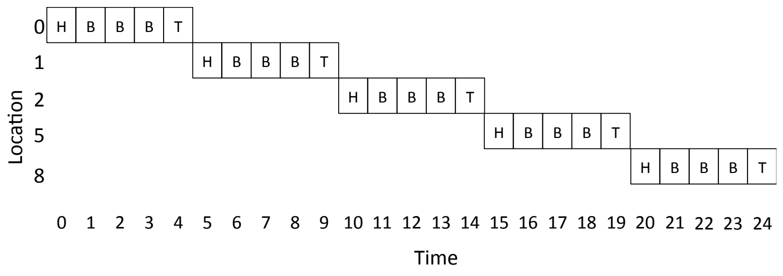


Figure 5.3: Store and Forward Example.

transfer to Core 8; however, the resources (links) necessary to reach Core 8 are already allocated to Core 0. Therefore, Core 2's request is stalled. At time 9, the acknowledgement request is received by Core 0 and the data transfers, *D* can begin. Once the required data are sent, a tail flit, *T* is sent by Core 0 to deallocate these resources. At time 19, Core 2 can now begin acquiring the resources recently deallocated by the tail flit. From this example, we see that there is significant wasted link bandwidth, during the setup time and when links have been reserved but there is no data that needs to be transmitted (wasted link bandwidth is shaded in grey). Core 2 also suffers significant latency waiting for resources that are mostly idle.

5.3 PACKET-BASED FLOW CONTROL

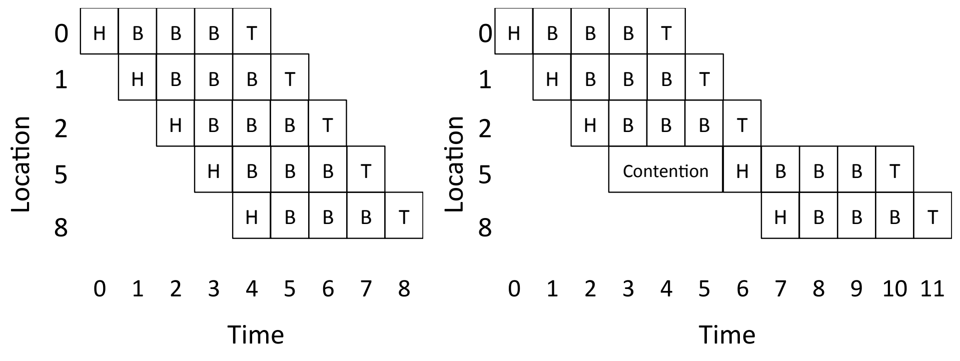
Circuit-switching allocates resources to messages and does so across multiple network hops. There are several inefficiencies to this scheme; next, we look at schemes that allocate resources to packets. Packet-based flow control techniques first break down messages into packets, then interleave these packets on the links, thus improving link utilization. Unlike circuit switching, the remaining techniques will require per-node buffering to store in-flight packets.

5.3.1 STORE AND FORWARD

With packet-based techniques, messages are broken down into multiple packets and each packet is handled independently by the network. In *store-and-forward* flow control [47], each node waits until an entire packet has been received before forwarding any part of the packet to the next node. As a result, long delays are incurred at each hop, which makes them unsuitable for on-chip networks that are usually delay-critical. Moreover, store and forward flow control requires that there be sufficient buffering at each router to buffer the entire packet. These high buffering requirements reduce store and forward switching's amenability to on-chip networks.

In Figure 5.3, we depict a packet traveling from Core 0 to Core 8 using store and forward switching. Once the tail flit has been buffered at each node, the head can then allocate the next link and depart for the next router. Serialization delay is paid for at each hop for the body and tail flits to

serialization delay



(a) VCT with no delay (b) VCT with delay

Figure 5.4: Virtual Cut Through Example.

catch up with the head flit. For a 5-flit packet, the latency is 5 cycles to transmit the packet at each hop.

5.3.2 CUT-THROUGH

To reduce the delay packets experience at each hop, *virtual cut-through* flow control [110] allows transmission of a packet to proceed to the next node before the entire packet is received at the current router. Latency experienced by a packet is thus drastically reduced over store and forward flow control, as shown in Figure 5.4a. In Figure 5.3, 25 cycles are required to transmit the entire packet; with virtual cut-through, this delay is reduced to 9 cycles. However, bandwidth and storage are still allocated in packet-sized units. Packets still move forward only if there is enough storage at the next downstream router to hold the entire packet. On-chip networks with tight area and power constraints may find it difficult to accommodate the large buffers needed to support virtual cut-through when packet sizes are large (such as 64- or 128-byte cache lines).

In Figure 5.4b, the entire packet is delayed when traveling from node 2 to node 5 even though node 5 has buffers available for 2 out of 5 flits. No flits can proceed until all 5 flit buffers are available.

5.4 FLIT-BASED FLOW CONTROL

To reduce the buffering requirements of packet-based techniques, flit-based flow control mechanisms exist. Low buffering requirements help routers meet tight area or power constraints on-chip.

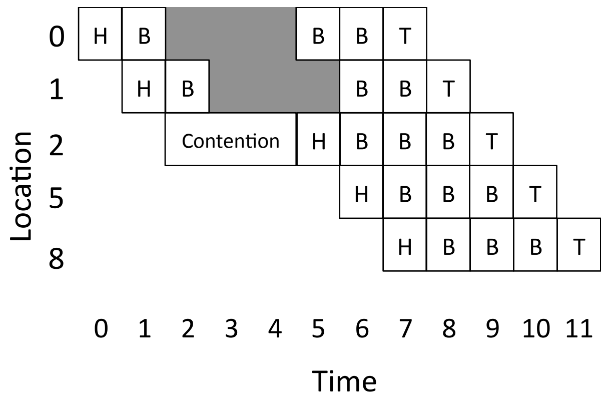


Figure 5.5: Wormhole Example.

5.4.1 WORMHOLE

Like virtual cut-through flow control, *wormhole* flow control [54] cuts through flits, allowing flits to move on to the next router before the entire packet is received at the current location. For wormhole flow control, the flit can depart the current node as soon as there is sufficient buffering for this flit. However, unlike store-and-forward and virtual cut-through flow control, wormhole flow control allocates storage and bandwidth to flits rather than entire packets. This allows relatively small flit-buffers to be used in each router, even for large packet sizes. While wormhole flow control uses buffers effectively, it makes inefficient use of link bandwidth. Though it allocates storage and bandwidth in flit-sized units, a link is held for the duration of a packet's lifetime in the router. As a result, when a packet is blocked, all of the physical links held by that packet are left idle. Since wormhole flow control allocates buffers on a flit granularity, a packet composed of many flits can potentially span several routers, which will result in many idle physical links. Throughput suffers because other packets queued behind the blocked packet are unable to use the idle physical links.

In the example in Figure 5.5, each router has 2 flit buffers. When the head flit experiences contention traveling from 1 to 2, the remaining two body and tail flits are stalled at Core 0 since there is no buffer space available at Core 1 until the head moves to Core 2. However, the channel is still held by the packet even though it is idle as shown in grey.

Wormhole flow control reduces packet latency by allowing a flit to leave the router as soon as a downstream buffer is available (in the absence of contention, the latency is the same as virtual cut through). Additionally, wormhole flow control requires much smaller buffers than packet-based techniques. Due to the tight area and power constraints of on-chip networks, wormhole flow control is the predominant technique adopted thus far.

5.5 VIRTUAL CHANNELS

Virtual channels have been explained as the 'swiss-army knife' of interconnection networks [47]. They were first proposed as a solution for deadlock avoidance [48], but has also been applied to skirt around head-of-line blocking in flow control, thus extending throughput. Head-of-line blocking occurs in all the above flow control techniques where there is a single queue at each input; when a packet at the head of the queue is blocked, it can stall subsequent packets that are lined up behind it, even when there are available resources for the stalled packets.

head-of-line blocking

Essentially, a virtual channel (VC) is basically a separate queue in the router; multiple VCs share the physical wires (physical link) between two routers. By associating multiple separate queues with each input port, head-of-line blocking can be reduced. Virtual channels arbitrate for physical link bandwidth on a cycle-by-cycle basis. When a packet holding a virtual channel becomes blocked, other packets can still traverse the physical link through other virtual channels. Thus VCs increase the utilization of the physical links and extend overall network throughput.

virtual channel

VCs can technically be applied to all the above flow control techniques to alleviate head-of-line blocking, though Dally first proposed it with wormhole flow control [48]. For instance, circuit switching can be applied on virtual channels rather than the physical channel, so a message reserves a series of VCs rather than physical links, and the VCs are time-multiplexed onto the physical link cycle-by-cycle, also called virtual circuit switching [73]. Store-and-forward flow control can also be used with VCs, with multiple packet buffer queues, one per VC, VCs multiplexed on the link packet-by-packet. Virtual cut-through flow control with VCs work similarly, except that VCs are multiplexed on the link flit-by-flit. However, as on-chip network designs overwhelmingly adopt wormhole flow control for its small area and power footprint, and use virtual channels to extend the bandwidth where needed, for the rest of this lecture, when we mention virtual channel flow control, we assume that it is applied to wormhole, with both buffers and links managed and multiplexed at the granularity of flits.

A walk-through example illustrating the operation of virtual channel flow control is depicted in Figure 5.6. Packet A initially occupies VC 0 and is destined for Node 4, while Packet B initially occupies VC 1 and is destined for Node 2. At time 0, Packet A and Packet B both have flits waiting in the west input virtual channels of Node 0. Both A and B want to travel outbound on the east output physical channel. The head flit of Packet A is allocated virtual channel 0 for the west input of router 1 and wins switch allocation (techniques to handle this allocation are discussed in Chapter 6). The head flit of packet A travels to router 1 at time 1. At time 2, the head flit of packet B is granted switch allocation and travels to router 1 and is stored in virtual channel 1. Also at time 2, the head flit of A fails to receive a virtual channel for router 4 (its next hop); both virtual channels are occupied by flits of other packets. The first body flit of A inherits virtual channel 0 and travels to router 1 at time 3. Also at time 3, the head flit of B is able to allocate virtual channel 0 at router 2 and continues on. At time 4, the first body flit of packet B inherits virtual channel 0 from the head flit and wins switch allocation to continue to router 2. By time 6, all of the flits of B have arrived at router 2, the

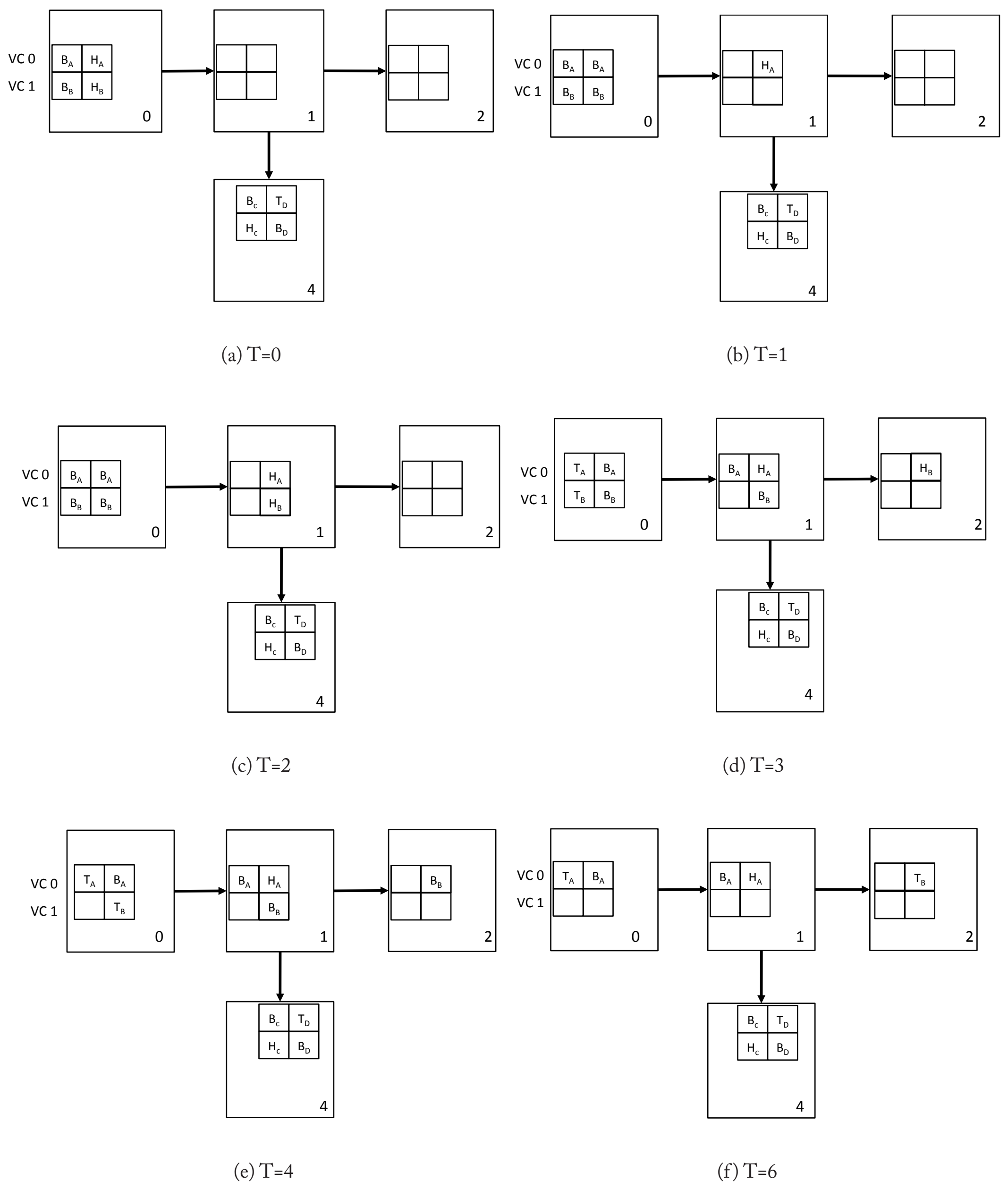


Figure 5.6: Virtual Channel Flow Control Walk-through Example. Two packets A and B are broken into 4 flits each (H: head, B: Body, T: Tail).

Table 5.1: Summary of flow control techniques.

	Links	Buffers	Comments
Circuit-switching	Messages	N/A (buffer-less)	Requires setup & acknowledgment
Store and Forward	Packet	Packet	Head flit must wait for entire packet before proceeding on next link
Virtual Cut Through	Packet	Packet	Head can begin next link traversal before tail arrives at current node
Wormhole	Packet	Flit	Head of line blocking reduces efficiency of link bandwidth
Virtual Channel	Flit	Flit	Can interleave flits of different packets on links

head and body flits have continued on and the tail flit remains to be routed. The head flit of packet A is still blocked waiting for a free virtual channel to travel to router 4.

With wormhole flow control using a single virtual channel, packet B would be blocked behind packet A at router 1 and would not be able to continue to router 2 despite the availability of buffers, links and the switch to do so. Virtual channels allow packet B to proceed towards its destination despite the blocking of packet A. Virtual channels are allocated once at each router to the head flit and the remainder of flits inherit that virtual channel. With virtual-channel flow control, flits of different packets can be interleaved on the same physical channel, as seen in the example between time 0 and 2.

Virtual channels are also widely used to break deadlocks, both within the network (see Section 5.6), and for handling system-level or protocol-level deadlocks (see Section 2.1.3).

The previous sections have explained how different techniques handle resource allocation and utilization. These techniques are summarized in Table 5.1.

5.6 DEADLOCK-FREE FLOW CONTROL

Deadlock freedom can be maintained either through the use of constrained routing algorithms that ensure no cycles ever occur (see Chapter 4), or through the use of deadlock-free flow control which allows any routing algorithm to be used.

Figure 5.7 illustrates how two virtual channels can be used to break a cyclic deadlock in the network when the routing protocol permits a cycle. Here, since each VC is associated with a separate buffer queue, and every VC is time-multiplexed onto the physical link cycle-by-cycle, holding onto a VC implies holding onto its associated buffer queue rather than locking down a physical link. By

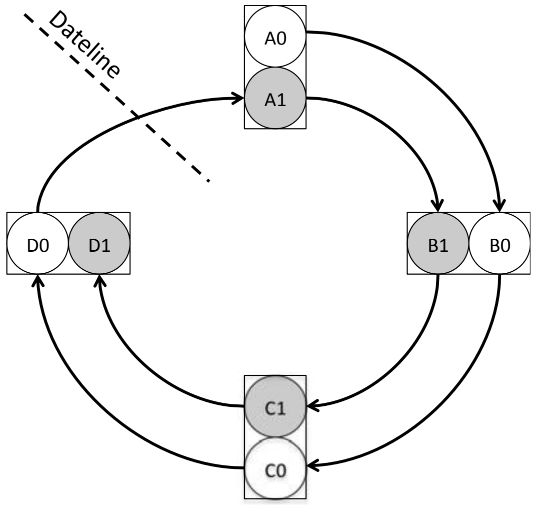


Figure 5.7: Two virtual channels with separate buffer queues denote with white and grey circles at each router are used to break the cyclic route deadlock in Figure 4.2.

enforcing an order on VCs, so that lower-priority VCs cannot request and wait for higher-priority VCs, there cannot be a cycle in resource usage. In Figure 5.7, all messages are sent through VC 0 until they cross the dateline. After crossing the dateline, messages are assigned to VC 1 and cannot be allocated to VC 0 at any point during the remainder of their network traversal [55].

dateline

At the system level, messages that can potentially block each other can be assigned to different message classes that are mapped to different virtual channels within the network, such as request and acknowledgment messages of coherence protocols. Implementation complexity of virtual channel routers will be discussed in detail next in Chapter 6 on router microarchitecture.

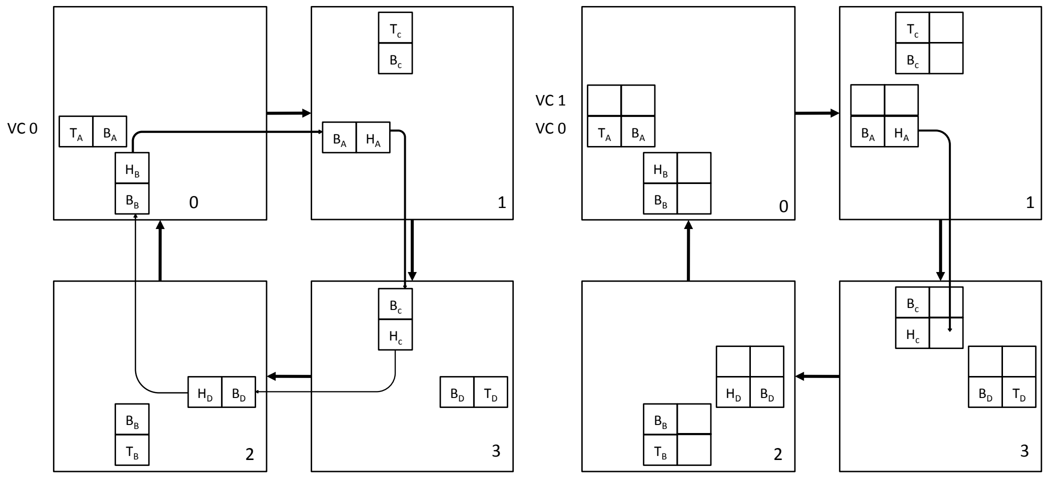
5.6.1 ESCAPE VCS

The previous section discussed the benefits of enforcing VC-ordering to prevent deadlocks. However, enforcing an order on VCs lowers their utilization, affecting network throughput when the number of VCs is small. In Figure 5.7, all packets are initially assigned VC 0 and remain on VC 0 until they cross the dateline. As a result, VC 1 is underutilized. A technique called Escape VCs is proposed to address this [61]: rather than enforcing a fixed order/priority between all VCs, it was proved that so long as there is a single escape VC that is deadlock-free, all other VCs can be used arbitrarily. This escape VC is typically made deadlock-free by routing it with a deadlock-free routing function. For instance, if VC 0 is designated as the escape channel, all traffic on VC 0 must be routed using dimension-ordered routing, while all other VCs can be routed with arbitrary routing functions. Explained simply, so long as access to VCs is arbitrated fairly, a packet always has a chance of

escape VC

landing on the escape VC, and thus of escaping a deadlock. This increases the utilization of VCs, or permits a higher throughput with a smaller number of VC, making for leaner routers.

In Figure 5.8a, we illustrate once again how unrestricted routing with a single virtual channel can lead to deadlock. Each packet is trying to acquire resources to make a clockwise turn. Figure 5.8b utilizes two virtual channels. Virtual channel 1 serves as an escape virtual channel. For example, Packet A could be allocated virtual channel 1 (and thus dimension order routed to its destination). By allocating virtual channel 1 at router 4 for packet A, all packets can make forward progress. The flits of packet A will eventually drain from VC 0 at router 1, allowing packet B to be allocated either virtual channel at router 1. Once the flits of packet B have drained, packet D can continue on virtual channel 0 or be allocated to virtual channel 1 and make progress before packet B has drained. The same goes for the flits of packet C.



(a) Deadlock without Escape VCs

(b) Escape VCs break deadlock cycle

Figure 5.8: Escape Virtual Channel Example. Virtual Channel 1 serves as an escape virtual channel that is dimension order XY routed.

5.7 BUFFER BACKPRESSURE

As most on-chip network designs cannot tolerate the dropping of packets, there must be buffer backpressure mechanisms for stalling flits. Flits must not be transmitted when the next hop will not have buffers available to house them. The unit of buffer backpressure depends on the specific flow control protocol; store-and-forward and virtual cut-through flow control techniques manage buffers in units of packets, while wormhole and virtual channel flow control manage buffers in units of flits.

Circuit switching, being a bufferless flow control technique, does not require buffer backpressure mechanisms. Two commonly used buffer backpressure mechanisms are credits and on/off signaling.

credit backpressure

Credit-based. Credits keep track of the number of buffers available at the next hop, by sending a credit to the previous hop when a buffer is vacated (when a flit/packet leaves the router), and incrementing the credit count at the previous hop upon receiving the credit. When a flit departs the current router, the current router decrements the credit count for the appropriate downstream buffer.

on-off backpressure

On/off. On/off signaling involves a signal between adjacent routers that is turned off to stop the previous hop from transmitting flits when the number of buffers drop below a threshold. This threshold must be set to ensure that all in-flight flits will have buffers upon arrival. Buffers must be available for flits departing the current router during the transmission latency of the off-signal. When the number of free buffers at the downstream router rises above a threshold, the signal is turned on and flit transmission can resume. The on threshold should be selected so that the next router will still have flits to send to cover the time of transmission of the on signal plus the delay to receive a new flit from the current router.

5.8 IMPLEMENTATION

The implementation complexity of a flow control protocol essentially involves the complexity of the entire router microarchitecture as well as the wiring overhead imposed in communicating resource information between routers. Here, we focus on the latter, as Chapter 6 elaborates on router microarchitectures and associated implementation issues.

When choosing a specific buffer backpressure mechanism, we need to consider its performance in terms of buffer turnaround time, and its overhead in terms of the number of reverse signaling wires.

5.8.1 BUFFER SIZING FOR TURNAROUND TIME

buffer turnaround time

Buffer turnaround time is the minimum idle time between when successive flits can reuse a buffer. A long buffer turnaround time leads to inefficient reuse of buffers, which results in poor network throughput. If the number of buffers implemented does not cover the buffer turnaround time, then the network will be artificially throttled at each router, since flits will not be able to flow continuously to the next router even when there is no contention from other ports of the router. As shown in Figure 5.9, the link between two routers is idle for 6 cycles while waiting for a free buffer at the downstream router.

For credit-based buffer backpressure, a buffer is held from the time a flit departs the current node (when the credit counter is decremented), to the time the credit is returned to inform the current node that the buffer has been released (so the credit counter can be incremented again). Only then can the buffer be allocated to the next flit, although it is not actually reused until the flit traverses the current router pipeline and is transmitted to the downstream router. Hence, the

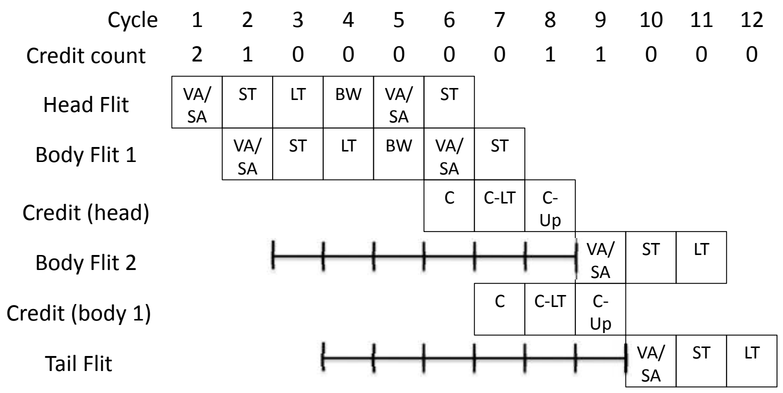


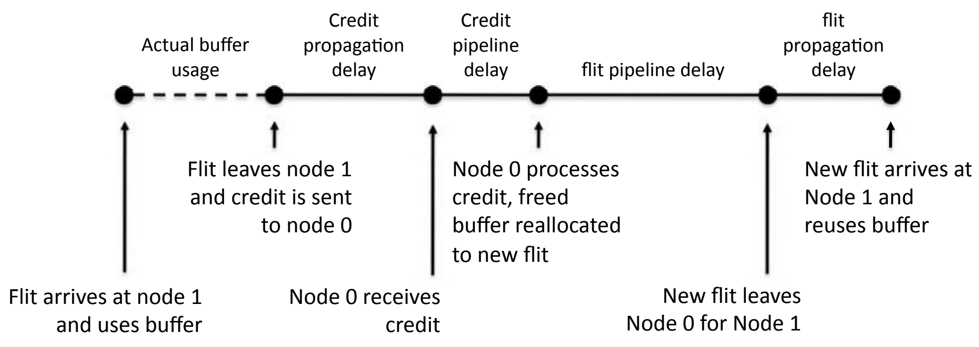
Figure 5.9: Throttling due to too few buffers. Flit pipeline stages discussed in Chapter 6. C: Credit send. C-LT: Credit Link Traversal. C-Up: Credit update.

turnaround time of a buffer is at least the sum of the propagation delay of a data flit to the next node, the credit delay back, and the pipeline delay, as is shown in Figure 5.10a.

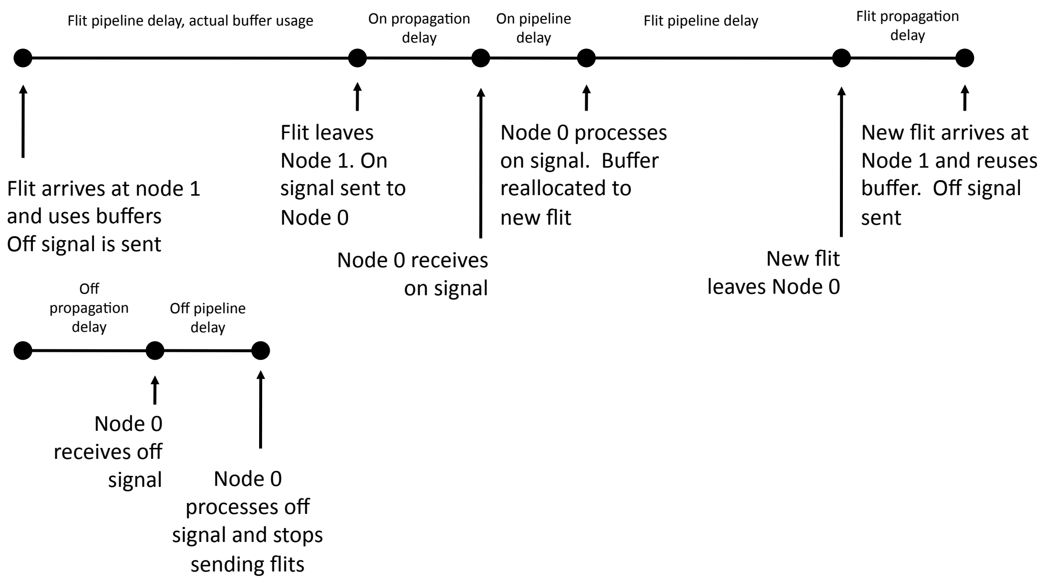
In comparison, in on/off buffer backpressure, a buffer is held from the time a flit arrives at the next node and occupies the last buffer (above the threshold), triggering the off signal to be sent to stop the current node from sending. This persists until a flit leaves the next node and frees up a buffer (causing the free buffer count to go over the threshold). Consequently, the on signal is asserted, informing the current node that it can now resume sending flits. This buffer is occupied again when the data flit arrives at the next node. Here, the buffer turnaround time is thus at least twice the on/off signal propagation delay plus the propagation delay of a data flit, and the pipeline delay, as shown in Figure 5.10b.

If we have a 1 cycle propagation delay for both data flits and reverse signaling between adjacent nodes, a 1-cycle pipeline delay for buffer backpressure signals, and a 3-cycle router pipeline, then credit-based backpressure will have a buffer turnaround time of at least 6 cycles, while on/off backpressure will have a buffer turnaround time of at least 9 cycles. Note that this implies that this network using on/off backpressure needs at least 9 buffers per port to cover the turnaround time, while if it chooses credit-based backpressure, it needs 3 fewer buffers per port. Thus, buffer turnaround time also affects the area overhead, since buffers take up a substantial portion of a router's footprint.

Note that it is possible to optimize the buffer turnaround time by triggering the backpressure signals (credits or on/off) once it is certain a flit will depart a router and no longer need its buffer, rather than waiting until the flit has actually been read out of the buffer.


Actual buffer usage
Credit propagation delay
Credit pipeline delay
flit pipeline delay
flit propagation delay
Flit leaves node 1 and credit is sent to node 0
Node 0 processes credit, freed buffer reallocated to new flit
New flit arrives at Node 1 and reuses buffer
Flit arrives at node 1 and uses buffer
Node 0 receives credit
New flit leaves Node 0 for Node 1

(a) Credit-based

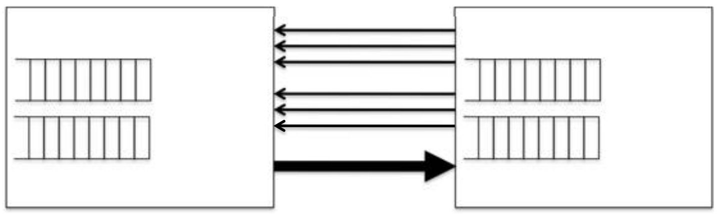

Flit pipeline delay, actual buffer usage
On propagation delay
On pipeline delay
Flit pipeline delay
Flit propagation delay
Flit arrives at node 1 and uses buffers Off signal is sent
Flit leaves Node 1. On signal sent to Node 0
Node 0 receives on signal
Node 0 processes on signal. Buffer reallocated to new flit
New flit leaves Node 0
New flit arrives at Node 1 and reuses buffer. Off signal sent
Off propagation delay
Off pipeline delay
Node 0 receives off signal
Node 0 processes off signal and stops sending flits

(b) On/Off-based

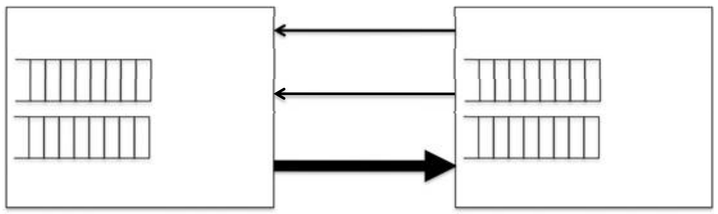
Figure 5.10: Buffer backpressure mechanisms time lines.

5.8.2 REVERSE SIGNALING WIRES

While on/off backpressure performs poorly compared to credit-based backpressure, it has lower overhead in terms of reverse signaling overhead. Figure 5.11 illustrates the number of reverse signaling wires needed for both backpressure mechanisms: credit-based requires log_B wires per queue



(a) Credit-based



(b) On/Off based

Figure 5.11: Reverse signalling overhead.

(virtual channel), where B is the number of buffers in the queue, to encode the credit count. On the other hand, on/off needs only a single wire per queue. With eight buffers per queue and two virtual channels, credit-based backpressure requires six reverse signalling wires (Figure 5.11a) while on/off requires just two reverse wires (Figure 5.11b).

In on-chip networks, where there is abundant on-chip wiring, reverse signaling overhead tends to be less of a concern than area overhead and throughput. Hence, credit-based backpressure will be more suitable.

5.9 FLOW CONTROL IMPLEMENTATION IN MPSOCS

MPSoCs typically rely on wormhole flow control for the same reasons that it has been adopted for more general purpose on-chip networks. Applications that run on MPSoCs often have real-time performance requirements. Such quality of service requirements can impact flow control design decisions. The network interface controller can regulate traffic injected into the network to reduce contention and ensure fairness [160, 206]. The use of time division multiplexing (TDM)that allocates a fixed amount of bandwidth to each node is one way to provide guaranteed throughput and avoid contention [78]. Time division multiplexing schedules communications so that each communication flow has their own time slot on network links. The size and number of time slots implemented governs

time division multiplexing (TDM)

the granularity at which network resources can be allocated. As the TDM slots are allocated a priori, deviation in actual bandwidth demands will lead to idling channels.

A custom network for an MPSoC may result in a heterogeneous set of switches; these switches may differ in terms of number of ports, number of virtual channels and number of buffers [30]. In terms of flow control implementation, different numbers of buffers may be instantiated at each node depending on the communication characteristics [99]. Buffering resources will impact the thresholds of on/off flow control or the reverse signalling wires required by credit-based flow control. Additionally, non-uniform link lengths in a customized topology will impact the buffer turn-around time of the flow control implementation. Regularity and modularity are sacrificed in this type of environment; however the power, performance and area gains can be significant.

5.10 BIBLIOGRAPHIC NOTES

5.10.1 CASE STUDIES

IBM Cell. As the Cell interfaces with the EIB through DMA bus transactions, the unit of communications is large DMA transfers in bursts, with flow control semantics of buses rather than packetized networks. Resource allocation (access to rings) is guarded by the ring arbiter, with highest priority given to the memory controller so requestors will not be stalled on read data. Other elements on the EIB have equal priority and are served in a round-robin manner.

Intel TeraFLOPS. The TeraFLOPS has a minimum packet size of two flits (38-bit flits comprised of 6 bits of control data and 32 bits of data), with no limits placed on the maximum packet size by the router architecture. The network uses wormhole flow control with two virtual channels, though the virtual channels are used only to avoid system-level deadlock, and not for flow control, as packets are pinned to a VC throughout their network traversal. This simplifies the router design since no VC allocation needs to be done at each hop. Buffer backpressure is maintained using on/off signaling, with software programmable thresholds.

Tilera TILE64. The iMesh's four dynamic networks use simple wormhole flow control without virtual channels to lower the complexity of the routers, trading off the lower bandwidth of wormhole flow control by spreading traffic over multiple networks. Credit-based flow control is used. The static network uses circuit switching to enable the software to pre-set arbitrary routes while enabling fast delivery for the subsequent data transfer; the setup delay is amortized over long messages.

ST Microelectronics STNoC. STNoC uses wormhole flow control, supporting flit sizes ranging from 16 to 512 bits depending on the bandwidth requirements of the application. Virtual channels are used to break deadlocks, using a dateline approach similar to what has been discussed, with the variant that nodes that do not route past the dateline need not be constrained to a specific VC, but can instead use all VCs. Buffer backpressure is tracked using credits. Actual flit size, number of buffers, virtual channels are determined for each application-specific design through design space exploration.

5.10.2 BRIEF STATE-OF-THE-ART SURVEY

Wormhole flow control is widely assumed in on-chip network proposals and chips thus far, e.g. MIT RAW [202], Tilera TILE64 [212], Intel 80-core TeraFLOPS [95], Austin TRIPS [80] and the SPIN network [12]. Layered switching [138] hybridizes wormhole and virtual cut-through flow control by allocating resources to groups of data words which are larger than flits but smaller than packets. The TRIPS operand network (OPN) has single flit packets, so naturally uses flit-based wormhole flow control rather than packet-based ones [81]. Additionally, some designs use multiple virtual channels to boost the bandwidth of wormhole flow control (e.g. Intel TeraFLOPS [95], Austin TRIPS [80]), while others use multiple networks instead (RAW [202], Tilera TILE64 [212]). Intel's Larrabee [190] uses message-based flow control; once a message is injected into the network no storage is required at the routers.

Asynchronous transfer mode (ATM) [60] establishes virtual circuit connections; before data can be sent, network resources must be reserved from source to destination (like circuit switching); however, data are switched through the network at a packet granularity rather than a message granularity.

Research into flow control for on-chip networks have targeted the unique opportunities and constraints in on-chip networks. For instance, several works leverage the availability of upper metal layers and heterogeneous interconnect sizing to harness links of different speeds and bandwidths in flow control [173, 20]. Others target the tight power constraint faced by on-chip networks, and propose flow control protocols that allow flits to bypass through router pipelines, lowering dynamic switching power at routers, while improving buffer turnaround time and latency-throughput [127, 126], or dynamically scale down the number of VCs needed to support the traffic demands [159]. The overheads of fault tolerance for various flow control strategies have been explored [182]. Elastic buffer flow control [145] uses pipelined channels for buffering, eliminating the need for virtual channel buffers. Several proposals aim to achieve the benefits of both circuit and packet switching [65, 214, 213].

CHAPTER 6

Router Microarchitecture

Routers must be designed to meet latency and throughput requirements amidst tight area and power constraints; this is a primary challenge designers are facing as many-core systems scale. As router complexity increases with bandwidth demands, very simple routers (unpipelined, wormhole, no VCs, limited buffering) can be built when high throughput is not needed, so area and power overhead is low. Challenges arise when the latency and throughput demands on on-chip networks are raised.

A router's architecture determines its critical path delay which affects per-hop delay and overall network latency. Router microarchitecture also impacts network energy as it determines the circuit components in a router and their activity. The implementation of the routing, flow control and the actual router pipeline will affect the efficiency at which buffers and links are used and thus overall network throughput. The area footprint of the router is clearly determined by the chosen router microarchitecture and underlying circuits.

6.1 VIRTUAL CHANNEL ROUTER MICROARCHITECTURE

Fig. 6.1 shows the microarchitecture of a state-of-the-art credit-based virtual channel (VC) router to explain how typical routers work. The example assumes a 2-dimensional mesh, so the router has five input and output ports corresponding to the four neighboring directions and the local processing element (PE) port. The major components which constitute the router are the input buffers, route computation logic, virtual channel allocator, switch allocator, and the crossbar switch. Most on-chip network routers are input-buffered, in which packets are stored in buffers only at the input ports, as input buffering permits the use of single-ported memories. Here, we assumed four VCs at each input port, each with its own buffer queue that is four flits deep.

input buffered

The buffers are responsible for storing flits when they enter the router, and housing them throughout their duration in the router. This is in contrast to a processor pipeline that latches instructions in buffers between each pipeline stage. If source routing is not used, the route computation block will compute (or lookup) the correct output port for this packet. The allocators (virtual channel and switch) determine which flits are selected to proceed to the next stage where they traverse the crossbar. Finally, the crossbar switch is responsible for physically moving flits from the input port to the output port.

6.2 PIPELINE

Fig. 6.2a shows the corresponding logical pipeline stages for this basic virtual channel router (BASE). Like the logical pipeline stages of a typical processor: instruction fetch, decode, execute, memory

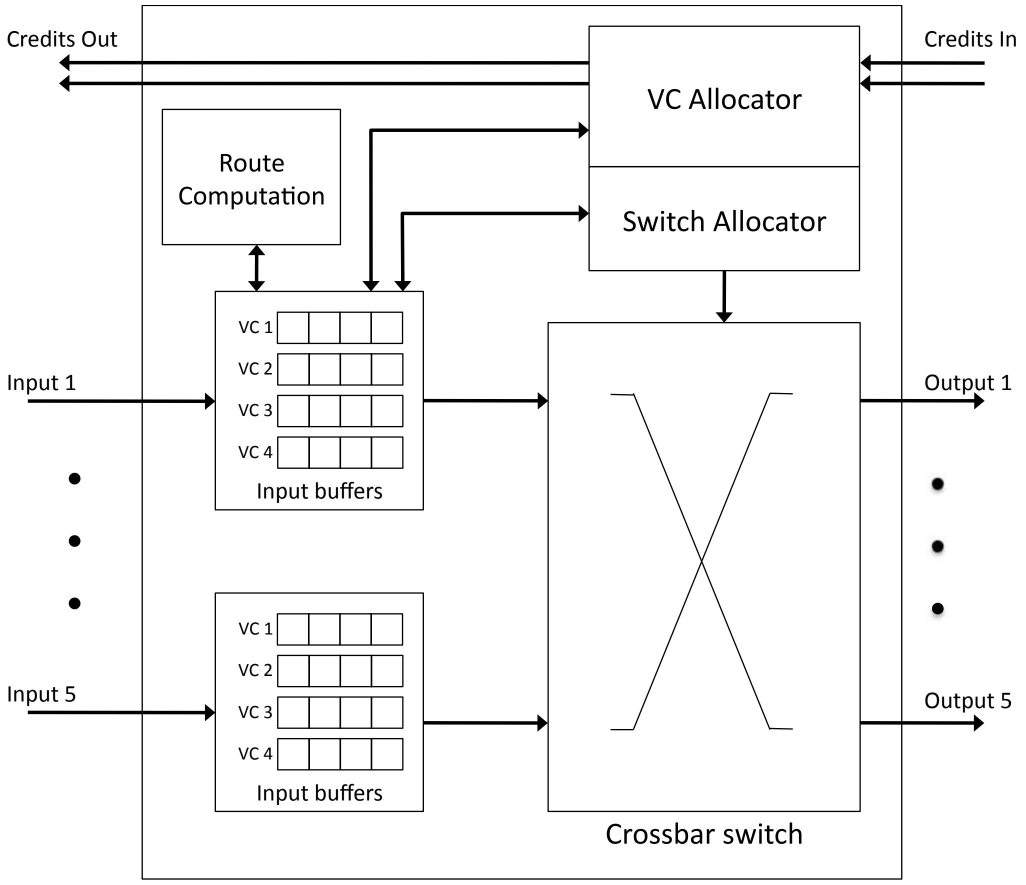
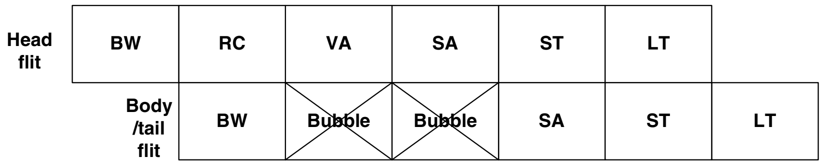


Figure 6.1: A credit-based virtual channel router microarchitecture.

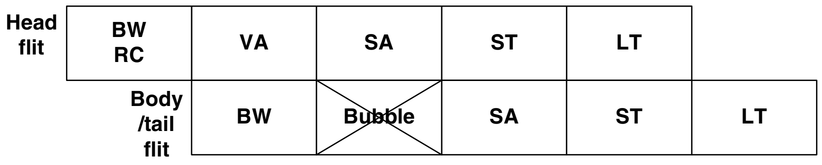
and writeback, these are logical stages that will fit into a physical pipeline depending on the actual clock frequency.

A head flit, upon arriving at an input port, is first decoded and buffered according to its input VC in the buffer write (BW) pipeline stage. In the next stage, the routing logic performs route computation (RC) to determine the output port for the packet. The header then arbitrates for a VC corresponding to its output port in the VC allocation (VA) stage. Upon successful allocation of a VC, the header flit proceeds to the switch allocation (SA) stage where it arbitrates for the switch input and output ports. On winning the output port, the flit is then read from the buffer and proceeds to the switch traversal (ST) stage, where it traverses the crossbar. Finally, the flit is passed to the next node in the link traversal (LT) stage. Body and tail flits follow a similar pipeline except that they do

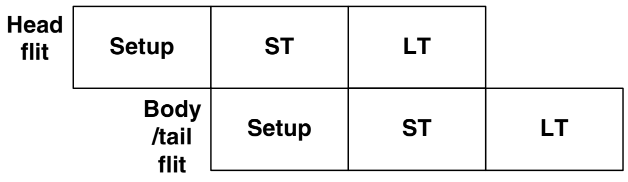
buffer write
route computation
VC allocation
switch allocation
switch traversal
link traversal



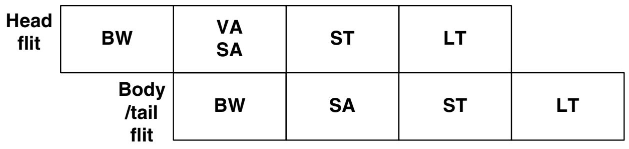
(a) Basic 5-stage pipeline (BASE)



(b) Lookahead pipeline (BASE+LA)



(c) Bypassing pipeline (BASE+LA+BY)



(d) Speculation pipeline (BASE+LA+BY+SPEC)

Figure 6.2: Router pipeline [BW: Buffer Write, RC: Route Computation, VA: Virtual Channel Allocation, SA: Switch Allocation, ST: Switch Traversal, LT: Link Traversal].

not go through RC and VA stages, instead inheriting the route and the VC allocated by the head flit. The tail flit, on leaving the router, deallocates the VC reserved by the head flit.

A wormhole router with no VCs does away with the VA stage, requiring just four logical stages. In Figure 6.1, such a router will not require a VC allocator, and will have only a single buffer queue in each input port.

6.2.1 PIPELINE IMPLEMENTATION

This logical virtual channel pipeline consists of 5 stages. A router that is running at a low clock frequency will be able to fit all 5 stages into a single clock cycle. For aggressive clock frequencies, the router architecture must be pipelined. The actual physical pipeline depends on the implementation of each of these logical stages and their critical path delay. If the physical pipeline has five stages just like the logical stages, then the stage with the longest critical path delay will set the clock frequency. Typically, this is the VC allocation stage when the number of VCs is high, or the crossbar traversal stage with very wide, highly-ported crossbars. The clock frequency can also be determined by the overall system clock, for instance sized by the processor pipeline's critical path instead.

Increasing the number of physical pipeline stages increases the per-hop router delay for each message, as well as the buffer turnaround time which affects the minimum buffering needed and affects throughput. Thus, pipeline optimizations have been proposed and employed to reduce the number of stages. Common optimizations targeting logical pipeline stages are explained next.

6.2.2 PIPELINE OPTIMIZATIONS

lookahead routing

To remove the RC stage from the critical path, prior work has proposed *lookahead routing* [71] where the route of a packet is determined one hop in advance. Precomputing the route enables flits to compete for VCs immediately after the BW stage, with the RC stage essentially responsible for just the decoding of the route header encoded within the head flit. The routing computation needed at the next hop can be computed in parallel with VC allocation since it is no longer needed to determine which VCs to allocate. Figure 6.2b shows the router pipeline with lookahead routing (BASE+LA).

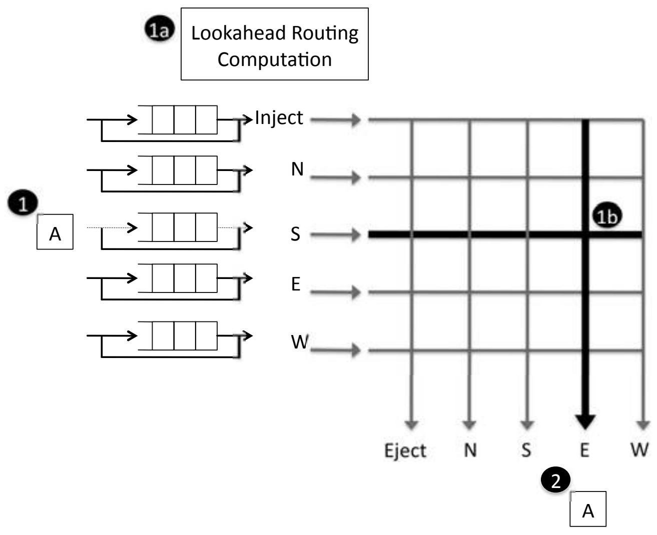
pipeline bypassing

Pipeline bypassing is another technique that is commonly used to further shorten the router critical path by allowing a flit to speculatively enter the ST stage if there are no flits ahead of it in the input buffer queue. Figure 6.2c shows the pipeline where a flit goes through a single stage of switch setup, during which the crossbar is set up for flit traversal in the next cycle while simultaneously allocating a free VC corresponding to the desired output port, followed by ST and LT (BASE+LA+BY). Upon a port conflict, the allocation is aborted and the flit is written into the buffer (BW).

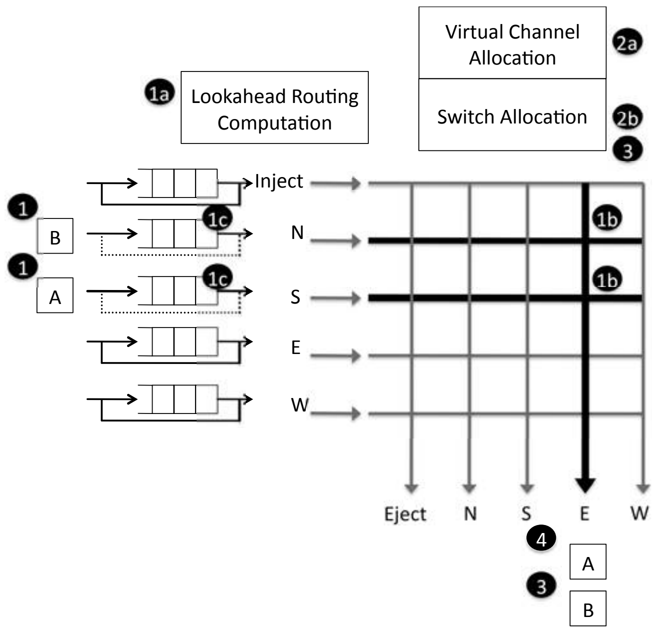
aggressive speculation

When the router ports are busy, thereby disallowing pipeline bypassing, aggressive *speculation* can be used to cut down the critical path [151, 153, 175]. Figure 6.2(d) shows the router pipeline where VA and SA take place in parallel in the same cycle (BASE+LA+BY+SPEC). A flit enters the SA stage speculatively after BW and arbitrates for the switch port while at the same time trying to acquire a free VC. If the speculation succeeds, the flit directly enters the ST pipeline stage. However, when speculation fails, the flit must go through some of these pipeline stages again, depending on where the speculation failed.

In Figure 6.3, two examples of pipeline optimizations are shown. Figure 6.3a shows pipeline bypassing. At Time 1, A arrives at the South input port and there are no buffered flits waiting in the input queue. The lookahead routing computation is performed in the first cycle (1a) and the crossbar connection between the south input and the east output is setup (1b). At Time 2, A traverses the crossbar and exits the router. Buffering and allocation are bypassed. In Figure 6.3b, two flits arrive at Time 1 (A on the South input and B on the North input); both have empty input queues and attempt to bypass the pipeline. However, during the crossbar setup (1b), a port conflict is detected



(a) Pipeline Bypassing



(b) Pipeline Bypassing Aborted with Speculation Example

Figure 6.3: Pipeline Optimization Examples.

as both flits attempt to setup the crossbar for the East output port. Now, both flits must be written into input buffers (1c). At time 2, both A and B will perform virtual channel and switch allocation in parallel (2a and 2b). Packet B successfully allocates an output virtual channel and the switch and traverses the switch at Time 3. Packet A succeeds in virtual channel allocation but fails in switch allocation. At Time 3, A will again attempt to allocate the switch. A's request is now non-speculative since it already obtained an output virtual channel (2a). A's request is successful and it traverses the switch and exits the router at Time 4.

In the next sections, we discuss the implementation of each of the router and pipeline components.

6.3 BUFFER ORGANIZATION

Buffer organization has a large impact on network throughput, as it heavily influences how efficiently packets share link bandwidth. Essentially, buffers are used to house packets or flits when they cannot be forwarded right away onto output links. Flits can be buffered on the input ports and on the output ports. Output buffering occurs when the allocation rate of the switch is greater than the rate of the channel. Crossbar speedup (discussed in Section 6.4.2) requires output buffering since multiple flits can be allocated to a single output channel in the same cycle.

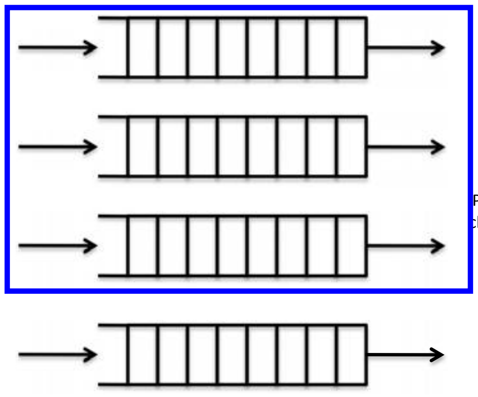
All previously proposed on-chip network routers have buffering at input ports, as input buffer organization permits area and power-efficient single-ported memories. We will, therefore, focus our discussion on input-buffered routers here, dissecting how such buffering is organized *within* each input port.

Single fixed-length queue. Figure 6.4a shows an input-buffered router where there is a single queue in each input port, i.e. there are no VCs. Incoming flits are written into the tail of the queue, while the flit at the head of the queue is read and sent through the crossbar switch and onto the output links (when it wins arbitration). The single queue has a fixed length, so the upstream router can keep track of buffer availability and ensure that a flit is forwarded only if there is a free buffer downstream.

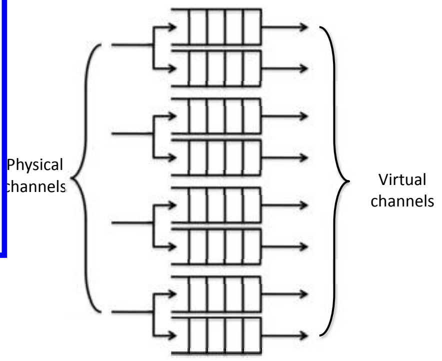
Clearly, a single queue can lead to scenarios where a packet at the head of the queue is blocked (as its output port is held by another packet), while a packet further behind in the queue whose output port is available could not make forward progress as it has to wait for the head of the queue to clear. Such unnecessary blocking is termed head-of-line blocking.

Multiple fixed-length queues. Having multiple queues at each input port helps alleviate head-of-line blocking. Each of these queues is termed a virtual channel, with multiple virtual channels multiplexing and sharing the physical channel/link bandwidth. Figure 6.4(b) shows an input-buffered router where there are two separate queues in each input port, corresponding to a router with 2 VCs.

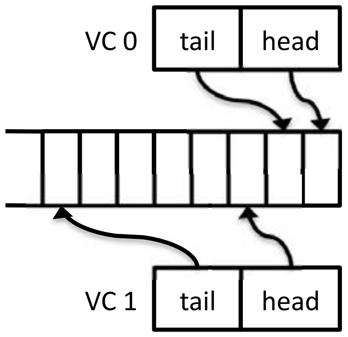
Multiple variable-length queues. In the above buffer organization, each VC has a fixed-length queue, sized at four flits in Figure 6.4(b). If there is imbalance in the traffic, there could be



(a) Single fixed-length queue



(b) Multiple fixed-length queues



(c) Multiple variable-length queues

Figure 6.4: Buffer Organizations.

a VC that is full and unable to accept more flits when another VC is empty, leading to poor buffer utilization and thus low network throughput.

To get around this, each VC queue can be variable-length, sharing a large buffer [201], as shown in Figure 6.4(c). This permits better buffer utilization, but at the expense of more complex circuitry for keeping track of the head and tail of the queues. Also, to avoid deadlocks, one flit buffer needs to be reserved for each VC, so that other VCs will not fill up the entire shared buffer and starve out a VC, ensuring forward progress.

With the same total amount of buffering per port, designers have the choice of using many shallow VCs or fewer VCs with deeper buffers. More VCs further ease head-of-line blocking and

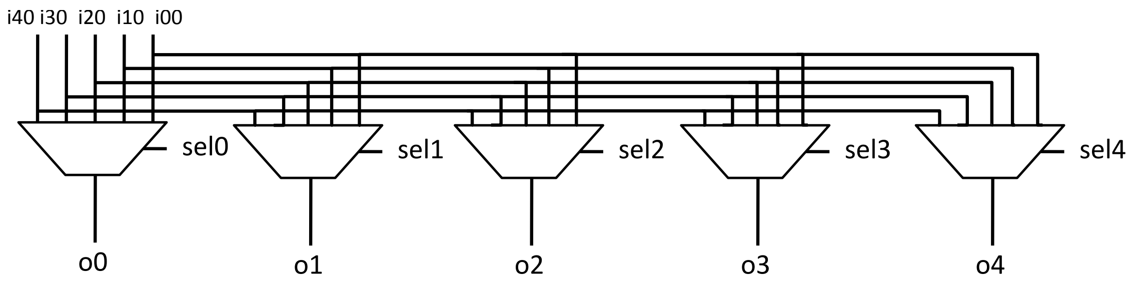


Figure 6.5: Crossbar composed of many multiplexers

thus improve throughput. It, however, comes at the expense of a more complex VC allocator and VC buffer management. Also, for fixed-length queues, there needs to be a minimum number of buffers per VC to cover the buffer turnaround time. Furthermore, the efficiency of many, shallow VCs versus few, deep VCs will depend on the traffic pattern. With light traffic, many shallow VCs will lead to under utilization of extra VCs. Under periods of heavy traffic, few, deep VCs will be less efficient as packets will be blocked due to a lack of available VCs.

6.4 SWITCH DESIGN

The crossbar switch of a router is the heart of the router datapath. It switches bits from input ports to output ports, performing the essence of a router's function.

6.4.1 CROSSBAR DESIGNS

Aggressive design of crossbar switches at high frequencies and low power is a challenge in VLSI design, such as the bit-interleaved or double-pumped custom crossbar used in the Intel TeraFLOPs chip [208]. Here, we just provide some background on basic crossbar designs, and discuss alternative microarchitectural organizations of crossbars.

Table 6.1 shows a Verilog module describing a crossbar switch, where input select signals to each multiplexer set up the connections of the switch, i.e. which input port(s) should be connected to which output port(s). Synthesizing this will lead to a crossbar composed of many multiplexers, such as that illustrated in Figure 6.5. Most low-frequency router designs will use such synthesized crossbars.

multiplexer crossbar

As designs push towards GHz clock and are faced with more stringent power budgets, custom-designed crossbars tend to be used. These have crosspoint based organizations with select signals feeding each crosspoint, setting up the connection of the switch, like that in Figure 6.6.

crosspoint crossbar

With either design, it is obvious that a switch's area and power scale at $O((pw)^2)$, where p is the number of crossbar ports and w is the crossbar port width in bits. A router architect thus has

Table 6.1: Verilog of a 4-bit 5-port crossbar.

```
module xbar (clk, reset, in0, in1, in2, in3, in4, out0, out1, out2
              out3, out4, colsel0, colsel1,colsel2, colsel3, colsel4);
input clk;
input reset;
input['CHANNELWIDTH:0] in0, in1, in2, in3, in4;
output['CHANNELWIDTH:0] out0, out1, out2, out3, out4;
input [2:0] colsel0, colsel1, colsel2, colsel3 colsel4;
reg [2:0] colsel0reg, colsel1reg, colsel2reg, colsel3reg, colsel4reg;

bitxbar bx0(in0[0],in1[0],in2[0],in3[0],in4[0],out0[0],out1[0],out2[0],out3[0],
            out4[0],colsel0reg,colsel1reg,colsel2reg,colsel3reg,colsel4reg,1'bx);

bitxbar bx1(in0[1],in1[1],in2[1],in3[1],in4[1],out0[1],out1[1],out2[1],out3[1],
            out4[1],colsel0reg,colsel1reg,colsel2reg,colsel3reg,colsel4reg,1'bx);

bitxbar bx2(in0[2],in1[2],in2[2],in3[2],in4[2],out0[2],out1[2],out2[2],out3[2],
            out4[2],colsel0reg,colsel1reg,colsel2reg,colsel3reg,colsel4reg,1'bx);

bitxbar bx3(in0[3],in1[3],in2[3],in3[3],in4[3],out0[3],out1[3],out2[3],out3[3]
            out4[3],colsel0reg,colsel1reg,colsel2reg,colsel3reg,colsel4reg,1'bx);

endmodule
```

```
module bitxbar(i0,i1,i2,i3,i4,o0,o1,o2,o3,o4,sel0,sel1,sel2,sel3,sel4,inv
input i0,i1,i2,i3,i4;
output o0,o1,o2,o3,o4;
[2:0] sel0, sel1, sel2, sel3, sel4;
input inv;

buf b0(i00, i0); //buffer for driving in0 to the 5 muxes
...
buf b4(i40, i4);

mux5_1 m0(i00, i10, i20, i30, i40, o0, sel0, inv);
...
mux5_1 m4(i00, i10, i20, i30, i40, o4, sel4, inv);

endmodule
```

to diligently choose p, a function of the topology, and w, which affects phit size and thus overall packet energy-delay.

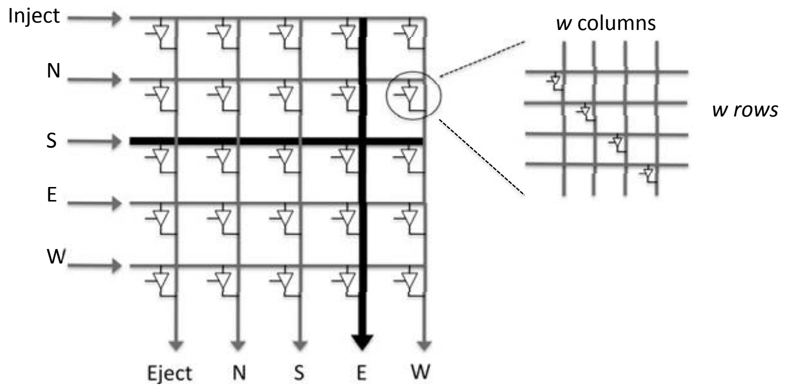


Figure 6.6: A 5x5 Crosspoint crossbar switch. Each horizontal and vertical line is w bits wide (1 phit width). The bold lines show a connection activated from the south input port to the east output port.

6.4.2 CROSSBAR SPEEDUP

A router microarchitect needs to decide on the crossbar switch speedup, i.e. the number of input and output ports in the crossbar relative to the number of router input and output ports. Figure 6.7 shows various alternative crossbar designs with different speedup factors: crossbars with higher speedups provide more internal bandwidth between router input and output ports, and thus ease the allocation problem and improving flow control. For instance, if each VC has its own input port to the crossbar, a flit can be read out of every VC every cycle, so multiple VCs need not contend for the same crossbar input port. A 10×5 crossbar (such as shown in Figure 6.7b) will achieve close to 100% throughput even with a simple allocator (allocators are discussed in the next section). By providing more inputs to select from, there is a higher probability that each output port will be matched (used) each cycle. The use of output speedup allows multiple flits to be sent to the same output port each cycle, thus reducing the contention. A crossbar with output speedup requires output buffers to multiplex flits onto single output port.

input speedup

output speedup

Crossbar speedup can also be achieved by clocking the crossbar at a higher frequency than the rest of the router. For instance, if the crossbar is clocked at twice the router frequency, it can then send two flits each cycle between a single pair of input-output ports, achieving the same performance as a crossbar with input and output speedup of 2. This is less likely in on-chip networks where a router tends to run off a single clock supply that is already aggressive.

6.4.3 CROSSBAR SLICING

With the crossbar taking up a significant portion of a router's footprint and power budget, microarchitectural techniques targeted towards optimizing crossbar power-performance have been proposed.

Dimension slicing a crossbar [161] in a 2-dimensional mesh uses two 3×3 crossbars instead

dimension slicing

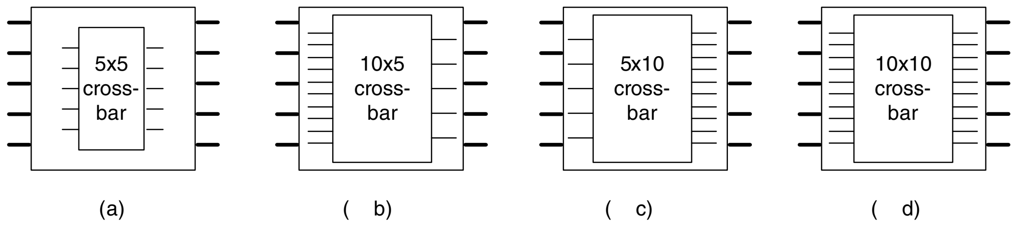


Figure 6.7: Crossbars with different speedups for a 5-port router. (a) No crossbar speedup, (b) Crossbar with input speedup of 2, (c) Crossbar with output speedup of 2, (c) Crossbar with input and output speedup of 2.

of one 5 × 5 crossbar, with the first crossbar for traffic that remains in the X-dimension, and the second crossbar for traffic remaining in the Y-dimension. A port on the first crossbar connects with a port on the second crossbar so traffic that turns from the X to Y dimension traverses both crossbars while those remaining within a dimension traverses only one crossbar. This is particularly suitable for the dimension-ordered routing protocol where traffic mostly stays within a dimension.

bit interleaving

Bit interleaving the crossbar targets w instead. It sends alternate bits of a link on the two phases of a clock on the same line, thus halving w. The TeraFLOPS architecture employs bit interleaving as will be discussed in Section 6.7.2.

6.5 ALLOCATORS AND ARBITERS

allocator: N to M
arbiter: N to 1

An allocator matches N requests to M resources while an arbiter matches N requests to 1 resource. In a router, the resources are VCs (for virtual channel routers) and crossbar switch ports.

switch arbiter

In a wormhole router with no VCs, the switch arbiter at each output port matches and grants that output port to requesting input ports. Hence, there are P arbiters, one per output port, each arbiter matching P input port requests to the single output port under contention.

VC allocator

switch allocator

In a router with multiple VCs, we need a virtual-channel allocator (VA) which resolves contention for output virtual channels and grants them to input virtual channels, as well as a switch allocator (SA) that grants crossbar switch ports to input virtual channels. Only the head flit of a packet needs to access the virtual-channel allocator, while the switch allocator is accessed by all flits and grants access to the switch on a cycle-by-cycle basis.

An allocator/arbiter that delivers high matching probability translates to more packets succeeding in obtaining virtual channels and passage through the crossbar switch, thereby leading to higher network throughput. Allocators and arbiters must also be fast and pipelinable so they can work under high clock frequencies.

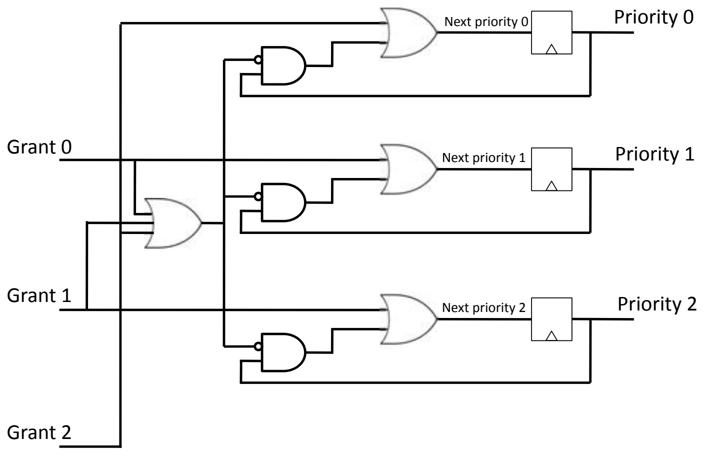


Figure 6.8: Round-robin arbiter.

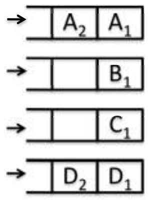


Figure 6.9: Request queues for arbiter examples.

6.5.1 ROUND-ROBIN ARBITER

With a round-robin arbiter, the last request to be serviced will have the lowest priority in the next round of arbitration. In Figure 6.8, the circuit required for a round robin arbiter is shown.

Next, we will walk through an example granting requests with a round-robin arbiter. A set of requests from 4 different requestors are shown in Figure 6.9. The last request serviced prior to this set of requests was from Requestor A. As a result, B has the highest priority at the start of the example. With the round-robin arbiter, requests are satisfied in the following order: B_1, C_1, D_1, A_1, D_2, A_2.

6.5.2 MATRIX ARBITER

A matrix arbiter operates so that the least recently served requestor has the highest priority [47]. A triangular array of state bits are stored to implement priorities since $w_{ij} = \neg w_{ji} \forall i \neq j$. When bit $[i, j]$ is set, i has a higher priority than j. The implementation of a matrix arbiter is shown in

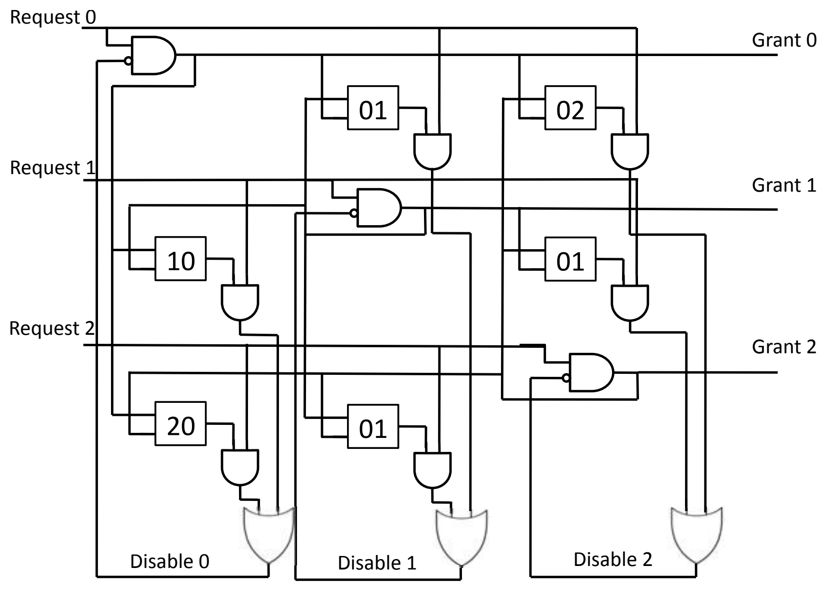


Figure 6.10: Matrix arbiter diagram.

Figure 6.10. When a request line is asserted, the request is AND-ed with the state bits in that row to disable any lower priority requests.

Next, we will walk-through the same set of requests from the previous example (Figure 6.9). The initial state of the matrix arbiter is given in Figure 6.11a. As each request is granted, the updated matrix values are shown. While only the upper triangle values of the matrix need to be stored all values are shown for clarity. We can see from the initial state of the matrix that requestor A has the lowest priority. Bits [1,0], [2,0], and [3,0] are all set to 1. In the first cycle, D_1 is granted. As a result, the bits in the 4^{th} column are set and bits in the 4^{th} row are cleared (Figure 6.11b). At T=2, request C_1 is granted with the resulting matrix in Figure 6.11c. Grants continue in this fashion with the resulting grant order being D_1, C_1, B_1, A_1, D_2, A_2.

6.5.3 SEPARABLE ALLOCATOR

Due to the need for pipelineable allocators, simple separable allocators are typically used, with an allocator being composed of arbiters, where an arbiter is one that chooses one out of N requests to a single resource. Figure 6.12 shows how each stage of a 3:4 separable allocator (an allocator matching 3 requests to 4 resources), is composed of arbiters. For instance, a separable VA will have the first stage of the allocator (comprised of four 3:1 arbiters) select one of the eligible output VCs, with the winning requests from the first stage of allocation then arbitrating for an output VC in the second

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 1 & x & 0 & 0 \\ 1 & 1 & x & 0 \\ 1 & 1 & 1 & x \end{pmatrix}$$

(a) T=0

$$\begin{pmatrix} x & 0 & 0 & 1 \\ 1 & x & 0 & 1 \\ 1 & 1 & x & 1 \\ 0 & 0 & 0 & x \end{pmatrix}$$

(b) T=1

$$\begin{pmatrix} x & 0 & 1 & 1 \\ 1 & x & 1 & 1 \\ 0 & 0 & x & 0 \\ 0 & 0 & 1 & x \end{pmatrix}$$

(c) T=2

$$\begin{pmatrix} x & 1 & 1 & 1 \\ 0 & x & 0 & 0 \\ 0 & 1 & x & 0 \\ 0 & 1 & 1 & x \end{pmatrix}$$

(d) T=3

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 1 & x & 0 & 0 \\ 1 & 1 & x & 0 \\ 1 & 1 & 1 & x \end{pmatrix}$$

(e) T=4

$$\begin{pmatrix} x & 0 & 0 & 1 \\ 1 & x & 0 & 1 \\ 1 & 1 & x & 1 \\ 0 & 0 & 0 & x \end{pmatrix}$$

(f) T=5

$$\begin{pmatrix} x & 0 & 0 & 0 \\ 1 & x & 0 & 1 \\ 1 & 1 & x & 1 \\ 1 & 0 & 0 & x \end{pmatrix}$$

(g) T=6

Figure 6.11: Matrix arbiter example.

stage (comprising three 4:1 arbiters). Different arbiters have been used in practice, with round-robin arbiters being the most popular due to their simplicity.

Figure 6.13 shows one potential outcome from a separable allocator. Figure 6.13a shows the request matrix. Each of the 3:1 arbiters selects one value of each row of the matrix; these first stage results of the allocator are shown in the matrix in Figure 6.13b. The second set of 4:1 arbiters will arbitrate among the requests set in the intermediate matrix. The final result (Figure 6.13c) shows that only one of the initial requests was granted. Depending on the arbiters used and the initial states, more allocations could result.

6.5.4 WAVEFRONT ALLOCATOR

Figure 6.14 shows a 4x4 wavefront allocator [200] which is used in the SGI SPIDER chip [71]. Non-square allocators can be realized by adding dummy rows or columns to create a square array. The 3x4 allocation example shown above with a separable allocator requires a 4x4 wavefront allocator.

The execution of a wavefront allocator begins with setting one of the four priority lines (p0..p3). This supplies row and column tokens to the diagonal group of cells connected to the selected priority line. If one of the cells is requesting a resource, it will consume the row and column tokens and its resource request is granted. Cells that cannot use their tokens pass row tokens to the right and column tokens down. To improve fairness, the initial priority group changes each cycle.

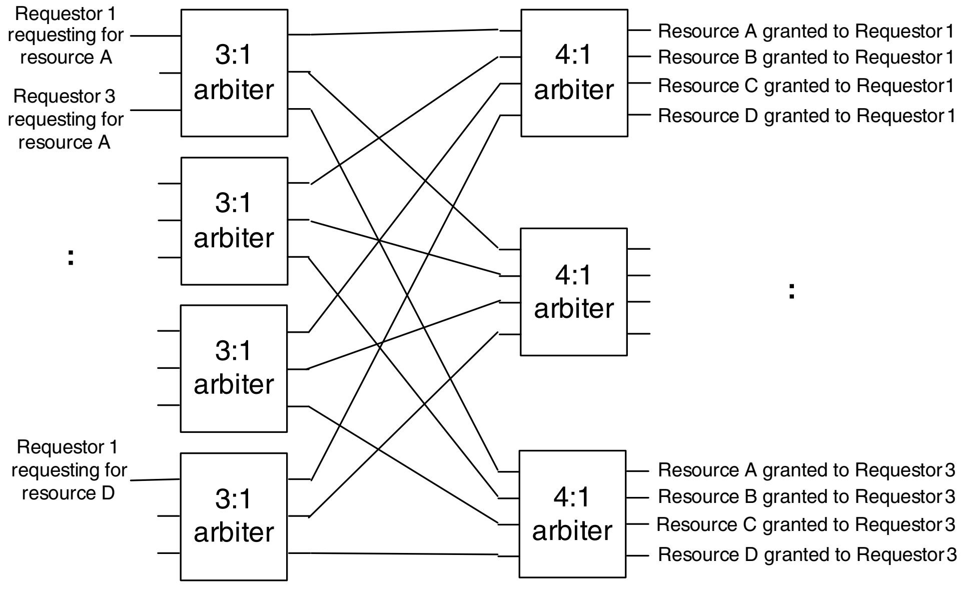

Requestor 1 requesting for resource A
Requestor 3 requesting for resource A
:
Requestor 1 requesting for resource D
3:1 arbiter
3:1 arbiter
3:1 arbiter
3:1 arbiter
4:1 arbiter
4:1 arbiter
4:1 arbiter
Resource A granted to Requestor 1
Resource B granted to Requestor 1
Resource C granted to Requestor 1
Resource D granted to Requestor 1
:
Resource A granted to Requestor 3
Resource B granted to Requestor 3
Resource C granted to Requestor 3
Resource D granted to Requestor 3

Figure 6.12: A separable 3:4 allocator (3 requestors, 4 resources) which consists of four 3:1 arbiters in the first stage and three 4:1 arbiters in the second. The 3:1 arbiters in the first stage decides which of the 3 requestors win a specific resource, while the 4:1 arbiters in the second stage ensure a requestor is granted just 1 of the 4 resources.

Using the same request matrix from Figure 6.13a, we next illustrate the function of a wavefront allocator. Shaded in light grey are the requests from the request matrix. The first diagonal wave of priorities starting with p0 is circled in Figure 6.15a. Entry [0,0] is the first to receive a grant (highlighted in dark grey). Next, the wave propagates down and to the right (shown in Figure 6.15b). Entry [0,0] consumed a token in the first stage when it received its grant; therefore, as the wave propagates, [0,1] and [1,0] do not receive a token in the second wave since it was already consumed. Entry [3,2] receives tokens from [3,1] and from [2,2] which results in its request being granted (Figure 6.15c). Figure 6.15c shows the 3rd priority wave; the remaining unused tokens are again passed down and to the right. Request [1,1] receives valid tokens in this wave and receives a grant.

After the wavefronts have fully propagated, the grant matrix that results is shown in Figure 6.16. The wavefront allocator is able to grant 3 requests (as opposed to the single request for this example with a separable allocator).

6.5.5 ALLOCATOR ORGANIZATION

The design of the virtual-channel allocator also depends on the implementation of the routing function. The routing function can be implemented to return a single virtual channel. This would lead

$$\begin{pmatrix} 1 & 1 & 1 \\ 1 & 1 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 1 \end{pmatrix}$$

(a) Request Matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \\ 1 & 0 & 0 \end{pmatrix}$$

(b) Intermediate Matrix

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

(c) Grant Matrix

Figure 6.13: Separable Allocator Example.

to a virtual channel allocator that needs to arbitrate only between input virtual channels contending for the same output virtual channel. If the routing implementation is more general and returns multiple candidate virtual channels for the same physical channel, the allocator needs to first arbitrate among v possible first stage requests before forwarding the winning requests to the second stage (can be done with the separable allocator described above). A routing function that returns all candidate virtual channels for all candidate physical channels is the most general and requires more functionality from the virtual-channel allocator.

With a speculative virtual channel router, non-speculative switch requests must have a higher priority than speculative requests. One way to achieve this is to have two parallel switch allocators. One allocator handles non-speculative requests, while the second handles speculative requests. With the output of both allocators, successful non-speculative requests can be selected over speculative ones. However, if there are no non-speculative requests in the router, the a speculative switch request will succeed in allocating the desired output port. It is possible for a flit to succeed in speculative switch allocation but fail in the parallel virtual channel allocation. In this case the speculation is incorrect and the crossbar passage that was reserved by the switch allocator is wasted. Only head flits are required to perform VC allocation. As a result, subsequent body and tail flits are marked as non-speculative (for their switch allocation) since they inherit the VC allocated to the head flit.

Adaptive routing can complicate the switch allocation for flits. For a deterministic routing algorithm, there is a single desired output port; the switch allocator's function is simply to bid for the single output port. With an adaptive routing function that returns multiple candidate output ports, the switch allocator can bid for all output ports. The granted output port must match the virtual channel granted by the virtual channel allocator. Alternatively, the routing function can return a single candidate output port and then retry routing (for a different output port) if the flit fails to obtain an output virtual channel.

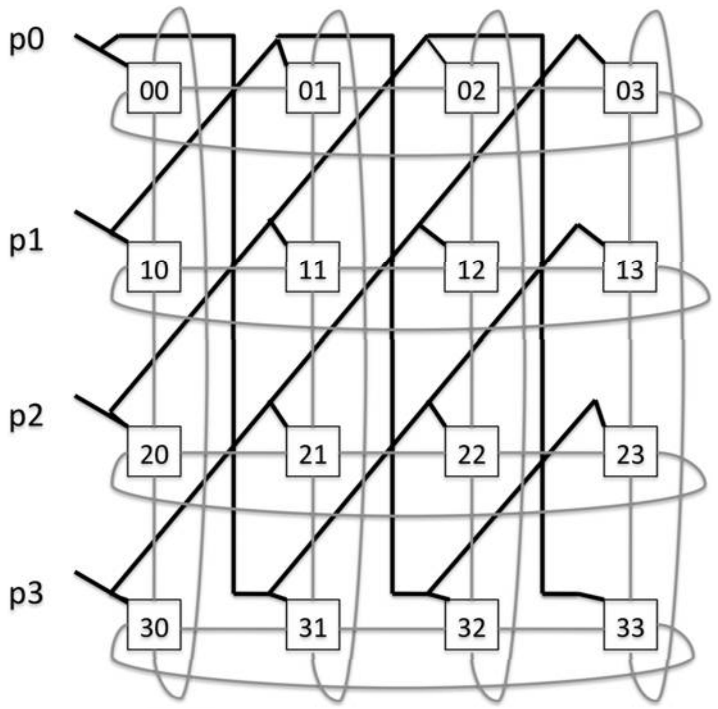


Figure 6.14: A 4x4 Wavefront allocator. Diagonal priority groups are connected with bold lines. Connections for passing tokens are shown with grey lines.

6.6 IMPLEMENTATION

6.6.1 ROUTER FLOORPLANNING

A key step in the backend design flow of a router is floorplanning: determining the placement of the different input and output ports of a router, along with the global allocator modules and the crossbar switch. Figure 6.17 shows two alternative router floorplans for a fairly similar microarchitecture: A 5-port router with virtual channel flow control.

Typically, the allocators (VA) or the crossbar switch traversal (ST) dictate the critical path. Hence, both floorplans optimized their layouts in order to target these two components, but in different ways. Both floorplans use the semi-global metal layers (typically M5, M6 in recent processes) for router-to-router interconnects, but Figure 6.17a drops to the local metal layers for intra-router wiring, such as the crossbar switch, with the inter-router link datapath continuing on the upper metal. Figure 6.17b, on the other hand, has the crossbar switch continuing on the semi-global metal layers.

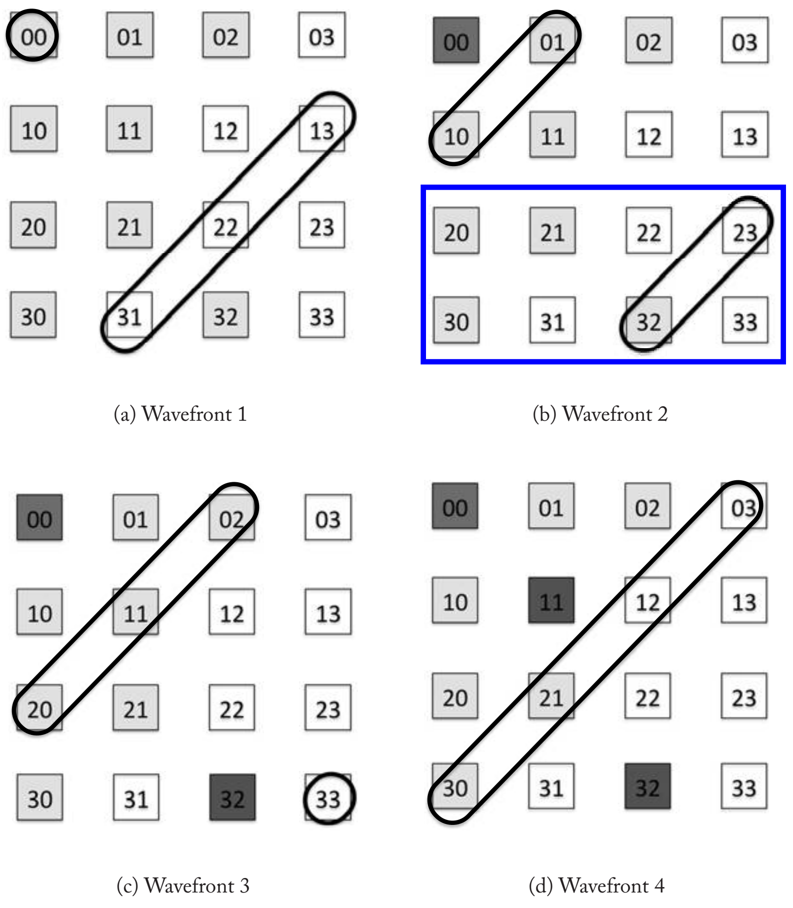


Figure 6.15: Wavefront allocator example.

This leads to a key difference apparent from the floorplan in the placement of the input ports. The placement in Figure 6.17a is fairly intuitive: the north input port is placed close to the north edge of the router, the east input port along the east edge and so on. The placement in Figure 6.17b puts all input ports side by side, on the left side of the switch, to free up the M5 and M6 layers for the crossbar wiring, while ensuring fast access to the allocators.

To target allocator delay, Figure 6.17a replicates the allocators at every input port, so allocator grant signals will not incur a large RC delay before triggering buffer reads, and the crossbar can also be setup more quickly as control signals now traverse a shorter distance. This comes at the cost of

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

Figure 6.16: Wavefront grant matrix.

increased area and power. Allocator request signals still have to traverse through the entire crossbar height and width, but their delay is mitigated as that router uses a pipeline optimization technique, advanced bundles, to trigger allocations in advance. Figure 6.17b, however, leverages their use of the semi-global metal layers for the crossbar to place the allocators in the middle, in the active area underneath the crossbar wiring, to lower wire delay to the allocators without replication.

Here, we just aim to illustrate the many possible back-end design decisions that can be made at floorplanning time to further optimize the router design. Note that as routers are just a component of a many-core chip, its floorplan also needs to be done relative to the positions of the network interfaces (NICs), cores and caches.

6.6.2 BUFFER IMPLEMENTATION

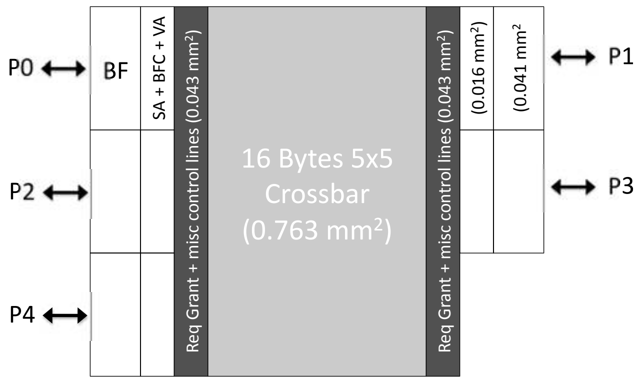
Router buffer cells can be implemented using flip-flops or generated memory cells (SRAM or register file), depending on the buffer size and access timing requirements. For very small buffers, flip flops suffice and can be readily synthesized without requiring memory generators. Flip flops, however, have much poorer area, power and delay characteristics compared to SRAMs and register files. Between SRAMs and register files, at smaller buffer sizes, register file cells tend to occupy a smaller footprint as they do not require differential sense amplifiers, and support faster access times. However, at larger buffer sizes, SRAMs prevail. The crossover point depends heavily on the specific process and memory cells used, so designers ought to carefully evaluate the alternatives.

6.7 BIBLIOGRAPHIC NOTES

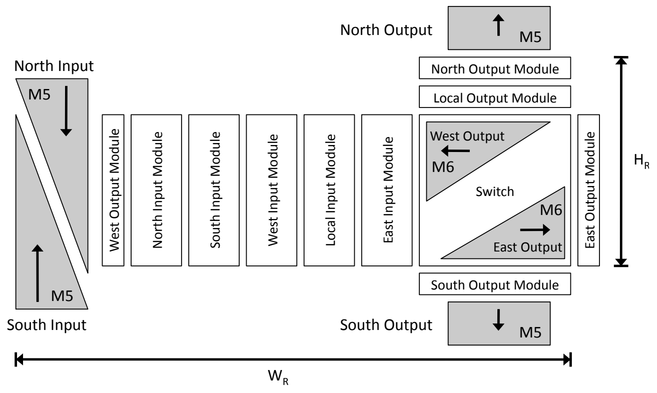
6.7.1 CASE STUDIES

IBM Cell. As the IBM Cell essentially mimics a bus using the four rings, it does not have switches or routers at intermediate nodes. Instead, bus interface units (BIU) arbitrate for dedicated usage of segments of the ring, and once granted, injects into the ring.

Intel TeraFLOPS. The high 5GHz (15 FO4s) frequency of the TeraFLOPS chip mandates the use of aggressively pipelined routers. The router uses a 5-stage pipeline: buffer write, route computation (extracting the desired output port from the header), two separable stages of switch allocation, and switch traversal. It should be noted that single hop delay is still just 1 ns. Each port has two input queues, one for each VC, that are each 16 flits deep. The switch allocator is separable in


P0
P2
P4
BF
SA + BFC + VA
Req Grant + misc control lines (0.043 mm²)
16 Bytes 5x5 Crossbar (0.763 mm²)
Req Grant + misc control lines (0.043 mm²)
(0.016 mm²)
(0.041 mm²
P1
P3

(a) Router Layout from Kumar et al [125]. BF: Buffer, BFC: Buffer Control, VA: VC Allocator, SA: Switch Allocator. P0: North port; P1: East; P2: West; P3: South; P4: Injection/Ejection.



(b) Router Layout from Balfour and Dally [21]. M5 and M6 indicate the metal layers used.

Figure 6.17: Two router floorplans.

order to be pipelineable, implemented as a 5-to-1 port arbiter followed by a 2-to-1 lane arbiter. The first stage of arbitration within a particular lane essentially binds one input port to an output port, for the entire duration of the packet, opting for router simplicity over the flit-level interleaving of multiple VCs. Hence, the VCs are not leveraged for bandwidth, but serve only deadlock avoidance purposes. The crossbar switch is custom-designed, using bit-interleaving, or double pumping, with alternate bits sent on different phases of a clock, reducing the crossbar area by 50%.

Tilera TILE64. The iMesh' wormhole networks have a single-stage router pipeline during straight portions of the route, and an additional route calculation stage when turning. Only a single buffer queue is needed at each of the 5 router ports, since no VCs are used. Only 3 flit buffers are used per port, just sufficient to cover the buffer turnaround time. This emphasis on simple routers results in a low area overhead of just 5.5% of the tile footprint.

ST Microelectronics STNoC. Since the STNoC targets MPSoCs, it supports a range of router pipelines and microarchitectures. Here we summarize the key attributes that are common across all STNoC routers. As MPSoCs do not require GHz frequencies, STNoC supports up to 1GHz in 65nm ST technology. Since the degree is three, the routers are four-ported, to left, right, across links as well as the NIC to IP blocks. Low-load bypassing is used to lower the latency. Buffers are placed at input ports, statically allocated to each VC, with the option of also adding output buffer queues at each output port to further relieve head-of-line blocking. Round-robin, separable allocators are used. Extra ports or links that are trimmed off are not instantiated at design time. The crossbar switch is fully synthesized.

6.7.2 BRIEF STATE-OF-THE-ART SURVEY

This chapter discussed fairly complex routers, where virtual channels are used to multiplex messages on the same links, increasing link utilization and overall network throughput. Most on-chip network routers to date are simple wormhole-based routers, with no virtual channels, and very limited buffering, as bandwidth demands are not high for current core counts and applications, and designers are faced with stringent area and power budgets. For instance, each of Tilera's networks has a single queue of 3 flit buffers per port just to cover the buffer turnaround time. Where virtual channels are used, designers chose limited numbers, such as 2 in the Intel Teraflops, 2 in the Austin TRIPS, to lower the complexity.

Lookahead signals for single-flit packets[81] and advanced bundles for multi-flit packets[125] have similar characteristics in their aim to reduce router pipeline to a single stage. A setup bundle will configure the switch prior to the arrival of the flit; a flit will experience only the switch traversal stage.

Router microarchitecture research can be classified into those that reduce the number of router pipeline stages in order to improve on-chip network latency [153, 125, 174, 81, 65, 144, 170]; those that specifically target router power [116, 210, 126, 127], and several that come up with new flow control protocols that naturally demand major modifications to the microarchitecture, such as the use of a ring as the switching fabric within a router rather than a crossbar [3]. In application-specific MPSoCs, router microarchitectures can also be customized; an example is custom buffer allocation [99].

CHAPTER 7

Conclusions

We conclude by synthesizing the information from the previous chapters to help the reader understand the current state-of-the-art for on-chip networks and where we believe on-chip network research is needed to move the field forward.

7.1 GAP BETWEEN STATE-OF-THE-ART AND IDEAL

Research proposals that drive the performance and cost of on-chip networks toward the ideal performance and cost will increase the adoption of these networks and provide scaling in many-core architectures. Ideal metrics serve as a consistent benchmark against which researchers can measure progress in this field. Next, we will describe the characteristics of the ideal interconnection network and then contrast it with the current state-of-the-art.

7.1.1 DEFINITION OF IDEAL INTERCONNECT FABRIC

Ideal latency. The lowest latency (or ideal latency) that can be achieved by the interconnection network is one that is solely dedicated to the wire delay between a source and destination. This latency can be achieved by allowing all data to travel on dedicated pipeline wires that directly connect a source to a destination. Wire latency assumes the optimal insertion of repeaters and flip-flops. The latency of this dedicated wiring would be governed only by the average wire length D between the source and destination (assumed to be the Manhattan distance), packet size L, channel bandwidth b, and propagation velocity v:

$$T_{ideal} = \frac{D}{v} + \frac{L}{b}$$

The first term corresponds to the time spent traversing the interconnect, while the second corresponds to the serialization latency for a packet of length L to cross a channel of bandwidth b. Ideally, serialization delay could be avoided with very wide channels. However, such a large number of wires would not be feasible given the projected chip sizes.

Ideal energy. The energy expended to communicate data between tiles should ideally be just the energy of interconnect wires as given by

$$E_{ideal} = \frac{L}{b} \cdot D \cdot E_{wire}$$

where D is again the distance between source and destination and E_{wire} is the interconnect transmission energy per unit length.

Ideal throughput. Throughput is defined as the data rate in bits per second that the network can accept per input port prior to saturation. Ideally, the throughput of the network should be solely a function of the bandwidth provided by the topology.

7.1.2 DEFINITION OF STATE-OF-THE-ART

State-of-the art latency. As dedicated wiring is impractical for systems with a large number of nodes, long wires are segmented through the insertion of routers. The use of routers and the presence of multiple communication flows sharing resources increases communication latency over the ideal latency presented previously. The packet latency of an on-chip network can be described by the following equation:

$$T_{actual} = \frac{D}{v} + \frac{L}{b} + H \cdot T_{router} + T_c$$

where H is the average hop count through the topology, T_{router} is the delay through a single router and T_c is the delay due to contention between multiple messages competing for the same network resources. T_{router} accounts for the time each packet spends in various stages at each router as the router coordinates between multiple packets; depending on the implementation, this can consist of several pipeline stages.

State-of-the-art energy. Just as the addition of routers increases communication latency, it also increases communication energy. Energy is no longer solely attributed to the transmission power of wires but rather is given by

$$E_{actual} = \frac{L}{b} \cdot (D \cdot P_{wire} + H \cdot P_{router})$$

where P_{router} is the average router power. Buffer read/write power, power spent arbitrating for virtual channels and switch ports and crossbar traversal power all contribute to P_{router} [211].

State-of-the-art throughput. Ideal throughput assumes perfect flow control and perfect load balancing from the routing algorithm. In reality, actual saturation throughput can be much lower. The inability of the arbitration schemes to make perfect matching between requests and available resources can degrade throughput. Likewise, limited number of buffers and buffer turnaround latency can drive down the throughput of the network. Routing algorithm can have a large impact; a deterministic routing algorithm is unable to balance traffic and respond to network load. Heavily used paths will saturate quickly, reducing the rate of accepted traffic.

7.1.3 NETWORK POWER-DELAY-THROUGHPUT GAP

In Figure 7.1, we present simulated data for a state-of-the-art virtual channel network that uses dimension-ordered routing, a large number of VCs and buffers empirically determined to lead to the best performance (8 VCs, 24 buffers per port), all pipeline optimizations discussed in the book (lookahead routing, speculative allocation, pipeline bypassing) in a 7-ary 2-cube targeting 65nm, 3GHz, and 1.1V. The full set of parameters used to obtain this data are given in Table 7.1. Ideal

Table 7.1: State of the Art Network Simulation Parameters.

Parameter	Value
Technology	65 nm
V_{dd}	1.1 V
$V_{threshold}$	0.17 V
Frequency	3 GHz
Topology	7-ary 2-mesh
Routing	Dimension-ordered (DOR)
Traffic	Uniform random
Number of router ports	5
VCs per port	8
Buffers per port	24
Flit size/channel width (c_{width})	128 bits
Link length	1 mm
Wire pitch (W_{pitch})	0.45 μm

latency is calculated based on data from [136]; the maximum wire length that can be driven in a single cycle assuming delay-optimal repeater and flip-flop insertion, with minimum wire pitch from ITRS [1] used to derive L_{edge}[1]. The chip edge length required to accommodate the total bisection wiring of dedicated connections between all tiles can be calculated as

$$L_{edge} = 2 \cdot \frac{N}{2} \cdot \frac{N}{2} \cdot W_{pitch} \cdot c_{width}$$

where N is the number of tiles, W_{pitch} is the wire pitch and c_{width} is the channel width. Assuming a uniform placement of tiles, the average wire length D can then be derived as:

$$D = H \cdot \frac{L_{edge}}{\sqrt{N}}$$

The on-chip network curve assumes the state-of-the-art router pipeline optimizations discussed in Chapter 6. Optimizations to reduce the number of pipeline stages traversed has a significant impact on the latency. If every message was required to traverse the baseline 5-stage pipeline presented, the latency gap at low loads would be much wider. Aggressive speculation and bypassing narrows the gap at low loads; however, it is not eliminated. Despite the use of these optimizations, the network latency is still higher than our ideal latency of only wire transmission latency. Without such pipeline optimizations, we would expect higher latency at all injected loads. Techniques that narrow the latency gap at moderate to high loads are needed as mis-speculation rates tend to increase as the offered traffic rates increase.

[1]Note that the resulting chip size is impractically large.

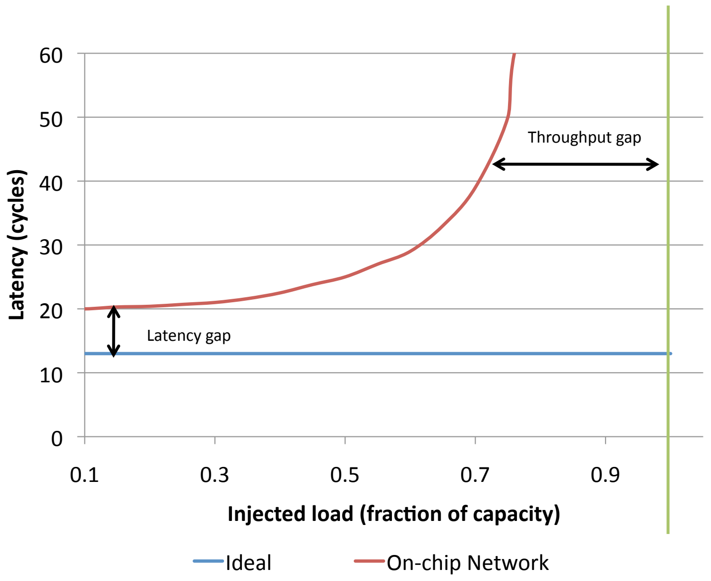


Figure 7.1: Latency and Throughput gap for a 7-ary 2-cube [127].

This aggressive design saturates before the ideal throughput is reached; even with separable allocator and aggressive optimization, this design still falls short of the ideal. Simpler router designs will increase this throughput gap significantly; wormhole flow control without virtual channels will saturate much earlier than the curve shown. A small number of buffers will also reduce the saturation throughput.

Figure 7.2 shows the gap between the energy consumption of an ideal network and a state-of-the-art baseline network. This baseline architecture incorporates many energy-efficient microarchitectural features but still significantly exceeds the energy consumed solely by wires. This gap exists due to the additional buffering, switching and arbitration that occurs at each router; the gap widens until the network saturates.

7.2 KEY RESEARCH CHALLENGES

The study of on-chip networks is a relatively new research field. Conference papers addressing them began appearing only in the late 1990s and on-chip networks have only recently began appearing in products in sophisticated forms. As such there clearly are many pressing research challenges that need to be addressed. Here we briefly discuss several major ones.

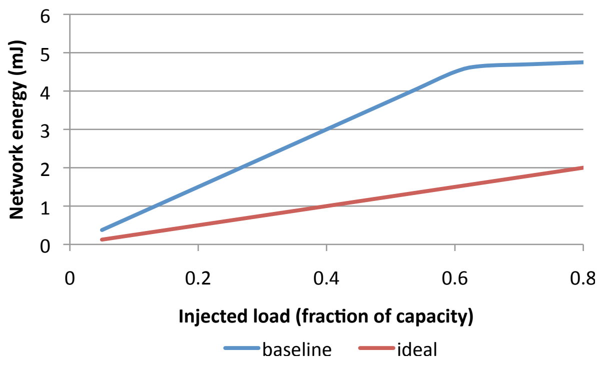


Figure 7.2: Energy Gap for a 7-ary 2-cube [127].

7.2.1 LOW-POWER ON-CHIP NETWORKS

Power is highlighted as one of the most pressing constraints in today's chip design. A 2006 workshop sponsored by the National Science Foundation and attended by about 50 leading researchers in the area of on-chip networks from academia and industry pointed out on-chip network power as a key problem, projecting that current designs consume an order of magnitude higher power than what can be supported in future many-core chips [167]. Research into low-power on-chip networks has made significant progress in recent years [27, 109, 211, 116, 130, 22, 163, 89], but as we show earlier, there still exists a large gap between the current state-of-the-art and the ideal interconnect in terms of power consumption.

The power consumed by a network is largely dependent on the bandwidth it has to support, while the amount of on-chip bandwidth needed depends on several factors. The application communication characteristics, number of cores and the cache hierarchy all dictate the on-chip bandwidth demand, while the memory controllers and I/O pins affect the off-chip bandwidth and thus the supply to the on-chip bandwidth. Of course, the power budget available to the on-chip network critically constrains the bandwidth it can support. To substantially lower the power consumption of on-chip networks, there needs to be a reexamination of every aspect, from topology, routing, flow control to router and link microarchitecture and circuits. In fact, ultimately, we believe there needs to be research into new multi-core system architectures that can optimize communications along with overall chip performance and scalability.

7.2.2 BEYOND CONVENTIONAL INTERCONNECTS

This lecture focused on router architecture and design, as all on-chip networks to date used conventional repeatered wires run at full voltage swing. Repeaters are inserted in these wires, appropriately trading off delay with energy [166].

To drive on-chip network power down, clearly power-efficient link designs can be used to lower transmission energy. Researchers have explored the leveraging of off-chip link signaling techniques [53, 37] for on-chip transmission lines [122, 111, 100, 90, 36, 35, 106].

With off-chip I/O being constrained by the number of pins and link bandwidth, researchers have also increasingly viewed 3-dimensional die stacking as a viable option for very high bandwidth off-chip I/O. 3-D stacking will lead not only to a substantial jump in the demand for on-chip bandwidth, but will also impact the design of on-chip networks [115, 217, 171, 134].

Finally, optics has effectively replaced electronics for long-distance links, and have also shown promise in addressing the tight power constraints faced in on-chip networks. Miller [149] projected that electronic on-chip networks will be unable to achieve the bandwidth demands and tight power envelope of future chips, detailing how the physical characteristics of optics and the progress in optical device research can potentially solve that. Several groups have investigated alternative optical on-chip network architectures [209, 169, 118, 192, 107, 24, 135]. Prior work in optical multi-chip interconnection networks can also be leveraged [177, 86].

7.2.3 RESILIENT ON-CHIP NETWORKS

As technology scales towards deep sub-micron lithographies, an on-chip network that is architected and designed assuming perfect, zero-fault fabrication and operation will no longer be possible. This is aggravated by the system overlaid atop the on-chip network typically assuming always-correct communication. For instance, a cache-coherent shared-memory CMP is architected assuming that every message will be delivered correctly to the designated destination, with no packets dropped midway. Similarly, a MPSoC assumes that every transaction through the on-chip network is successfully completed.

Shrinking gate lengths will lead to manufacturing defects and variability, with the large die sizes of many-core chips making post-fabrication faults highly likely [19]. When combined with errors that are likely to occur during chip operation, such as soft errors and wearout errors, it will be critical to design on-chip networks that are resilient and continue correct operation in the face of many faults/errors.

Fault-tolerant routing has been investigated substantially in the past, in the domain of clusters of workstations [218, 63, 178] and large-scale multi-chassis multi-computers [51], and can be leveraged. In on-chip networks, resilient mechanisms have to be designed under very tight power and area constraints, and yet work in the face of many faults. Early works in this area explored the addition of redundant components in the router microarchitecture [40] as well as the use of coding techniques to trade off link power and reliability [216].

7.3 NOC INFRASTRUCTURES

There are several different evaluation tools and methodologies that can facilitate on-chip network research. Performance simulation can be done with a stand-alone interconnect simulator or as part of a full-system simulation environment. The Wisconsin GEMS toolset [141] is a full-system simulation environment that has integrated Garnet, a detailed interconnect model [8]. SESC is another cycle-accurate simulation environment that has an integrated network model [185]. Several stand-alone simulators are also available [181, 56, 188, 84].

In addition to performance simulators to produce latency and throughput data, models have been developed to understand the delay, area and power consumption of various interconnection network design points. Analytical models that model router pipelines and estimate per-hop delay have been developed [175]. Architectural power modeling has been proposed for interconnection networks [211, 109]. Low-level power estimation can also be done using RTL models [152]. High-level models that estimate the area of proposed on-chip networks given microarchitectural parameters have also been released [21, 109].

7.4 ON-CHIP NETWORK BENCHMARKS

There has not yet been a standard suite of NoC benchmarks that is widely used by researchers, though there is clearly such a need [167, 105, 82]. Researchers in the architecture area have used on-chip network traces obtained from the Austin TRIPS [187] and MIT RAW [202] many-core chips as representative of fine-grained operand networks. Network traces have also been culled from full-system simulations of CMPs running widely-used parallel benchmark suites such as SPLASH [215] and PARSEC [31]. In the MPSoC area, researchers have used large task graphs composed of many IP blocks, such as the VOPD task graph shown in Figure 2.10, as benchmarks for evaluating NoC mapping algorithms and NoC designs. There has also been an initiative in OCP-IP to define a common standard for benchmarks and gather a suite of benchmarks that can be widely used to evaluate future NoCs [82].

Network traces provide a fairly realistic way of exploring the effectiveness of proposed on-chip network designs, but clearly, it should be noted that their characteristics depend heavily on the simulated many-core platform. The number of cores/IP blocks, the memory hierarchy, the number of memory controllers, etc. significantly influence the network trace. The lack of feedback effects when using network traces also impacts the accuracy. Full-system simulations provide more accurate evaluations but take up substantial simulation time. In general, a combination of synthetic traces, which exercise and test the limits of a proposed approach; real network traces, which give an idea of the effectiveness of an approach; and full-system simulations, which more accurately evaluate the approach in a specific system; should be used to have a good understanding of the pros and cons of a proposed technique.

7.5 ON-CHIP NETWORKS CONFERENCES

Research into on-chip networks is appearing in many venues spanning several disciplines. Table 7.2 highlights some major conferences in Architecture, CAD, VLSI and NoCs that publish innovative research into on-chip networks.

Table 7.2: On-chip network conferences.

Field	Conference
Architecture	International Symposium on Computer Architecture (ISCA)
	International Symposium on Microarchitecture (MICRO)
	International Symposium on High Performance Computer Architecture (HPCA)
	International Conference on Parallel Architectures and Compilation Techniques (PACT)
	International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)
CAD	International Conference on Computer-Aided Design (ICCAD)
	Design Automation Conference (DAC)
	Design Automation and Test in Europe (DATE)
VLSI	International Conference on VLSI (VLSI)
	International Solid State Circuits Conference (ISSCC)
Network on Chip	International Network on Chip Symposium (NOCS)

This list is not exhaustive; in addition to major conferences, recent years have seen workshops specifically focused on on-chip networks at many of these conferences. A repository of many network on chip papers is maintained by Robert Mullins at Cambridge [155].

7.6 BIBLIOGRAPHIC NOTES

We hope that this book has provided researchers with the necessary foundation for ongoing study of and research into on-chip networks. As the number of cores integrated on a single chip continues increasing, the communication fabric will play an increasingly critical role. Performance, power and reliability can all be impacted by the design of the on-chip interconnection network; we encourage researchers to explore this area at the system and microarchitectural levels. Each chapter has concluded with references to various state-of-the-art on-chip network research proposals to provide understanding of the current snapshot of on-chip network design, as well as motivate them with what open challenges remain. In addition to these works, we refer the interested reader to other summary and overview papers [139, 28, 32, 167, 162, 87] to help guide them in further study of on-chip networks.

Bibliography

[1] International technology roadmap for semiconductors. http://public.itrs.net.

[2] Pablo Abad, Valentin Puente, and José Ángel Gregorio. MRR: Enabling fully adaptive multicast routing for CMP interconnection networks. In *International Symposium on High Performance Computer Architecture*, pages 355–366, 2009. DOI: 10.1109/HPCA.2009.4798273

[3] Pablo Abad, Valentin Puente, José Angel Gregorio, and Pablo Prieto. Rotary router: An efficient architecture for CMP interconnection networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 116–125, June 2007. DOI: 10.1145/1273440.1250678

[4] Dennis Abts, Abdulla Bataineh, Steve Scott, Greg Faanes, Jim Schwarzmeier, Eric Lundberg, Tim Johnson, Mike Bye, and Gerald Schwoerer. The Cray BlackWidow: a highly scalable vector multiprocessor. In *Proceedings of the conference on Supercomputing*, page 17, 2007. DOI: 10.1145/1362622.1362646

[5] Dennis Abts, Natalie Enright Jerger, John Kim, Mikko Lipasti, and Dan Gibson. Achieving predictable performance through better memory controller placement in many-core CMPs. In *Proceedings of the International Symposium on Computer Architecture*, 2009.

[6] Saurabh N. Adya and Igor L. Markov. Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Transactions on VLSI Systems*, 11(6):1120–1135, December 2003. DOI: 10.1109/TVLSI.2003.817546

[7] Anant Agarwal, Liewei Bao, John Brown, Bruce Edwards, Matt Mattina, Chyi-Chang Miao, Carl Ramey, and David Wentzlaff. Tile processor: Embedded multicore for networking and multimedia. In *Proceedings of Hot Chips: Symposium on High Performance Chips*, 2007.

[8] Niket Agarwal, Tushar Krishna, Li-Shiuan Peh, and Niraj K. Jha. GARNET: A detailed on-chip network model inside a full-system simulator. In *Proceedings of the IEEE International Symposium on Performance Analysis of Systems and Software*, pages 33–42, April 2009. DOI: 10.1109/ISPASS.2009.4919636

[9] Niket Agarwal, Li-Shiuan Peh, and Niraj K. Jha. In-network snoop ordering (INSO): Snoopy coherence on unordered interconnects. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 67–78, February 2009. DOI: 10.1109/HPCA.2009.4798238

[10] Thomas William Ainsworth and Timothy Mark Pinkston. On characterizing performance of the Cell broadband engine element interconnect bus. In *Proceedings of the International Network on Chip Symposium*, pages 18–29, May 2007. DOI: 10.1109/NOCS.2007.34

[11] Adrijean Andriahantenaina, Herve Charlery, Alain Greiner, Laurent Mortiez, and Cesar Albenes Zeferino. SPIN: a scalable, packet switched, on-chip micro-network. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 70–73, 2003.

[12] Adrijean Andriahantenaina and Alain Greiner. Micro-network for SoC: Implementation of a 32-port SPIN network. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1128–1129, March 2003.

[13] Padma Apparao, Ravi Iyer, Xiaomin Zhang, Don Newell, and Tom Adelmeyer. Characterization and analysis of a server consolidation benchmark. In *Proceedings of the International Conference on Virtual Execution Environments*, pages 21–30, 2008. DOI: 10.1145/1346256.1346260

[14] ARM. Amba 3 overview. http://www.arm.com/products/solutions/ AMBA3AXI.html.

[15] ARM. Amba system architecture. http://www.arm.com/products/solutions/ AMBAHomePage.html.

[16] ARM. Amba axi protocol specification v1.0, 2003.

[17] Krste Asanović, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The landscape of parallel computing research: A view from Berkeley. Technical Report Technical Report No. UCB/EECS-2006-183, Electrical Engineering and Computer Sciences, University of California at Berkeley, 2006.

[18] Infiniband Trade Association. http://www.infinibandta.org/about.

[19] Todd Austin, Valeria Bertacco, Scott Mahlke, and Yu Cao. Reliable systems on unreliable fabrics. *IEEE Design and Test of Computers*, 25(4):322–332, July 2008. DOI: 10.1109/MDT.2008.107

[20] Rajeev Balasubramonian, Naveen Muralimanohar, Karthik Ramani, Liqun Cheng, and John Carter. Leveraging wire properties at the microarchitecture level. *IEEE MICRO*, 26(6):40–52, Nov/Dec 2006. DOI: 10.1109/MM.2006.123

[21] James Balfour and William J. Dally. Design tradeoffs for tiled CMP on-chip networks. In *Proceedings of the International Conference on Supercomputing*, pages 187–198, 2006. DOI: 10.1145/1183401.1183430

[22] Arnab Banerjee, Robert Mullins, and Simon Moore. A power and energy exploration of network-on-chip architectures. In *Proceedings of the First International Symposium on Networks-on-Chips*, pages 163–172, May 2007. DOI: 10.1109/NOCS.2007.6

[23] Luiz A. Barroso, Kourosh Gharachorloo, Robert McNamara, Andreas Nowatzyk, Shaz Qadeer, Barton Sano, Scott Smith, Robert Stets, and Ben Verghese. Piranha: a scalable architecture based on single-chip multiprocessing. In *Proceedings of the International Symposium on Computer Architecture*, pages 282–293, 2000.

[24] Christopher Batten, Ajay Joshi, Jason Orcutt, Anatoly Khilo, Benjamin Moss, Charles Holzwarth, Miloš Popović, Hanqing Li, Henry Smith, Judy Hoyt, Franz Kärtner, Rajeev Ram, Vladmir Stojanović, and Krste Asanović. Building manycore processor-to-DRAM networks with monolithic silicon photonics. In *Proceedings of the IEEE Symposium on High Performance Interconnects*, pages 21–30, August 2008. DOI: 10.1109/HOTI.2008.11

[25] Shane Bell, Bruce Edwards, John Amann, Rich Conlin, Kevin Joyce, Vince Leung, John MacKay, Mike Reif, Liewei Bao, John Brown, Matthew Mattina, Chyi-Chang Miao, Carl Ramey, David Wentzlaff, Walker Anderson, Ethan Berger, Nat Fairbanks, Durlov Khan, Froilan Montenegro, Jay Stickney, and John Zook. TILE64 processor: A 64-core SoC with mesh interconnect. In *International IEEE Solid-State Circuits Conference*, pages 88–89, Feb 2008. DOI: 10.1109/ISSCC.2008.4523070

[26] Luca Benini, Davide Bertozzi, Alessandro Bogliolo, Francesco Menichelli, and Mauro Olivieri. MPARM: Exploring the multi-processor SoC design space with SystemC. *Journal of VLSI Signal Processing Systems*, 41(2):169–182, September 2005. DOI: 10.1007/s11265-005-6648-1

[27] Luca Benini and Giovanni De Micheli. Powering networks on chips. In *Proceedings of the 14th International Symposium on System Synthesis*, pages 33–38, 2001. DOI: 10.1145/500001.500009

[28] Luca Benini and Giovanni De Micheli. Networks on chips: a new SoC paradigm. *IEEE Computer*, 35(1):70–78, January 2002. DOI: 10.1109/2.976921

[29] Luca Benini and Giovanni De Micheli. *Networks on Chips: Technology and tools*. Academic Press, 2006.

[30] Davide Bertozzi, Antoine Jalabert, Srinivasan Murali, Rutuparna Tamhankar, Stergios Stergiou, Luca Benini, and Giovanni De Micheli. NoC synthesis flow for customized domain specific multiprocessor systems-on-chip. *IEEE Transactions on Parallel and Distributed Systems*, 16(2):113–129, February 2005. DOI: 10.1109/TPDS.2005.22

[31] Christian Bienia, Sanjeev Kumar, Jaswinder Pal Singh, and Kai Li. The parsec benchmark suite: Characterization and architectural implications. In *Proceedings of the 17th International*

Conference on Parallel Architectures and Compilation Techniques, pages 72–81, October 2008. DOI: 10.1145/1454115.1454128

[32] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. *ACM Computer Surveys*, 38(1), 2006. DOI: 10.1145/1132952.1132953

[33] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodny. QNoC: QoS architecture and design processor for cost-effective network on chip. *Special issue on Networks on Chip, The Journal of Systems Architecture*, 50(2):105–128, February 2004. DOI: 10.1016/j.sysarc.2003.07.004

[34] Evgeny Bolotin, Israel Cidon, Ran Ginosar, and Avinoam Kolodony. Routing table minimization for irregular mesh NoCs. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 942–947, 2007.

[35] M. Frank Chang, Jason Cong, Adam Kaplan, Chunyue Liu, Mishali Naik, Jagannath Premkumar, Glenn Reinman, Eran Socher, and Sai-Wang Tam. Power reduction of CMP communication networks via RF-interconnects. In *Proceedings of the 41st Annual International Symposium on Microarchitecture*, pages 376–387, November 2008. DOI: 10.1109/MICRO.2008.4771806

[36] M. Frank Chang, Jason Cong, Adam Kaplan, Mishali Naik, Glenn Reinman, Eran Socher, and Sai-Wang Tam. CMP network-on-chip overlaid with multi-band RF-interconnect. In *Proceedings of the 14th International Symposium on High-Performance Computer Architecture*, pages 191–202, February 2008. DOI: 10.1109/HPCA.2008.4658639

[37] M. Frank Chang, V.P. Roychowdhury, Liyang Zhang, Hyunchoi Shin, and Yongzi Qian. RF/wireless interconnect for inter- and intra-chip communications. In *Proceedings of the IEEE, 89(4)*, pages 456–466, April 2001. DOI: 10.1109/5.920578

[38] Andrew A. Chien and Jae H. Kim. Planar-adaptive routing: low-cost adaptive networks for multiprocessors. In *Proceedings of the International Symposium on Computer Architecture*, pages 268–277, 1992. DOI: 10.1145/139669.140383

[39] Ge-Ming Chiu. The odd-even turn model for adaptive routing. *IEEE Transactions on Parallel and Distributed Systems*, pages 729–738, July 2000. DOI: 10.1109/71.877831

[40] Kypros Constantinides, Stephen Plaza, Jason Blome, Bin Zhang, Valeria Bertacco, Scott Mahlke, Todd Austin, and Michael Orshansky. BulletProof: A defect tolerant CMP switch architecture. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 5–16, 2006. DOI: 10.1109/HPCA.2006.1598108

[41] Pat Conway and Bill Hughes. The AMD Opteron Northbridge architecture, present and future. *IEEE Micro Magazine*, 27:10–21, March 2007. DOI: 10.1109/MM.2007.43

[42] Marcello Coppola, Milto D. Grammtikakis, Riccardo Locatelli, Giuseppe Maruccia, and Lorenzo Pieralisi. *Design of Cost-Efficient Interconnect Processing Units: Spidergon STNoC*. CRC Press, 2008.

[43] Marcello Coppola, Riccardo Locatelli, Giuseppe Maruccio, Lorenzo Pieralisi, and A. Scandurra. Spidergon: a novel on chip communication network. In *International Symposium on System on Chip*, page 15, November 2004. DOI: 10.1109/ISSOC.2004.1411133

[44] Cisco CRS-1. http://www.cisco.com.

[45] D. E. Culler and J. P. Singh. *Parallel Computer Architecture: A Hardware/Software Approach*. Morgan Kaufmann Publishers Inc, 1999.

[46] Matteo Dall'Osso, Gianluca Biccari, Luca Giovannini, Davide Bertozzi, and Luca Benini. xpipes: A latency insensitive parameterized network-on-chip architecture for multi-processor SoCs. In *International Conference on Computer Design*, pages 536–539, 2003.

[47] William Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann Pub., San Francisco, CA, 2003.

[48] William J. Dally. Virtual-channel flow control. In *Proceedings of the International Symposium on Computer Architecture*, 1990. DOI: 10.1109/71.127260

[49] William J. Dally. Express cubes: Improving the performance of k-ary n-cube interconnection networks. *IEEE Transactions on Computers*, 40(9):1016–1023, Sept 1991. DOI: 10.1109/12.83652

[50] William J. Dally and Hiromichi Aoki. Deadlock-free adaptive routing in multicomputer networks using virtual channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, 1993. DOI: 10.1109/71.219761

[51] William J. Dally, Larry R. Dennison, David Harris, Kinhong Kan, and Thucydides Xanthopoulos. The reliable router: A reliable and high-performance communication substrate for parallel computers. In *Proceedings of the First International Workshop on Parallel Computer Routing and Communication*, pages 241–255, 1994. DOI: 10.1007/3-540-58429-3_41

[52] William J. Dally, J. A. Stuart Fiske, John S. Keen, Richard A. Lethin, Michael D. Noakes, Peter R. Nuth, Roy E. Davison, and Gregory A. Fyler. The message-driven processor – a multicomputer processing node with efficient mechanisms. *IEEE Micro*, 12(2):23–39, April 1992. DOI: 10.1109/40.127581

[53] William J. Dally and John W. Poulton. *Digital Systems Engineering*. Cambridge University Press, 1998.

[54] William J. Dally and Charles L. Seitz. The torus routing chip. *Journal of Distributed Computing*, 1(3):187–196, 1986. DOI: 10.1007/BF01660031

[55] William J. Dally and Charles L. Seitz. Deadlock-free message routing in multiprocessor interconnection networks. *IEEE Transactions on Computers*, 36(5):547–553, 1987. DOI: 10.1109/TC.1987.1676939

[56] William J. Dally and Brian Towles. http://cvs.stanford.edu/books/ppin.

[57] William J. Dally and Brian Towles. Route packets, not wires: On-chip interconnection networks. In *Proceedings of the 38th Conference on Design Automation*, pages 684–689, 2001. DOI: 10.1109/DAC.2001.935594

[58] Reetuparna Das, Soumya Eachempati, Asit K. Mishra, N. Vijaykrishnan, and Chita R. Das. Design and evaluation of hierarchical on-chip network topologies for next generation CMPs. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 175–186, February 2009.

[59] Luca Benini Giovanni de Micheli. Networks on chip: A new paradigm for systems on chip design. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 418–419, 2002. DOI: 10.1109/DATE.2002.998307

[60] Martin De Prycker. *Asynchronous Transfer Mode: Solution for Broadband ISDN*. Prentice Hall, 3rd edition, 1995.

[61] José Duato. A new theory of deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 4(12):1320–1331, December 1993. DOI: 10.1109/71.250114

[62] José Duato. A necessary and sufficient condition for deadlock-free adaptive routing in wormhole networks. *IEEE Transactions on Parallel and Distributed Systems*, 6(10):1055–1067, Oct 1995. DOI: 10.1109/71.473515

[63] José Duato, Sudhakar Yalamanchili, and Lionel M. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2nd edition, 2003.

[64] Noel Eisley, Li-Shiuan Peh, and Li Shang. In-network cache coherence. In *Proceedings of the 39th International Symposium on Microarchitecture*, pages 321–332, December 2006. DOI: 10.1109/MICRO.2006.27

[65] Natalie Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Circuit-switched coherence. In *Proceedings of the International Network on Chip Symposium*, pages 193–202, April 2008. DOI: 10.1109/NOCS.2008.22

[66] Natalie Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Virtual circuit tree multicasting: A case for on-chip hardware multicast support. In *International Symposium on Computer Architecture*, pages 229–240, June 2008. DOI: 10.1109/ISCA.2008.12

[67] Natalie Enright Jerger, Li-Shiuan Peh, and Mikko H. Lipasti. Virtual tree coherence: Leveraging regions and in-network multicast trees for scalable cache coherence. In *Proceedings of the 41st International Symposium on Microarchitecture*, pages 35–46, November 2008. DOI: 10.1109/MICRO.2008.4771777

[68] Natalie Enright Jerger, Dana Vantrease, and Mikko H. Lipasti. An evaluation of server consolidation workloads for multi-core designs. In *IEEE International Symposium on Workload Consolidation*, pages 47–56, September 2007. DOI: 10.1109/IISWC.2007.4362180

[69] Jose Flich, Andres Mejia, Pedro López, and José Duato. Region-based routing: An efficient routing mechanism to tackle unreliable hardware in networks on chip. In *Proceedings of the Network on Chip Symposium*, pages 183–194, May 2007. DOI: 10.1109/NOCS.2007.39

[70] Jose Flich, Samuel Rodrigo, and José Duato. An efficient implementation of distributed routing algorithms for NoCs. In *Proceedings of the International Network On Chip Symposium*, pages 87–96, April 2008. DOI: 10.1109/NOCS.2008.4492728

[71] Mike Galles. Scalable pipelined interconnect for distributed endpoint routing: The SGI SPIDER chip. In *Proceedings of Hot Interconnects Symposium IV*, pages 141–146, 1996.

[72] Alan Gara, Matthias A. Blumrich, Dong Chen, George L.-T. Chiu, Paul Coteus, Mark E. Giampapa, Ruud A. Haring, Philip Heidelberger, Dirk Hoenicke, Gerard V. Kopcsay, Thomas A. Liebsch, Martin Ohmacht, Burkhard D. Steinmacher-Burow, Todd Takken, and Pavlos Vranas. Overview of the Blue Gene/L system architecture. *IBM Journal of Research and Developement*, 49(2–3):195–212, 2005.

[73] Patrick T. Gaughan and Sudhakar Yalamanchili. Pipelined circuit-switching: a fault-tolerant variant of wormhole routing. In *Proceedings of the Symposium on Parallel and Distributed Processing*, pages 148–155, December 1992. DOI: 10.1109/SPDP.1992.242751

[74] N. Genko, D. Atienza, G. De Micheli, J. Mendias, R. Hermida, and F. Catthoor. A complete network-on-chip emulation framework. In *Proceedings of the Conference on Design Automation and Test in Europe*, pages 246–251, March 2005. DOI: 10.1109/DATE.2005.5

[75] Christopher J. Glass and Lionel M. Ni. The turn model for adaptive routing. In *Proceedings of the International Symposium on Computer Architecture*, pages 278–287, May 1992. DOI: 10.1145/146628.140384

[76] Nitin Godiwala, Jud Leonard, and Matthew Reilly. A network fabric for scalable multiprocessor systems. In *Proceedings of the Symposium on Hot Interconnects*, pages 137–144, 2008. DOI: 10.1109/HOTI.2008.24

[77] Kees Goossens, John Dielissen, Om Prakash Gangwal, Santiago Gonzalez Pestana, Andrei Radulescu, and Edwin Rijpkema. A design flow for application-specific networks on chip with guaranteed performance to accelerate SoC design and verification. In *Proceedings of the Design, Automation and Test in Europe Conference*, pages 1182–1187, March 2005. DOI: 10.1109/DATE.2005.11

[78] Kees Goossens, John Dielissen, and Andrei Radulescu. Æthereal network on chip: Concepts, architectures, and implementations. *IEEE Design and Test*, 22(5):414–421, September 2005. DOI: 10.1109/MDT.2005.99

[79] Paul Gratz, Boris Grot, and Stephen W. Keckler. Regional congestion awareness for load balance in networks-on-chip. In *Proceedings of the 14th IEEE International Symposium on High Performance Computer Architecture*, pages 203–214, Feb 2008. DOI: 10.1109/HPCA.2008.4658640

[80] Paul Gratz, Changkyu Kim, Robert G. McDonald, Stephen W. Keckler, and Doug Burger. Implementation and evaluation of on-chip network architectures. In *IEEE International Conference on Computer Design*, pages 477–484, October 2006. DOI: 10.1109/ICCD.2006.4380859

[81] Paul Gratz, Karthikeyan Sankaralingam, Heather Hanson, Premkishore Shivakumar, Robert G. McDonald, Stephen W. Keckler, and Doug Burger. Implementation and evaluation of a dynamically routed processor operand network. In *Proceedings of the International Network on Chip Symposium*, pages 7–17, 2007. DOI: 10.1109/NOCS.2007.23

[82] Cristian Grecu, Andrè Ivanov, Partha Pande, Axel Jantsch, Erno Salminen, Umit Ogras, and Radu Marculescu. An initiative toward open network on chip benchmarks. OCP-IP website http://www.ocpip.org/socket/whitepapers/NoC-Benchmarks-WhitePaper-15.pdf, March 2007.

[83] Boris Grot, Joel Hestness, Stephen W. Keckler, and Onur Mutlu. Express cube topologies for on-chip networks. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 163–174, February 2009. DOI: 10.1109/HPCA.2009.4798251

[84] SMART Interconnects Group. Flexsim 1.2. http://ceng.usc.edu/smart/tools.html.

[85] M. Gschwind, B. D'Amora, K. O'Brien, and A. Eichenberger. Cell broadband engine - enabling density computing for data-rich environment. Tutorial held in conjunction with the International Symposium on Computer Architecture, June 2006.

[86] Cory Hawkins, Benjamin A. Small, D. Scott Willis, and Keren Bergman. The data vortex, an all optical path multicomputer interconnection network. *IEEE Transactions on Parallel and Distributed Systems*, 18(3):409–420, 2007. DOI: 10.1109/TPDS.2007.48

[87] Jörg Henkel, Wayne Wolf, and Srimat T. Chakradhar. On-chip networks: A scalable, communication-centric embedded system design paradigm. In *Proceedings of VLSI Design*, pages 845–851, January 2004. DOI: 10.1109/ICVD.2004.1261037

[88] John L. Hennessy and David A. Patterson. *Computer Architecture: A Quantitative Approach*. Morgan Kaufmann Publishers Inc, 4th edition, 2006. DOI: 10.1145/1077603.1077692

[89] Seongmoo Heo and Krste Asanović. Replacing global wires with an on-chip network: A power analysis. In *Proceedings of ISLPED*, pages 369–374, 2005. DOI: 10.1109/JSSC.2007.910807

[90] Ron Ho, T. Ono, R.D. Hopkins, A. Chow, J. Schauer, F.Y. Liu, and R. Drost. High speed and low energy capacitively driven on-chip wires. *IEEE Journal of Solid-State Circuits*, 43(1):52–60, January 2008. doi:10.1109/JSSC.2007.910807. DOI: 10.1109/JSSC.2007.910807

[91] Wai Hong Ho and Timothy Mark Pinkston. A methodology for designing efficient on-chip interconnects on well-behaved communication patterns. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 377–388, February 2003. DOI: 10.1109/HPCA.2003.1183554

[92] Wai Hong Ho and Timothy Mark Pinkston. A design methodology for efficient application-specific on-chip interconnects. *IEEE Transactions on Parallel and Distributed Systems*, 17(2):174–190, February 2006. DOI: 10.1109/TPDS.2006.15

[93] Jeff Hoffman, David A. Ilitzky, Anthony Chun, and Aliaksei Chapyzhenka. Architecture of the scalable communications core. In *Proceedings of the First International Symposium on Networks-on-Chip*, pages 40–52, May 2007. DOI: 10.1109/NOCS.2007.11

[94] H. Peter Hofstee. Power efficient processor architecture and the cell processor. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 258–262, February 2005. DOI: 10.1109/HPCA.2005.26

[95] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar. A 5-GHz mesh interconnect for a Teraflops processor. *IEEE MICRO*, 27(5):51–61, 2007. DOI: 10.1109/MM.2007.4378783

[96] Jingcao Hu and Radu Marculescu. Exploiting the routing flexibility for energy/performance aware mapping of regular NoC architectures. In *Proceedings of the Conference on Design, Automation and Test Europe*, pages 688–693, March 2003.

[97] Jingcao Hu and Radu Marculescu. DyAD–smart routing for networks-on-chip. In *Proceedings of the Design Automation Conference*, pages 260–263, June 2004. DOI: 10.1145/996566.996638

[98] Jingcao Hu and Radu Marculescu. Energy- and performance-aware mapping for regular NoC architectures. *IEEE Transactions on Computer Aided Design for Integrated Circuits Systems*, 24(4):551–562, April 2005. DOI: 10.1109/TCAD.2005.844106

[99] Jingcao Hu, Umit Y. Ogras, and Radu Marculescu. System-level buffer allocation for application specific networks-on-chip router design. *IEEE Transactions on Computer-Aided Design for Integrated Circuits System*, 25(12):2919–2933, December 2006. DOI: 10.1109/TCAD.2006.882474

[100] Paolo Ienne, Patrick Thiran, Giovanni De Micheli, and Frédéric Worm. An adaptive low-power transmission scheme for on-chip networks. In *Proceedings of the International Symposium on Systems Synthesis*, pages 92–100, 2002. DOI: 10.1145/581199.581221

[101] Cadence Inc. http://www.cadence.com/products/di/edi_system/pages/default.aspx.

[102] Intel. Intel© QuickPath technology. http://www.intel.com/technology/quickpath.

[103] Intel. From a few cores to many: A Tera-scale computing research overview, 2006.

[104] Antoine Jalabert, Srinivasan Murali, Luca Benini, and Giovanni De Micheli. xpipesCompiler: A tool for instantiating application specific networks on chip. In *Proceedings of the Conference on Design, Automation and Test in Europe*, volume 2, pages 884–889, February 2004. DOI: 10.1109/DATE.2004.1268999

[105] Axel Jantsch, Ian Mackintosh, John Bainbridge, Radu Marculescu, Timothy Pinkston, and Drew Wingard. Proliferating the use and acceptance of NoC benchmark standards. Panel Session in Conjunction with the 1st International Symposium on Networks-on-Chip, May 2007. http://2007.nocsymposium.org/panel_discussion.html.

[106] A.P. Jose, G. Patounakis, and K.L. Shepard. Near speed-of-light on-chip interconnects using pulsed current-mode signaling. In *Symposium on VLSI Circuits*, pages 108–111, June 2005. DOI: 10.1109/VLSIC.2005.1469345

[107] Norman P. Jouppi. System implications of integrated photonics. In *Proceedings of the International Symposium on Low Power Electronics and Design*, pages 183–184, August 2008. DOI: 10.1145/1393921.1393923

[108] J. A. Kahl, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the Cell multiprocessor. *IBM Journal of Research and Development*, 49(4):589–604, 2005.

[109] Andrew Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. Orion 2.0: A fast and accurate NoC power and area model for early-stage design space exploration. In *Proceedings of the Conference on Design, Automation and Test in Europe*, April 2009.

[110] Parviz Kermani and Leonar Kleinrock. Virtual cut-through: a new computer communication switching technique. *Computer Networks*, 3(4):267–286, 1979.

[111] B. Kim and V. Stojanović. A 4Gb/s/ch 356fj/b 10mm equalized on-chip interconnect with nonlinear charge-injecting transmitter filter and transimpedance receiver in 90nm CMOS technology. In *IEEE Solid-State Circuits Conference*, February 2009.

[112] Changkyu Kim, Doug Burger, and Stephen W. Keckler. An adaptive, non-uniform cache structure for wire-delay dominated on-chip caches. In *Proceedings of the 10th International Conference on Architectural Support for Programming Languages and Operating System*, pages 211–222, 2002. DOI: 10.1145/605432.605420

[113] John Kim, James Balfour, and William Dally. Flattened butterfly topology for on-chip networks. In *Proceedings of the 40th International Symposium on Microarchitecture*, pages 172–182, December 2007.

[114] John Kim, William Dally, Steve Scott, and Dennis Abts. Technology-driven, highly-scalable dragonfly topology. In *Proceedings of the International Symposium on Computer Architecture*, pages 194–205, June 2008. DOI: 10.1109/ISCA.2008.19

[115] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Reetuparna Das, Yuan Xie, N. Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. A novel dimensionally-decomposed router for on-chip communication in 3d architectures. In *International Symposium on Computer Architecture*, pages 138–149, June 2007. DOI: 10.1145/1273440.1250680

[116] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, N. Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. A gracefully degrading and energy-efficient modular router architecture for on-chip networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 4–15, June 2006. DOI: 10.1145/1150019.1136487

[117] Jongman Kim, Dongkook Park, T. Theocharides, N. Vijaykrishnan, and Chita R. Das. A low latency router supporting adaptivity for on-chip interconnects. In *International Conference on Design Automation*, pages 559–564, 2005. DOI: 10.1145/1065579.1065726

[118] Nevin Kirman, Meyrem Kirman, Rajeev K. Dokania, Jose F. Martinez, Alyssa B. Apsel, Matthew A. Watkins, and David H. Albonesi. Leveraging optical technology in future bus-based chip multiprocessors. In *Proceedings of the International Symposium on Microarchitecture*, pages 492–503, December 2006. DOI: 10.1109/MICRO.2006.28

[119] Michael Kistler, Michael Perrone, and Fabrizio Petrini. Cell multiprocessor communication network: Built for speed. *IEEE MICRO*, 26(3):10–23, May 2006. DOI: 10.1109/MM.2006.49

[120] Poonacha Kongetira, Kathirgamar Aingaran, and Kunle Olukotun. Niagara: A 32-way multithreaded SPARC processor. *IEEE Micro*, 25(2):21–29, 2005. DOI: 10.1109/MM.2005.35

[121] Rajesh Kota. HORUS: Large scale SMP using AMD Opteron™. http://www.hypertransport.org/docs/tech/horus_external_white_paper_final.pdf.

[122] Tushar Krishna, Amit Kumar, Patrick Chiang, Mattan Erez, and Li-Shiuan Peh. NoC with near-ideal express virtual channels using global-line communication. In *Proceedings of Hot Interconnects*, pages 11–20, August 2008. DOI: 10.1109/HOTI.2008.22

[123] John Kubiatowicz and Anant Agarwal. The anatomy of a message in the Alewife multiprocessor. In *Proceedings of the International Conference on Supercomputing*, pages 195–206, July 1993. DOI: 10.1145/165939.165970

[124] Matthias Kühnle, Michael Hübner, Jürgen Becker, Antonio Deledda, Claudio Mucci, Florian Ries, Antonio Marcello Coppola, Lorenzo Pieralisi, Riccardo Locatelli, Giuseppe Maruccia, Tommaso DeMarco, and Fabio Campi. An interconnect strategy for a heterogeneous, reconfigurable SoC. *IEEE Design and Test of Computers*, 25(5):442–451, Sept/Oct 2008. DOI: 10.1109/MDT.2008.150

[125] Amit Kumar, Partha Kundu, Arvind Singh, Li-Shiuan Peh, and Niraj K. Jha. A 4.6Tbits/s 3.6GHz single-cycle NoC router with a novel switch allocator in 65nm CMOS. In *Proceedings of the International Conference on Computer Design*, pages 63–70, October 2007. DOI: 10.1109/ICCD.2007.4601881

[126] Amit Kumar, Li-Shiuan Peh, and Niraj K Jha. Token flow control. In *Proceedings of the 41st International Symposium on Microarchitecture*, pages 342–353, Lake Como, Italy, November 2008. DOI: 10.1109/MICRO.2008.4771803

[127] Amit Kumar, Li-Shiuan Peh, Partha Kundu, and Niraj K. Jha. Express virtual channels: Toward the ideal interconnection fabric. In *Proceedings of 34th Annual International Symposium on Computer Architecture*, pages 150–161, San Diego, CA, June 2007. DOI: 10.1145/1273440.1250681

[128] Shashi Kumar, Axel Jantsch, Juha-Pekka Soininen, M. Forsell, Mikael Millberg, Johnny Öberg, Kari Tiensyrjä, and Ahmed Hemani. A network on chip architecture and design methodology. In *Proceedings of the IEEE Computer Society Annual Symposium on VLSI*, pages 105–112, April 2002.

[129] James Laudon and Daniel Lenoski. The SGI Origin: a ccNUMA highly scalable server. In *Proceedings of the 24th Annual International Symposium on Computer Architecture*, pages 241–251, May 1997. DOI: 10.1145/384286.264206

[130] Kangmin Lee, Se-Joong Lee, Sung-Eun Kim, Hye-Mi Choi, Donghyun Kim, Sunyoung Kim, Min-Wuk Lee, and Hoi-Jun Yoo. A 51mW 1.6GHz on-chip network for low-power heterogeneous SoC platform. In *Proceedings of the International Solid-State Circuits Conference*, pages 152–153, Feb 2004. DOI: 10.1109/ISSCC.2004.1332639

[131] Se-Joong Lee, Kangmin Lee, Seong-Jun Song, and Hoi-Jun Yoo. Packet-switched on-chip interconnection network for system-on-chip applications. *IEEE Transactions on Circuits and Systems, Part II: Express Briefs*, 52(6):308–312, June 2005. DOI: 10.1109/TCSII.2005.848972

[132] Whay Sing Lee, William J. Dally, Stephen W. Keckler, Nicholas P. Carter, and Andrew Chang. An efficient protected message interface in the MIT M-Machine. *IEEE Computer Special Issue on Design Challenges for High Performance Network Interfaces*, 31(11):69–75, November 1998. DOI: 10.1109/2.730739

[133] Charles Leiserson. Fat-trees: Universal networks for hardware-efficient supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, October 1985.

[134] Feihui Li, Chrysostomos Nicopoulos, Thomas Richardson, Yuan Xie, N. Vijaykrishnan, and Mahmut Kandemir. Design and management of 3D chip multiprocessors using network-in-memory. In *Proceedings of the International Symposium on Computer Architecture*, pages 130–141, June 2006. DOI: 10.1145/1150019.1136497

[135] Zheng Li, Jie Wu, Li Shang, Robert Dick, and Yihe Sun. Latency criticality aware on-chip communication. In *Proceedings of the IEEE Conference on Design, Automation, and Test in Europe*, March 2009.

[136] Weiping Liao and Lei He. Full-chip interconnect power estimation and simulation considering repeater insertion and flip-flop insertion. In *Proceedings of the International Conference on Computer-Aided Design*, pages 574–580, November 2003. DOI: 10.1109/ICCAD.2003.75

[137] Erik Lindholm, John Nickolls, Stuart Oberman, and John Montrym. NVIDIA Tesla: A unified graphics and computing architecture. *IEEE Micro*, 28(2):39–55, March-April 2008. DOI: 10.1109/MM.2008.31

[138] Zhonghai Lu, Ming Lui, and Axel Jantsch. Layered switching for networks on chip. In *Proceedings of the conference on Design Automation*, pages 122–127, San Diego, CA, June 2007.

[139] Radu Marculescu, Umit Y. Ogras, Li-Shiuan Peh, Natalie Enright Jerger, and Yatin Hoskote. Outstanding research problems in NoC design: System, microarchitecture, and circuit perspectives. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 28(1):3–21, January 2009. DOI: 10.1109/TCAD.2008.2010691

[140] Théodore Marescaux, Jean-Yves Mignolet, Andrei Bartic, W. Moffat, Diederik Verkest, Serge Vernalde, and Rudy Lauwereins. Networks on chip as a hardware component of an OS for

reconfigurable systems. In *Proceedings of Field Programmable Logic*, pages 595–605, 2003. DOI: 110.1007/b12007

[141] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Michael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. Hill, and David A. Wood. Multifacet's general execution-driven multiprocessor simulator (GEMS) toolset. *Computer Architecture News*, 33(4):92–99, September 2005.

[142] Michael R. Marty and Mark D. Hill. Coherence ordering for ring-based chip multiprocessors. In *Proceedings of the 39th International Symposium on Microarchitecture*, pages 309–320, December 2006. DOI: 10.1109/MICRO.2006.14

[143] Michael R. Marty and Mark D. Hill. Virtual hierarchies to support server consolidation. In *Proceedings of the International Symposium on Computer Architecture*, pages 46–56, June 2007. DOI: 10.1145/1250662.1250670

[144] Hiroki Matsutani, Michihiro Koibuchi, Hideharu Amano, and Tsutomu Yoshinaga. Prediction router: Yet another low latency on-chip router architecture. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 367–378, February 2009. DOI: 10.1109/HPCA.2009.4798274

[145] George Michelogiannakis, James Balfour, and William J. Dally. Elastic-buffer flow control for on-chip networks. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 151–162, February 2009. DOI: 10.1109/HPCA.2009.4798250

[146] ST Microelectronics. http://www.st.com.

[147] ST Microelectronics. STBus interconnect. http://www.st.com/stonline/products/ technologies/soc/stbus.htm.

[148] Mikael Millberg, Erland Nilsson, Rikard Thid, and Axel Jantsch. Guaranteed bandwidth using looped containers in temporally disjoint networks within the Nostrum network-on-chip. In *Proceedings of the conference on Design, Automation and Testing in Europe (DATE)*, pages 890–895, 2004. DOI: 10.1109/DATE.2004.1269001

[149] D.A.B. Miller. Rationale and challenges for optical interconnects to electronic chips. *Proceedings of the IEEE*, 88(6):728–749, June 2000. DOI: 10.1109/5.867687

[150] Thomas Moscibroda and Onur Mutlu. A case for bufferless routing in on-chip networks. In *Proceedings of the 36th International Symposium on Computer Architecture*, June 2009.

[151] Shubhendu S. Mukherjee, Peter Bannon, Steven Lang, Aaron Spink, and David Webb. The Alpha 21364 network architecture. *IEEE Micro*, 22(1):26–35, 2002. DOI: 10.1109/40.988687

[152] Robert Mullins. Netmaker interconnection networks. http://www-dyn.cl.cam.ac.uk/~rdm34/wiki/index.php?title=Main_Page.

[153] Robert Mullins, Andrew West, and Simon Moore. Low-latency virtual-channel routers for on-chip networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 188–197, June 2004. DOI: 10.1109/ISCA.2004.1310774

[154] Robert Mullins, Andrew West, and Simon Moore. The design and implementation of a low-latency on-chip network. In *Proceedings of the 11th Asia and South Pacific Design Automation Conference*, pages 164–169, Yokohama, Japan, January 2006. DOI: 10.1109/ASPDAC.2006.1594676

[155] Robert D. Mullins. An on-chip network bibliography. http://www.cl.cam.ac.uk/users/rdm34/onChipNetBib/browser.htm, 2007.

[156] Srinivasan Murali and Giovanni De Micheli. Bandwidth-constrained mapping of cores onto NoC architectures. In *Proceedings of the Conference for Design, Automation and Test in Europe*, pages 896–901, February 2004.

[157] Srinivasan Murali, Paolo Meloni, Federico Angiolini, David Atienza, Salvatore Carta, Luca Benini, Giovanni De Micheli, and Luigi Raffo. Designing application-specific networks on chips with floorplan information. In *International Conference on Computer-Aided Design*, pages 355–362, November 2006. DOI: 10.1109/ICCAD.2006.320058

[158] Ted Nesson and S. Lennart Johnsson. ROMM routing on mesh and torus networks. In *Proceedings of the Symposium on Parallel Algorithms and Architectures*, pages 275–287, 1995.

[159] Chrysostomos A. Nicopoulos, Dongkook Park, Jongman Kim, N. Vijaykrishnan, Mazin S. Yousif, and Chita R. Das. ViChaR: A dynamic virtual channel regulator for network on-chip routers. In *International Symposium on Microarchitecture*, pages 333–344, December 2006.

[160] Erland Nilsson, Mikael Millberg, Johnny Oberg, and Axel Jantsch. Load distribution with proximity congestion awareness in a network on chip. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 1126–1127, March 2003.

[161] Peter R. Nuth and William J. Dally. The J-Machine network. In *Proceedings of the International Conference on Computer Design*, pages 420–423, October 1992.

[162] Umit Y. Ogras, Jingcao Hu, and Radu Marculescu. Key research problems in NoC design: A holistic perspective. In *Proceedings of the International Conference on Hardware-Software Codesign Systems and Synthesis*, pages 69–74, September 2005.

[163] Umit Y. Ogras and Radu Marculescu. Energy- and performance-driven NoC communication architecture synthesis using a decomposition approach. In *Proceedings of the conference on Design, Automation and Test in Europe*, pages 352–357, March 2005.

[164] Umit Y. Ogras and Radu Marculescu. "It's a small world after all": NoC performance optimization via long-range link insertion. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems - Special Section Hardware/Software Codesign System Synthesis*, 14(7):693–706, July 2006.

[165] Kunle Olukotun, Basem A. Nayfeh, Lance Hammond, Ken Wilson, and Kun-Yung Chang. The case for a single-chip multiprocessor. In *Proceedings of the International Symposium on Architectural Support for Parallel Languages and Operating Systems*, pages 2–11, October 1996.

[166] Ralph H. Otten and Robert K. Brayton. Planning for performance. In *Proceedings of the conference on Design Automation*, pages 122–127, June 1998.

[167] John D. Owens, William J. Dally, Ron Ho, D. N. Jayasimha, Stephen W. Keckler, and Li-Shiuan Peh. Research challanges for on-chip interconnection networks. *IEEE Micro, Special Issue on On-Chip Interconnects for Multicores*, 27(5):96–108, September/October 2007. DOI: 10.1109/MM.2007.4378787

[168] Maurizio Palesi, Rickard Holsmark, Shashi Kumar, and Vincenzo Catania. A methodology for design of application specific deadlock-free routing algorithms for NoC systems. In *Proceedings of the International Conference on Hardware-Software Codesign Systems and Synthesis*, pages 142–147, October 2006. DOI: 10.1145/1176254.1176289

[169] Yan Pan, Prabhat Kumar, John Kim, Gokhan Memik, Yu Zhang, and Alok Choudhary. Firefly: Illuminating future network-on-chip with nanophotonics. In *Proceedings of the International Symposium on Computer Architecture*, June 2009.

[170] Dongkook Park, Reetuparna Das, Chrysostomos Nicopoulos, Jongman Kim, N. Vijaykrishnan, Ravishankar Iyer, and Chita R. Das. Design of a dynamic priority-based fast path architecture for on-chip interconnects. In *Proceedings of the 15th IEEE Symposium on High-Performance Interconnects*, pages 15–20, August 2007. DOI: 10.1109/HOTI.2007.1

[171] Dongkook Park, Soumya Eachempati, Reetuparna Das, Asit K. Mishra, Yuan Xie, N. Vijaykrishnan, and Chita R. Das. MIRA: A multi-layered on-chip interconnect router architecture. In *Proceedings of the International Symposium on Computer Architecture*, pages 251–261, June 2008.

[172] Sudeep Pasricha and Nikil Dutt. *On-Chip Communication Architectures: System on Chip Interconnect*. Morgan Kaufmann, 2008.

[173] Li-Shiuan Peh and William J. Dally. Flit-reservation flow control. In *Proceedings of the 6th International Symposium on High Performance Computer Architecture*, pages 73–84, February 2000.

[174] Li-Shiuan Peh and William J. Dally. A delay model and speculative architecture for pipelined routers. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 255–266, January 2001.

[175] Li-Shiuan Peh and William J. Dally. A delay model for router microarchitectures. *IEEE Micro*, 21(1):26–34, January 2001. DOI: 10.1109/40.903059

[176] D. Pham, T. Aipperspach, D. Boerstler, M. Bolliger, R. Chaudhry, D. Cox, P. Harvey, H. P. Hofstee, C. Johns, J. Kahle, A. Kameyama, J. Keaty, Y. Masubuchi, M. Pham, J. Pille, S. Posluszny, M. Riley, D. Stasiak, M. Suzuoki, O. Takahashi, J. Warnock, S. Weitzel, D. Wendel, and K. Yazawa. Overview of the architecture, circuit design, and physical implementation of a first-generation cell processor. *IEEE Journal of Solid-State Circuits*, 41(1):179–196, 2006. DOI: 10.1109/JSSC.2005.859896

[177] Timothy Mark Pinkston, Yungho Choi, and Mongkol Raksapatcharawong. Architecture and optoelectronic implementation of the WARRP router. In *Proceedings of Symposium on Hot Interconnects*, 1997.

[178] Timothy Mark Pinkston and José Duato. Appendix E: Interconnection networks. In John L. Hennessy and David A. Patterson, editors, *Computer Architecture: A Quantitative Approach*, pages 1–114. Elsevier Publishers, 4th edition, September 2006.

[179] Alessandro Pinto, Luca P. Carloni, and Alberto L. Sangiovanni-Vincentelli. Efficient synthesis of networks on chip. In *Proceedings of the International Conference on Computer Design*, pages 146–150, October 2003. DOI: 10.1109/ICCD.2003.1240887

[180] Open Core Protocol. http://www.ocpip.org/home/.

[181] V. Puente, J. A. Gregorio, and R. Beivide. SICOSYS: An integrated framework for studying interconnection networks in multiprocessor systems. In *Proceedings of the IEEE 10th Euromicro Workshop on Parallel and Distributed Processing*, pages 15–22, January 2002. DOI: 10.1109/EMPDP.2002.994207

[182] Antonio Pullini, Federico Angiolini, Davide Bertozzi, and Luca Benini. Fault tolerance overhead in network-on-chip flow control schemes. In *Proceedings of the Symposium on Integrated and Circuits System Design*, pages 224–229, Sept 2005. DOI: 10.1109/SBCCI.2005.4286861

[183] Antonio Pullini, Federico Angiolini, Paolo Meloni, David Atienza, Srinivasan Murali, Luigi Raffo, Giovanni De Micheli, and Luca Benini. NoC design implementation in 65nm technology. In *Proceedings of the First International Symposium on Networks-on-Chip*, pages 273–282, May 2007. DOI: 10.1109/NOCS.2007.30

[184] Antonio Pullini, Federico Angiolini, Srinivasan Murali, David Atienza, Giovanni De Micheli, and Luca Benini. Bringing NoCs to 65 nm. *IEEE Micro*, 27(5):75–85, 2007. DOI: 10.1109/MM.2007.4378785

[185] José Renau, B. Fraguela, James Tuck, Wei Liu, Milos Prvulovic, Luis Ceze, Karin Strauss, Smruti Sarangi, Paul Sack, and Pablo Montesinos. SESC simulator. http://sesc.sourceforge.net.

[186] Samuel Rodrigo, Jose Flich, José Duato, and Mark Hummel. Efficient unicast and multicast support for CMPs. In *Proceedings of the International Symposium on Microarchitecture*, pages 364–375, November 2008.

[187] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Jaehyuk Huh, Changkyu Kim, Doug Burger, Stephen W. Keckler, and Charles R. Moore. Exploiting ILP, TLP, and DLP using polymorphism in the TRIPS architecture. In *Proceedings of the 30th Annual International Symposium on Computer Architecture*, pages 422–433, June 2003.

[188] Graham Schelle and Dirk Grunwald. On-chip interconnect exploration for multicore processors utilizing FPGAs. In *Proceedings of the 2nd Workshop on Architecture Research using FPGA Platforms*, February 2006.

[189] Steve Scott, Dennis Abts, John Kim, and William J. Dally. The BlackWidow high-radix Clos network. In *Proceedings of the International Symposium on Computer Architecture*, pages 16–27, June 2006.

[190] Larry Seiler, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, Jeremy Sugerman, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, and Pat Hanrahan. Larrabee: A many-core x86 architecture for visual computing. *ACM Transactions on Graphics*, 27, August 2008.

[191] Daeho Seo, Akif Ali, Won-Taek Lim, Nauman Rafique, and Mithuna Thottenhodi. Near-optimal worst-case throughput routing for two-dimensional mesh networks. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 432–443, June 2005.

[192] Assaf Shacham, Keren Bergman, and Luca P. Carloni. On the design of a photonic network on chip. In *Proceedings of International Symposium on Network on Chip*, pages 53–64, May 2007. DOI: 10.1109/NOCS.2007.35

[193] Arjun Singh, William J. Dally, Amit K. Gupta, and Brian Towles. GOAL: A load-balanced adaptive routing algorithm for torus networks. In *Proceedings of the International Symposium on Computer Architecture*, pages 194–205, June 2003.

[194] Arjun Singh, William J. Dally, Brian Towles, and Amit K. Gupta. Locality-preserving randomized oblivious routing on torus networks. In *SPAA*, pages 9–13, 2002.

[195] Yong Ho Song and Timothy Mark Pinkston. A progressive approach to handling message-dependent deadlocks in parallel computer systems. *IEEE Transactions on Parallel and Distributed Systems*, 14(3):259–275, March 2003. DOI: 10.1109/TPDS.2003.1189584

[196] Sonics Inc. http://www.sonicsinc.com/home/htm.

[197] Krishnan Srinivasan and Karam S. Chatha. A low complexity heuristic for design of custom network-on-chip architectures. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 130–135, 2006.

[198] S. Stergiou, E Angiolini, D. Bertozzi, S. Carta, L. Raffo, and G. De Micheli. xpipesLite: A synthesis-oriented design flow for networks on chip. In *Proceedings of the Conference on Design, Automation and Test Europe*, pages 1188–1193, 2005.

[199] Steven Swanson, Ken Michelson, Andrew Schwerin, and Mark Oskin. Wavescalar. In *Proceedings of the 36th International Symposium on Microarchitecture*, pages 291–302, 2003.

[200] Y. Tamir and H. C. Chi. Symmetric crossbar arbiters for VLSI communication switches. *IEEE Transactions Parallel and Distributed Systems*, 4(1):13–27, 1993. DOI: 10.1109/71.205650

[201] Yuval Tamir and Gregory L. Frazier. Dynamically-allocated multi-queue buffers for VLSI communication switches. *IEEE Transactions on Computers*, 41(6):725–737, June 1992. DOI: 10.1109/12.144624

[202] Michael Bedford Taylor, Walter Lee, Saman Amarasinghe, and Anant Agarwal. Scalar operand networks: On-chip interconnect for ILP in partitioned architectures. In *Proceedings of the International Symposium on High Performance Computer Architecture*, pages 341–353, February 2003.

[203] J. Tendler, J. Dodson, J.S. Fields, H. Le, and B. Sinharoy. Power4 system microarchitecture. *IBM Journal of Research and Development*, 46(1):5–26, 2002.

[204] Marc Tremblay and Shailender Chaudhry. A third-generation 65nm 16-core 32-thread plus 32-scout-thread CMT SPARC processor. In *Proceedings of the International Solid-State Circuits Conference*, 2008.

[205] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 263–277, 1981.

[206] J.W. van den Brand, C. Ciordas, K. Goossens, and T. Basten. Congestion-controlled best-effort communication for networks-on-chip. In *Proceedings of the Conference on Design, Automation and Test in Europe*, pages 948–953, April 2007.

[207] S. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, P. Iyer, A. Singh, T. Jacob, S. Jain, S. Venkataraman, Y. Hoskote, and N. Borkar. An 80-tile 1.28 TFLOPS network-on-chip in 65nm CMOS. In *International Solid-State Circuits Conference*, pages 98–99, Feb 2007.

[208] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, V. Erraguntla, C. Roberts, Y. Hoskote, N. Borkar, and S. Borkar. An 80-tile sub-100-w TeraFLOPS processor in 65-nm CMOS. *IEEE Journal of Solid State Circuits*, 43(1):29–41, January 2008. DOI: 10.1109/JSSC.2007.910957

[209] Dana Vantrease, Robert Schreiber, Matteo Monchiero, Moray McLaren, Norman P. Jouppi, Marco Fiorentino, Al Davis, Nathan L. Binkert, Raymond G. Beausoleil, and Jung Ho Ahn. Corona: System implications of emerging nanophotonic technology. In *International Symposium on Computer Architecture*, pages 153–164, June 2008.

[210] Hang-Sheng Wang, Li-Shiuan Peh, and Sharad Malik. Power-driven design of router microarchitectures in on-chip networks. In *Proceedings of the 36th International Symposium on Microarchitecture*, pages 105–116, November 2003.

[211] Hang-Sheng Wang, Xinping Zhu, Li-Shiuan Peh, and Sharad Malik. Orion: A power-performance simulator for interconnection networks. In *Proceedings of the 35th International Symposium on Microarchitecture*, pages 294–305, November 2002.

[212] David Wentzlaff, Patrick Griffin, Henry Hoffman, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John Brown III, and Anant Agarwal. On-chip interconnection architecture of the Tile processor. *IEEE Micro*, 27(5):15–31, 2007. DOI: 10.1109/MM.2007.4378780

[213] Daniel Wiklund and Dake Lui. SoCBus: Switched network on chip for hard real time embedded systems. In *Proceedings of the International Parallel and Distributed Processing Symposium*, pages 8–16, April 2003. DOI: 10.1109/IPDPS.2003.1213180

[214] Pascal T. Wolkotte, Gerard J .M Smit, Gerard K. Rauwerda, and Lodewijk T. Smit. An energy-efficient reconfigurable circuit-switched network-on-chip. In *Proceedings of the 19th International Parallel and Distributed Processing Symposium*, pages 155–162, 2005. DOI: 10.1109/IPDPS.2005.95

[215] Steven C. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta. The SPLASH-2 programs: Characterization and methodological considerations. In *Proceedings of the International Symposium on Computer Architecture*, pages 24–36, June 1995.

[216] Frédéric Worm, Paolo Ienne, Patrick Thiran, and Giovanni De Micheli. An adaptive low-power transmission scheme for on-chip networks. In *ISSS '02: Proceedings of the 15th international symposium on System Synthesis*, pages 92–100, New York, NY, USA, 2002. ACM. DOI: 10.1145/581199.581221

[217] Yi Xu, Yu Du, Bo Zhao, Xiuyi Zhou, Youtao Zhang, and Jun Yang. A low-radix and low-diameter 3D interconnection network design. In *International Symposium on High Performance Computer Architecture*, pages 30–42, February 2009. DOI: 10.1109/HPCA.2009.4798234

[218] Bilal Zafar, Timothy Mark Pinkston, Aurelio Bermúdez, and José Duato. Deadlock-free dynamic reconfiguration over Infiniband™ networks. *Parallel Algorithms Appl.*, 19(2–3):127–143, 2004.

Biography

NATALIE ENRIGHT JERGER

Natalie Enright Jerger is an Assistant Professor at the University of Toronto. She joined the faculty in the Edward S. Rogers Sr. Department of Electrical and Computer Engineering in January 2009. Prior to that she completed her PhD at the University of Wisconsin-Madison in 2008. She received her Master of Science degree from the University of Wisconsin-Madison and Bachelor of Science in Computer Engineering from Purdue University in 2004 and 2002 respectively. Her research focuses on parallel architectures, on-chip networks and cache coherence protocols and is funded by the Natural Sciences and Engineering Research Council of Canada.

LI-SHIUAN PEH

Li-Shiuan Peh is Associate Professor of Electrical Engineering at Princeton University and has been at Princeton since 2002. She graduated with a Ph.D. in Computer Science from Stanford University in 2001, and a B.S. in Computer Science from the National University of Singapore in 1995. Her research focuses on low- power interconnection networks, on-chip networks and parallel computer architectures, and is funded by several grants from the National Science Foundation, the DARPA MARCO Gigascale Systems Research Center and Interconnect Focus Center as well as Intel Corporation. She was awarded the CRA Anita Borg Early Career Award in 2007, Sloan Research Fellowship in 2006, and the NSF CAREER award in 2003.