ARCHITECTING ON-CHIP INTERCONNECTION NETWORK
FOR FUTURE MANY-CORE CHIP-MULTIPROCESSORS

A Dissertation in
Computer Science and Engineering
by
Jin Ouyang

© 2012 Jin Ouyang

Submitted in Partial Fulfillment
of the Requirements
for the Degree of

Doctor of Philosophy

May 2012
The dissertation of Jin Ouyang was reviewed and approved* by the following:

Yuan Xie  
Associate Professor of Computer Science and Engineering  
Dissertation Advisor, Chair of Committee

Vijaykrishnan Narayanan
Professor of Computer Science and Engineering

Mahmut Kandemir
Professor of Computer Science and Engineering

Zhiwen Liu
Associate Professor of Electrical Engineering

Raj Acharya
Professor of Computer Science and Engineering  
Department Head

*Signatures are on file in the Graduate School.
Abstract

The rapid process scaling trend of the silicon industry has provided the resources to concurrently execute multiple instruction sequences on the same chip, a capability conventionally only available with bulky and expensive parallel computers and distributed multiprocessors. A range of chips have emerged with simultaneous multi-tasking capability, and received a well-known name “chip-multiprocessors” (CMP) which emphasizes their connections with the much bulkier predecessors, conventional distributed multiprocessors. As possibly the only cost-effective approach to keep Moore’s law alive, CMP architectures have received tremendous research and developments efforts further boosting their multi-tasking capabilities (increased number of computing units, improved shared cache architecture, better scheduling and power management, etc.). In particular, as a crucial part of the on-chip system, on-chip interconnection network has become a heated field that draws great research interests and yet presents numerous challenges.

The most critical challenges facing on-chip interconnection network researches are the large design space and the disparaging requirements imposed by different applications. Lacking standards and demanding high performance, most proposed on-chip interconnection network architectures adopt technologies transferred from off-chip interconnection networks developed for conventional multiprocessors. However, the constraints imposed by the on-chip environment and the even stringent demand of low latency and high throughput makes the on-chip interconnection network essentially different from the off-chip counterparts, and elicits innovative approaches to efficient on-chip network architectures.

In this thesis an extensive view of researches on on-chip interconnection networks is presented. In the first two chapters, preliminary knowledge about and recent work on on-chip interconnection network is reviewed. In the following chapters, our work on exploring and improving the design of on-chip interconnection networks is presented. Particularly, in Chapter 4 novel architectures that pro-
vide quality-of-service at the interconnect-level are presented, and in Chapter 3 emerging interconnect techniques are leveraged to further improve the efficiency of on-chip interconnection networks. The final chapter highlights the observations and the findings obtained from foregoing chapters, as well as discusses open issues in this field.
Table of Contents

List of Figures viii
List of Tables xii
Acknowledgments xiii

Chapter 1
Overview of On-Chip Interconnection Network 1
1.1 Emerging Chip-Multiprocessors (CMP) . . . . . . . . . . . . . . . . 1
1.2 Basics of On-Chip Interconnection Network . . . . . . . . . . . . . 6
  1.2.1 A Taxonomy of OCINs . . . . . . . . . . . . . . . . . . . . . 6
  1.2.2 Router and Channel Architectures . . . . . . . . . . . . . . . 9
    1.2.2.1 On-Chip Router Architecture . . . . . . . . . . . . 10
    1.2.2.2 Physical Channels (Links) . . . . . . . . . . . . . . 12
  1.2.3 Algorithms and Protocols . . . . . . . . . . . . . . . . . . . 13
    1.2.3.1 Service Layers in OCINs . . . . . . . . . . . . . . . 13
    1.2.3.2 Algorithms of Each Service Layer . . . . . . . . . 16
  1.2.4 Design Considerations and Metrics . . . . . . . . . . . . . . . 19
    1.2.4.1 Performance . . . . . . . . . . . . . . . . . . . . . . 19
    1.2.4.2 Cost . . . . . . . . . . . . . . . . . . . . . . . . . . 20

Chapter 2
Recent Work on On-Chip Interconnection Network 23
2.1 General Architecture and Topology Researches . . . . . . . . . . . . . 23
  2.1.1 Router Architecture Innovations . . . . . . . . . . . . . . . 23
  2.1.2 Novel Topologies Tailored for OCIN . . . . . . . . . . . . . 26
2.2 Application-Aware Networks . . . . . . . . . . . . . . . . . . . . 27
Chapter 3

Leveraging Emerging Interconnect Technologies

3.1 A 3D Network-on-Chip Based on AC-Coupling Links
3.1.1 Introduction
3.1.2 Vertical Interconnections in 3DIC
3.1.2.1 Through Silicon Vias
3.1.2.2 AC-Coupling Vertical Links
3.1.3 3D NoC Router Architecture
3.1.4 Addressing the Area Overhead with Serialization
3.1.5 Data Compression
3.1.5.1 The Concept
3.1.5.2 Implementation
3.1.6 Experiment Result
3.1.6.1 Study with Synthetic Traffics
3.1.6.2 Study with Multi-Thread Applications
3.1.6.3 Area Efficiency
3.1.7 Summary

3.2 F²BFLY: A Free-Space Optical Network-on-Chip
3.2.1 Interconnect Challenges for Many-Core CMPs
3.2.2 Designing Free-Space Optical Interconnects
3.2.3 Free Flattened Butterfly NoC
3.2.3.1 Topology
3.2.3.2 Router Architecture
3.2.3.3 Comparison to Prior Approaches
3.2.4 Experiment Setup
3.2.4.1 Methodology
3.2.4.2 Network Configuration
3.2.4.3 CMP Configuration
3.2.5 Experiment Results
3.2.5.1 Results for Synthetic Traffics
3.2.5.2 Results for Applications
3.2.6 Summary of F²BFLY

Chapter 4

Quality-of-Service

4.1 Introduction and Background
4.1.1 Quality-of-Service Objectives
4.1.2 Related Work in Providing QoS for OCIN ............... 75
4.2 LOFT: A High-Performance Network-on-Chip Providing Quality-
of-Service ............................................ 76
  4.2.1 Introduction ....................................... 76
  4.2.2 Principles of LSF and FRS ......................... 77
    4.2.2.1 Frame-Based Scheduling ......................... 77
    4.2.2.2 Flit-Reservation ................................. 81
  4.2.3 Integration of LSF and FRS ......................... 85
    4.2.3.1 Combining Local Frames and Reservation Tables . 85
    4.2.3.2 Output Scheduling Anomaly .................. 86
    4.2.3.3 Optimizations ................................ 90
  4.2.4 Simulation Setup .................................. 92
    4.2.4.1 LOFT .......................................... 92
    4.2.4.2 GSF ........................................... 94
    4.2.4.3 Hardware Cost ................................ 95
  4.2.5 Experiment Results ................................ 95
    4.2.5.1 Fairness ...................................... 95
    4.2.5.2 Performance .................................. 96
    4.2.5.3 Case Studies of Performance Isolation ........ 97
  4.2.6 Summary of LOFT .................................. 98
4.3 Enabling Quality-of-Service in Optical OCIN ............. 98
  4.3.1 Introduction ...................................... 98
  4.3.2 Nanophotonic Interconnect Components .............. 100
  4.3.3 Optical NoC Architecture .......................... 100
    4.3.3.1 Baseline Architecture ......................... 101
    4.3.3.2 Optical NoC with QoS Support .............. 102
  4.3.4 Experiment ........................................ 107
    4.3.4.1 Fairness and Performance .................... 108
    4.3.4.2 Energy and Hardware Overheads ............. 109
  4.3.5 Summary of QoS Support in Optical NoC ............. 110

Chapter 5
  Conclusion and Future Work ................................ 120

Appendix A
  Formal discussion of condition (4.1) .................... 123

Bibliography ........................................... 127
List of Figures

1.1 Feature size and supply voltage scaling predicted by ITRS roadmap 2007. [1] .......... 2
1.2 Density (million transistors per mm²), frequency (GHz), and number of wiring levels predicted for process generations. [1] .... 2
1.3 An example CMP with 16 tiles interconnected by routers (R) and channels ......................................... 4
1.4 A structural taxonomy of OCINs .................................................. 7
1.5 Example networks ................................................................. 7
1.6 (a) Baseline router architecture. (b) Virtual channels of an input port. .................................................. 10
1.7 Structure of the message. In this example, the message consists of multiple flits, each of which in turn are two phits. ................. 14
1.8 Message pipelining ............................................................... 16
1.9 Three different flow-control mechanisms. (c) shows an example for wormhole-switching, where a 2-flit message (grey) is being forwarded from router A to router B. For clarity, only input buffers in the port of interests are shown, and have only 1 virtual channel and 2 slots. It is clear that the grey message starts next hop even there is no sufficient buffer. A previous message (mosaic) has not yet been drained. ............................................. 18

2.1 Router pipeline stages for different architectures: (a) Canonical router. (b) Speculative SA. (c) Lookahead routing. ................. 24
2.2 1-stage pipeline enabled by pseudo-circuit ............................... 25
2.3 Concentrated mesh. (a) Topology. (b) Internal concentration. (c) External concentration. ............................................... 26
2.4 Interconnections in a row in on top of a 2D mesh. .................. 27

3.1 A conceptual of vertical interconnects based on three different physical media ........................................... 32
3.2 (a) A 7-port 3D NoC router with dedicated vertical ports (Up/Down). (b) Serial transmitter. (c) Serial receiver. 
3.3 Area overheads when the serialization ratio varies, for different vertical link pitches. The results have been normalized to the case where the area of vertical links is not included. The trend line shows the normalized bandwidth reduction with increasing serialization ratio.
3.4 Zero-pattern compression in principle.
3.5 Operations in a single data transaction. The text in shaded boxes refers to additional operations of data compression.
3.6 Data compression hardware details: (a) Compression. (b) Decompression.
3.7 Message latencies for different traffic patterns without data compression.
3.8 Energy consumption normalized to baseline $T_{SV}$ for different traffic patterns.
3.9 Message latency reduction with higher transmission frequencies.
3.10 Effectiveness of data compression in improving IPC.
3.11 Impacts of higher transmission frequency on IPC.
3.12 Energy of interconnection network for multi-thread applications without data compression.
3.13 Energy of interconnection network for multi-thread applications with data compression.
3.14 Area efficiency of the interconnection network when different techniques are successively applied.
3.15 Free-space optical interconnects: the actively modulated VCSEL emits encoded optical signals, which are then collimated and redirected by a series of optical devices, and finally detected by the photodetector.
3.16 The optical system with 3D stacked RGL, optical substrate, and CMOS substrate.
3.17 The two RGL orientations directing laser beams to two orthogonal directions.
3.18 1-d FSOI links and crossbar formed by wavelength-switching.
3.19 Free flattened butterfly: (a) concept of dimension-decomposition; (b) router architecture.
3.21 Performance results for synthetic traffics. The “-I” versions of networks refer to the cases with internal concentration. For the 64-node network, the results with internal concentration for the hotspot traffic are not shown, since we already show with the 64-node network it does not improve performance for the hotspot traffic.

3.22 Energy consumption for uniform and transpose traffics, when the injection rate is 0.05 packets/cycle/node.

3.23 Energy-delay products calculated from results in Figure 3.22.

3.24 Network performance and energy results for applications.

4.1 (a) Abstraction of network contention into a server model. The left part shows a piece of NoC; and the right part shows the converted server model. The rectangles labeled “P” represent PEs; the circles labeled “R” represent routers. In this figure three contending flows are shown. Flows are merged at each scheduling point, where a “MUX” picks flits from different queues in a certain order. For simplicity, PEs other than sources or destinations and routers without contention are omitted. (b) and (c) Frame-based arbitration: (b) shows the ideal setup while (c) shows the practical setup.

4.2 A pathological scenario: while the stripped is not contending with the grey nodes, its throughput is still reduced by GSF due to the global synchronization.

4.3 Format of look-ahead flits.

4.4 Router architecture for flit-reservation flow-control.

4.5 The formats of reservation tables.

4.6 Comparison of three different flow-control mechanisms. The black rectangles reflect credit turn-around time. For wormhole, the rectangles labeled “F” represent flits. For VCT and FRS, the rectangles labeled “Pkt k” represent packets. The rectangles labeled “L k” in FRS refer to look-head flits.

4.7 Integration of LSF and FRS: the time slots in output reservation tables are grouped into frames.

4.8 An example showing output scheduling anomaly. The entry tagged by “N” shown in (c) indicates buffer underflow.

4.9 The speculative and the non-speculative buffer sharing an input link.

4.10 Fairness of throughput allocation for hotspot traffic pattern. The tables show the maximum, minimum, average, and standard deviation of throughputs for each group of flows.
4.11 Packet latency and network throughput for (a) uniform and (b) hotspot traffic patterns. Note that throughput results are normalized to that of GSF. For each LOFT architecture, \( \text{spec} = N \) means the speculative buffer size is \( N \). .................................................. 112

4.12 Case Study I: per flow average packet latency and accepted throughput are plotted against aggressor injection rates, when (a) GSF and (b) LOFT are used. .................................................. 113

4.13 Case Study II: accepted throughputs of grey and stripped nodes in Figure 4.2 for (a) GSF and (b) LOFT. It is assumed that both kinds of nodes are injecting at the same rates. ......................... 113

4.14 Channel-guided principle: A conceptual nanophotonic link, which consists of a laser source, waveguides, and micro-rings as modulators or detectors. .................................................. 114

4.15 (a) A single MWSR ring. Black boxes refer to O/E and E/O converters. P0-P4 are processors that send and receive signals. (b) A connected 4-node network with 4 MWSR rings. For clarity, O/E and E/O converters are omitted. ................................. 114

4.16 Floorplan of the 256-core CMP interconnected by optical token-rings (Corona [2]). .................................................. 114

4.17 Bandwidth allocation models. .................................................. 115

4.18 All-optical frame-switching mechanism. The inner ring is the completion ring while the outer is the frame-switching ring. ......................... 115

4.19 State machines for source and home nodes. ................................. 116

4.20 Accepted throughput for (a) the baseline architecture and for (b–d) the QoS-enabled architecture with equal and differentiated allocations. The red arrow indicates the hotspot node. ......................... 117

4.21 Average flit latency (left), achieved throughput (right) for (a) uniform, (b) hotspot, and (c) transpose. .................................................. 118

4.22 (a) Maximum throughputs of the QoS-enabled network normalized to the baseline, with different frame sizes (128–640 flits). (b) Energy decomposition with different offered loads. .................................................. 119

A.1 Divide frame window into frame regions. .................................. 124
List of Tables

1.1 Summary of OCIN Metrics .................................................. 22

3.1 Recent Work on Inductive/Capacitive-Coupling Vertical Links ..... 33
3.2 TSV Characteristics [3] ......................................................... 34
3.3 Implementation Results of 128-bit Transceiver with STMicro 65nm technology at 5GHz ......................................................... 37
3.4 Baseline Router Configuration ............................................. 37
3.5 Optical System Design ......................................................... 57
3.6 Configurations of Different Network Architectures. The "I" versions of networks refer to the cases with internal concentration. ... 64
3.7 Configurations for the CMP ..................................................... 66

4.1 Simulation Setup .............................................................. 93
4.2 Per Router Storage Requirements (bits) ................................. 95
4.3 Summary of notations .......................................................... 105
4.4 Optical Component Budget .................................................. 110
Acknowledgments

I am in deep gratitude for my thesis advisor and committees, for their invaluable guidance, inspiration, stimuli, and perhaps sometimes tortures that enable me to finish this thesis.

I would also like to acknowledge my single mother, my family and friends for supporting and accompanying me through all these years of study, and shaping me to be who I am now.
Overview of On-Chip Interconnection Network

1.1 Emerging Chip-Multiprocessors (CMP)

The semiconductor industry has never stopped the pursuit of higher performance per unit cost, as predicted by Intel’s co-founder Gordon Moore. As of today, feature size scaling is still following the exponential trend, and doubling the transistor density every 18–24 months. The unceasing improvement of device density provide the opportunities for designers to continuously improve the performance of and integrate more functionalities on a single die, as we have observed from generations of processors throughout recent years.

On the downside, however, not all aspects of semiconductor are scaling at the same pace as the feature size. One outstanding example is the supply voltage scaling that is constrained by device non-idealities and has been long lagging feature size scaling. As shown by Figure 1.1, ITRS [1] projected a seven-fold decrease in feature size but only a two-fold decrease in supply voltage until 2022. The direct impact of slow voltage scaling is the slow increase of operation frequency constrained by the power supply and the heat dissipation challenges. This is illustrated in Figure 1.2, where the device density is increasing much faster than the operation frequency. In addition, the number of wiring levels scales even slower with only 4 additional levels available until 2022. Overall, the technology scaling
trends imply that designers have to primarily leverage the abundant transistors rather than increasing frequency to improve the performance of future processors, and that architectures requiring high interconnect complexity is less favorable than those with regular and loosely coupled components requiring structured interconnects.
Another hard to manage challenge is the design complexity associated with the exponentially increasing resources on a single chip. The burgeoning scale of circuits stresses all aspects of design flow:  

- **a)** Design productivity is challenged as the size of design is increasing faster than the progress of automation tools (pure human productivity will probably remain constant for millennia).  
- **b)** Power consumption will become more and more the central problem of design, which is further compounded with the following problems.  
- **c)** Design-for-test (DFT) and design-for-manufacture (DFM) becomes essential to control cost and yield faced with the ramping process variations.  
- **d)** Circuits and interconnects will be more vulnerable to interference and dynamic faults (*e.g.* soft error), whose impacts cannot be ignored during the design process. The aggregate effect of the above problems is the rapidly increasing design cost, and a thrust for design paradigm improvement.

The idea of chip-multiprocessors (CMP) is motivated by above aforementioned difficulties, and proposed to fully exploit the exponential increase of transistor density. Unlike conventional, single-processor chips, CMPs contain many cores that can run multiple concurrent instruction sequences. Each core can have a relatively simple micro-architecture and a moderate operating frequency, but collectively all cores can create multiple folds of throughput improvement over single-processor chips. CMP addresses the above challenges in following ways:

- **Performance improvement with CMPs** does not rely on single thread performance improvement that is conventionally enabled by intricate micro-architecture innovations (*e.g.* latency-hiding techniques such as speculation and out-of-order execution) and increasing operating frequency. Instead, it utilizes the exponentially increasing transistor budget to execute multiple instruction sequences (threads) and easily multiplies overall throughput without exacerbating power and heat dissipation problems.

- **CMP is composed of** regular and modular cores which are loosely coupled by on-chip communication system. This facilitates design productivity as each computing nodes has moderate size and complexity, and can be easily integrated together once designed. Due to the reduced design complexity and well-structured hierarchy, it is also easier to verify and test the chip.
• Each computing node is less power hungry and distribute power and heat density more uniformly across the chip.

• The whole processor with abundant cores enables various fault-tolerant approaches to be used to improve the resilience against interference and dynamic faults.

Figure 1.3: An example CMP with 16 tiles interconnected by routers (R) and channels

Besides the commercial products with only a few cores [4–6], a number of research prototypes from both academia and industry have showcased tens to hundreds of cores on a single die [7–11] signaling future advances allowing for more and more cores in the CMPs. Driven by the ease of design, most contemporary many-core CMPs have a “tiled” organization, where identical “tiles” form a regular floorplan on the chip. One example of such CMPs with 16 tiles is shown in Figure 1.3. Each tile typically contains one core plus some local memory (e.g. private L1 caches), and an interface with the on-chip communication system. In the example shown in Figure 1.3, the on-chip communication system is essentially an
on-chip interconnection network composed of multiple routers connected by channels; each tile has one router, which serves as both the interface between the owner tile and the network, and the intermediate router for communication among other tiles. The tiles and routers are surrounded by off-chip communication interfaces, \textit{e.g.}, memory controllers, I/O interfaces, and inter-chip communication interfaces, \textit{etc}.. The off-chip communication interfaces are also interconnected with the tiles by the on-chip network.

The tiled CMP both enables and necessitates on-chip interconnection network (OCIN) as the underlying communication system. The regular floorplan and similarities of communicating components (in our example, the communicating components are mostly identical tiles) eases the design of well-structured OCIN. Using OCIN instead of dedicated global wiring to interconnect tiles have multitudes of advantages: \textit{a)} OCIN is composed of regular and structured channels whose electrical properties are highly predictable and can be specially controlled and optimized. \textit{b)} By defining a standard interface, OCIN facilitates modularity and design reuse. \textit{c)} OCIN provides scalable bandwidth to accommodate increasing communication requirements of future many-core processors. Overall, compared with global wiring, OCIN is essentially a more cost-effective communication solution for future CMPs with tens to hundreds of communicating components.

Advances in many-core CMPs create a strong thrust to OCIN researches dedicated to develop higher throughput, lower latency, power, and cost OCINs. However, the OCIN architectures are far from being converging, leaving lots of open issues in this field. OCIN researchers often find themselves faced with a large design space to explore, and disparaging even conflicting requirements of different applications. Many OCIN architectures are derived from off-chip interconnection networks developed for conventional multiprocessors. However, the on-chip environment imposes unique constraints on network design and demands even higher performance interconnection networks, making OCIN essentially different from their off-chip counterparts, and eliciting innovative approaches to efficient on-chip network architectures. The rest of this thesis is devoted exclusively to the topic of OCIN architectures for future CMPs.
1.2 Basics of On-Chip Interconnection Network

On-chip interconnection network (OCIN), or a more popular and concise name, network-on-chip (NoC)\(^1\), has become a well-established research field due to its importance for future advances in CMPs. A precise definition for OCIN is difficult, as it could refer to as broad as any on-chip communication fabrics even including dedicated global wiring (a flat, point-to-point topology). Here a descriptive definition that tries to capture widely-accepted features of OCIN is presented:

1.2.1. Definition. *On-chip interconnection network is an emerging design paradigm for communication fabrics in large-scale on-chip systems within a single chip. It is constructed from multiple routers interconnected with channels, and multiplexed by different on-chip communicating peers. The messages sent from the source component to the destination component are relayed by one or multiple routers and channels that effectively form a path from the source to the destination. OCIN offers superior performance and bandwidth scalability due to the potential of simultaneously delivering different messages through different links and routers. While it is not strictly required, OCINs are often well-structured with regular topologies and distributed algorithms to coordinate message movement.*

In the rest of this section, we present several building blocks to understand the broad concept of OCIN: *a) Taxonomy of OCINs (Section 1.2.1). b) Basic structures of routers and channels (Section 1.2.2). c) Algorithms and protocols (Section 1.2.3). d) Design considerations and metrics (Section 1.2.4).* These basic building blocks also facilitate the understanding of state-of-the-art OCIN architectures presented in Chapter 2, 3, and 4.

1.2.1 A Taxonomy of OCINs

One way to facilitate understanding the large design space of OCINs is to identify cardinal dimensions where different OCINs architectures form distinct clusters according to most important features. Here we follow the approach adopted in the classic textbook of Duato *et al* [12], and classify OCINs primarily based on their topological features as shown in Figure 1.4.

\(^1\)OCIN and NoC will be used interchangeably in this thesis.
Figure 1.4: A structural taxonomy of OCINs

Figure 1.5: Example networks
**Shared-medium networks** is the simplest form of OCINs where a single channel is shared by all communicating peers. In this case, only one communication transaction can be carried out at a time, and therefore arbitration is needed to select which component can use the channel. Typical arbitration mechanisms include arbitration buses (as shown in Figure 1.5a) and token rings. Shared-medium networks are easy to design and use, however provides limited performance and bandwidth scalability. Therefore they are not covered in depth in following chapters and only listed here for completeness.

**Direct networks** is a large family of scalable networks, many of which are widely adopted in recent work. Direct networks consist of multiple nodes, each being a communicating device interfaced with a router. Each router is directly connected by channels to its neighboring routers to form the network. From the perspective of topology, there is no distinction among node, communication device, and router. Therefore direct networks are usually represented by a graph $G(V,E)$ where graph nodes ($V$) represent the set of nodes (routers) and edges ($E$) represents the channels. Examples are shown by Figure 1.5b and 1.5c. Typically the channel between a pair of nodes is bidirectional (or consists of two unidirectional channels). Unless otherwise specified, in the rest of the thesis, each edge in a topology graph refers to a bidirectional channel.

One major branch of direct networks is the **orthogonal networks** in which typically nodes are arranged in orthogonal dimensions in the space, and each node is connected only to its closest neighbors in each dimension. This is exactly the case for *k-ary n-mesh*, where $k$ refers the number of nodes in each dimension and $n$ refers to the number of dimensions. An example of 4-ary 2-mesh is shown in Figure 1.5b. Mesh is a very popular topology of OCIN due to its regular structure especially suitable for tiled CMPs. However, one drawback of mesh is the imbalance of network bandwidth, since nodes close to the center of the network enjoys higher bandwidth than nodes close to the boundaries and corners of the network. Torus network (*k-ary n-cube*) addresses this problem by adding wrap-around channels to the network (Figure 1.5c) and therefore ensuring every node receives the same bandwidth. However, both wiring complexity and wire length are increased in torus, resulting in lower per channel bandwidth when wiring budget is limited.

Recently, a range of different novel direct networks are proposed to address
different issues of conventional orthogonal networks [13–16]. For example, flattened butterfly network [14, 15] is an interesting topology that transform an indirect network (butterfly) to a direct network. We will revisit these novel networks in Chapter 2 and Chapter 3.

**Indirect networks** is another family of scalable networks that also gains lots of research interests. Different from direct networks, indirect network networks consist of intermediate switches \(^2\) that are interfaced with zero, one, or more communicating devices. Therefore there is a distinction between intermediate nodes (switches with zero communicating devices) and terminal nodes (communicating devices). Figure 1.5d shows an example of an indirect network, where intermediate and terminal nodes are differentiated by different symbols.

The simplest form of indirect network is a single switch connecting all communicating devices, or equivalently, a crossbar. However, crossbar has scalability issues as its complexity increase quadratically with the ports. It is also least favorable to implement in future process generations with abundant transistors but limited wiring levels and power supply. A branch of more scalable indirect networks is the multistage networks, where stages of switches with moderate complexity form the network to provide better scalability. Figure 1.5d shows one example of such networks (bidirectional butterfly).

**Hybrid networks** usually combines features of different networks according to the unique requirements of applications, in order to maximize the efficiency of OCIN. One way to construct hybrid networks is to implement hierarchies in the network, and use possibly different technologies in different hierarchies. For instance, Figure 1.5e shows a hierarchical network that combines rings and a crossbar. Another possible way is to replicate the same network multiple times, which is the case for address interleaved buses [17].

### 1.2.2 Router and Channel Architectures

As mentioned above, scalable OCINs are constructed by routers (switches) and channels. Therefore it is helpful to understand the router and channel architec-

---

\(^2\)The term “switch” in indirect networks is equivalent to “router” in direct network. This term is created in order to distinguish intermediate nodes (switches) from terminal nodes (communicating devices). In this thesis, both terms can be used interchangeably.
Figure 1.6: (a) Baseline router architecture. (b) Virtual channels of an input port.

In order to understand how the overall OCIN works.

1.2.2.1 On-Chip Router Architecture

The design space for router architecture is almost as large as, if not larger than, that of network topology. However, to be pertinent to contemporary researches, we limit our focus on a baseline router architecture that is currently widely-assumed in recent work. Figure 1.6a illustrates the basic components of this architecture:

- **Switch.** The central component of the baseline router is a switching fabric
that connects input ports to output ports. Typically, this switch is itself a non-blocking\(^3\) network, which, in most cases, is a flat crossbar or multiplexer trees. Another commonly made assumption is that the number of input ports is equal to the number of output ports, since typically in a OCIN channels are bidirectional (see Section 1.2.1). Therefore the number of input ports (output ports) to the crossbar (router) is often called the radix ($\sigma$) of the crossbar (router). Crossbar radix is an important metric of router complexity, since crossbar is one of the most power and area consuming components and its cost scales up quadratically with the radix. High-radix routers only adopts hierarchical crossbars [18] or the network-in-network approach recently proposed [19].

- **Buffers.** When a packet cannot be forwarded through the switch due to contention, buffers are necessary to temporarily hold the packet. It is an important approach to address congestion and improve throughput. Most contemporary OCIN routers assume only input buffers to reduce cost (avoid double-buffering within the router), but introduces some performance overheads due to head-of-line (HOL) blocking. To alleviate HOL blocking, input buffers are often decomposed into several parallel buffers, usually denoted as virtual channels sharing the same input port as shown in Figure 1.6b. When the packet in a virtual channel is blocked, packets in other virtual channels can advance instead. In addition, virtual channels are critical for deadlock-free adaptive routing algorithms. Because buffer is a scarce on-chip resource and virtual channels are critical to network performance, the number of virtual channels ($\nu$) and the depth per virtual channel ($\delta$) are also important metrics for an OCIN router.

- **Routing, allocation, and flow-control logics.** When a packet arrives, the router needs to determine which output to forward it. This is the task of routing logic. In addition, buffers and switch connections need to be allocated or set up for the arriving packet, which is handled by the virtual channel allocator (VA) and the switch allocator (SA). Finally, flow-control is needed to avoid dropping packets when there is insufficient buffer in the

---

\(^3\)Trafics from input ports will not block each other if they head for different output ports.
downstream routers. All the control logics will be discussed in depth in Section 1.2.3.

- **Input/output ports and link controllers.** As we mentioned, routers are interconnected by physical channels, which are essentially on-chip links implemented by wires. An input (output) port contains a link controller that is interfaced with the input (output) channels to neighboring routers or injection (ejection channels) to the local devices. In the case of intermediate routers (switches) in indirect networks, the input/output ports are always connected to other routers/switches.

Link latency, area and power consumption significantly affects the efficiency of the network. Several different link technologies will be briefly discussed below.

### 1.2.2.2 Physical Channels (Links)

To be distinguished from virtual channels, in the rest of the thesis the channels physically connecting neighboring routers are referred to as physical channels or links. Physical channels not only determines the latency and the throughput between neighboring routers, but also consumes a significant portion of overall network power. Therefore, optimizing link performance and power consumption is of central importance in the design of OCIN.

In the simplest form, physical channels are essentially metal wires on top of silicon. Given the typical length of channels (spanning one to several tiles), it is usually assumed that physical channels are implemented with the top 2 wiring levels shared with power supply. The area taken by the physical channels can be subsumed by loose logics, but dense structures such as memory modules can hardly absorb the repeaters that are required for medium to global range wires. Therefore, in contrast to common beliefs, there are limited on-chip resources to be exploited by the physical channels. In addition, the repeaters along the physical channels, necessary to reduce latency and improve signal integrity, consume substantial power from the total on-chip power budget. Due to above constraints, physical channels using simple repeated wires usually have limited bandwidths.

To reduce power consumption, one can implement the physical channels using
low-swing wires which has reduced voltage swing and therefore significantly lower power [20]. While low-swing wires are shown to have improved energy-efficiency, their operating frequency is usually low. Therefore, they can only be used in the cases of non-critical signal path or low-power chips.

Emerging on-chip interconnect technologies provides alternatives that have higher bandwidth and lower power consumption. In 3D-stacked chips, the through-silicon-vias (TSV) have low parasitics and can be used to replace global interconnects, in order to provide a high-bandwidth and low-power signal path. Very recently, on-chip transmission lines and optical interconnects have been proposed as candidates to replace global wires, and provide extremely high bandwidth and low power [21,22]. We will discuss recent work on emerging interconnects in Chapter 2, and our contributions to this field in Chapter 3.

1.2.3 Algorithms and Protocols

So far this section only presents more a structural than a algorithmic view of OCINs. To complete the picture, this subsection discusses the algorithms and the protocols that operate on top of the hardware in OCINs. We first present a simplistic layered structure of OCINs and then discuss algorithms/protocols associated with each layer.

1.2.3.1 Service Layers in OCINs

Engineers of computer and electronic systems often embrace a “layered” view of the target system, which helps decompose the design problem and isolate the focuses of different parts of the system. This is also the approach that has been long taken by the network community, separating the responsibilities of different layers to facilitate design and maintenance. One salient example is the OSI model that is the de facto universal standard of Internet (http://en.wikipedia.org/wiki/OSI_model). Other interconnection networks also have very similar layer structures that are usually a subset of OSI model. In OCIN, a layered view is also adopted to help architect efficient protocols. However, due to the regular structure and the somewhat homogeneity of agents and traffic patterns, OCIN’s layering structure is much simpler and shallower. In this thesis, we follow the simplistic
approach used in Duato et al’s textbook [12] and view the OCIN as composed of three layers: **routing layer, switching layer, and physical layer.**

**Routing layer** is an essential layer comprising algorithms and protocols that determine the path taken by the messages from the source to the destination nodes. Upon receiving a message, the router in OCIN needs to determine which output port will be used to forward this message. To do so, the router needs to first decode the routing information usually contained in the header of the message (see Figure 1.9), and then uses the routing logics to compute the output port.

**Switching layer** coordinates the movement of messages through the router once routing is done. The messages injected by source nodes into OCIN is likely to be longer than the width of datapath of the router. Therefore, they need to be split into small blocks when moving from the input ports to the output ports. Each of the block is called a *flit* (flow-control unit; flow-control is an integral part of switching layer as discussed below) which is the basic unit manipulated by switching layer (Figure 1.7).

One required task that switching layer performs is *switch allocation* (SA in Figure 1.6a), which for each output port selects the message stored in one input port to forward. The throughput of OCIN largely depends on the efficacy of the switch allocator since it determines the maximum number of successful connections in the router in each switching cycle.

![Figure 1.7: Structure of the message. In this example, the message consists of multiple flits, each of which in turn are two phits.](image)

The majority of OCINs, like their off-chip multiprocessor predecessors, guarantees lossless delivery of messages once they are injected. This is because the applications can hardly tolerate dropping packets, and because the simplicity of layering structure is insufficient to support retransmission. In these cases, the switching layer also performs *flow-control* (FC in Figure 1.6a) which prevents buffer overflow
and packet drop within the network. The unit of flow control is the above defined flit (Figure 1.7). That is, buffers are monitored and managed in the unit of flit, and the switching layer only forwards a flit to downstream router if it will not cause the input buffer of that router to overflow. The flow-control latency could become the bottleneck of OCIN bandwidth, its efficiency also largely determines the performance of OCIN.

When virtual channels are used, as is common for most proposed OCINs, switching layer has to take on another responsibility—*virtual channel allocation* (VA in Figure 1.6a). VA allocates an available virtual channel to a message waiting in the input port, and in a sense, is a flow-control mechanism operating at the granularity of an entire message. Virtual channel management is tightly related to routing algorithms and flow-control mechanisms, and also affects the overall throughput of OCIN.

**Physical Layer** handles link level transmission of messages from the output port of one router to the input port to another. As is the case with switching layer, link width is often smaller than the message length, enforcing the message to be split into blocks. The unit block of link layer is called *phit* (physical layer unit), and when the width of the channel is the same as the width of router datapath, is equivalent to flit\(^4\). Link transmission is on the critical path of message delivery and therefore determines the bandwidth of OCIN, and its power consumption is a major component of overall OCIN power as we already discussed. Depending on application requirements and design objectives, link protocols can be synchronous, asynchronous, or mesochronous when the designers see appropriate.

**Putting it together.** Here we summarize the typical stages a message go through for a router. As virtual channels have multitudes of advantages and are commonly used, we assume in this paragraph that virtual channels exist in routers and VA is needed. Below is the sequence of events in timing order: 1) When a message arrives at the input port a router, it is first enqueued into the allocated virtual channel. 2) When this message becomes the head of the virtual channel (all previous flits stored in the channel have been drained), routing computation is performed to decode the information stored in the header of the message and

---

\(^4\)This is usually the case for most OCINs. Therefore in this thesis phit and flit are used interchangeably unless otherwise specified.
select the output ports. 3) After the output port is determined, virtual channel allocation is performed to select an available virtual channel in downstream input port. If no available virtual channel is found, this message will be blocked in the current virtual channel until at least one is available. 4) For messages with selected output port and virtual channel, they can request for the switch in order to traverse to the output port, one flit by one flit. At the this stage, flow-control mechanism will monitor the buffer occupancies of downstream virtual channels, and block the message whose destination virtual channel is full. Switch allocator will select flits from unblocked messages to use the switch and reach individual output ports. 5) Physical channels convey phits (flits) from the output ports to the downstream input ports, which are again enqueued into corresponding virtual channels.

<table>
<thead>
<tr>
<th>Message A</th>
<th>RC</th>
<th>VA</th>
<th>SA</th>
<th>ST</th>
<th>LT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message B</td>
<td>RC</td>
<td>VA</td>
<td>SA</td>
<td>ST</td>
<td>LT</td>
</tr>
</tbody>
</table>

RC: Routing computation VA: Virtual channel allocation SA: Switch allocation ST: Switch traversal LT: Link traversal

Figure 1.8: Message pipelining

OCINs usually have stringent requirements on latency and throughput. Therefore to-date most proposed OCIN routers are pipelined, where the above stages overlap for different messages as shown in Figure 1.8. In addition, a message is finally removed from the network when the routing computation logic selects the ejection port (for routers connected to the destinations or switches connected to the terminals).

1.2.3.2 Algorithms of Each Service Layer

We primarily discuss algorithms for the routing and the switching layers since they are more pertinent to the architecture of OCIN. Physical layer protocols are more concerned with signaling and circuit issues and therefore less interesting to us.

**Routing Algorithms.** A plurality of algorithms have been proposed to perform routing computation in OCINs. They differ primarily in target topologies, adaptivity, implementation cost, and application considerations.
In terms of intended topologies, routing algorithms have to ensure that the nodes are fully connected. That is, an node can send message to any other node. In addition, it should optimize performance considering topological features. For example, to reduce latency, the routing algorithm should choose paths with minimum length. This kind of algorithms are called **minimal routing algorithms**. As another example, to improve throughput, it is often desirable to uniformly distribute the traffics across the network, and if possible, dynamically avoid routing packets through regions that are already congested. This is usually achieved by an important family of routing algorithms—adaptive routing algorithms as described below.

The adaptivity of a routing algorithm is the measure of its flexibility in selecting routes according to network conditions. The most restrictive routing algorithms are **deterministic routing algorithms** that select only a single port at each hop regardless of network condition. A less restrictive kind of algorithms is called **oblivious routing algorithms**, which is in fact the superset of deterministic routing algorithms. Oblivious routing algorithms are still ignorant of network condition, however they provide the flexibility of selecting multiple output ports at each hop. The **adaptive routing algorithms**, in some sense a superset of the above two, has the flexibility in selecting multiple possible output ports according to the network condition.

Adaptive routing algorithms are very effective in improving the network throughput by alleviating congestion. Most adaptive routing algorithms consist of essentially two subroutines. The first routine is the **routing routine** that computes a set of valid possible output ports for a message at a router; the second routine is the **selection routine** that selects one output port from the set of output ports supplied by the routing routing, according to the network condition.

While adaptive routing algorithms can achieve higher performance, they are usually more difficult to implement and complicate the problem of **deadlock** which is the subject to be covered shortly. Deterministic and oblivious algorithms have lower implementation cost due to simplicity, and can easily avoid deadlock by design. To minimize routing logic complexity, a technique called **source routing** can be applied which pre-computes the path at the source node, and encode the output port to use at each hop in the message. In this case, the routing logic of the
router is simply a decoder that extract the pre-computed output port. However, source routing will impose overheads of routing information carried by the message, and can only be used when simplistic routing algorithm is desired and the network is small.

Considering application requirements, adaptive routing algorithms need to not only consider impact of port selection on network condition, but also that on application objectives. We will discuss routing algorithms that consider performance isolation in a many-core CMP in Chapter 2.

![Flow-control mechanisms](image)

Figure 1.9: Three different flow-control mechanisms. (c) shows an example for wormhole-switching, where a 2-flit message (grey) is being forwarded from router A to router B. For clarity, only input buffers in the port of interests are shown, and have only 1 virtual channel and 2 slots. It is clear that the grey message starts next hop even there is no sufficient buffer. A previous message (mosaic) has not yet been drained.

**Switching Algorithms.** There are two sets of switch algorithms, that perform arbitration (SA and VA) and that perform buffer allocation (flow-control). We will briefly discuss both below.

Both the switch allocator and the virtual channel allocator need to perform arbitration to select one winner from all contenders for a shared resource (switch output port or virtual channel in the downstream router). Arbitration can be as simple as FIFO or round-robin, which performs well in general cases but does not reflect system-level requirements. More complex arbitration mechanisms take into
considerations the importance of different messages (priority- and age-based arbitration), or the overall system throughput (application-level priority and fairness). Again, arbitration mechanisms are tightly coupled with routing and flow-control, and more complex arbitration mechanisms are even more difficult to implement.

There are primarily three kinds of flow-control mechanisms, differing in the granularity of buffer management. The first kind called store-and-forward (SAF in Figure 1.9a) only forward a message when it has been completed received. This also requires that a message is blocked when there is no sufficient buffer in the downstream router to hold the whole packet. When the message length is long, the time spent at each hop is primarily constrained by the time needed to deliver the whole message to the downstream router. To address this problem, a second kind of flow-control mechanisms is proposed and termed virtual cut-through (VCT in Figure 1.9b). In VCT, the router can start forward a message as long as its header is received (the routing information is contained in the header), essentially pipelining the movement of the message. This significantly reduces per hop latency for a packet when the network is lightly loaded. Finally, the third kind called wormhole-switching (WH in Figure 1.9c) differs from VCT in that once the downstream buffer has empty slots (not necessarily to hold the whole message), the current router can start to forward the remaining flits of message. Therefore in wormhole-switching, flits of a message can potentially occupy buffers across multiple routers. Wormhole-switching especially suits OCIN with very shallow buffers constrained by resource budget, and stringent latency requirements imposed by the applications.

1.2.4 Design Considerations and Metrics

So far we already describe the basic components and mechanisms of OCIN, and will proceed to present design considerations and metrics to characterize an OCIN, which will be used frequently in following chapters.

1.2.4.1 Performance

Throughput and latency are usually first-order performance metrics for OCINs. From the topological point of view, the bisection bandwidth ($\beta_b$) is a useful estimate of the total throughput. Bisction bandwidth is defined as the minimum bandwidth
across any intersection of the network. For instance, the bisection bandwidth of a \textit{k-ary n-mesh} can be computed as:

\[ \beta_b = \frac{2 \times k \times (n - 1) \times \omega}{\Delta} \]  

(1.1)

where \( \omega \) is the size of a flit and \( \Delta \) is the link cycle period which is often equal to the clock cycle period of the whole OCIN. Similarly, the \textit{network diameter} \( (D) \), defined as the maximum distance between any two nodes in the network, and the \textit{average hop count} \( (H) \) which is the expected hops an arbitrary message take, are also good estimates of network latency.

However, the actual performance of OCIN largely depends not only on design, but also on dynamics of the network, such as traffic pattern, injection rate, and burstiness, \textit{etc}. More confident estimates can be obtained by measurement when the designed OCIN is presented with the dynamics in interest. In the measurement-based approach, throughput is characterized as the \textit{saturation bandwidth} \( (\beta_s) \) which is the bandwidth measured when the network latency starts to ramp significantly. Latency measurement is trickier; a good characterization should consider the latencies \( (L) \) measured at low throughput, at a throughput close to saturation, and at a median point of the above two points.

There are more advanced performance metrics of OCIN, most of which are related to specific application requirements. For instance, quality-of-service and fairness is becoming important to improve the throughput of on-chip systems. Therefore the robustness and the capability of providing performance isolation is also of interests for OCINs. We would like to denote this kind of performance metrics as \textit{second-order performance metrics} and defer the discussion of them when related topics are encountered.

\subsection*{1.2.4.2 Cost}

The cost of on-chip VLSI systems primarily comes in two flavors: \textit{area} and \textit{power}. Under a given budget, performance can always be improved by designs with higher area- or power-efficiency. Here we discuss several factors and design decisions that impact the cost of OCINs.

As we have discussed, the radix of router \( (\sigma) \) has a strong impact on area and
power. The complexity of major router components, *e.g.* switch, switch allocator, and virtual channel allocator, increases super-linearly with radix. Area and power of those components, to the first order, is proportional to their complexities. In addition, change of datapath width (flit width, $\omega$) also results in super-linear change of the complexity of the switch.

Memory modules are often expensive in VLSI. The buffers used in OCIN routers is another major contributor to the area and the power consumption. For the baseline router we assumed, the total buffer used can be calculated as the product of radix $\omega$, number of virtual channels $\nu$, and buffer depth $\delta$.

The area taken by the physical channels can be roughly calculated as the product of link length ($\alpha$) and link width (we assume to be the flit width $\omega$). The power consumed in the physical channel is primarily the dynamic power to drive all the wires including the repeaters, and the leakage power consumed the drivers and the repeaters\(^5\).

As can be seen, the design factors strongly impacting cost are also determinants of performance as we discussed above. For example, the flit width $\omega$ determines the bisection bandwidth and thus the overall throughput, but increasing which also has a strong impact on switch and physical channel complexity which will be translated to area and power overheads. Therefore performance and cost are essentially a pair of conflicting requirements. Well-designed OCINs should try to balance them.

Table 1.1 summarizes the metrics of OCINs, which will be revisited in following chapters. Note that they may have different meanings if specifically redefined.

\(^5\)We assume for now electrical links (metal wire, TSV) are used. Emerging interconnect technologies will be discussed in depth in Chapter 3
Table 1.1: Summary of OCIN Metrics

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Radix</td>
<td>$\sigma$</td>
<td>Number pairs of input/output ports of the router</td>
</tr>
<tr>
<td>VC count</td>
<td>$\upsilon$</td>
<td>Number of virtual channels per input port</td>
</tr>
<tr>
<td>Buffer depth</td>
<td>$\delta$</td>
<td>Buffer depth per virtual channel</td>
</tr>
<tr>
<td>Flit width</td>
<td>$\omega$</td>
<td>Width of the flit, datapath, and physical channel</td>
</tr>
<tr>
<td>Bisection bandwidth</td>
<td>$\beta_b$</td>
<td>The minimum bandwidth across any intersection</td>
</tr>
<tr>
<td>Saturate bandwidth</td>
<td>$\beta_s$</td>
<td>The bandwidth when congestion starts to form</td>
</tr>
<tr>
<td>Network diameter</td>
<td>$D$</td>
<td>The maximum distance between any two nodes</td>
</tr>
<tr>
<td>Average hop count</td>
<td>$H$</td>
<td>Expected hops taken by a message</td>
</tr>
<tr>
<td>Latency</td>
<td>$L$</td>
<td>Measured message latency in a certain scenario</td>
</tr>
<tr>
<td>Link length</td>
<td>$\alpha$</td>
<td>Length of the physical channel</td>
</tr>
<tr>
<td>Area</td>
<td>$S$</td>
<td>Area taken by OCIN</td>
</tr>
<tr>
<td>Power</td>
<td>$P$</td>
<td>Power consumed by OCIN</td>
</tr>
<tr>
<td>Clock period</td>
<td>$\Delta$</td>
<td>Clock cycle period of different components</td>
</tr>
</tbody>
</table>
Recent Work on On-Chip Interconnection Network

Motivated by its importance in CMP design, there have been significant research and development efforts invested in the field of OCIN. In this chapter, we review a number of seminal work in this field, by which our proposed architectures (Chapter 3 and 4) are inspired.

2.1 General Architecture and Topology Researches

While inheriting many basic building blocks from off-chip interconnection network, the fundamental differences between OCINs and the off-chip counterparts necessitate specific tailoring to further improve the performance and the cost-effectiveness. Many seminal prior researches exist in this sub-field and are reviewed below.

2.1.1 Router Architecture Innovations

Fine-granularity multi-tasking and scheduling in CMP places even more stringent requirement on OCIN’s latency. In addition, in shared-memory CMPs (the architecture in contemporary CMPs), the short, yet latency-critical coherence messages dominate the network traffics in OCIN. For these kind of messages, even with message pipelining, the intrinsic latency of the router (that is the total pipeline length
of the pipelined router) dominates the end-to-end delivery latency. Therefore reducing router latency, or equivalently the number of pipeline stages, is a topic draws considerable research interests. Peh et al’s seminal work is the first to propose a framework that characterize the router latency depending on architecture features [23]. In that work, a canonical router architecture, which serves as the baseline architecture, is proposed and consists of 4 stages (similar to that in Figure 1.8 excluding the LT stage, also shown in Figure 2.1a). The canonical router is wormhole-switched and pipelined at the granularity of flits (datapath width, physical channel width, and flow-control unit size are all equal to the flit size), and therefore requires only shallow buffers and achieves with the most amenable traffic pattern a sustainable throughput of $\frac{\sigma \omega}{\Delta}$. Based on this canonical router. Peh et al further propose a speculative switch allocator [23] that allows for overlapping the VA stage and the SA stage as shown in Figure 2.1b. Speculative switch allocator optimistically allocates switch ports to messages even if virtual channel allocation has not been finished. At the next cycle (switch traversal), if it turns out a message is assigned a switch port but not a virtual channel (VA fail), that message is held in the input buffer and retries VA and SA again; otherwise that message is forwarded through the switch to the output port. To avoid affecting performance
when speculation is wrong, non-speculative requesters (with VC assigned) have always higher priority in switch allocation than speculative requesters (with VC not yet assigned). As shown in Figure 2.1b, speculative SA can be fully overlapped with VA to reduce one stage in router pipeline.

Another approach to further reduce pipeline stages is look-ahead routing [24] which computes the output to take one hop ahead the current router. Look-ahead routing effectively removes the routing stage from the critical path of the router pipeline, and further reduces the number of pipeline stages by one (Figure 2.1c). An even aggressive pipeline reduction approach, built upon aforementioned approaches, is proposed by Ahn et al [25]. In their paper [25], they present the observation that a considerable portion of flits reuse an existing switch connection set up by previous flits. Therefore they propose a mechanism called pseudo-circuit that allows the flits reusing an existing switch connection to bypass both the input buffer and the switch allocation, making the router pipeline stages experienced by a flit be only the ST stage in the best case (Figure 2.2).

Besides router pipeline latencies, flow-control latency adds to the delivery latency and is significant for latency-critical messages. In another paper of Peh et al [26], flit-reservation flow-control is proposed that minimize the latency overhead caused by flow-control especially when the network load is high. In this approach, two overlapped physical networks are required: a look-ahead network and a data network. In order to deliver a message, a look-ahead flit needs to first be sent via the look-ahead network to the destination. At each hop, the look-ahead flit performs buffer allocation and flow-control, and the corresponding message follows the look-ahead flit in the data network, and reclaims the buffers pre-allocated by the look-ahead flit. Since look-ahead flits are short and simple in structure, delivering and processing them incurs very low latency and overheads. In this way, flow-control is removed from the critical path of message delivery. We will revisit this mechanism in Chapter 4 where our quality-of-service work is shown to built on top of it.
2.1.2 Novel Topologies Tailored for OCIN

Figure 2.3: Concentrated mesh. (a) Topology. (b) Internal concentration. (c) External concentration.

Topology is another first-order factor affecting network performance, and draws a lot of research interests. The primary differences between off-chip network and OCIN, in terms of impacting topology, are that OCIN does not have I/O ping restrictions and can potentially have higher radix and link bandwidth, and that OCIN is not suitable for high-dimensional topologies due to the restriction of 2D silicon process [27]. Balfour et al [13] first explores topology design for OCIN. In the topologies evaluated in their paper, the concentrated mesh (Figure 2.3) is found to be the best cost-effective topology that also provides the best performance. The idea of concentration behind this topology is to share a router among several cores so that the network diameter is reduced. In this sense concentrated mesh can also be viewed as a hybrid topology combining direct and indirect topologies. Sharing a router among cores increases the radix of the router, which significantly increases router cost. One work-around is called external concentration where the traffics from the cores are merged before going through the attached router. Both internal concentration (directly attaching cores to the router) and external concentration are shown in Figure 2.3.

The potential to implement higher-radix routers and wider link width inspires researchers to increase wiring complexity of the topology in order to improve the performance. One example is the flattened butterfly topology proposed by Kim et al [15] that creates strongly connected crossbars in each dimension on top of a 2D mesh. Figure 2.4a shows an example of the connections from a router to
other routers in the same row in flattened butterfly. Flattened butterfly effectively enables a 2-hop network with enhanced connectivity. However, the quadratical increase of wiring intensity makes it costly to be implemented. As a work-around, Grot et al [16] propose the MECS topology which replaces the dedicated physical channels originating from a same node by a single shared bus. Figure 2.4b illustrates an example of such interconnection strategy. Any other node in the same dimension has a sink to divert messages destined to itself. MECS reduces the implementation cost while still maintaining the benefits of low hop-count. Express virtual channels [28] and dragonfly network [29] are similar approaches inserting long range links into a regular, baseline mesh network. In a sense, all these networks are hybrid networks inheriting features from different families of topologies.

2.2 Application-Aware Networks

Network performance always has a strong dependence on the features of traffics going through it. In turn, the applications using the network may have different requirements and objectives that should be differentiated by the network. Designing OCIN in a close-loop with application requirements has been shown to achieve better overall system throughput.

A major branch of researches in this field are focused on provide guaranteed services (e.g. bandwidth) to different tasks, especially tasks requiring real-time
performance. A plurality of papers have been published on this topic [30–38]. Among them two principles stand out due to their suitability for on-chip implementation: 1) **Pre-scheduled time-slots.** The Æthereal network [35] is an outstanding example adopting this approach. In Æthereal, each router has a small amount of registers that are used to record the service time slots for pre-scheduled messages (usually from real-time tasks). Scheduling can be done in design- or compile-time, and programmed into the registers when the system boots. The pre-scheduled messages have well-defined latency and bandwidth, but the non-pre-scheduled messages can only request service time slots that are not assigned to pre-scheduled messages, and receives only best-effort service. 2) **Frame-based scheduling.** *Globally-synchronized frames* (GSF) [37] and *preemptive virtual clock* (PVC) [38] are two interesting works that propose economic implementations of frame-based scheduling in OCIN. Frame-based scheduling is an approximation of age-based scheduling, by coarsening the concept of “age” to be “frame”. GSF further extends the approximation by allowing the whole network sharing a common set of frames. PVC tries to address some drawbacks of GSF by re-introducing a rate-based scheduler for messages having the same frame mark.

Another set of researches aims at improving the overall system throughput rather than providing hard service guarantees. For example, Das *et al* [39, 40] propose to consider application-level criticalness with message priorities, in order to improve the overall progress of different applications. Jang *et al* [41] propose to sort SDRAM request messages inside the NoC to improve the effectiveness of memory controller and thus the memory throughput. Ma *et al* [42] propose the idea of *virtual networks*, which are essentially isolated regions in a common OCIN. Each virtual network or region is exclusively assigned to a virtual machine to facilitate workload consolidation, and a routing algorithm called *DBAR* is proposed to isolate interference of traffics from different regions. As attempts to improve energy-efficiency of OCINs, Mishra *et al* [43] propose DVFS mechanisms in OCIN that step down the frequency of lightly-used links and routers. Similar moralities and mechanisms have also been presented by Shang *et al* [44] and Chen *et al* [45] with slight differences in technical details.
2.3 Emerging On-Chip Interconnect Enabled OCINs

The characteristics of physical channels have a significant impact on OCIN’s performance and cost. Metal wires are not likely to scale at the same pace as transistors (Figure 1.2 shows one aspect of slow improvement of on-chip interconnects: the slow increase of wiring levels). In order to overcome the intrinsic inefficiency of conventional planar metal wires, a range of new interconnect topologies have been proposed and leveraged by novel OCIN architectures.

**3D-stacking.** In 3D-stacked processors, TSVs can potentially have low parasitics and high density, and serve as a very efficient interconnect alternative. In addition, with 3DIC, it is straightforward to implement high-dimensionality topologies that potentially have lower network diameter and better performance. For example, Li et al [46] propose a 3D mesh network of a 3D-stacked CMP. In their work, in order to better leverage the high efficiency of TSVs, they propose a hybrid vertical-bus-and-planar-mesh network that interconnects routers in the vertical dimension by a shared bus (an approach with some similarities with MECS [16]). Later researches further improve the 3D mesh network by reducing the router complexity [47] or by fine-granularity partitioning the router components [48].

**Transmission line.** On-chip transmission lines are shown to be more energy-efficient than conventional wires when used as global-links [49]. Therefore it is possible to leverage transmission lines as an express channel overlaid on a regular 2D topology (*e.g.* mesh), as is proposed by Chang et al [49]. In their more recent work [21], they further propose to take application-level features, such as spatial distribution of traffic density, in to account when dynamically allocate the transmission line to different communicating parties.

**Optical interconnect.** There have been heated interests in on-chip optical interconnects due to the strong belief that they clearly outperform electrical interconnects as have been demonstrated by off-chip optical interconnects. A plurality of researches have been done, leverage optical interconnects in on-chip bus [50], circuit-switched torus [51], token-crossbar [2, 52], internal network inside memory array [53], inter-die network for multi-chip module [54], switching fabrics for high-radix data-center switch [55], and many more. Conventionally, researchers have
been focused on on-chip interconnects that are implemented with passive waveguides. However, such implementation has drawbacks such as wasting laser power and thermal instability. Very recently, researchers start to explore alternative free-space optical interconnects that are free of those problems [56, 57]. Overall, while promising, optical interconnects for OCIN still remain a questionable candidate whose applicability largely depends on the maturity of the interconnect technology.

2.4 Simulation and Modeling Tools

Architectural researches in NoC largely relies on simulation and modeling tools, especially those early-stage tools that can rapidly explore the design space. Many cycle-accurate simulators have been developed, and available to be used under public-domain license. Booksim [24] is a simulator widely used due to its nice structure and built-in utilities to model a wide-range of OCIN architectures. OMNet++ [58] provides building blocks to construct both off-chip and on-chip networks. Agarwal et al develop an OCIN simulator called GARNET [59] that is then integrated into GEMS and gem-5 full-system simulators [60] to enable close-loop simulation. Early-stage power and area models are also available, such as Orion 2.0 [61] that generates detailed area and power reports for a specified OCIN architecture. Orion 2.0 is then integrated into McPAT [62], a area/power model and optimizer for full CMPs.
Chapter 3

Leveraging Emerging Interconnect Technologies

We have reviewed basic knowledge of OCIN and recent work in previous chapters. In this chapter, we will first present our work on OCIN architectures leveraging emerging on-chip interconnect technologies. In the first section, an 3D OCIN architecture based on AC-coupling vertical links is presented [108]; in the second section, we explore the applicability of free-space optical interconnects in OCIN and compare it with prior work using conventional optical waveguides [111].

3.1 A 3D Network-on-Chip Based on AC-Coupling Links

3.1.1 Introduction

In recent years 3D integration (3DIC) is becoming a heated research area and useful technology to enable the massive integration of transistors and devices in a single chip, and to address the interconnect challenges facing large-scale integrated circuits. One extreme example is Intel’s 80-core Teraflop processor [7] which stacks memories on top of cores. As demonstrated by Intel’s 80-core processor as well as a number of other chips, 3DICs bring brand new opportunities to integrate many and heterogeneous components in a single chip, to deliver performance and
Despite the merits mentioned above, the additional complexities introduced by 3DIC to the fabrication process cast a shadow on the picture. In through-silicon-via (TSV) based 3DIC, forming vertical links to interconnect different layers requires additional and unconventional processing steps involving high aspect-ratio etching, wafer thinning, bonding, etc. Defects that occur during the extra processes cause performance and yield loss, hindering the wide adoption of 3DIC. Consequently, vertical interconnects have been identified as one of the major technical difficulties of 3DIC technology [63].

This hurdle of fabrication has led to the advent of AC-coupling communication schemes recently. In these schemes, electromagnetic fields serve as the medium through which signals are transmitted vertically. The proposed AC-coupling links can be categorized based on the carrier field in capacitive-coupling links [64–66] and inductive-coupling links [67–70]. Fig. 3.1 provides a conceptual view of the three different communication media. The advantage of inductive/capacitive-coupling links is that the transceiver can be implemented solely by conventional 2D process. Additional 3D processing steps to form TSVs are eliminated, so that the fabrication complexities are reduced. In addition, it is possible to improve the performance of inductive/capacitive-coupling links with technology scaling [71].

The fourth row in Table 3.1 shows the single link data rates of inductive/capacitive-links developed by most recent work. It can be seen that the single link bandwidth of such links can reach 11Gbps, which is sufficient for vertical links in 3DIC. However, the area of these links is likely to be significantly larger than TSVs, as can be seen in the fifth and the sixth rows of Table 3.1. As area is closely coupled to interconnect density, chip yield and cost, and even performance, the inferior area efficiency of inductive/capacitive links renders their
Table 3.1: Recent Work on Inductive/Capacitive-Coupling Vertical Links

<table>
<thead>
<tr>
<th></th>
<th>Xu05 [70]</th>
<th>Ishikuro07 [67]</th>
<th>Miura08 [68]</th>
<th>Miura09 [69]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Inductive</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Energy (pJ/b)</td>
<td>17</td>
<td>&gt;390</td>
<td>0.14</td>
<td>1.4</td>
</tr>
<tr>
<td>Data Rate (GB/s)</td>
<td>2.8</td>
<td>0.02</td>
<td>1</td>
<td>11</td>
</tr>
<tr>
<td>Inductor/Capacitor Size</td>
<td>150×150um²</td>
<td>&gt;0.6mm²</td>
<td>29um (diameter)</td>
<td>120um (diameter)</td>
</tr>
<tr>
<td>Inductor/Capacitor Pitch</td>
<td>200um</td>
<td>NA</td>
<td>30um</td>
<td>NA</td>
</tr>
<tr>
<td>Active Area (mm²)</td>
<td>NA</td>
<td>&gt;0.006</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Total Area (mm²)</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>0.0015</td>
</tr>
<tr>
<td>Distance (um)</td>
<td>90</td>
<td>1200</td>
<td>15</td>
<td>15</td>
</tr>
<tr>
<td>Process</td>
<td>.35um</td>
<td>.25um</td>
<td>90nm</td>
<td>180nm</td>
</tr>
<tr>
<td><strong>Capacitive</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Energy (pJ/b)</td>
<td>3</td>
<td>0.08</td>
<td>0.27</td>
<td>0.39</td>
</tr>
<tr>
<td>Data Rate (GB/s)</td>
<td>1.8</td>
<td>1.7-2.46</td>
<td>10</td>
<td>11</td>
</tr>
<tr>
<td>Inductor/Capacitor Size</td>
<td>NA</td>
<td>8×8-20×20um²</td>
<td>48×18um²</td>
<td>72×38um²</td>
</tr>
<tr>
<td>Inductor/Capacitor Pitch</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Active Area (mm²)</td>
<td>NA</td>
<td>NA</td>
<td>0.0012</td>
<td>0.0021</td>
</tr>
<tr>
<td>Total Area (mm²)</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Distance (um)</td>
<td>&lt;10</td>
<td>1</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>Process</td>
<td>180nm</td>
<td>130nm</td>
<td>180nm</td>
<td></td>
</tr>
</tbody>
</table>

applicability in high density 3DIC a question.

On the other hand, TSV-based 3D NoC is believed to achieve better performance and energy efficiency than 2D NoCs [72]. However, the additional complexity and cost introduced by TSVs make the use of 3D NoC hard to justify. In addition, none of previous studies have considered using inductive/capacitive-coupling links as the vertical interconnect, which has low cost but introduces area overheads. As we will show shortly, in the worst case using inductive/capacitive-coupling links can increase the NoC router area by over two times.

Our focus of this section is to improve the area efficiency of inductive/capacitive-coupling links, in order to improve their practicality in 3D NoC. To achieve this goal, three techniques are proposed:

- **Serial communication** on the vertical links is firstly proposed to constrain the area overheads. However, while it is an effective way to save area, serial communication incurs performance penalty since the effective vertical bandwidth is reduced.

- **Data compression** is then proposed to recover the performance by reducing the bandwidth requirements for vertical links. Different from previous work, we implement an in-transceiver data compression scheme where compression is performed simultaneously with serial communication, incurring no latency overhead nor changes to original NoC.
• **High-speed asynchronous transmission** is lastly exploited to further boost the performance. Inductive/capacitive-coupling vertical links combined with serial transceiver become a natural asynchronous interface capable of running at a frequency much higher than the 2D links. This provides opportunities to further improve the area efficiency of vertical links.

In what follows, we perform a comprehensive evaluation of a 3D CMP interconnected by NoC using inductive/capacitive-coupling, taking into consideration performance, energy, and area. With the combination of serialization, data compression, and asynchronous transmission, experiment results show that the overheads of performance, energy, and area caused by using inductive/capacitive-coupling links can all be bounded under 10%. In addition, the area efficiency of these links is also brought to over 90% of TSV. Considering the benefit of reducing fabrication difficulties, this is a positive result to support inductive/capacitive-coupling communication as a competitive alternative in real 3D designs.

### 3.1.2 Vertical Interconnections in 3DIC

#### 3.1.2.1 Through Silicon Vias

TSVs are the most commonly used vertical interconnects. Usually the dimensions of TSV are quite small, resulting in low parasitics which in turn translate into high bandwidth and low energy consumption. The specifications of TSV depend on the process and can vary diversely. For example, IBM, IMEC, and many more organizations have presented different pitches ranging from several microns up to tens of microns [3]. In this work we assume a moderate TSV whose dimensions and RC data is generalized in Table 3.2 [3]. The delay and energy results are obtained from SPICE simulation by using an RC network model taking into account parasitics of both TSV and substrate, and assuming that simple buffers are needed at each end of the TSV [73].

<table>
<thead>
<tr>
<th>Size</th>
<th>10×10um²</th>
</tr>
</thead>
<tbody>
<tr>
<td>Height</td>
<td>50um</td>
</tr>
<tr>
<td>Pitch</td>
<td>20um</td>
</tr>
<tr>
<td>$R_{TSV}$</td>
<td>14mΩ</td>
</tr>
<tr>
<td>$C_{TSV}$</td>
<td>284fF</td>
</tr>
<tr>
<td>Delay</td>
<td>60.6ps</td>
</tr>
<tr>
<td>Energy</td>
<td>0.125pJ</td>
</tr>
</tbody>
</table>
3.1.2.2 AC-Coupling Vertical Links

Table 3.1 summarizes most recently demonstrated designs of both inductive-coupling links (the top table) and capacitive-coupling links (the bottom table). Note that some of the entries do not have data because not all publications reveal a complete set of data.

Both kinds of vertical links rely on AC-coupling through different electromagnetic fields. Inductive-coupling uses magnetic flux generated by the current flowing in metal solenoids while capacitive-coupling uses the electric field produced by the potential difference on two metal plates (Fig. 3.1a and Fig. 3.1b). Due to the different natures of the carrier fields, their characteristics also differ. The communication distance of inductive-coupling is relatively long. This long communication distance is exploited in [67] to realize an attachable wireless chip probe and most recently in [74] for a memory-on-processor chip where the communication distance is 120um. Capacitive-coupling, compared to inductive-coupling, has much shorter communication distance which constrains it to be used only in face-to-face stacking. Nevertheless, the footprints of capacitive-coupling are likely to be much smaller than inductive-coupling, incurring much lower area overhead than inductive-coupling.

Existing inductive-coupling and capacitive-coupling designs can achieve a data rate as high as 11Gb/s or energy consumption as low as 0.14pJ/b, which are both comparable to TSV (the third and fourth rows in Table 3.1). In addition, they can be implemented with conventional 2D process without additional processing steps. The metal solenoids and capacitive plates can be formed simply by the top metal of the integrated circuit. However, as Table 3.1 shows (the fifth and sixth rows), most reported inductive/capacitive-coupling links have much larger area than TSV, ranging from hundreds of square microns to tens of thousands of square microns.

In the remaining evaluations for AC-coupling links, the characteristics of inductive/capacitive-coupling links are based on Table 3.1 where more conservative processes (90nm or older) are used. However, for the transceivers, and the 3D NoC and the CMP introduced in the following sections, a more contemporary technology node of 65nm is assumed. Hence, our evaluations hereafter can be deemed as the worst-case results, since it is possible to improve inductive/capacitive-coupling
links’ performance with technology scaling [71]. Moreover, due to the incompleteness and large variation of reported data on inductive/capacitive-coupling links, we design our evaluations to consider a range of possible characteristics of inductive/capacitive-coupling links. That is, we take into consideration of the sensitivity of results to parameters of underlying technologies.

3.1.3 3D NoC Router Architecture

Fig. 3.19b depicts the architecture of a generic 3D NoC router, which contains the components we have discussed in Section 1.2.2.1 (physical channels, input buffers, routing, arbitration, flow-control logics, and crossbar). However, for NoC routers in the 2D mesh topology, there are only five ports: East, North, West, South, and a local port connected to the processing element (PE). As a baseline 3D NoC router, we assume a 3D mesh topology and add two ports (Up/Down) to the router for vertical communication. In addition, the output links of Up/Down ports are connected to the inductive/capacitive-coupling transceiver instead of TSVs. Fig. 3.19b illustrates these modifications in the dotted box. Finally, We assume wormhole switching and minimal adaptive routing with the NoC router, and use ORION [61] to obtain its power and area for 65nm node. The router specifications are generalized in Table 3.4.

3.1.4 Addressing the Area Overhead with Serialization

One way to address the large area overhead of inductive/capacitive-coupling vertical links is to adopt serial communication where the data is transmitted on a single link. For serial communication to carry out on the vertical links, the inductive/capacitive-coupling transceivers need to be able to do data serialization when data is delivered from parallel 2D links to the serial 3D link. The serialization ratio is defined as the ratio between the number of parallel and serial links.

We adopt the transceiver design proposed in [75] originally for TSV, shown in Figure 3.2b and 3.2c. This transceiver effectively implements the start-1 stop-0 serial protocol where the data payload is sandwiched between a start bit (bit “1”) and a stop bit (bit “0”). This scheme does not require dedicated sideband signals for handshaking and is quite efficient in terms of area and speed. The
Table 3.3: Implementation Results of 128-bit Transceiver with STMicro 65nm technology at 5GHz

<table>
<thead>
<tr>
<th>Serialization Ratio</th>
<th>Energy (pJ/b)</th>
<th>Area (um$^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:1</td>
<td>0.3581</td>
<td>4776</td>
</tr>
<tr>
<td>8:1</td>
<td>0.5987</td>
<td>4102</td>
</tr>
<tr>
<td>16:1</td>
<td>0.9063</td>
<td>3890</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Serialization Ratio</th>
<th>Energy (pJ/b)</th>
<th>Area (um$^2$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>4:1</td>
<td>0.7785</td>
<td>8473</td>
</tr>
<tr>
<td>8:1</td>
<td>0.9528</td>
<td>7009</td>
</tr>
<tr>
<td>16:1</td>
<td>1.4801</td>
<td>6334</td>
</tr>
</tbody>
</table>

Table 3.4: Baseline Router Configuration

<table>
<thead>
<tr>
<th>Configurations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pipeline 2-stage</td>
</tr>
<tr>
<td>with look-ahead routing</td>
</tr>
<tr>
<td>Switching Wormhole</td>
</tr>
<tr>
<td>Routing Algorithm Minimal, Fully-Adaptive</td>
</tr>
<tr>
<td>Flit Size 128-bit</td>
</tr>
<tr>
<td>No. of Ports 7</td>
</tr>
<tr>
<td>Virtual Channels per Port 6</td>
</tr>
<tr>
<td>Buffer Depth 8</td>
</tr>
</tbody>
</table>

receiver only needs to detect a “0” to “1” transition to capture the correct data. In addition, the transceiver owns private clock generators (ring-oscillator) which allows for asynchronous transmission.

We implement the transceiver with STMicroelectronics 65nm technology. The transceiver can run at as high as 5Ghz using this technology. The first half of Table 3.3 shows power and area of 128-bit transceivers (transmitter + receiver) with different serialization ratios. Note that a 128-bit transceiver may consist of thirty-two 4-bit sub-transceivers for 4:1 serialization, sixteen 8-bit sub-transceivers for 8:1 serialization, etc.

The impact of serialization on the area of an NoC router is evaluated based on the implementation results. The specification of baseline router is in Table 3.4. In this evaluation, we sweep over a range of different vertical link pitches and see how much overhead they each introduce.

Figure 3.3 shows the total router area including area taken by vertical links, normalized to the router area without vertical links. As a guidance, Figure 3.3 also plots relative bandwidth reduction with serialization. We notice that with a
pitch as small as $20\mu m$ (same as the TSV), serialization does not help much to reduce the area overhead, which is only 3% even without serialization (ratio = 1:1). Considering the bandwidth loss, we consider it not attractive to use serialization with TSV to reduce area. However, with the large pitches of inductive/capacitive-coupling links, up to two times of area overhead is observed without serialization. In addition, we see that with a serialization ratio of 8:1, the area of the router with inductive/capacitive links is only 1.1 times larger than the router with TSV, even when the pitch is increased to 120um.
Figure 3.3: Area overheads when the serialization ratio varies, for different vertical link pitches. The results have been normalized to the case where the area of vertical links is not included. The trend line shows the normalized bandwidth reduction with increasing serialization ratio.

3.1.5 Data Compression

As shown by Fig. 3.3 and subsequently in Section 3.1.6, serialization can incur considerable performance penalty. Therefore, we propose to use data compression to reduce the performance penalty and develop an in-transceiver data compression design.

3.1.5.1 The Concept

It has been observed that only a few data patterns dominate the overall on-chip communications [76]. Furthermore consecutive-zero patterns (hereafter zero-pattern) are found to cover a significant portion of the frequent patterns. In [77], frequent-pattern compression not only reduces cache pressure but also improves NoC performance, boosting the overall performance. While data compression is performed in NIC or PE in [77], producing variable length packets, in this work we are focused on a simple and efficient approach specifically to reduce the data volume traveling on the vertical links. To achieve this goal, we extend the serial transceiver discussed in the previous section with zero-pattern compression/decompression modules. The benefits of this in-transceiver data compression are (a) data compression is overlapped with serialization, incurring no extra delays and (b) it has no impacts on the NoC architecture which can be optimized obliviously of data...
Zero-pattern compression is performed at the granularity of single 32-bit words. Conceptually, if one word in a flit contains all zeros, this word is omitted for transmission and instead only a few bits to encode this situation are sent. For instance, consider a 128-bit wide flit that contains four 32-bit words (Fig. 3.4). If some word in the flit contains all zeros, it is eliminated and only the remaining three words are sent. Encoding can be done in hardware by generating a 4-bit mask. A “1” in the mask represents a valid word while a “0” indicates the corresponding word is omitted. This mask is sent together with the compressed data, and used by the receiver to recover the data.

In addition, the bits of the flit have to be interleaved across the sub-transceivers to really save cycles. This can be explained by the example shown in Fig. 3.4, where the serialization ratio is 4:1 and 32 4-bit sub-transceivers are used to deliver the 128-bit flit. On the transmitter side, each sub-transmitter \(i\) of the thirty-two sub-transmitters sends the \(i\)th bits of word 0, 1, 2, and 3 of the flit respectively. If, for example, word 1 contains all zeros, then the second bit in each sub-transmitter will be skipped, effectively saving one cycle. On the receiver side, the received bits can be rearranged to restore the data.

### 3.1.5.2 Implementation

Here we use an 128-bit 4:1 transceiver to illustrate the implementation of data-compression enabled transceiver. Except for some minor changes, the implementation is similar for other serialization ratios. The synthesis results of the enhanced transceivers with 65nm technology are listed in the second half of Table 3.3.
Cycles
0 1 2 ... k+2
0. Load data into data register
Generate mask
1. Send start-bit on the serial link
Send mask on parallel links
(k+2). Reset transmitter state
... Reset mask register
Send k valid data bits

(a) Transmitter Operations

Cycles
0 1 2 ... k+2
0. Rising-edge detected, latch start-bit
Latch mask into register
1. Receive the 1st valid bit
Arrange the valid bit to proper position
Receive all valid data bits
Reset mask register
... Receive k valid data bits
(k+2). Send data to input buffer

(b) Receiver Operations

Figure 3.5: Operations in a single data transaction. The text in shaded boxes refers to additional operations of data compression.

Figure 3.6: Data compression hardware details: (a) Compression. (b) Decompression.

Data Compression Operations — Fig. 3.5 shows the additional operations needed by data compression during each transmission cycle. In Fig. 3.5a, simultaneously with loading data into shift register, the transmitter also generates and latches the 4-bit mask into a register. This mask is sent on parallel links, together with the start-bit on the serial link in the second cycle, and each valid data bit is sent one-by-one in following cycles. At the end of the transaction, all valid data bits are sent, and the mask register is reset. In Fig. 3.5b, when detecting a rising-edge on the serial link, the receiver latches the start-bit into its data register and the mask on the parallel links into its mask register. Then when following bits come in, the mask register is used to rearrange the bits to recover the original data. At the end of the transaction, the decompressed data is delivered and the mask register is cleared.

Hardware Details — Supporting data compression only adds a few changes
to the transceiver shown in Fig. 3.2. Specifically, only the shaded parts in Fig. 3.2b and 3.2c are modified.

The modified shaded part in Fig. 3.2b is shown in Fig. 3.6a. For clarity, clock and data signals are omitted. As can be seen, there are two major changes:

- The ring-counter is replaced by the mask register, combined with some logic, to control the transmission sequence.
- The data register is not a pure shift register, but instead is equipped with a 4-to-1 multiplexer to forward any bits to the head of the register.

The data and mask registers are loaded when crossbar buffer becomes ready, as shown in Fig. 3.5a. In each following transmission cycle, with a first-one detector, the mask register controls the multiplexer to select the next valid bit to be sent. The output from the first-one detector is also fed-back to clear the leading “1” in the mask register in each cycle. After all valid bits are sent, the mask register should contain only “0”s and reset the transmitter state.

The modifications to the receiver, as shown in Fig. 3.6b, are very similar to the transmitter. The difference is that instead of using a multiplexer, the data register de-multiplexes the incoming serial data using data enables generated by the mask register. Again, after all valid bits are received, the mask register is cleared and the state of receiver is reset. One subtlety with receiver is that the data register needs to be cleared to ‘0’ before receiving the first valid bit, to eliminate obsolete bits from last transaction.

### 3.1.6 Experiment Result

Inductive/capacitive-coupling vertical links are evaluated in the context of a 3D NoC with a cycle accurate simulator adapted from NoX [77]. In the simulator we model a 2-tier, 16-core 3D CMP with a shared L2 cache distributed into 16 banks. These 32 nodes are interconnected by a 3D NoC configured as a 3D mesh (4×4 by 2 layers). We choose a mosaic floorplan where each core is surrounded by 5 L2 cache banks in the cardinal directions (except for cores on edges). The NoC router specification is given in Table 3.4. The area and energy of proposed transceivers is given in Table 3.3, for STMicro 65nm technology and 5GHz frequency. The whole CMP (processor, cache, NoC) is clocked at 1.5Ghz. For synthetic traffics, we
study the impact of serialization and asynchronous transmission; for multi-thread applications, we evaluate all the three proposed approaches.

**Note:** For comparison purpose, we define baseline$_{TSV}$ as the case that TSV is used without serialization. As discussed in Section 3.1.4, serialization with TSV only gains negligible area savings, but incurs significant performance loss. The two proposed approaches (data compression and asynchronous transmission) to recover performance are only effective when serialization is used, where vertical link bandwidth is the bottleneck. Otherwise, speeding up the vertical links by either approach does not help to improve the performance which is instead limited by the bandwidth of 2D links. Therefore, none of serialization, data compression, and asynchronous transmission is used in baseline$_{TSV}$.

### 3.1.6.1 Study with Synthetic Traffics

![Graph](image)

Figure 3.7: Message latencies for different traffic patterns without data compression.

![Graph](image)

Figure 3.8: Energy consumption normalized to baseline$_{TSV}$ for different traffic patterns.
Fig. 3.7 shows the average message latencies for four synthetic traffics with different load ratios and serialization ratios. Note that the performance when the serialization ratio is 1:1 is just the same as baseline\(T_{SV}\). It can be seen from the results using serialization with the vertical links negatively affects the performance of NoC. With the serialization ratio increasing, the message latency under the same load ratio increases, and the saturation load ratio decreases. Within the four traffic patterns, nearest neighbor shows least performance reduction, while bit complement shows worst performance loss.

The energy consumption of the interconnection network normalized to baseline\( T_{SV}\) is shown in Fig. 3.8. Since the energy consumption of inductive/capacitive-coupling vertical links can range from 0.1pJ to several hundred pJ, we study how the energy overhead changes over a range of link energies. This figure shows that both high serialization ratio and large inductive/capacitive-coupling link energy increase the energy overhead as compared to TSV. In addition, inductive/capacitive-coupling link energy affects the energy overhead to a greater extent. In order to constrain the overhead under 10% at 8:1 serialization ratio, the link energy has to be less than 0.5pJ/b.

Then, we study the effectiveness of high-speed asynchronous transmission in recovering performance loss. In this evaluation, a moderate load ratio of 0.05, and an serialization ratio of 4:1\(^1\) are assumed. The message latencies with different transmission frequencies are plotted in Fig. 3.9 and normalized to the baseline\( T_{SV}\).

\(^1\)The injection ratio is selected to be the saturation point of uniform traffic with 4:1 serialization ratio.
The $y$-axis of this figure only ranges from 1 to 3, as we consider that latencies that are three times higher are already beyond saturation [78]. It can be seen that with the successively increasing transmission frequency, the packet latency is continuously decreasing. At 5Ghz, the average latency overheads is reduced to 30%.

![Figure 3.10: Effectiveness of data compression in improving IPC.](image1)

![Figure 3.11: Impacts of higher transmission frequency on IPC.](image2)

### 3.1.6.2 Study with Multi-Thread Applications

To study the impact of proposed techniques on system-level performance, four multi-thread benchmarks ($sap$, $sjas$, $sjbb$, and $tpc$) are simulated. Each of the benchmarks runs 16 threads on the 16 cores. In every run we first warm up the cache states by 3 million instructions for each thread, and then collect statistics for the next 5 million instructions. The IPCs of the benchmarks both without data compression and with data compression are plotted in Fig. 3.10 and also normalized to the baseline. It can be seen that serialization does negatively impact the performance, and for a serialization ratio of 8:1, the performance is reduced
to around 60% of original performance. With data compression used, this performance gap shrinks, especially for sap and tpc that have relatively high zero-pattern ratios \(^2\).

The energy consumption of interconnection network normalized to baseline\(_{TSV}\) is plotted in Fig. 3.12 without data compression, and in Fig. 3.13 with data compression. Without data compression, the energy overheads of the benchmarks are similar to that of synthetic traffics. With 0.5pJ/b link energy, the overhead can be limited under 10%. With data compression, on the other hand, the average energy overhead is reduced to 5% with 0.5pJ/b link energy.

Although effective, data compression is not able to recover the performance to satisfaction (only 70% to 75% in the 8:1 case). Asynchronous transmission is then exploited to further reduce the performance gap. For the serialization ratio of 8:1, we improve the transmission frequency to 2, 3, 4, and 5Ghz respectively.

\(^2\)It is found that for sap and tpc, 60-70% of transmitted words contains only “0” bits.
Figure 3.14: Area efficiency of the interconnection network when different techniques are successively applied.

The results normalized to the baseline are shown in Fig. 3.11, which validate that increasing transceiver frequency improves the IPC. At 5Ghz, the performance losses of the benchmarks have been reduced to under 6%.

3.1.6.3 Area Efficiency

For a more comprehensive evaluation, we study the area efficiencies of the interconnection network in terms of the performance of multi-thread applications. We define the area efficiency to be \( \frac{IPC}{Router\ Area} \). Three cases are compared: (a) no data compression nor asynchronous transmission is used; (b) only data compression is used; and (c) both data compression and asynchronous transmission are used. For all cases, the serialization ratio is 8:1. For asynchronous transmission, the transmission frequency is set to 5Ghz. The inductive/capacitive-coupling link pitch is set to 70um, a moderate value among existing designs. The area efficiencies of using inductive/capacitive-coupling vertical links are normalized to baseline TSV.

The results are shown in Fig. 3.14. Without data compression and asynchronous transmission, the average area efficiency of the four benchmarks is only 55% of baseline TSV, with the highest efficiency being 60%, and lowest 48%. When data compression is applied, the average area efficiency is increased to 67%, with the highest being 70%, and lowest 65%. Finally, the area efficiencies of all four benchmarks are improved to over 90% when both data compression and asynchronous transmission are used, with an average of 92%.
3.1.7 Summary

In this section we propose design methodologies for using inductive/capacitive-coupling vertical links in 3D NoC. To address the area overheads of these links, serial communication is adopted and it is shown that to reduce the overhead under 10%, the serialization ratio has to be at least 8:1. However, high serialization ratios will incur significant performance losses. Hence, data compression and high-speed asynchronous transmission are proposed to reduce the performance penalty incurred by serialization. We design and implement asynchronous transceivers that can do in-place zero-pattern compression simultaneously with serialization. Data compression is shown to not only increase performance, but also reduce energy overhead. Combining both techniques increase the performance of multi-thread applications to over 94% of the baseline case. Finally, the area efficiency is brought to above 90% of TSV by the combination of the three techniques. The energy overheads are shown to be bounded under 10% with a link energy of 0.5pJ/b.

3.2 \( F^2 \text{BFLY: A Free-Space Optical Network-on-Chip} \)

In the previous section, we presented design methodologies for 3D-stacked OCIN using wireless interconnects. However, in that work the use of wireless interconnects only reduces fabrication difficulties, but does not improve the performance and the energy-efficiency. The horizontal channels are still implemented with metal wires whose performance scales much slower than transistor. To overcome the intrinsic inefficiency of electrical interconnects, researchers have leveraged recent developments in chip photonics to design novel optical network-on-chip (NoC). However, existing optical NoCs are mostly based on passively switched, channel-guided optical interconnect in which large amount of power is wasted in heating the micro-rings and maintaining the optical signal integrity.

In the rest of this chapter we present an optical NoC based on free-space optical interconnect in which optical signals emitted from the transmitter is propagated in the free space in the package. With lower attenuation and no coupling effects, free-space optical interconnects have less overheads to maintain the signal.
integrity, and no energy waste for heating micro-rings. In addition, we propose a novel cost-effective wavelength-switching method where a refractive grating layer directs optical signals in different wavelengths to different photodetectors without collision. Based on the above interconnect and switching technologies, we propose free flattened butterfly (F$^2$BFLY) NoC which features both high-radix network and dense free-space optical interconnects to improve the performance while reducing the power. Our experiment results, comparing F$^2$BFLY with state-of-the-art electrical and optical on-chip networks, show that it is a highly competitive interconnect substrate for many-core architectures.

3.2.1 Interconnect Challenges for Many-Core CMPs

As we have discussed in Chapter 1 and 2, the continuous scaling-up of many-core processors places rapidly increasing communication burden on the on-chip interconnection networks. While many researchers have been focused on designing novel router architectures and network topologies, some researchers take more futuristic steps and study the use and impact of emerging on-chip interconnects such as the optical interconnects on the design and the performance of on-chip interconnection networks.

A major branch of contemporary NoC researches inherit ideas from the long-established computer and multiprocessor networks [12], and build on-chip networks based on metal wires on the silicon. Driven by the requirements of low latency and low complexity, researchers have been working on reducing the router pipeline stages and efficient switching mechanisms [23, 25, 26]. An interesting recent trend is to leverage the ample on-chip wiring resources to build high-radix, richly interconnected NoCs that features both high bi-section bandwidth and low network diameter [15, 16, 28, 79]. Grot et al. [16] generalize these high-radix NoCs as express cube topologies and perform a comprehensive study of the design space of express cubes. Their results show that by wisely utilizing the abundant wiring resources, express cubes can produce significantly higher performance and consume lower power than conventional topologies.

However, future performance elevation with electrical NoC is impeded by the slow improvement in the on-chip interconnect technology. The wiring density is not
increasing as fast as device density, since to maintain acceptable signal integrity, the feature size of wires cannot scale at the same pace as devices. In addition, the increase of metal levels has almost stopped, and it is projected that from now to the 11nm node by 2022, only 4 more metal levels will be added (from 11 to 15 levels) [1]. Furthermore, the power spent by on-chip interconnects starts to dominate the overall chip power [80, 81]. The express cube topologies will suffer even more from above issues as the wiring intensity of these topologies increases with the network size \( O(N) \) for flattened butterfly [15] and \( O(\sqrt{N}) \) for MECS [16, 79], where \( N \) is the number of nodes in the network. This sets researchers on the quest of more scalable and power-efficient alternatives for on-chip interconnects.

Recent developments in nanophotonics have motivated researchers to design NoCs based on optical interconnects, which is intrinsically faster and less power consuming than electrical interconnects. The proposed optical NoCs [2, 22, 50–52, 82, 83] are mostly based on channel-guided optical interconnects, where optical signals are confined and propagated in on-chip waveguides.

**Channel-Guided Optical Interconnects**— Figure 4.14 provides a conceptual view of channel-guided optical interconnects and illustrates the necessary components. On-chip waveguides (fabricated with poly-Si, crystalline Si [2, 22], or polymer) serve as the physical channel carrying optical signals. A continuous laser beam is provided by an external laser source and modulated into on-off digital signals. The modulated light continues to travel downstream, possibly duplicated by a splitter, and finally detected and removed from the waveguides. Modulation and detection are both performed by micro-rings placed beside the waveguides, which can be electrically tuned into and out of resonance with a certain wavelength. In the resonant state, the coupling between the micro-rings and the waveguides becomes so strong that the light is dropped by the micro-rings. Therefore, electrically tuning the micro-rings achieves on-off modulation, and dropping the optical signals to a photodetector completes detection. To further increase the bandwidth, dense-wavelength-division-multiplexing (DWDM) is used to enable the waveguide to carry multiple wavelengths simultaneously. The micro-ring can selectively modulate or drop a given wavelength by adjusting its resonance frequency.

While channel-guided optical interconnects with DWDM are shown to be more energy-efficient than electrical interconnects, it has significant energy wastes that
Figure 3.15: Free-space optical interconnects: the actively modulated VCSEL emits encoded optical signals, which are then collimated and redirected by a series of optical devices, and finally detected by the photodetector.

offset the energy-efficiency of optical interconnects. First, the external laser source needs to continuously produce lights used for modulation. To make things worse, existing optical NoCs typically have a large number of long waveguides due to the lack of optical buffering. This requires the external laser source to provide high laser power on the order of tens of watts. Second, to achieve high bandwidth thousands of micro-rings are used as modulators and detectors. Thermal trimming is needed to ensure all the micro-rings to work with the right resonance frequencies. Besides technical difficulties, the total power needed for trimming is so high to be comparable with the laser source power. Analysis shows that the laser source power and the trimming power accounts for 75% of total power in channel-guided optical networks [84]. Note that the above two power components are static powers and pure overheads.

**Free-Space Optical Interconnects**—Most recently, free-space optical interconnects (FSOI) gains interests as it overcomes the inefficiencies of channel-guided optical interconnects. Figure 3.15 shows a conceptual view of FSOI. Compared with channel-guided interconnects, the major differences include:

- **Laser source and modulation**: Light is emitted from on-chip integrated laser sources [85–87] (in this work, we consider two different laser source technologies with different characteristics: vertical-cavity surface-emitting laser (VCSEL) and Kerr frequency combs. Thanks to 3D integration, there is no intrinsic constraint on what laser technology to be used within the chip. We will discuss the laser source selection issue in more details in the next subsection.). By switching on and off the integrated laser sources, active modulation is achieved with the laser source.
• **Light propagation:** Modulated lightwave propagates in the free-space in the package. In order to direct lightwave onto intended photodetectors, optical elements are inserted to achieve multiple functionalities of collimation, diffraction, and collection of light signal. We will discuss more about how to control the light’s direction below and in the following section.

• **Detection:** Instead of using micro-rings to couple light out of waveguides for detection, photo diodes are employed. Light beams are projected onto corresponding photo diodes under the control of optical elements.

From the above discussion, we can see that free-space optical interconnects eliminate the needs of external laser source and micro-rings, which are the sources of significant energy wastes. In addition, optical signals propagating in the free space experience no coupling and lower attenuation, which further improves the efficiency of optical links.

However, new challenges are also introduced: directing the emitted laser beam to the intended photodetector. Relying on nanofabrication processes, existing on-chip laser sources and photodetectors usually have fixed orientation (normal direction of the chip in our case) and cannot be freely aimed at each other. Therefore, optical elements are placed in the optical path to control the direction of beam propagation. Xue *et al.* [56] propose to use angled micro-mirrors in the package to re-direct the beams. However, this method suffers from technological difficulties and complexities. Fabricating micro-mirrors requires unconventional processes that are incompatible with planar CMOS processes. To make things worse, aligning the micro-mirrors requires precise control of 5 free variables in the 3D space \((x, y, z, \text{inclination angle } \theta, \text{and azimuth angle } \phi)\), whose feasibility remains a question. In addition, Xue *et al.* propose an all-to-all network where a laser beam will be reflected by multiple mirrors before reaching the destination. Planning the optical paths for all links in the complex all-to-all network is an intimidating task and adds more difficulties to alignment. Last but not the least, this also introduces problems to designing network architectures, which will be detailed in Section 3.2.3.

In contrast to previous approaches, we propose to achieve beam redirection with a single **Refractive Grating Layer** (RGL). A RGL is a planar thin-film diffractive optical element. One interesting feature of RGL is that the refractive angle
extent to which the light direction is changed) depends on the wavelength of the incident light. That is, controlling the optical path is as simple as selecting the wavelength.

Figure 3.16 provides the high level view of our optical system. We leverage 3D integration to stack the optical substrate (GaAs) on CMOS substrate (Si). On the ceiling of the package, a reflective RGL is deposited, which bounces back the beams incident normally to the RGL towards the corresponding photodetectors on the optical substrate. Since the diffraction angles of different wavelengths are different, we can direct the beams to intended photodetectors without collision. The advantages of this approach are: (a) the optical elements for beam redirection (RGL) can be fabricated with mature processes and (b) we only need to control 3 free variables (wavelength $\lambda$ and 2D locations of lasers and detectors) to ensure that the laser beams are projected onto intended detectors. The relative positions of laser sources and detectors can be precisely defined when fabricating the optical substrate; wavelengths $\lambda$ is well controlled in contemporary multi-wavelength on-chip lasers [85–87]. With the absence of alignment issues, the proposed optical system is more feasible to realize. Detailed analysis of the system is provided in Section 3.2.2.

**Free Flattened Butterfly (F\textsuperscript{2}BFLY):** We leverage the above proposed free-space optical interconnects to design an optical NoC called free flattened butterfly. Based on the special features of underlying interconnect technology, F\textsuperscript{2}BFLY fea-
Figure 3.17: The two RGL orientations directing laser beams to two orthogonal directions.

3.2.2 Designing Free-Space Optical Interconnects

In this subsection, we provide specific design considerations for the free-space optical interconnects illustrated in Figure 3.16, with a focus on enabling wavelength-switching with a high efficiency RGL:

- **RGL Design:** RGL is a thin-film component with periodic pattern structure, which diffracts different wavelengths of light from a common input direction into different angular output directions. When a light beam passes through this structure, it is diffracted (or redirected) and the angle of the output beam \( \theta \) depends on the wavelength \( \lambda \) and period \( \Lambda \), as given by \( \sin(\theta) = \lambda/\Lambda \). In this work, we use the reflection grating as RGL due to its advantages in diffraction efficiency and system packaging. The diffraction efficiency of the this kind of RGL is high, ranging from 80% up to 100% [88–90].

The index pattern of RGL is generated with conventional nanofabrication processes. However, each process step can only define *one specific pattern* (period and
orientation). Therefore it is impractical to generate arbitrary patterns on a single film, which requires highly complex mask and tremendous amount of fabrication costs. In addition, it also makes alignment much more challenging during the integration step, and the network routers highly complicated and power consuming. In this work we only assume two orientations on the RGL used: an $x$-orientation enabling $x$-dimension links and a $y$-orientation enabling $y$-dimension links (Figure 3.17). **Ideally, these two patterns will have the same grating period and therefore fabrication cost and alignment difficulty are both largely reduced.** However, due to the limitations of the laser sources (as discussed below), we may need to *spatially multiplex*\(^3\) different grating periods on the same RGL which slightly increase the complexity of RGL. Finally, restricting the complexity of grating patterns presents both challenges and opportunities for designing an free-space optical network, as we will discuss in detail in Section 3.2.3.

- **Laser Sources and Photodetectors:** Contemporary integrated laser sources feature small footprints on the order of tens of microns, high modulation speeds up to 40Gbps [91]. In our setting, we also require the laser technology to be able to generate a wide range of wavelengths on the chip, in order for a single source node to reach different destination nodes in a certain dimension. There are several on-chip laser technologies providing this capabilities [85–87]. However, they each have own limitations, and need to be designed together with the RGL pattern. In this work, we propose two configurations using state-of-the-art multi-wavelength laser sources:

  - **VCSEL:** VCSELs are preferred and more practical laser sources to be used in our system, and have been used pervasively as integrated laser sources in telecommunication scenarios. However, the limitation of VCSEL is that the range of wavelength that can be generated from a multi-wavelength VCSEL array is relatively narrow, typically 200nm around the communication frequency [85, 86]. While this range is sufficient for fiber communication application, it cannot provide enough span in our work to reach all possible destinations in a same dimension with a single grating period. Therefore,

\(^3\)Spatially multiplexing the RGL means that during the fabrication process, different grating patterns are generated at different locations of RGL. In our case, we first spatially multiplex the orientation of the grating patterns as shown in Figure 3.17, and then multiplex the periods for each of the $x$- and $y$-patterns.
when using multi-wavelength VCSEL array, we also multiplex different grating periods on the same RGL in addition to multiplexing the grating orientations as shown in Figure 3.17.

More specifically, at each node, the multi-wavelength VCSEL array is further sub-divided into three sub-arrays. Each sub-array contains identical VCSELs providing a wavelength range of 800–1000nm. However, the grating periods of the RGL on top of each sub-array are different. Based on our calculation, the three grating periods are chosen to be 3.3um, 1.2um, and .88um (see Table 3.5), in order for the beams generated from different VCSEL sub-arrays to reach different, non-overlapping sets of destinations. Put in another way, the spatially multiplexed grating periods and VCSEL sub-arrays provide the initial offsets of the diffraction angles \( \theta_0 \), and the wavelength variation (800–1000nm) inside each sub-array introduces additional angle shifts \( \theta_1 \). And the total diffraction angle \( \theta \) can be roughly thought as the combination of the above two angles.

- **Broadband Sources:** The Kerr frequency combs can provide the wavelength range on the order of 500–1000nm [87]. While it has the potential to provide wide wavelength range and small footprints, its mass integration on the chip and maturity in fabrication still needs more researches. Nevertheless, there is no intrinsic limitation preventing them from being used as multi-wavelength arrays as in the case of VCSELs. Using the broadband sources, we are able to use only a single RGL period along each dimension which further reduces the difficulty in RGL fabrication and alignment in integration. In this configuration, we only vary the wavelength across a wide range (780nm–1500nm), and use a single RGL period of 1.6um.

In summary, using VCSEL array is a more practical choice, however requires slightly increased complexity in RGL fabrication and alignment. The broadband laser sources (Kerr frequency combs) eliminate the need of multiplexed RGL and reduce fabrication and alignment difficulties. However, their maturity and mass integration issue can be challenging. Since VCSEL is a more mature integrated laser source and has been widely used, in this work we still
prefer it and design the NoC architecture based on it. However, as we have mentioned, there is no intrinsic constraint ruling out the use of broadband sources considered here. And their impact can be taken into consideration once mature mass integration of such sources has been demonstrated.

The key parameters of the overall optical system is summarized in Table 3.5. Note that the VCSEL case refers to the configuration using multi-wavelength VCSELs, and the Broadband case refers to the configuration using broadband multi-wavelength sources as possible alternatives. The resonant cavity photodiodes incorporates the resonant cavity similar to VCSELs to amplify the received signal, and offers high sensitivity and bandwidth [92]. For the free-space optical links in this work, we assume the high-speed VSCEL and photodetector design in [56].

<table>
<thead>
<tr>
<th>Package Height</th>
<th>VCSEL: 5mm</th>
<th>Broadband: 3mm</th>
</tr>
</thead>
<tbody>
<tr>
<td>Communication Distances</td>
<td>Max. 12mm</td>
<td>Min. 1.5mm</td>
</tr>
<tr>
<td>RGL and Laser Sources</td>
<td>VCSEL: .8–1um 300, 830, 1130 l/mm</td>
<td>Broadband: .78–1.5um 660 l/mm</td>
</tr>
<tr>
<td>VCSEL [56]</td>
<td>Frequency 40GHz</td>
<td>Power 6.3mW</td>
</tr>
<tr>
<td>Photodetector [56]</td>
<td>Frequency 40GHz</td>
<td>Power 4.2mW</td>
</tr>
</tbody>
</table>

Table 3.5: Optical System Design

### 3.2.3 Free Flattened Butterfly NoC

Leveraging the FSOI based on wavelength-switching in the previous section, we design F²BFLY with realistic hardware implementation in mind and its architecture is detailed below.

#### 3.2.3.1 Topology
While an all-to-all network seems attractive to efficiently use optical interconnects, practical issues mentioned in the previous section prevent implementing FSOI links with arbitration directions. In addition, the hardware complexity per node of an all-to-all network grows in the order of $O(N)$, which will become difficult to accommodate on-chip when the network scale goes up.

Instead, we propose a topology that both exploits optical efficiency and has good scalability. First, the FSOI links in our system can have only two possible directions, chosen from the cardinal directions ($x$ and $y$ directions). This eliminates the complexities associated with building links with arbitrary directions. In addition, coupled with a dimensionally decomposed router architecture (see Section 3.2.3.2), RGL fabrication and alignment difficulties are further reduced.

Using only the two cardinal directions, a node in the network can reach any node in the same row or column with wavelength-switching as discussed in the previous section. Figure 3.18 shows an example of a 16-node network, where 3 FSOI links using 3 different wavelengths from the node at the upper-left corner to all other nodes in the first row are shown. With FSOI links connecting any pair of nodes in the same dimension, a 1-dimensional crossbar is formed by the FSOI links for each row/column. For example, Figure 3.18 shows a 1-dimensional crossbar formed in the first column. Overall, the resulted network topology is equivalent to a 2-level flattened butterfly (FBFLY) [15], where the crossbars along the rows form the first level and those along the columns form the second level. With this topology, packets sent between nodes in the same row or column can reach the destination in one hop. For a packet sent to a node in both a different row and a different column, it is first routed in the first level (row dimension) and then in the second level (column dimension) to reach the destination. Therefore, any packet can reach the destination in at most 2 hops (the network diameter is 2 hops). Compared to all-to-all optical networks F$^2$BFLY introduces only one more hop and one more pair of O/E and E/O conversions, while both the implementation
difficulty and hardware complexity per node (now $O(\sqrt{N})$) become significantly lower. In addition, it enables an efficient router architecture suitable for on-chip implementation, which will be detailed in the following subsection.

3.2.3.2 Router Architecture

Unlike 2D mesh whose routers have a constant number of output ports, F$^2$BFLY requires the output port counts of routers to increase in the order of $O(\sqrt{N})$. For example, a 16-node and a 64-node F$^2$BFLY requires routers with 7 and 15 output ports, respectively, including the ejection ports. Simply adopting the generic wormhole-switching router architecture [23] for such high-radix routers is inefficient, since the hardware complexity of generic wormhole-switching routers grows quadratically with the radix. To design an efficient, high-radix router architecture for F$^2$BFLY, we propose to use dimension-decomposition to reduce the router complexity. In addition, we use only one virtual channel per input port and source routing to further simplify the router pipeline.

- **Dimension-Decomposition**: The idea of dimension-decomposition is to divide the router into a row module and a column module [93], as shown in Figure 3.19a. As a result, the original 5×5 crossbar of the router is decomposed into 2 2×2 crossbars inside the row and column modules respectively. The incoming flits are guided to the row/column modules in order to traverse in the row/column dimen-
sion. Since the complexity of the crossbar grows quadratically with its port counts, the decomposed crossbars have lower hardware cost and power consumption than the original crossbar. Using the same approach, we can reduce the crossbar complexity of routers in \( F^2 \)-BFLY. For example, the 7×7 crossbar for a 16-node \( F^2 \)-BFLY can be decomposed into 2 4×4 crossbars whose overall complexity is lower.

There are further opportunities for optimization if we assume XY-routing is used. First, note that with XY-routing a flit can only turn from the row dimension to the column dimension but not reversely. Second, since in \( F^2 \)-BFLY a flit only takes at most 1 hop in each dimension, an incoming flit that is currently traversing in the row/column dimension will not continue to traverse in the same dimension. That is, an incoming flit using the row dimension will either turn to the column dimension or go to the ejection port; an incoming flit using the column dimension will definitely go to the ejection port (it cannot turn to the row dimension with XY-routing). Based on these observations, the complexities of switching fabrics in each dimension module can be further reduced.

The resulted overall router architecture for 16-node \( F^2 \)-BFLY is shown in Figure 3.19b. For clarity, input buffers are omitted in the figure. The router is decomposed into a row and a column module. Each module has a transceiver block (Tx-Rx Block in the figure) and switching fabrics (crossbar or multiplexers). The transceiver block consists of VCSEL and photodetector arrays, which forms the input and output FSOI links for the corresponding dimension. For a 16-node \( F^2 \)-BFLY, each transceiver block has 3 input ports and 3 output ports. The switching fabric directs an incoming flit to one of the output ports of the transceiver block or to the ejection port. The switching fabric of the column module is a 4×4 crossbar, whose inputs are the injection port and the 3 input ports from the transceiver block in the row module, and whose outputs are the ejection port and the 3 output ports to the transceiver block in the column module. The switching fabric of the row module contains only one 3-to-1 multiplexer and one 1-to-3 de-multiplexer. The inputs to the multiplexer are the input ports from the transceiver block in the column module and the output is the ejection port. The de-multiplexer, on the other hand, takes the injection port as the input and the output ports to the transceiver block in the row module as the outputs. Since a flit can only turn from the row module to the column module and
cannot continue traversing a same dimension, the optimized router architecture can maintain the connectivity of $F^2$BFLY while using low-complexity switching fabrics instead of a $7 \times 7$ crossbar.

- **Router Pipeline and Source Routing:** As network deadlock is already avoided with XY-routing, our $F^2$BFLY router uses only 1 virtual channel which removes virtual channel allocation (VA) stage from the pipeline. In addition, the fact that the network diameter is 2 hops motivates us to use source routing to eliminate the route computation (RC) stage. In source routing, the source of the packet computes the output port to use at each hop during the injection process and encodes this information in the packet header. In $F^2$BFLY, the packet header only needs to record 2 output ports along the path, which introduces very low overhead. For example, the first half of Figure 3.20 compares the routing information contained in the header for per hop routing and source routing respectively, in a 16-node $F^2$BFLY. For source routing, the first bit (the FD bit) of routing information indicates whether to traverse the row dimension or the column dimension first. The two pairs of bits of the following 4 bits (OP bits) record the output ports to use when traversing the row dimension and the column dimension respectively. If the FD bit indicates that the first dimension to traverse is column dimension, the output port information for the row dimension is ignored. As we can see, the source routing information is only 1 bit longer than the per hop routing information, adding negligible overheads to the header. Finally, the second half of Figure 3.20 shows the resulted router pipeline where only the switch arbitration (SA) and switch traversal (ST) stages are left.

- **Flow Control:** We use credit-based flow control with $F^2$BFLY to ensure no packet is dropped when being transferred in the network. Each link employs one additional VCSEL-photodetector pair to transfer credits.

### 3.2.3.3 Comparison to Prior Approaches

All-to-all topologies gain favor in prior researches of optical networks, since these approaches seems to better leverage the efficiency of optical communication. In Corona NoC [2, 52], an optical crossbar is formed using multiple optical token rings to interconnect all nodes in the network. Collisions on the rings are avoided
by using optical tokens to grant accesses to the rings. In Xue et al.’s work [56], every node in the network has FSOI links to all other nodes, and collectively these FSOI links form an optical crossbar. However, all-to-all networks scale poorly. Both Corona and Xue et al.’s work require a daunting number of optical devices. In Corona the large number of waveguides and micro-rings also introduces great static power consumed by the external laser source and ring trimming. In Xue et al.’s work, accurately controlling all FSOI links becomes an intimidating design and implementation issue. Moreover, as the bandwidth density holds constant, the large number of FSOI links also reduces the bandwidth per link and increases serialization latency.

In addition to the difficulties with the optical technology, all-to-all topologies also complicates the router design at each node. For Corona, each router has at least a wide de-multiplexer to direct flits to queues on corresponding rings, and a complex arbiter to nominate a fix number of queues for optical arbitration and transmission [52]. For Xue et al.’s work [56], collision is handled by the exponential back-off algorithm with an initial window size of 3 and a fractional base of 1.1. It is not clear from their work how such a back-off algorithm can be implemented efficiently in the hardware. Adding to the difficulties, each of the $O(N^2)$ FSOI links needs to run a back-off algorithm independently. However, the costs of implementing the back-off algorithm is not considered in their paper.
In contrast, F$^2$BFLY trades some of the optical efficiency for better scalability, lower power consumption, and easy hardware implementation. Compared with Corona [2, 52], F$^2$BFLY has at most one more hop (3 network cycles) and one more pair of O/E and E/O conversion, but much lower static power consumption. Compared with Xue et al.’s work [56], it has fewer FSOI links and higher link bandwidth, and is more feasible to implement. In either case, the router of F$^2$BFLY has lower complexity and is more suitable for on-chip implementation.

### 3.2.4 Experiment Setup

#### 3.2.4.1 Methodology

We perform a comparative evaluation of F$^2$BFLY and other state-of-the-art electrical and optical networks using a cycle-accurate NoC simulator [94]. Besides synthetic traffic patterns, we also run trace-driven simulations of parallel benchmarks by interfacing the NoC simulator with an x86 CMP simulator [40]. We use ORION 2.0 [61] to estimate the power consumption of electrical routers. In addition, we use reported data in [22, 52, 83] to estimate the power for the external laser, ring modulation, and ring trimming. Finally, the power expended by VC-SELs and photodetectors is estimated from the analysis presented in section 3.2.2. The network architectures evaluated are detailed in the following subsection.

#### 3.2.4.2 Network Configuration

Table 3.6 lists configurations of the network architectures evaluated, including three electrical networks: mesh, flattened butterfly (FBFLY) [15], and multidrop express channels (MECS) [16]; and two optical networks: Corona and F$^2$BFLY. For each architecture, two network sizes (16-node and 64-node) are used in the evaluation.

Both FBFLY and MECS belong to the express cube family, which features high-radix routers and high wiring complexity. FBFLY is designed to leverage the rich on-chip wires and have a network diameter of 2 hops. From the topology perspective, FBFLY is exactly the same as our proposed F$^2$BFLY. However, FBFLY uses electrical links and suffers from the limited bandwidth density and power efficiency of electrical interconnects [16]. In contrast, F$^2$BFLY is built with
Table 3.6: Configurations of Different Network Architectures. The “-I” versions of networks refer to the cases with internal concentration.

<table>
<thead>
<tr>
<th>Network</th>
<th>Mesh (-I)</th>
<th>FBFLY (-I)</th>
<th>MECS (-I)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of nodes</td>
<td>16</td>
<td>64</td>
<td>16</td>
</tr>
<tr>
<td>Number of PEs</td>
<td>64</td>
<td>256</td>
<td>64</td>
</tr>
<tr>
<td>Concentration degree</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>Network diameter</td>
<td>6</td>
<td>14</td>
<td>2</td>
</tr>
<tr>
<td>Link width</td>
<td>576</td>
<td>576</td>
<td>144</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Router</th>
<th>Corona</th>
<th>F^2-BFLY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input ports</td>
<td>5 (8)</td>
<td>5 (8)</td>
</tr>
<tr>
<td>Output ports</td>
<td>5 (8)</td>
<td>5 (8)</td>
</tr>
<tr>
<td>Switching fabrics</td>
<td>(8x8)</td>
<td>(8x8)</td>
</tr>
<tr>
<td>VC per port</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>VC depth</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>

FSOI links which provides much higher bandwidth with lower power consumption. In addition, in the original FBFLY work, the authors simply adapts a generic wormhole switching router as the high-radix router, which incurs high hardware and power overheads. In this work, we adopt a dimension-decomposed router architecture which together with other optimizations significantly reduces hardware complexity and power consumption (see Section 3.2.3). The MECS architecture is proposed to overcome the high interconnect intensity of FBFLY. MECS has lower router radix but also more restricted connectivity, which makes it a compromise between complexity and connectivity. Corona is a channel-guided optical network. As we have discussed in Section 3.2.3.3, Corona is based on an optical crossbar formed by multiple optical token rings and has high implementation complexities and power consumption.

- **Concentration:** According to prior studies [13, 16], concentrated on-chip network provides better power-performance efficiency. In this work, we use 4-way concentration for all networks evaluated. The second and the third rows of Table
3.6 show the network sizes in terms of number of nodes and number of PEs. In addition, there are two kinds of concentration: external concentration first merges traffics of the 4 PEs and sends using a single port of the router; internal concentration adds more inject/eject ports to the router and directly connects the PEs to the additional ports. In general, internal concentration provides better performance but has higher hardware and power overheads. In our evaluation, we exam both internal concentration and external concentration for the electrical NoCs.

- **Link Bandwidth:** To perform a fair comparison, we use the iso-bandwidth analysis as that used by Grot et al. [16]: we keep the bi-section bandwidth constant (4,608-bit and 18,432-bit for the 16-node and the 64-node networks respectively), and calculate the link bandwidth for each of the three electrical networks. As we can see from the sixth row (link width) of Table 3.6, mesh has highest link bandwidth due to its lowest wiring complexity, while FBFLY has the lowest link bandwidth due to its highest wiring complexity. The results of MECS show a compromise between complexity and link bandwidth.

  For Corona, we assume each waveguide carries 72 wavelengths and 4 waveguides forms a physical channel, leading to a 288-bit link. For F²BFLY, by conservatively assuming that VCSELs and photodetectors have a pitch of 50μm and the node area is 1.5mm×1.5mm, each FSOI link is 72-bit and 36-bit for the 16- and 64-node networks respectively. The frequencies of optical links are 10GHz and 40GHz for Corona and F²BFLY respectively, conforming to the settings in [52] and [56].

- **Virtual Channels:** Only the mesh network employs multiple virtual channels, with each channel having a depth of 8-flit. For FBFLY and MECS, the depth of the input buffer is designed to account for the worst-case round-trip credit return latency. For Corona, the depth of the ejection queue is designed to cover the round-trip latency of optical tokens. Finally, since the FSOI link has a single cycle latency, the buffer size is fixed for both network sizes, also according to the credit return latency.

  From our experiment results, power and performance of the two optical networks are both insensitive to concentration types.
Table 3.7: Configurations for the CMP

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Frequency</strong></td>
<td>5GHz</td>
</tr>
<tr>
<td><strong>Processor Core</strong></td>
<td></td>
</tr>
<tr>
<td>Issue/Commit width</td>
<td>2</td>
</tr>
<tr>
<td>ROB size</td>
<td>128</td>
</tr>
<tr>
<td><strong>L1 Cache</strong></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>32KB per core</td>
</tr>
<tr>
<td>Associativity</td>
<td>4</td>
</tr>
<tr>
<td>Block size</td>
<td>64-byte</td>
</tr>
<tr>
<td><strong>L2 Cache</strong></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>1MB bank</td>
</tr>
<tr>
<td>Associativity</td>
<td>16</td>
</tr>
<tr>
<td>Block size</td>
<td>64-byte</td>
</tr>
<tr>
<td><strong>Memory</strong></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>4GB</td>
</tr>
<tr>
<td>Latency</td>
<td>260 cycles</td>
</tr>
<tr>
<td>Controllers</td>
<td>4 MCs on the corners</td>
</tr>
</tbody>
</table>

3.2.4.3 CMP Configuration

The CMP consists of 64 (256) PEs for networks with 16 (64) nodes. Each PE consists of an x86-like core, a private L1 cache and a shared L2 cache bank, and is connected to the corresponding router through a network interface. The whole CMP system including routers runs at 5GHz. Table 3.7 lists the detailed configuration for the CMP.

3.2.5 Experiment Results

3.2.5.1 Results for Synthetic Traffics

We first study different networks with synthetic traffics. For each synthetic traffic, packets with a size of 576-bit (equivalent to a cache line transfer) are generated with the specified injection rates. The measurements are taken after the network reaches a steady state.

- **Performance**: Figure 4.21 shows the average packet latency against the injection rate for different networks. Let us first look at the results for the 16-node network. We notice that when the injection rate is low, FBFLY and MECS have comparable or slightly lower packet latencies than mesh. However, the packet latencies of both topologies rise rapidly when the inject rate increases, and the saturate throughputs are much lower than mesh. This is because, while the two high-radix networks have lower network diameter, the lower link bandwidth of them results in a longer serialization latency, which will cause the network to satu-
Figure 3.21: Performance results for synthetic traffics. The “-I” versions of networks refer to the cases with internal concentration. For the 64-node network, the results with internal concentration for the hotspot traffic are not shown, since we already show with the 64-node network it does not improve performance for the hotspot traffic.

rate early. On the other hand, both optical networks have consistently lower packet latencies than all the electrical networks, thanks to the significantly higher bandwidth of the optical links. It is worth noting that due to the higher hop counts, F²BFLY has slightly higher packet latencies than Corona. However, both networks have roughly the same saturation throughputs. Moreover, we also see that internal concentration improves the performances for the electrical networks for uniform and transpose traffics, due to the increased injection and ejection bandwidth. The performances for the hotspot traffic see almost no improvement, because in the hotspot traffic the performance is restricted by the single destination instead of the injection/ejection ports.

In the 64-node network, the packet latencies of MECS at low loads are even lower than mesh, due to the increased network scale. In contrast, the performance
Figure 3.22: Energy consumption for uniform and transpose traffics, when the injection rate is 0.05 packets/cycle/node.

of FBFLY becomes worse, exhibiting largest packet latencies among all networks for all loads. The primary cause for this phenomenon is the halved link bandwidth for FBFLY when the network size increases from 16-node to 64-node. This implies that FBFLY has inferior scalability than other networks. In the 64-node network, the two optical architectures continue to perform better than electrical networks. However, we notice that F^2BFLY now has significantly lower saturation throughput than Corona. This is because we conservatively halved the link bandwidth of F^2BFLY for the 64-node network, but optimistically keep the link bandwidth for Corona. Finally, internal concentration continue to improve the performances for electrical networks for uniform and transpose traffics. The results with internal concentration for the hotspot traffic are not shown, since internal concentration
Figure 3.23: Energy-delay products calculated from results in Figure 3.22.

does not improve the performance in this case.

**Power:** We set the injection rate to 0.05 packets/cycle/node and measure the energy consumption for the uniform and the transpose traffics. Figure 3.22 shows the results for all architectures for both 16- and 64-node networks. The black curve in each plot shows the average packet latencies. For the 16-node network, the F$^2$BFLY has the lowest energy consumption. While Corona has slightly lower packet latency than F$^2$BFLY, it has significantly higher energy consumption due to the more complex router architecture and the static power consumed by the external laser source and ring trimming. For the 64-node network, we see a rapid increase of power spent by the laser source and ring trimming: the static energy accounts for more than 60% of total energy consumption of Corona, largely offsetting the power efficiency. This result implies that when the network size goes up,
the static energy waste impedes the energy efficiency of Corona. Finally, we also notice that internal concentration results in slightly lower packet latency but also higher energy consumption.

For a more comprehensive comparison, we calculate the energy-delay products (EDP) from results in Figure 3.22 and plot them in Figure 3.23. As we can see, in the 16-node network, the two optical networks have the lowest EDP. In addition, the EDP of $F^2$BFLY is 41% lower than that of Corona. In the 64-node network, while the optical networks continue to have the lowest EDP, Corona’s EDP increases significantly due to the large static power consumption. In this case, $F^2$BFLY’s EDP is 80% lower than Corona’s EDP. In addition, we see that the EDP results with and without internal concentration are roughly the same, because internal concentration slightly decreases the packet latency, but increases the power consumption.

### 3.2.5.2 Results for Applications

We further evaluate networks by simulating a CMP running different applications. To make the simulation time tractable, we only simulate a 16-node network with 64 processing elements. Furthermore, for the electrical networks we only evaluate internal concentration since our studies above show that internal concentration re-
sults in slightly better performance and roughly the same EDP. The network and CMP configurations are detailed in Table 3.6 and 3.7. We collect traces from a diverse set of 33 benchmarks including SPEC CPU2006 benchmarks, applications from SPLASH-2 and SPEC OMP benchmark suites, and four commercial workloads (sap, sjas, sjbb, tpcw). In addition, we choose representative execution phases using [95] for all the workloads.

Figure 3.24 shows the results for network performance and power when running the applications. Due to space limit, we only show results for 10 representative applications out of the 33 benchmarks. The last set of bars in each figure show the average results for all 33 benchmarks. From the results, we first notice that the two optical networks have the lowest packet latencies and power consumptions across all benchmarks. Second, the high-radix electrical networks (FBFLY and MECS) have slightly lower latencies than mesh for some benchmarks (gcc, astart), but in general higher latencies than mesh for most benchmarks. A closer look at the benchmarks reveals that gcc and astart have quite low injection rates (less than 0.02 flits/cycle/node), however on average the injection rates of all benchmarks are high (around 0.1 flits/cycle/node) partly because of the 4-way concentration used.

Figure 3.24b shows the normalized energy-delay products. We notice that the EDPs for the high-radix electrical networks (FBFLY and MECS) are lower than mesh for some benchmarks, but higher for others. On average, the EDP for FBFLY is 10% lower and the EDP for MECS is 10% higher compared with mesh. Our proposed F$^2$BFLY is shown to have the lowest EDP, which is 15% better than Corona. While the saving in EDP is not as significant as with the synthetic traffics shown above, we believe the energy advantage of F$^2$BFLY will increase as the network size scales up as predicted by the synthetic traffic results.

### 3.2.6 Summary of F$^2$BFLY

F$^2$BFLY network-on-chip which is based on novel FSOI technologies. Compared to state-of-the-art optical NoCs, it has lower hardware complexity and power consumption, and also better scalability. We evaluate F$^2$BFLY together with recently developed electrical and optical NoCs, and our results show that F$^2$BFLY has low-
est power consumption and energy-delay products for both synthetic traffics and benchmarks. In addition, when the network scale grows up energy efficiency of F$^2$BFLY is even better than Corona, a representative channel-guided optical network. These results from comparative evaluation show that F$^2$BFLY is a potential solution for future efficient on-chip interconnection substrates.
Quality-of-Service

In this chapter, two OCIN architectures that provide quality-of-service guarantees are presented. The first architecture, named LOFT, combines flit-reservation flow-control and locally-synchronized frames to provide robust QoS guarantees while maintaining high performance [109]. The second architecture is built on top of Corona, a high-performance optical OCIN, in order to provide fairness in the token-based arbitrated optical rings [110].

4.1 Introduction and Background

Besides good overall throughput, it is becoming increasingly important to provide QoS support in many-core architectures for individual tasks, especially for real-time tasks. QoS support helps real-time tasks to maintain the expected performance and functionalities. It also enables composability and simplifies system integration during design time. Finally, QoS support enables performance isolation, which benefits the effectiveness of system-level task scheduler. Consequently, we have seen significant recent research efforts on QoS support in on-chip shared resources such as caches [96–98] and memory controllers [99–101].

However, providing QoS guarantees with NoC turns out to be difficult, because tens and hundreds of tasks compete for bandwidth while the scheduling is distributed to each router. The lack of global knowledge hinders implementing robust QoS guarantees with straightforward schemes. To address this difficulty, existing approaches to provide QoS support rely on resource reservations or relax-
ing guarantees, resulting in either low network throughput or weak guarantees.

4.1.1 Quality-of-Service Objectives

The QoS objectives at the network level can be defined for each communication flow in the network. In a multi-hop NoC connected by routers, multiple processing elements (PE) send data to each other. Then a communication flow (or flow for short) refers to the unique traffic sent from one PE (source) to another PE (destination). Flows are uni-directional. We use flow$_{ij}$ to refer to the traffic from node $i$ to node $j$ and vice versa. With this model, the desired features of QoS support in NoC is specified as follows:

(a) **Guaranteed minimum throughput**: For flow$_{ij}$ requiring quality-of-service, its data rate is at least $r_{ij}$ regardless of other flows contending for bandwidth.

(b) **Guaranteed maximum delay**: During design phase, it should be possible to calculate the maximum packet delay $b_{ij}$ according to the path taken by flow$_{ij}$. This can be used by various design-time procedures such as task binding and route computation. During run-time, the actual maximum packet delay should be always less than the previously calculated bound.

(c) **Fairness in throughput allocation**: It should allow contending flows to specify either equal or differentiated data rates (fair allocation or differentiated allocation). During run-time the actual data rates obtained are proportional to the specified data rates. As a priori, requirements (a) and (b) should be satisfied.

(d) **Low level of under-utilization**: It is not likely that a flow will be constantly using its booked throughput. Also, the full capacity of the network might not be totally booked. In either case, other flows should be able to scavenge the excess bandwidth to help performance.

(e) **Low hardware complexity**: The QoS mechanism is desirable to introduce little hardware overheads.

(f) **Good scalability**: The QoS mechanism should scale well, providing high performance and strong guarantees even when the size of NoC scales up.

The first two features above provide flows requiring guaranteed service with the illusion of a private network maintaining at least the requested performance. The
third feature ensures heterogeneous flows with differentiated throughput allocation. The rest of features shape out a cost-effective QoS mechanism.

Measuring a guaranteed service mechanism against features specified above is straightforward. Per flow accepted bandwidth and worst-case packet latency can be directly measured to check features (a), (b) and (c). Overall utilization and throughput can be measured to check feature (d). Hardware complexity and scalability can be analytically estimated.

4.1.2 Related Work in Providing QoS for OCIN

A number of previous studies have developed mechanisms to provide quality-of-service and guaranteed service in NoC. Time-division-multiplexing (TDM) circuit-switching is proposed to provide bandwidth and latency guarantees in Æthereal NoC [35]. Each guaranteed flow is mapped to a virtual circuit, by reserving a time slot for each link along the path. By time-slot reservation, Æthereal architecture can provide the exact bandwidth booked by the flow and analyzable delay bounds. However, it does not allow guaranteed flows to use excess bandwidth when the network is under-utilized. Also, how to arbitrate amongst throughput requests is not clear from the original work. Nostrum NoC [32] also exploits the idea of TDM to create virtual circuits and uses the so-called “containers” to provide bandwidth guarantee. It only works for deflective routing and shares the same problem with Æthereal. In MANGO NoC [33], a flow needs to reserve the virtual channels along its path to construct a virtual circuit before sending packets on the virtual circuit. Therefore MANGO implements a large amount of buffers and complex switch modules. In addition, this connection-based approach tends to penalize short-lived transactions and compromises the overall throughput. In SonicsMX [36], sources insert interval markers to acquire bandwidth. The benefit of this approach is the absence of circuit setup. However, it does not provide hard guarantees of either minimum bandwidth or maximum delay.

Globally-Synchronized Frames (GSF) [37,38] is a recent proposed approach to provide QoS for OCIN. GSF adopts the frame-based scheduling principles proposed in Rotate Combined Queues (RCQ) [102] and offers excellent throughput guarantees. In GSF, time is coarsely quantized into frames, and each flow can reserve a
fraction in the frame to inject data flits. Frames are prioritized according to their ages, and the data flits belonging to older frames are switched first in any router. GSF also allows bursty flows to utilize excess bandwidth by providing multiple on-the-fly frames and fast frame recycling. Without circuit setup and dedicated virtual channels, GSF has relatively low hardware complexity and good scalability. On the downside, GSF also suffers from several problems: a) To maintain a high throughput and QoS guarantees, GSF requires a large frame and source queue size, which introduces significant hardware overheads to the OCIN. b) The flexibility of GSF is constrained by the need to synchronize frame recycling within the network. c) Finally, GSF does not allow different packets to share a virtual channel, effectively reducing flow-control efficiency. We will discuss the drawbacks of GSF in more details shortly when introducing our work of LOFT to provide both high performance and robust QoS guarantees in NoC.

4.2 LOFT: A High-Performance Network-on-Chip Providing Quality-of-Service

4.2.1 Introduction

As we discussed previously, providing quality-of-service (QoS) for concurrent tasks in many-core architectures is becoming important, but even challenging for network-on-chip (NoC). This is because unlike other shared resources, NoC does not typically have central arbitration of access to the shared resource. Instead, each router shares the responsibility of resource allocation. While such distributed nature benefits the scalable performance of NoC, it also dramatically complicates the problem of providing QoS support for individual flows.

We propose a novel NoC architecture that can provide strong and flexible QoS support, and at the same time can sufficiently utilize network bandwidth to achieve high overall performance. Our proposed LOFT network-on-chip architecture achieves the above goal based on the combination of two mechanisms:

- **Locally-Synchronized Frame** (LSF) is a frame-based scheduling mechanism implemented locally in each router, in order to provide strong and flexible service guarantees to respective flows. In LSF, the distributed scheduling within
each router only requires local information exchange between adjacent routers, without the need of global knowledge. Therefore, LSF is well scalable, especially suitable for future large scale multiprocessor architectures.

- **Flit-Reservation** (FRS) is a flow-control mechanism bound to LSF and improves overall network utilization and throughput. In FRS, a look-ahead flit is sent before the data flits, to reserve bandwidth and buffers along the path to the destination. The data flits follow the look-ahead flit and reclaim resources booked by the look-ahead flit to make progress. Look-ahead flits are routed on a separate look-ahead network without interfering with data flits. Since look-ahead flits are much shorter than data flits, the look-ahead network is lightly loaded and quite fast.

The contribution of LOFT can be generalized as follows: a) LOFT can provide robust throughput guarantee to each rate-observing flow even with interference of malicious aggressors that try to exhaust network bandwidth; b) LOFT also guarantees delay bound at a much finer granularity than similar approaches [37,38]; c) LOFT can fully exploit under-utilized bandwidth, and achieve excellent overall performance; d) the extra hardware introduced by LOFT is lightweight, incurring very low overheads. Our experiment results show that LOFT delivers flexible and reliable QoS guarantees while sufficiently utilizes available network capacity to gain high overall throughput.

### 4.2.2 Principles of LSF and FRS

Our proposed LOFT architecture is based on the integration of locally-synchronized frames (LSF) and flit-reservation flow-control (FRS). We first separately present the principles for LSF and FRS, but defer the discussion of integration of LSF and FRS to the next section.

#### 4.2.2.1 Frame-Based Scheduling

Our proposed *locally-synchronized frames* (LSF) is based on the idea of grouping time slots into frames, to coarsely approximate the ideal deadline-based scheduling [102]. The term “scheduling” in the NoC sense is to determine the switching order of flits contending for an output physical link. Figure 4.1a showcases how
Figure 4.1: (a) Abstraction of network contention into a server model. The left part shows a piece of NoC; and the right part shows the converted server model. The rectangles labeled “P” represent PEs; the circles labeled “R” represent routers. In this figure three contending flows are shown. Flows are merged at each scheduling point, where a “MUX” picks flits from different queues in a certain order. For simplicity, PEs other than sources or destinations and routers without contention are omitted. (b) and (c) Frame-based arbitration: (b) shows the ideal setup while (c) shows the practical setup.

contentions in the NoC can be abstracted by a server model. As can be seen, “server” here simply refers to each physical link, and the “service” it provides is the link bandwidth. At each server, queues (buffers) are provided to account for dynamic mismatch between arrival and service rates. Scheduling is performed conceptually by a “MUX”, which at each cycle dequeue one flit from a certain queue to service. Therefore the queuing order determines the service order.

Using this model, frame-based scheduling can be explained with the help of Figure 4.1b and 4.1c. First, Figure 4.1b shows the ideal setup. At each “MUX” (the scheduling point), the buffers are divided into isolated queues, the frames. Each frame has a size of $F$ flits. Frames are associated with a frame number, and
the smaller this number is, the higher priority the corresponding frame has. The “MUX” services the flits in the frames in the decreasing order of their priorities. That is, “MUX” will first dequeue flits from frame 0, and only after all flits in frame $k$ are drained, the flits in frame $(k+1)$ can be serviced. A head frame pointer points to the frame currently being serviced, the head frame. Figure 4.1c shows a more practical setting, where there is a finite window of frames, which has $WF$ frames. The head frame pointer walks across frames in the round-robin order.

Incoming flits are queued into frames as follows. For each contending flow $ij$ at “MUX”, it is assigned a fixed number of buffer slots, denoted by its allocated reservation $R_{ij}$, from each frame. The sum of $R_{ij}$s cannot be larger than $F$. $flow_{ij}$ can only inject up to $R_{ij}$ flits of its incoming flits into a frame, starting from the highest priority frame. The flow is throttled if its injection frame hits the head frame; that is, its reservations in all the frames have been used up. A throttled flow can restart to inject flits when the head frame pointer advances, at which time the frame window is shifted by 1. Frame window shifting effectively recycles the oldest frame to a fresh one.

Allocating buffer slots inside each frame essentially divides the total bandwidth into shares. If we assume that the “MUX” can dequeue 1 flit at a cycle, and $\sum R_{ij} \leq F$, then the throughput allocated to $flow_{ij}$ is $R_{ij}/\sum R_{ij}$ of the ideal bandwidth. In addition, since at most only $F$ flits can be injected into a frame, the time to drain the head frame is upper bounded.

Globally-synchronized frames (GSF) is proposed recently to leverage frame-based scheduling in enabling QoS in OCIN. The key to GSF is to use a small window of large frames to reduce the complexity in determining the relative priorities among in-flight packets. This frame window is maintained inside each source node and router, but synchronized across the network by a barrier network. GSF have been shown to provide robust and flexible bandwidth guarantees to different flows in the network, with low hardware overheads.

However, GSF suffers from several problems. First, as suggested by Grot et al [38] and Das et al [39], to maintain a good performance GSF needs large frame sizes and source queues. For example, to achieve good throughput in a 64-node NoC, GSF needs a 2000-flit source queue [38,39] which translates to a significant buffer overhead. In addition, the use of a large frame size causes the delay bounds
to be too loose.

Figure 4.2: A pathological scenario: while the stripped is not contending with the grey nodes, its throughput is still reduced by GSF due to the global synchronization.

Second, the globally synchronized frame recycling in GSF limits its ability to utilize excess bandwidth. Figure 4.2 shows an example pathological scenario. The grey nodes on the first column of the network are sending packets to the black node (hotspot) at the center of the network. The stripped node, in contrast, is sending to its nearest neighbor. Without prior knowledge of the actual traffic pattern (which is typically the case for CMP), all the nodes may be assigned equal reservations in the frame. In the actual traffic pattern, the stripped node could exploit full link speed since it experiences no contention. However, the pressure created on the hotspot node slows down the global frame recycling, which indirectly reduces the accepted throughput of the stripped node.

Third, GSF reduces the efficiency of virtual channel flow-control. To prevent priority inversion, GSF does not allow flits belonging to different packets to reside in the same virtual channel. This requirement essentially elongates the time to return virtual credits and further compromises the network throughput.

While based on the same principle, GSF and LSF fundamentally differs in where frames are managed. GSF abstracts the whole NoC as a single “MUX”. In GSF, frame window shifting is globally synchronized by a barrier network. While this saves some complexities in the router, it also incurs serious problems as discussed above. In contrast, in LSF each output link of a router manages frame recycling of its own frames regardless of other output links or routers. The benefit
of doing so will become clear from the following sections. In addition, LSF allows for integration of flit-reservation, an efficient flow-control mechanism to improve performance.

4.2.2.2 Flit-Reservation

Flit-reservation flow-control (FRS), originally proposed by Peh et al [26], is an integral part of LOFT. It meshes with LSF well since it is also a distributed resource scheduling mechanism. As [26] presented a thorough discussion, here we only briefly reiterate the key ideas.

FRS proposes using look-ahead flits to pre-schedule bandwidth and buffers for data flits, and thus obliterates the need of arbitration and flow-control for data flits. A look-ahead network is dedicated to route look-ahead flits. The look-ahead network is simply a second physical network overlaid on the data network. Before the injection of any data flit, a leading look-ahead flit must be injected into the look-ahead network to reserve resources for the data flit.

Figure 4.3 shows the format of look-ahead flits. The look-ahead flit contains the departure times ($t_d$) and the flit numbers ($flt\_no$) for the data flits it leads. A look-ahead flit can leads only data flits belonging to a single flow. A source field uniquely identifies the source node of the flow, and a destination field identifies the destination node. Each $flt\_no$ field uniquely identifies a data flit led by the look-ahead flit, and each $t_d$ following the $flt\_no$ records the departure time of that data flit from the previous router. While originally in [26] $flt\_no$ and the source field are not needed, for integration with LSF we introduce these two fields, whose usage will be clear in Section 4.2.3.

The router architectures for both the look-ahead network and the data network
are shown in Figure 4.4. The router of the look-ahead network closely resembles the generic wormhole router with virtual channels and a sequence of routing stages: route computing (RC), virtual channel allocation (VA), switch allocation (SA), and switch traversal. In addition, each output/input port of the router contains an output/input scheduler. The output scheduler schedules the departure times of data flits to the next router. The input scheduler allocates space in the input buffer for data flits, and schedules data flits to traverse the switch to output ports in time before departure. Both schedulers program the scheduled events in the output and input reservation tables respectively.

The data network router contains an output reservation table in each output port, and an input reservation table in each input port. These tables contain the scheduling results from the schedulers to manipulate the movement of data flits. In addition, the input buffer is arranged as a central buffer, rather than virtual channels.

The formats of output reservation table and input reservation tables are shown in Figure 4.5. The output reservation table keeps track of the status of the output port within a time window, which in this figure has 8 time slots. For each time slot, it contains a busy flag to record if the output port is busy. It also contains
Figure 4.5: The formats of reservation tables.

A field to record the number of free buffer slots in the input buffer of the next router. We refer to this field “virtual credits” as it represents free buffers in future time slots rather than the current time slot. The grey arrow in the figure indicates the current time slot. The input reservation table is organized similarly, recording the information of arriving data flits (flow and flit number). In addition, it stores buffer allocation, output ports, and switching times of arriving data flits.

The procedure of scheduling the departure time of a data flit is sketched as follows:

1) The look-ahead flit arrives at a router, and writes the information of data flits it leads to the input reservation table. Buffers are also allocated for the data flits.
2) The look-ahead flit passes the router pipeline as in an ordinary router.
3) When the look-ahead flit reaches the output port, it tries to schedule the earliest departure times (when the output port is not busy and the virtual credit count is positive). Upon successful scheduling, it updates the status of the output reservation table.
Figure 4.6: Comparison of three different flow-control mechanisms. The black rectangles reflect credit turn-around time. For wormhole, the rectangles labeled “F” represent flits. For VCT and FRS, the rectangles labeled “Pkt k” represent packets. The rectangles labeled “L k” in FRS refer to look-head flits.

4) The output scheduling result is returned to the input scheduler, which updates the switching time and the output port. The input scheduler also returns virtual credits to the previous router.

5) When data flits arrive, the input scheduler lazily allocates buffer slots for them.

For example, Figure 4.5 shows the status of reservation tables after scheduling one data flit which arrives at cycle 2 and is scheduled to departure at cycle 5 from the east port.

Figure 4.6 presents a comparison of flow-control mechanisms in conventional wormhole switching, GSF, and FRS, which highlights the efficiency of FRS. This time graph shows the back-to-back transfer of 4 packets each having 4 flits between two routers when the input buffer is close to full. As can be seen, even assuming 1-cycle credit turn-around time, flow-control introduces a non-negligible throughput overhead to wormhole switching. This overhead is even worse for GSF. On the other hand, FRS can achieve zero turn-around time and has the highest efficiency. In addition, as the look-ahead flits are short and a single look-ahead flit can schedule multiple data flits, the look-ahead network is lightly loaded and fast.
4.2.3 Integration of LSF and FRS

Locally-synchronized frames and flit-reservation share fundamental similarities. Both mechanisms are forms of distributed resource scheduler that allocates buffer space and link bandwidth. In addition, the central buffer used by FRS can be readily used to emulate the frame queues used by LSF. The similarities motivate us to combine the two mechanisms for a high performance network-on-chip providing guaranteed services. We discuss the integration of LSF and FRS in this section.

4.2.3.1 Combining Local Frames and Reservation Tables

The first step we take is to divide the output reservation table, rather than input buffers, into frames (Figure 4.7). The frame size $F$ now equals to the number of time slots in each segment of the framed output table. A share of time slots, denoted as $R_{ij}$, from each frame is assigned to each flow $ij$ contending for the output. Using the same terminology as frame-based scheduling, successfully scheduling a time slot in a frame $k$ is referred to as injecting a data flit into frame $k$. As with frame-based scheduling, each flow $ij$ can only inject up to $R_{ij}$ data flits into each frame, and it is required that $\sum R_{ij} \leq F$.

Each output scheduler maintains a current time slot pointer (denoted as $CP$) indicating the current time slot, and a head frame pointer (denoted as $HF$) pointing to the frame being serviced. Each flow $ij$ contending for the output link maintains its current injection frame $IF_{ij}$ and remaining reservation $C_{ij}$. $WF$ is used
to denote the frame window size and $WT$ is used to denote the time window size, where $WT = F \times WF$.

Algorithm 1 formally shows the injection procedure for $flow_{ij}$. When the network is powered-up, $IF_{ij}$ is initialized to $HF$ and $C_{ij}$ is initialized to $R_{ij}$. Upon each scheduling request from look-ahead flits belonging to $flow_{ij}$, it is first checked if $C_{ij}$ is positive. If so, procedure $try\_schedule()$ is invoked to try to inject the data flit into frame $IF_{ij}$. Upon successful scheduling, $C_{ij}$ is decremented by one; otherwise, $IF_{ij}$ is incremented and $C_{ij}$ is updated to the minimum of $R_{ij}$ and $(C_{ij} + R_{ij})$, until the reservations in all frames have been exhausted.

Algorithm 2 defines procedure $try\_schedule()$. In the algorithm, we use $output\_table(k).busy$ and $output\_table(k).virtual\_credits$ to refer to the busy flag and the virtual credit count of entry $k$ in the output reservation table. We define a valid time slot as the slot with a false busy flag and a positive virtual credit count. Procedure $try\_schedule()$ only returns true if a valid time slot is found in the given injection frame.

Algorithm 3 shows the procedure to shift the frame window. Initially $CP$ and $HF$ are both assigned “0”. At each cycle $CP$ is updated to $(CP + 1) \text{Mod} \ WT$, while $HF$ is updated to $(HF + 1) \text{Mod} \ WF$ for every $F$ cycles.

It is worth to mention that GSF [37] does not allow flows to inject into the head frame, in order to guarantee that the head frame is drained in a finite time. In contrast, in our work injection to head frame is permitted, as it is guaranteed that the head frame can be recycled for every $F$ cycles.

4.2.3.2 Output Scheduling Anomaly

While the construction in the previous subsection seems plausible, it has an inherent severe problem, which is explained by the following example.

Consider a simplified case where two flows $flow_{ij}$ and $flow_{mn}$ contend for an output link. Suppose $F = 4$, $WF = 4$, and the size of input buffer is 4 flits. We equally distribute the link bandwidth to the two flows; that is, $R_{ij} = R_{mn} = 2$. The initial status of the output reservation table is shown in Figure 4.8a.

Now suppose two look-ahead flits from $flow_{ij}$ arrive in the first two cycles, and each of them leads two data flits. The first look-ahead flit will successfully schedule two time slots in the first frame. Then the second look-ahead flit finds its
Algorithm 1 Injection procedure for flow$_{ij}$

1: Initialize: IF$_{ij}$ ← HF  
2: Initialize: C$_{ij}$ ← R$_{ij}$  
3: UPON EVERY REQUEST FROM LOOK-AHEAD FLITS:  
4: scheduled ← FALSE  
5: while scheduled = FALSE do  
6: if C$_{ij}$ > 0 then  
7: scheduled ← try_schedule(IF$_{ij}$)  
8: end if  
9: if scheduled then  
10: C$_{ij}$ ← C$_{ij}$ - 1  
11: else  
12: if (IF$_{ij}$ + 1)Mod WF ≠ HF then  
13: C$_{ij}$ ← MIN(R$_{ij}$, C$_{ij}$ + R$_{ij}$)  
14: IF$_{ij}$ ← (IF$_{ij}$ + 1)Mod WF  
15: else  
16: break  
17: end if  
18: end if  
19: end while

Algorithm 2 try_schedule(IF$_{ij}$)

1: if IF$_{ij}$ = HF then  
2: candidate ← CP + 1  
3: else  
4: candidate ← F × IF$_{ij}$  
5: end if  
6: scheduled ← FALSE  
7: while scheduled = FALSE and candidate ≠ ((IF$_{ij}$ + 1) Mod WF) × F do  
8: if output_table(candidate).busy = FALSE and output_table(candidate).virtual_credit > 0 then  
9: schedule ← TRUE {Scheduling is successful if a valid slot is found}  
10: else  
11: candidate ← candidate + 1  
12: end if  
13: end while  
14: if scheduled then  
15: Update output and input reservation tables  
16: end if  
17: return scheduled

reservation in frame 0 is used up. Therefore it advances IF$_{ij}$ and injects two data flits in the second frame. Furthermore, the virtual credits consumed in the first frame are not returned by slot 7 (this is possible, for example, due to contention in the next hop). The output reservation table immediately after the previous scheduling events is shown in Figure 4.8b (shaded entries indicate changes). It
Algorithm 3 Manipulation of $CP$ and $HF$

1: Initialize: $CP \leftarrow 0 \quad HF \leftarrow 0$
2: AT EVERY CLOCK TICK:
3: $CP \leftarrow (CP + 1)\text{Mod} \ WT$
4: if $CP \text{Mod} \ F = 0$ then
5: for all flow$_{ij}$ contending for the output link do
6: if IF$_{ij} = HF$ then
7: IF$_{ij} \leftarrow (IF_{ij} + 1)\text{Mod} \ WF$
8: $C_{ij} \leftarrow \text{MIN}(R_{ij}, C_{ij} + R_{ij})$
9: end if
10: end for
11: $HF \leftarrow (HF + 1)\text{Mod} \ WF$
12: end if

shows that the next input buffer is full at slot 5 with zero virtual credits. A problem may occur if a look-ahead flit from flow$_{mn}$ arrives in the third cycle, leading one data flit. This look-ahead flit will inject to frame 0, since it has $C_{mn} > 0$ and finds slot 3 is valid. However, the virtual credit count at slot 5 will become negative after flow$_{mn}$ injects to slot 3 (Figure 4.8c)! The outcome is disastrous since the buffer is “silently” overbooked without any flow being aware. At cycle 5, the actual delivery of the scheduled data flit may fail due to insufficient buffer.

We call this problem output scheduling anomaly, which is a direct consequence of out-of-order scheduling as shown in the above example. An aggressive flow can schedule buffer slots in more distant future, as forced by frame-based scheduling. A more moderate flow not aware of the aggressive flow may inject to a more imminent time slot, which may cause buffer underflow in the future. There are two straightforward approaches to address the problem. The first approach prevents injection into a time slot if any future time slot has zero credits. The second approach enforces in-order scheduling by requiring new data flits to be scheduled in more distant time slots than any already scheduled data flits. While either approach eliminates the anomaly, both of them will force a moderate flow to discard unused reservations in more imminent frames. This would cause bandwidth underutilization and break the fairness of scheduling. A better solution to the anomaly problem should let aggressive flows voluntarily yield buffer space to moderate flows.

Our solution to the problem introduces an additional counter skipped$(i)$ associated with each frame $i$, where $i \in [0, WF - 1]$. Informally, skipped$(i)$ records the total yielded reservations by all flows. The manipulation of skipped$(i)$ is as
Figure 4.8: An example showing output scheduling anomaly. The entry tagged by “N” shown in (c) indicates buffer underflow.

follows:

1. Upon initialization, \( \text{skipped}(i) \leftarrow 0 \).

2. When the injection pointer \( IF_{ij} \) of a flow\(_{ij}\) advances, \( \text{skipped}(IF_{ij}) \) is incremented by \( C_{ij} \). That is, we add \( \text{skipped}(IF_{ij}) \leftarrow \text{skipped}(IF_{ij}) + C_{ij} \) after the if statement in line 12 of Algorithm 1.

The newly added counter \( \text{skipped}(i) \) is used as follows. For an arbitrary flow \( \text{flow}_{ij} \), let \( Prior \) be the index of the entry immediately prior to the injection frame \( IF_{ij} \) in the output reservation table. That is, \( Prior \) equals to \( (IF_{ij} \times F + WT - 1) \mod WT \). Then \( \text{flow}_{ij} \) can only inject flits into frame \( IF_{ij} \) if the following condition holds:

\[
F - \text{skipped}(IF_{ij}) \leq \text{out_table}(Prior).\text{virtual_credit} \quad (4.1)
\]

Condition (4.1) is appended to the conditions of the if statement at line 6 of Algorithm 1. With this change, output scheduling anomaly is eliminated when
the size of input buffer is $F$ flits, as stated by Theorem I in the Appendix. In addition, aggressive flows will voluntarily yield buffer space to moderate flows to ensure fair allocation in buffer space. *Detailed discussion of condition (4.1) can be found in the Appendix.*

Reconsider our example with the addition of condition (4.1). Now $flow_{ij}$ cannot inject into frame 1 since $F - \text{skipped}(1) = 4 > \text{out}_\text{table}(3).\text{virtual\_credit} = 2$. Therefore it will advance $IF_{ij}$ to 2, and as a result $\text{skipped}(1) = C_{ij} = 2$. Since now $F - \text{skipped}(2) = 4 \leq \text{out}_\text{table}(7).\text{virtual\_credit} = 4$, $flow_{ij}$ can inject two data flits into frame 2. After that, when $flow_{mn}$ schedules slot 3, no buffer underflow will occur.

### 4.2.3.3 Optimizations

The techniques presented so far frame output reservation tables to provide fair bandwidth allocation. However, although each output scheduler independently manages output reservation tables and frames, frame recycling is still globally synchronized across all routers using Algorithm 3. Consider the stripped node in Figure 4.2. Despite the ample bandwidth it could leverage, on average it can only inject $R_{ij}$ flits for every $F$ cycles and the bandwidth is under-utilized. In this subsection, we propose optimizations to improve utilization and overall throughput. The first optimization speculatively forwards flits to reduce latency. The second optimization breaks the global synchronization among schedulers to exploit excess bandwidth.

**Speculative Flit Switching**— In the present switching mechanism, data flits are switched at the time slot scheduled by look-ahead flits, even when no other flits are scheduled before them. This unnecessarily prolongs the delivery latency.
when the network is lightly loaded.

To address this problem, we modify the switching mechanism to forward data flits as soon as possible. A data flit can be forwarded to the next router if there is no on-going transmission on the output link and if the next buffer has free space. This requires two changes to the data network router shown in Figure 4.4. First, additional signals are added to return actual credits in the input buffer. Second, an output arbiter similar to the switch arbiter is needed to pick one flit from the ready data flits to forward. This arbiter can be a simple round-robin arbiter, since its choice will not compromise scheduling made by LSF and FRS, except for one case explained below.

The input scheduler picks a data flit that has arrived (input buffer has been allocated) with earliest scheduled departure time as the candidate for switching. In practice, it just picks the first non-empty entry in the “buffer out” row of the input reservation table. If the candidate wins the output arbitration, it will be forwarded and its entry in the input and output reservation tables will be cleared. The input scheduler will mark a candidate as emergent if it is scheduled to be forwarded in the current cycle. Emergent candidate is guaranteed to win arbitration. This causes no problem since an output port can only have at most one emergent candidate for each cycle. This is the only exception for the round-robin arbiter.

Two caveats deserves explanation. First, as now the data flits may arrive ahead of the scheduled time (but still after the look-ahead flit), the input reservation table needs to record the flow number and flit number, in order to uniquely identify arriving data flits. This information is provided by the look-ahead flit.

The second problem is more subtle but severe. With the modified switching mechanism, data flits may be serviced in a different order other than the scheduled order. This brings back the problem of output scheduling anomaly, where out-of-order forwarding may cause buffer underflow at a later time. The outcome is the potential risk that emergent flits miss their switching times due to insufficient buffer. We solve this problem by adding a “speculative buffer” to each input port, as shown in Figure 4.9. The speculative buffer is used to hold data flits forwarded out of order, and the original non-speculative buffer is used for in-order flits. The output reservation table only tracks buffer usage of the non-speculative buffer. For a winning candidate, the output scheduler checks if it is the first scheduled
flit in the output reservation table. If so, the data flit is forwarded to the non-speculative buffer; if not, it is forwarded to the speculative buffer. In either case, if the corresponding buffer is full the data flit will be denied access to the output link. At the other end of the link, the input scheduler allocates space in the speculative buffer for speculative flits and vice versa. Using a separate speculative buffer prevents out-of-order flits from blocking emergent flits.

**Local Status Reset**— Speculative forwarding only saves latency but not improves throughput. A flow cannot inject more data flits once its reservation in a frame window is used up, even if all scheduled data flits have been delivered by speculative switching. The intrinsic bottleneck to throughput is the constant frame recycling rate for both lightly loaded and heavily loaded links. Thus we propose to do local status reset when an output link is idle. During a local status reset event, 

- a) $CP, HF \leftarrow 0$
- b) $IF_{ij} \leftarrow HF$, $C_{ij} \leftarrow R_{ij}$, for all contending flow$_{ij}$, and
- c) out.table($i$).virtual.credit $\leftarrow B_N$, $\forall i \in [0, WT - 1]$, where $B_N$ is the size of the non-speculative buffer.

The reset event is triggered on an output link when the conditions below are both met to ensure the reset is safe:

- All the busy flags in the output reservation table are $FALSE$.
- The non-speculative buffer in the next input port is empty. This can be checked by the credits returned from the non-speculative buffer.

Local status reset essentially recycles all the frames in the frame window to be fresh frames. It reduces the idle time of output link and improves utilization. Note that local status reset also breaks the global synchronization of frame recycling, allowing lightly loaded links to recycle frames at faster rates.

### 4.2.4 Simulation Setup

#### 4.2.4.1 LOFT

We model LOFT adhering to the principles presented in Section 4.2.3 in a cycle-accurate NoC simulator [103]. Our current model simulates a $8 \times 8$ mesh topology. Each node is numbered by $(x + y \times 8)$ according to its coordinates $(x, y)$. The injection rates of the source nodes are variable for the purpose of performance study. In contrast, the ejection rate of any destination is constant, 1 flit/cycle.

The detailed specifications are generalized in Table 4.1. Each data packet has
Table 4.1: Simulation Setup

<table>
<thead>
<tr>
<th>Common Specification</th>
<th>LOFT</th>
<th>Data network</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size &amp; topology</td>
<td>64-node 2D mesh</td>
<td></td>
</tr>
<tr>
<td>Routing algorithm</td>
<td>Dimension-order</td>
<td></td>
</tr>
<tr>
<td>Maximum flows</td>
<td>64</td>
<td></td>
</tr>
<tr>
<td>Packet size</td>
<td>4 flits</td>
<td></td>
</tr>
<tr>
<td>Frame size</td>
<td>256</td>
<td></td>
</tr>
<tr>
<td>Frame window size</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>Time window size</td>
<td>128</td>
<td></td>
</tr>
<tr>
<td>No. of virtual channels</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>Buffer size of each channel</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>No. of router stages</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>Look-ahead flit width</td>
<td>64-bit</td>
<td>Data flit width</td>
</tr>
<tr>
<td></td>
<td></td>
<td>128-bit</td>
</tr>
<tr>
<td>Reservation table size</td>
<td>256</td>
<td></td>
</tr>
</tbody>
</table>

Look-ahead Network— The router architecture of look-ahead network adheres to that shown in Figure 4.4. We add one output scheduling stage (input scheduling is not on the critical path) to a baseline 2-stage wormhole router with look-ahead routing and speculative switch allocation [23]. Each output scheduler maintains the status (injection frame $IF_{ij}$, allocated reservation $R_{ij}$, and remaining reservation $C_{ij}$) for 64 flows as well as the head frame pointer $HF$ and current slot pointer $CP$ shared by all flows.

In the current setup, each look-ahead flit leads a single data quantum composed by 2 data flits. Therefore the flit number (departure time) simply becomes the quantum number (departure time). The 32-bit look-ahead flit contains a 6-bit destination field, a 6-bit flow number field, a 10-bit quantum number field, and a 10-bit departure time field.

Data Network— The router in the data network also has 3 stages: input
buffer allocation, output request and arbitration, and switch traversal. The depth of the central buffer (non-speculative) is set to the same as the frame size to eliminate output scheduling anomaly (see Section 4.2.3.2). The depth of the speculative buffer is varied from 0 to 16 flits, and the impact of its size is studied.

Each data flit is 128-bit wide, same as the link width. The unique flow number and flit number is stored in the first 16 bits. Note that data flits do not contain any routing and scheduling information, as routing and scheduling has been done by the leading look-ahead flits.

4.2.4.2 GSF

For comparison purpose, we also implement GSF in the simulator. The suggested parameters in [38] and [39] are used, which are shown in Table 4.1. The network size, topology, injection and ejection rates, and routing algorithm are the same as LOFT. To maintain an acceptable performance, GSF requires a source queue as large as the frame size (2000 flits) for each node.

Delay Bounds and Hardware Cost— From the specifications in Table 4.1, we can calculate delay bounds and hardware cost of both LOFT and GSF.

Delay Bounds— As explained by [37], injected packets in GSF are guaranteed to be drained in one frame window. However, a tight bound of the period of the frame window is difficult to estimate, and the original work [37] relies on simulation to obtain an empirical bound. Nevertheless, a worst-case estimation can be made by assuming a frame window full of flits are injected to node 0 and destined to node 63. Then the worst-case time to drain a frame window is over ($k \times WF \times F$). The factor $k$ is due to the flow-control overhead (Section 4.2.2.2) and equals to 2 with the router architecture we model. Therefore the worst-case latency amounts to 24000 cycles. This bound is not only too long but regardless of paths taken by data flits. In contrast, LOFT provides much tighter bounds. In the worst case, all links in the network are heavily loaded, such that speculative switching and local status reset are not effective. In this case the worst-case end-to-end latency of LOFT is the same as that of RCQ [102]:

$$F \times WF \times Num\_Hops$$

which amounts to 512 cycles per hop in our setting. Compared to GSF, this
worst-case latency is not only much tighter but related to the paths taken by the flows.

### 4.2.4.3 Hardware Cost

As shown by Table 4.2, LOFT uses 32% less storages than GSF (assuming a 12-flit speculative buffer for LOFT). In addition, for a rough estimation of area and power, we use McPAT [62] with configurations to emulate the LOFT router (e.g., a worm-hole NoC router with one virtual channel and 256-flit input buffer depth). The area and the power of a 64-node LOFT NoC are estimated to be 32mm² and 50W, which is 7% of a 64-node CMP [2] and 19% of total chip power (265W estimated by McPAT).

### 4.2.5 Experiment Results

We measure and compare the quality-of-service metrics of LOFT and GSF, using the simulation setup presented in the previous section. In the simulation we run 4 types of synthetic traffic patterns: uniform, hotspot, and two pathological cases that evaluate the quality of performance isolation. For uniform traffic, each source is treated as a separate flow, while in other cases each source-destination pair is treated as a distinct flow. For hotspot traffic, all sources send packets to node 63. The configurations of the two pathological cases will be discussed later with their results. We run each simulation until a stable network state is reached. For clarity, in this section we use flow\textsubscript{i→j} to refer to flow\textsubscript{ij}.

#### 4.2.5.1 Fairness

We first evaluate the general fairness of LOFT. Figure 4.10a, 4.10b and 4.10c show the results for equal and differentiated allocations respectively for hotspot traffic. For equal allocation, the bandwidth is distributed equally to all flows; in the two

<table>
<thead>
<tr>
<th>GSF</th>
<th>Source queue</th>
<th>Virtual channels</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>256000</td>
<td>15360</td>
<td>271379</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>LOFT</th>
<th>Input buf.</th>
<th>Reserv. tables</th>
<th>Flow stat.</th>
<th>Look-ahead network</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>139264</td>
<td>40960</td>
<td>2308</td>
<td>1536</td>
<td>184203</td>
<td></td>
</tr>
</tbody>
</table>
differentiated allocation cases, the network is divided into 4 and 2 partitions, and differentiated services are provided to different partitions. It can be seen that LOFT achieves excellent fairness in bandwidth allocation as GSF [37]. We repeat the same experiment for uniform pattern and confirm that the actual bandwidths received by different flows also comply with the allocation.

4.2.5.2 Performance

For performance evaluation, two metrics are used: a) average packet latency against offered load rate, and b) network throughput. We obtain the metrics from the simulation of uniform and hotspot traffics. For LOFT, we also vary the size of the speculative buffer to study its impact on performance. Note that setting the speculative buffer size to 0 is equivalent to turning off all optimizations proposed in Section 4.2.3.3. For comparison purpose, the performance of GSF is also measured and included in the results.

Figure 4.11a shows the result for uniform traffic pattern. The first observation from the result is that the average packet latency levels out when the offered load increases beyond a certain point for both LOFT and GSF. This is an expected result as to provide guaranteed service, both LOFT and GSF regulate the injection rates of flows and prevent unbounded packet latency. Second, we observe that increasing the speculative buffer size helps improve LOFT’s performance, and LOFT outperforms GSF when the speculative buffer size is at least 8. However, increasing the speculative buffer size beyond 12 flits sees only marginal gains.

Figure 4.11b shows the results for hotspot traffic pattern. The performance of LOFT is even better than GSF in this case, where network contention is more fierce. The performance of LOFT is better for all speculative buffer sizes. However, different from the uniform case we notice that the speculative buffer size does not significantly impact the performance. This is due to the fact that with high contention on the links speculative switching is least effective.

According to the result above, we choose a 12-flit speculative buffer size, and use this size for the rest of experiment.
4.2.5.3 Case Studies of Performance Isolation

We study two cases to exam the effectiveness in performance isolation of LOFT compared with GSF.

\textit{a) Case Study I} is adapted from the hotspot traffic to model the denial-of-service attack. In this case, only nodes 0, 48, and 56 actively send packets to the hotspot, node 63. Each flow is allocated 1/4 of the link bandwidth, that is, 0.25 flits/cycle. \textit{flow}_{0\rightarrow63} \text{is a regulated flow, injecting at a constant average rate of 0.2 flits/cycle.} \textit{flow}_{48\rightarrow63} \text{and} \textit{flow}_{56\rightarrow63} \text{are aggressors. The injection rates of them varies significantly, and may be well beyond the allocated rate. We assume that both aggressors are injecting at same rates, and measure the performance of each flow versus the injection rate of aggressors.}

The results for GSF is shown in Figure 4.12a. We notice that with the injection rate of aggressors increasing, the average packet latency of all three flows rises significantly. The latency for the regulated flow increases from 60 cycles up to 2000 cycles, which shows that the performance of the regulated flow is severely degraded by the aggressors. In addition, the aggressors also cause the accepted throughput of the regulated flow to slightly decline. Finally, we see that the aggregate throughput is always below 60% of link bandwidth, due to the inefficiency of the flow-control mechanism.

Compared to GSF, the robustness of LOFT is more satisfying as shown in Figure 4.12b. The packet latencies of all flows do not significantly increase until the aggressor injection rate reaches 0.4 flits/cycle. Beyond that point, all flows see a packet latency increase. However, the latency increase (from 42 cycles to 55 cycles) of the regulated flow is to a much less extent than the aggressors (from \(\sim 25 \) cycles to \(\geq 400 \) cycles). Therefore LOFT provides better performance isolation than GSF, and tends to penalize aggressors that try to exhaust bandwidth. In addition, the network utilization of LOFT (over 90%) is much higher than GSF.

\textit{b) Case Study II} uses the pathological example we previously show in Figure 4.2. We allocate equal reservations to all flows assuming no prior knowledge of the actual traffic pattern. Furthermore we assume all flows are injecting at same rates, and measure the accepted throughput versus the injection rate. Figure 4.13 shows the results for both (a) GSF and (b) LOFT. It can be seen that in GSF the stripped node is throttled with the grey nodes due to contention.
around the hotspot region. In contrast, in LOFT while the throughput of grey nodes saturates early, the stripped node is able to exploit the ample bandwidth exposed to it. LOFT clearly isolates the lightly loaded region from the heavily loaded region in an asymmetrically loaded network. Therefore the overall network utilization and throughput of LOFT is significantly higher than GSF.

### 4.2.6 Summary of LOFT

In this section we present LOFT, a high performance network-on-chip that provides guaranteed service and overcomes the weaknesses of previous approaches. LOFT combines *locally-synchronized frames*, where each output port independently implements frame-based scheduling, and *flit-reservation*, a pre-scheduling mechanism for efficient flow-control. We compare LOFT with *globally-synchronized frames*, another state-of-the-art QoS framework for NoC. The experiment results show that LOFT achieves equally satisfying fairness, better robustness against denial-of-service attack, and higher network utilization than GSF. We believe LOFT serves as a promising candidate for guaranteed service NoC architectures.

### 4.3 Enabling Quality-of-Service in Optical OCIN

In the previous section, we presented a work on efficient QoS support in electrical NoC. On the other hand, QoS support for optical NoC architectures is an even critical yet under-explored problem. Next, we study the problem of QoS in optical NoC and propose a quality-of-service framework for optical network-on-chip based on *frame-based arbitration*. We show that the proposed approach achieves excellent differentiated bandwidth allocation with only simple hardware additions and low performance overheads.

#### 4.3.1 Introduction

As we have discussed in Chapter 3, the emerging nanophotonic technology enables on-chip optical interconnects that are faster and less power-consuming than electrical wires [22]. Channel-guided optical NoC has been popular due to the maturity of on-chip waveguides and other passive optical components. In this Chapter, we
will look more closely at how these channel-guided optical NoCs are built, and how to provide QoS for optical NoCs based on their characteristics.

Kirman et al. [50] propose to use optical components to build on-chip buses, which however has limited scalability when the network size increases. A major branch of optical network researches are focused on direct network topologies, such as meshes and tori [51, 104, 105]. These researches migrate the topologies widely used in electrical networks to optical networks. A common feature of these networks is that the optical network is overlaid over an electrical network with the same topology. The optical network uses circuit-switching to avoid intermediate buffering. Circuit set-up is done by sending set-up packets in the packet-switching electrical network. Another major branch of researches are focused on token-ring based networks such as Corona [2] and its extension [52], Firefly [82] and MPNoC [83]. All these networks implement all-optical arbitration and flow control mechanisms to exploit the full strength of nanophotonic technology.

While direct network topologies win popularity in electrical networks, optical direct networks have several severe problems. First, the circuit set-up latency is long since set-up packets are delivered in the electrical network. To amortize this overhead and achieve good utilization, the length of data packets needs to be over 2KB [51]. However, in CMPs, the majority of traffics are coherency data which are typically 10s of bytes long but require very short delivery latency. Second, the hop-by-hop electrical network consumes significant power, which offsets the power efficiency of the optical network. Third, direct networks inevitably introduce large number of waveguide crossings which severely affects the integrity of optical signal [106]. In contrast, token-ring based optical networks do not have overheads of a second electrical network, and there are few or no waveguide crossings even for a large scale network. Therefore token-ring based optical networks are likely to outperform direct optical networks in future many-core CMPs. However, token-ring based networks suffer from severe fairness issues since aggressive sources can easily starve other sources on the same ring. According to our knowledge, there is no existing work to provide QoS support in nanophotonic NoCs.

In the rest of this section, we propose a QoS-enabled optical network-on-chip that uses frame-based arbitration to provide differentiated bandwidth allocation. Due to the simplicity of proposed architectural innovations, the QoS-enabled op-
tical NoC architecture incurs low hardware and performance overheads compared to a baseline optical NoC, while achieving excellent fairness in bandwidth allocation. Due to its low overheads, we believe the proposed QoS-enabled architecture is suitable to be implemented in future nanophotonic network-on-chips.

4.3.2 Nanophotonic Interconnect Components

Although we have discussed the basics of nanophotonic links in Chapter 3, here we briefly reiterate the key points as the fundamental knowledge for the following discussions. A nanophotonic interconnect consists of a laser source (typically located off-chip), waveguides carrying light injected by the laser source, and micro-rings to modulate and detect optical signals. A conceptual view of a nanophotonic link is shown in Figure 4.14. With dense-wavelength-division-multiplexing (DWDM), up to 128 wavelengths can be generated and carried by the waveguides [22,83], which increases the bandwidth density to over 320Gb/s/um. Micro-rings can be electrically tuned into resonance (the “on” state) and remove light from waveguides; or out of resonance (the “off” state) and let light pass by unaffected. This mechanism is leveraged to modulate light into on-off signals. Doping Ge in a micro-ring turns it into an optical detector. When the doped micro-ring is turned on, it removes light from the waveguide and converts optical signals to electrical ones. Detecting is destructive which means if a detector is turned on then downstream detectors will not be able to detect light. A splitter is used to direct a fraction of light power to another waveguide without affecting modulated signals. It is needed to implement broadcast in nanophotonic links. The pitches of nanophotonic components are small, on the order of 5–10um.

4.3.3 Optical NoC Architecture

In this subsection, we first describe the baseline optical NoC architecture without QoS support, followed by our proposed QoS enhancements to the baseline architecture.
4.3.3.1 Baseline Architecture

Our baseline architecture is derived from Corona [2, 52], which we consider to be more promising than other alternatives for CMPs, as discussed in Section 4.3.1.

**MWSR Token Rings.** The on-chip network in Corona consists of multiple token rings, each of which is a Multiple Write, Single Read (MWSR) ring. On a MWSR ring, there is a single destination, and multiple sources that send data to the destination. Light flows unidirectionally in the ring, passing each source and finally terminated by the destination. The sources and the destination modulate and senses the light with micro-rings in the same way as described in Section 4.3.2. Figure 4.15a shows a single MWSR ring with three sources (P1, P2, P3) and one destination (P0). For a connected $n$-node network, $n$ MWSR rings are needed. Figure 4.15b shows an example of a connected 4-node network. In Corona’s terminology, the destination node terminating a MWSR ring is called the *home node* of that ring. For example, P0 is the home node of the MWSR ring in Figure 4.15a.

**Arbitration and Flow Control.** Arbitration is needed to avoid data collision on the MWSR rings. A token-based arbitration mechanism called *token slot* is proposed in [52] that achieves all-optical arbitration and up to 100% bandwidth utilization. For a single MWSR ring, the home node emits a one-bit token at every clock cycle. The requesters with data to send try to seize the tokens. Capturing a token grants the requester with the right to send one phit of data (1 phit=1 flit in Corona). As long as the delay between capturing a token and sending the data is a constant, the data sent from different requesters will not collide. To do flow control, the home node can simply stop emitting tokens when there are no sufficient buffers considering the round-trip latency on the ring.

**Overall Architecture.** Corona assumes a CMP with 256 cores aggregated into 64 clusters. Each cluster contains 4 cores and one optical router. 64 MWSR rings form a wide optical waveguide bundle that visits every cluster. The approximate floorplan is shown in Figure 4.16. Corona uses DWDM with 64 wavelengths per waveguide. With 64 wavelengths, a single waveguide is used to carry arbitration tokens of all MWSR rings. The length of the rings is estimated to be 160mm [2, 52, 83], leading to an 8-cycle round-trip latency with a 5GHz clock. In addition, Corona uses virtual output queues (VOQs) [52], which means each source queue is decomposed into multiple virtual queues. Each virtual queue is dedicated
to buffering flits for a different destination. VOQs prevents flits destined for different nodes from blocking each other and improves performance. It also allows us to provide QoS support with frame-based arbitration as discussed in the next subsection.

4.3.3.2 Optical NoC with QoS Support

With token slot arbitration, Corona suffers from a severe fairness issue: since the tokens flow unidirectionally, upstream requesters have absolutely higher priority than downstream requesters in seizing tokens; in the worst case, one requester can starve all other requesters on the same MWSR ring (e.g., P1 may starve P2 and P3 in Figure 4.15a). In the original work, the authors proposed fair token slot to address this problem. While this approach tries to provide equal bandwidths to contending requesters, it does not provide bandwidth differentiation and is ignorant of weights of different requesters. Hence, fair token slot can hardly be considered as providing QoS support.

In order to develop a QoS mechanism for optical NoC, we first derive an abstract model that is focused on bandwidth allocation in token-based MWSR rings. Figure 4.17a shows this model for the single 4-node MWSR ring in Figure 4.15a, which is very similar to the model we presented in Figure 4.1. Each source is represented solely by a source queue, and the destination is represented by a destination queue. To study bandwidth allocation, we ignore all processing and propagation latencies for the moment and view the role of the shared MWSR ring as a multiplexer which picks a flit at each cycle from one source queue and pushes it to the destination queue (similar to the per-router model in Section 4.2.2.1). For example, when all sources are backlogged, with the default token slot mechanism the multiplexer always picks flits from the requester with the highest priority (P1 in this example). On the other hand, with fair token slot it picks flits from source queues in a round-robin order, achieving equal bandwidth allocation. Note that since VOQs are used, each MWSR ring works independently. This allows us to study the model of a single ring and the conclusion about bandwidth allocation can be readily applied to multiple rings. To enable QoS in optical NoC, we exploit frame-based arbitration [37, 38] to provide differentiated bandwidth allocation according to the weights of requesters. Here the definitions and properties of frames
presented in the previous section will be reused. Specifically, we also define a frame as a batch of flits that is delivered in entirety, with the number of flits in a frame denoted as $F$. From each frame, a share of flits are allocated to each source $Pi$ in the network. Here we denote the number of flits assigned to source $Pi$ as $R_{Pi}$. And for the discussion in this section, we denote the frame number as $F_N$.

Frame-based arbitration in optical token-rings works very similarly to electrical multi-hop NoCs (see Section 4.2). However, there are some difference due to the differences in underlying technology and topology. Below we discuss how frame-based arbitration is implemented in optical token-rings, which consists of two parts: injection process and frame-switching:

- **Injection Process**—When the network is initially powered up, all queues are empty and no frame contains any flits. When a source, say $Pi$, pushes its first flit into the source queue, it marks it as belonging to frame 0. Using the same terminology with [37], we call this action *injecting* a flit into frame 0. Further incoming flits of $Pi$ are also injected into frame 0 until the total number of flits injected into frame 0 reaches $R_{Pi}$. After that, $Pi$ updates its injection frame to be frame 1 and fills it with further incoming flits, until the total number of injected flits reaches $R_{Pi}$ again. This process is repeated forever when the network is operating. That is, each source node injects its share into frames with increasing frame number. Figure 4.17b shows an example for the 4-node MWSR ring, where flits filled with different patterns belong to different frames. In this example, $R_{P1}=1$, $R_{P2}=1$, $R_{P3}=2$, and $F=4$.

In practice, implementing frame-based arbitration only requires each node to track the status of one frame—the head frame. A source node needs to be throttled if its flits belonging to the head frame have all been delivered, but there are flits belonging to the head frame left in some other source queues (that is, the head frame is not drained). Only when the current head frame is drained, source nodes can generate a new head frame by admitting flits to the new head frame in compliance with $R_{Pi}s$. We call this action *frame-switching*, which is the key to implementing frame-based arbitration.

- **Frame-Switching**— We leverage optical interconnect to provide an efficient and all-optical frame-switching mechanism, which is suitable to be used with our baseline architecture. We propose to introduce two additional rings:
The **Completion Ring** is used to gather local status of each source node on a MWSR ring. On the completion ring, each source node has a micro-ring which is tuned into resonance when it still has flits of the head frame. The home node injects a continuous light into the ring that passes each source node. Therefore the completion ring essentially implements a “NOR” function. When there is at least one flit of the head frame remaining in the network, the source node owning that flit will remove light from the completion ring (Figure 4.18a). The home node has a detector at the end of the ring, and only detects light when the current head frame is drained (Figure 4.18b).

The **Frame-Switching Ring** is used to broadcast the global status to each source node on the MWSR ring. When the home node detects light on the completion ring, it will send one-bit *frame-switching* signal on the frame-switching ring, which reaches the detectors of all source nodes on this ring (Figure 4.18c). Receiving a *frame-switching* signal triggers the source nodes to perform frame-switching the operation and tune their micro-rings on the completion ring into resonance again.

Note that it is possible to use a single broadcast ring to realize the functions of both the completion ring and the frame-switching ring. However, this means this single broadcast ring needs to pass each node twice and its length is doubled. According to our analysis, to account for exponential signal attenuation, this long broadcast ring requires a laser power of 32 Watts, while the total laser power needed by both completion and frame-switching rings is only 0.85 Watts. Therefore we choose the double-ring architecture with a small hardware overhead.

**State Machine for Source/Home Node Behaviors.** The source node and the home node are both controlled by state machines, which are described by Figure 4.19 and Procedure 1 and 2. The notations used in the figure and pseudo-codes are summarized in Table 4.3. Note that the 3 loops in Procedure 1 runs independently and concurrently.

The source node has two states: *spin* and *busy* (Figure 4.19a). Initially each source node is put into the busy state, during which the micro-ring on the completion ring is turned on. Upon the generation of a new flit, it is marked as a “ready”
Table 4.3: Summary of notations

<table>
<thead>
<tr>
<th>Var. Name</th>
<th>Range</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$F$</td>
<td>$1..\infty$</td>
<td>frame size in number of flits</td>
</tr>
<tr>
<td>$R_{pi}$</td>
<td>$1..F$</td>
<td>node $pi$’s share in a frame</td>
</tr>
<tr>
<td>$C_{pi}$</td>
<td>$1..R_{pi}$</td>
<td>node $pi$’s remaining share in the current head frame</td>
</tr>
<tr>
<td>$T_{CHN}$</td>
<td>$1..\infty$</td>
<td>round-trip latency ($8$ cycles for Corona)</td>
</tr>
<tr>
<td>$T_{PROC}$</td>
<td>$1..\infty$</td>
<td>node’s processing delay of frame-switching</td>
</tr>
<tr>
<td>$W$</td>
<td>$0..(T_{CHN}+T_{PROC})$</td>
<td>wait cycle counter for the home node</td>
</tr>
<tr>
<td>$L$</td>
<td>$1..\infty$</td>
<td>idle cycle threshold to trigger early frame-switching</td>
</tr>
<tr>
<td>$Q_{pi}$</td>
<td>$0..L$</td>
<td>idle cycle counter for node $pi$</td>
</tr>
</tbody>
</table>

A flit if there are still available share in the current head frame ($C_{pi} > 0$). The source node goes to the spin state when its share is used up ($C_{pi} = 0$) and all “ready” flits have been delivered. In the spin state, the source node is throttled and turns off its micro-ring on the completion ring. The transition from the spin state to the busy state is triggered by the frame-switching signal. During this transition, the source node refreshes its share ($C_{pi} \leftarrow R_{pi}$), admits existing flits to the head frame (line 31–34 of Procedure 1), and turns on its micro-ring on the completion ring.

The home node also has two states: spin and wait. It is initialized to be in the spin state. Upon detecting light on the completion ring, it emits the frame-switching signal and goes to the wait state. In the wait state, the home node waits for $(T_{CHN}+T_{PROC})$ cycles—the time for all source nodes to receive frame-switching signal and complete frame-switching. This is necessary to prevent the home node from sending multiple frame-switching signals during the on-going frame-switching process.

**Early Frame-Switching.** While providing bandwidth differentiation, frame-based arbitration has the risk to under-utilize the network capacity. Consider the extreme case that one source node does not generate flits at all. This “silent” node blocks frame-switching and other nodes are throttled even if they have data to send. In this extreme case network utilization is reduced to 0. Similarly, network utilization is low when low-injection-rate nodes prevent high-injection-rate nodes from exploiting unused bandwidth.
Procedure 4 Behavior of source node $P_i$.

1: Initialize: $C_{P_i} \leftarrow R_{P_i}$, $Q_{P_i} \leftarrow 0$
2: Initialize: Initial state $\leftarrow$ busy
3: Initialize: Turn on the micro-ring on the completion ring
4: loop // Flit generation loop
5: AT EACH FLIT GENERATION EVENT:
6: Push the flit into the source queue
7: if $C_{P_i} > 0$ then
8: Mark the flit as “ready”
9: $C_{P_i} \leftarrow C_{P_i} - 1$
10: end if
11: end loop
12: loop // Flit delivery loop
13: if there exists a “ready” flit then
14: Request to send the “ready” flit
15: $Q_{P_i} \leftarrow 0$
16: else
17: $Q_{P_i} \leftarrow Q_{P_i} + 1$
18: end if
19: end loop
20: loop // Node state manipulation loop
21: WHILE IN THE busy STATE:
22: if ($C_{P_i} = 0$ and no “ready” flit exists) or $Q_{P_i} = L$ then
23: Go to the spin state
24: Turn off the micro-ring on the completion ring
25: end if
26: WHILE IN THE spin STATE:
27: if detect the frame-switching signal then
28: Go to the busy state
29: Turn on the micro-ring on the completion ring
30: $C_{P_i} \leftarrow R_{P_i}$
31: repeat
32: Mark the first existing “unready” flit as “ready”
33: $C_{P_i} \leftarrow C_{P_i} - 1$
34: until $C_{P_i} = 0$ or all existing flits are “ready”
35: end if
36: end loop

To address this problem we propose to do early frame-switching: frame-switching is triggered even when some nodes have not used up their shares, based on the prediction that those nodes are unlikely to generate flits for a prolonged period. In this work we use the length of current idle period ($Q_{P_i}$) as a predictor. When the current idle period is longer than some threshold ($L$), a source node also goes from the busy state to the spin state. The threshold $L$ can be determined statically or adaptively according to the network status. Currently we statically set $L = 2$, an empirical value found in experiment to provide good utilization
Procedure 5 Behavior of the home node.

1: Initialize: Initial state ← spin
2: loop  // Node state manipulation loop
3:  WHILE IN THE spin STATE:
4:    if detect light on the completion ring then
5:      Go to the wait state
6:    Emit frame-switching signal
7:    \( W \leftarrow T_{CHN} + T_{PROC} \)
8:  end if
9:  WHILE IN THE wait STATE:
10: \( W \leftarrow W - 1 \)
11: if \( W = 0 \) then
12:   Go to the spin state
13: end if
14: end loop

and bandwidth allocation. Adaptive methods to determine \( L \) is the subject of our future work. Note that with this modification, a source node may go to the spin state even if its share is not used up; therefore it may still admit and send “ready” flits even in the spin state.

**Multiple MWSR Rings.** The discussion up to this point assumes a single MWSR ring. Since with VOQs flits destined for different nodes will not affect each other, the above mechanisms can be straightforwardly extended to multiple MWSR rings. Each MWSR ring simply implements the aforementioned frame-based arbitration independently. With 64 wavelengths per waveguide, two waveguides can implement all completion and frame-switching rings in a 64-node network. A source node \( P_i \) can have different \( R_{P_i}^H \)s on different rings. If we denote node \( P_i \)'s share on ring \( H_i \) as \( R_{P_i}^{H_i} \), it is still required that \( \sum R_{P_i}^{H_i} \leq F, \forall P_i \) on ring \( H_i \).

**Bandwidth Allocation.** With frame-based arbitration, the bandwidth allocated to a source node \( P_i \) on ring \( H_i \) is \( R_{P_i}^{H_i}/F \) of the maximum bandwidth of ring \( H_i \).

### 4.3.4 Experiment

We evaluate and compare the original Corona and our enhanced QoS-enabled network using an cycle-accurate NoC simulator. Each simulation is run until results are stabilized. We model a 64-node token-ring based network as shown in Figure 4.16. The baseline configuration is exactly the same as Corona, while for the
QoS-enabled network enhancements discussed in Section 4.3.3.2 are added. The default frame size \( (F) \) is set to 128 flits. Synthetic traffic patterns are used to exercise both networks. In addition, considering the features of the baseline architecture, the synthetic traffics are divided into two classes:

- For *uniform* and *hotspot* traffics, multiple sources may send data to one destination. In the token-ring based network, these sources will contend for bandwidth of a single or multiple MWSR rings.
- The other class of traffics are those based-on permutation patterns: *transpose*, *bit-reversal*, *perfect-shuffle*, *complement*, etc. In all these traffics, data are sent only between pairs of nodes; and a given destination only receives data from one source. For the baseline architecture, this means on each MWSR ring, there is only one active source, and no bandwidth contention exists.

### 4.3.4.1 Fairness and Performance

We use the hotspot traffic to evaluate the fairnesses of the baseline and the QoS-enabled architectures. In this experiment, we pick node \((0, 0)\) as the hotspot, and each other node sends data to this node at a rate of 0.05 flits/cycle. The resulted aggregate offered load exceeds the network capacity. The result for the baseline architecture is shown in Figure 4.20a. As can be seen, in this case the upstream nodes exhaust all available bandwidth, while downstream nodes only drips traffics. Figure 4.20b shows the result for the QoS-enabled network with equal allocation, and Figure 4.20c and 4.20d show the results with differentiated allocation. In Figure 4.20c the 64-node network is divided into 4 quadrants and differentiated services are provided to different quadrants. In Figure 4.20d, the network is partitioned in to \(2 \times 2\) node groups and bandwidth is allocated in a checkerboard pattern. We see that in all cases, the accepted throughput of each source is compliant with allocation.

We next examine the performance with synthetic traffics. For uniform and hotspot, we assign \(\lfloor F/64 \rfloor\) of a frame to each source. For the second class of traffics, we allocate the whole frame to each source since there is no contention. The results are shown in Figure 4.21, where offered load and throughput are normalized to network capacity. Due to space limit, we only show results for transpose from traffics in the second class, since their performances share identical traits. As can be seen, the flit latency of QoS-enabled network is almost identical to the baseline architec-
ture. However, beyond the saturation point the flit latency of QoS-enabled network rises more rapidly, especially in hotspot and transpose. The maximum accepted throughputs of QoS-enabled network are 17% and 7% lower than the baseline for uniform and hotspot respectively. This is due to the overheads of frame-switching latency and idle cycles of source nodes. On the other hand, the throughput overhead of transpose is negligible. This is because the share of each source node is a whole frame, and those extra latencies are amortized by the large share (128 flits).

A large frame size can potentially improve throughputs by amortizing overheads of frame-switching. This is reflected by Figure 4.22a, where the throughputs of the QoS-enabled network improve with increasing frame size. The improvement saturates beyond the frame size of 512 flits. With a frame size of 512 flits, the throughput reductions of uniform and hotspot are only 10% and 2% respectively.

4.3.4.2 Energy and Hardware Overheads

Energy consumption of an optical network consists of both static and dynamic components. The static component includes external laser power and ring heating power. The dynamic power is expended by ring modulation and electrical back-end components including pre-driver, analog receiver, sampling circuits, and amplifier. We use data from [83] and [22] for 22nm node to calculate the overall energy consumption. The results for uniform traffic with different offered loads are plotted in Figure 4.22b. First, we observe that for both architectures the static energy dominates overall energy when the offered load is low. With offered load increasing, the contribution of static energy is amortized by the increased data rate. On the other hand, the dynamic energy is almost constant for the baseline architecture. Second, we observe that QoS-enabled network incurs little static energy overhead since only few additional optical components are used. On the other hand, the dynamic energy overhead is significant, due to the activities associated with frame-switching. As expected, the dynamic energy overhead is more prominent with low offered loads when source nodes become idle more frequently; and it is much lower with high offered loads. Overall, the total energy overhead ranges from 32% at 0.2 load rate to 8% at 0.8 load rate. It is possible to adaptively adjust $L$ to control the frame-switching rate, which can reduce the overhead at low loads. This is the subject of our future work.
Table 4.4: Optical Component Budget

<table>
<thead>
<tr>
<th>Photonic Subsystem</th>
<th>Waveguides</th>
<th>Micro-rings</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data MWSR rings</td>
<td>256</td>
<td>1024K</td>
</tr>
<tr>
<td>Arbitration ring</td>
<td>1</td>
<td>4K</td>
</tr>
<tr>
<td><strong>Comp. ring</strong></td>
<td>1</td>
<td>4K</td>
</tr>
<tr>
<td><strong>Frame-switch. ring</strong></td>
<td>1</td>
<td>4K</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td>259</td>
<td>1036K</td>
</tr>
<tr>
<td><strong>QoS overhead</strong></td>
<td>0.8%</td>
<td>0.8%</td>
</tr>
</tbody>
</table>

The optical component budget is shown in Table 4.4, where the overheads introduced by the QoS enhancements are in bold. Frame-based arbitration only introduces 0.8% overheads for both waveguides and micro-rings. Due to the small sizes of optical components, the resulted area overhead is also likely to be small.

4.3.5 Summary of QoS Support in Optical NoC

Emerging nanophotonic technology has the potential to boost performance and reduce power of future many-core CMPs. To address the fairness problem in existing optical NoC architectures, we propose a nanophotonic network-on-chip architecture with quality-of-service support. Our frame-based QoS enhancements achieve excellent bandwidth allocation, while only introducing simple extra hardware and small performance overheads. Based on this initial work, we are currently working on adaptively adjusting the idle cycle threshold ($L$) to further reduce energy overheads.
Figure 4.10: Fairness of throughput allocation for *hotspot* traffic pattern. The tables show the maximum, minimum, average, and standard deviation of throughputs for each group of flows.
Figure 4.11: Packet latency and network throughput for (a) uniform and (b) hotspot traffic patterns. Note that throughput results are normalized to that of GSF. For each LOFT architecture, $spec=N$ means the speculative buffer size is $N$. 
Figure 4.12: Case Study I: per flow average packet latency and accepted throughput are plotted against aggressor injection rates, when (a) GSF and (b) LOFT are used.

Figure 4.13: Case Study II: accepted throughputs of grey and stripped nodes in Figure 4.2 for (a) GSF and (b) LOFT. It is assumed that both kinds of nodes are injecting at the same rates.
Figure 4.14: Channel-guided principle: A conceptual nanophotonic link, which consists of a laser source, waveguides, and micro-rings as modulators or detectors.

Figure 4.15: (a) A single MWSR ring. Black boxes refer to O/E and E/O converters. P0-P4 are processors that send and receive signals. (b) A connected 4-node network with 4 MWSR rings. For clarity, O/E and E/O converters are omitted.

Figure 4.16: Floorplan of the 256-core CMP interconnected by optical token-rings (Corona [2]).
Figure 4.17: Bandwidth allocation models.

(a) Abstract model for a 4-node MWSR ring

(b) Frame-based arbitration for a 4-node MWSR ring

Figure 4.18: All-optical frame-switching mechanism. The inner ring is the completion ring while the outer is the frame-switching ring.
Spin Busy
Receive frame-switching signal/
Refresh \( C_P \), admit existing flits as "ready" flits, turn on the micro-ring

1. Incoming flits and \( C_P > 0 \)/
   Mark flits as "ready" flits, decrement \( C_P \)
2. Have "ready" flits/
   Request to send "ready" flits

(a) Source node behavior

Detect light on the completion ring/
Broadcast one-bit frame-switching signal

Eject flits, send tokens if buffer available

Wait for \( (T_{CHN} + T_{PROC}) \) cycles, eject flits, send tokens if buffer available

After \( (T_{CHN} + T_{PROC}) \) cycles

(b) Home node behavior

Figure 4.19: State machines for source and home nodes.
Figure 4.20: Accepted throughput for (a) the baseline architecture and for (b–d) the QoS-enabled architecture with equal and differentiated allocations. The red arrow indicates the hotspot node.
Figure 4.21: Average flit latency (left), achieved throughput (right) for (a) uniform, (b) hotspot, and (c) transpose.
Figure 4.22: (a) Maximum throughputs of the QoS-enabled network normalized to the baseline, with different frame sizes (128–640 flits). (b) Energy decomposition with different offered loads.
Conclusion and Future Work

This chapter is dedicated to distilling previous chapters and discussing future work.

**Importance of OCIN research**—The technology of many-core chip-multiprocessors is rapidly advancing and foreshadowing an upcoming variety of CMPs with tens, hundreds, and even one thousand cores [107]. The importance and complexity of OCINs grow together with the intricacies of those CMPs, in order to meet the performance requirements. OCINs differ from off-chip interconnection networks in several aspects. First, as an integrated on-chip subsystem, it takes precious silicon and power resources that could be otherwise used by computing hardwares. Therefore cost-efficiency is a *life-or-death* metric in evaluating OCINs. Second, there are even stringent requirements of latency and throughput for OCINs. Shared-memory CMPs are likely to be a dominating breed, due to their well-understood program model. However, they also require very fast OCINs to support latency-critical coherence traffics. Other genres of many-core chips, such as *many-core system-on-chip* (MPSoC) and *general-purpose graphic-processing-unit* (GPGPU), demand either predictable and robust performance or extremely high aggregate throughput. All of these often conflicting goals have to be pursued under the tight resource budget mentioned above. Finally, the silicon fabrication process places other constraints on the design of OCINs. For example, physical channels are routed using planar metal wires, making it difficult to build high-dimensional OCINs. As another example, the ratio of delay and router delay of OCINs is much smaller than off-chip interconnection networks, and therefore router delay dominates end-to-end delivery latency and becomes a bottleneck.
for latency-critical messages. Bearing various objectives, constraints, and a huge design-space, current OCIN researches are far from being converged. A plurality of topics in this field still remain to be studied.

At the beginning of this thesis, we present the fundamentals of OCIN. We have seen the basic structure and the simplistic layering model of OCIN, as well as a taxonomy system that tries to decompose the design space of OCINs into sectors. All these tools can serve as building blocks on top of which advanced architectures can be built, or they can be themselves improved to become a better framework. Chapter 2 enumerates influential OCIN researches that largely rely on the principles presented in Chapter 1. We believe the advance of OCIN architectures is an iterative process in which innovations depend on and facilitate each other.

**Towards novel OCIN architectures**—We proceed to present our contributions to this field, which comprises primarily two broad topics: 1) Emerging on-chip interconnect technologies are taken into consideration during the design process of OCINs. The key objective is to fully exploit the advantages of those technologies and meanwhile reduce their disadvantages to a minimum. 2) Architectural modifications are made to OCINs, in order to account for application features in each layer. Specifically, we present architectures that support service guarantees while maintaining high performance.

In Chapter 3, the two proposed networks are shown to be feasible and highly efficient based on emerging on-chip interconnects. They show the potential of accommodating the communication requirements of CMPs with tens of cores, with the scalability to support even larger-scale CMPs. The major contributions we make are to enrich the library of alternative OCIN libraries, and elicit a variety of novel interconnect technologies and OCIN architectures.

In Chapter 4, we present LOFT, a QoS framework for electrical OCIN. LOFT is based on the principle of frame-based scheduling, inherits simplicity, and provides high performance. It addresses the key conflicts between the distributed nature of OCINs and the requirement of service guarantees. With LOFT, designing systems involving real-time tasks becomes possible; and aggregate application throughput and fairness is improved thanks to performance isolation. In addition, it is possible to integrate LOFT with other frame-based schedulers for other shared resources, *e.g.* cache, memory controller, *etc.* Eventually a complete QoS framework can be
achieved to facilitate application consolidation in many-core CMPs. Finally, we extend the principles of LOFT and propose a framework to enable QoS support in optical OCINs.

**Future work**—The architectures and the solutions presented in this thesis is neither optimal nor complete. We expect a even rich variety of novel OCIN architectures to emerge in this exciting field. As for future work, several topics can be explored:  

a) Application-specific OCINs for emerging processor architectures, such as SoCs and GPGPUs that exhibit disparate on-chip communication traffic patterns than general purpose CMPs.  
b) Heterogeneous OCINs that can combine advantages of different approaches are likely to improve the efficiency over homogeneous OCINs. However, heterogeneous OCINs further enlarge the already huge design space of OCIN, and introduce many open questions remaining to be solved.  
c) Finally, it is interesting to explore integrating the OCIN-level QoS framework with system-level QoS framework and developing resource managers and task schedulers that are aware of interference at the OCIN-level to effectively improve system throughput.
Formal discussion of condition (4.1)

In this appendix, we first formally prove the following theorem.

**THEOREM I.** With the constraint of condition (4.1), if the input buffer size is $F$ flits, then at any moment of time and any output port, $\text{out_table}(i).\text{virtual_credit} \geq 0, \forall i \in [0, WT - 1]$.

Before proving Theorem I, we introduce some necessary mathematical tools: Preliminaries—First, for the moment we assume infinite frame window and time window. This transformation does not affect the soundness of proof, since we can view the infinite frame/time window as the concatenation of infinite fixed-size frame/time windows. It only removes the ambiguity introduced by wrap-arounds and allows us to conveniently and uniquely identify a frame/time slot. Also for convenience, we use time slot $(X,a)$ ($a \in [0, F - 1]$) to refer to the $(a+1)$th time slot in frame $X$.

Furthermore we define

$$b(X,a) = \text{number of data flits scheduled in slot } (X,a)$$

$$B(X) = \sum_{a=0}^{F-1} b(X,a)$$

Note that for all valid $X$ and $a$, $b(X,a)$ is either 0 or 1, and $B(X) \geq 0$. We further define

$$u(X,a) = \text{number of returned virtual credit at slot } (X,a)$$

$$U(X) = \sum_{a=0}^{F-1} u(X,a)$$

and for all valid $X$ and $a$, $u(X,a), U(X) \geq 0$. Given the above notations and supposing the input buffer size is $F$ flits, the virtual credit count of some time slot
$(X, a)$ can be calculated as:

$$\text{out_table}(X, a).\text{virtual_credit} = F - \sum_{i=0}^{X-1} B(i) - \sum_{j=0}^{a} b(X, j) + \sum_{i=0}^{X-1} U(i) + \sum_{j=0}^{a} u(X, j)$$

(A.1)

Using equation (A.1), condition (4.1) can be transformed into the following equivalent form

$$\text{skipped}(IF_{ij}) \geq \sum_{i=0}^{IF_{ij}-1} B(i) - \sum_{i=0}^{IF_{ij}-1} U(i)$$

(A.2)

**Frame Region**—We define frame regions as non-overlapping groups of adjacent frames. Figure A.1 shows an example of three frame regions. By our convention, a frame region has to contain at least one frame. In addition, we number the frame regions with the frame numbers of the first frames they contain. Finally, we artificially isolate frame regions, by requiring that injection frame pointers cannot advance across boundaries of frame regions. This constraint is used to decompose the proof of Theorem I into isolated subproblems (see Lemma I), and construct the proof inductively.

Furthermore, we introduce following notations:

$$FR(k) = \text{Frame Region } k \quad R^{FR}(k) = \sum_{i \in FR(k)} R_{ij}$$

$$B^{FR}(k) = \sum_{X \in FR(k)} B(X) \quad U^{FR}(k) = \sum_{X \in FR(k)} U(X)$$

**Lemma I.** With the constraint of condition (4.1), if the input buffer size is $F$ flits, then for frame region $FR(0)$ containing the first frame (frame 0), $R^{FR}(0) \geq B^{FR}(0) - U^{FR}(0)$.

The proof is constructed inductively. First we divide the frame window into regions each containing only a single frame. Then we combine regions with the constraint of condition (A.2), to form an arbitrary region containing the first frame. We use the convention to number frame regions with the frame numbers of the first frames they contain. For example, in the base case, $k = 0, 1, 2, \ldots, \infty$ for all $FR(k)$.

**a) Base Case**—In this case, each single frames forms a unique frame region. For an arbitrary frame region $FR(k)$, it is obvious that $R^{FR}(k) \geq B^{FR}(k)$ since...
each flow cannot schedule more data flits than its allocation. Therefore, \( R^{FR}(0) \geq B^{FR}(0) - U^{FR}(0) \) is trivially true.

\[ \text{b) Inductive Case} - \text{Now suppose that frame region } FR(0) \text{ has a size of } m \text{ frames } (m \geq 1), \text{ and } R^{FR}(0) \geq B^{FR}(0) - U^{FR}(0) \text{ is true. Note that } FR(0) \text{ contains frame 0 to frame } (m - 1). \text{ We grow the size of } FR(0) \text{ by one by merging it with frame region } FR(m), \text{ for which we know from the base case that } R^{FR}(m) \geq B^{FR}(m) - U^{FR}(m). \text{ Furthermore we denote the newly formed frame region as } FR'(0). \]

Merging frame region \( FR(0) \) and \( FR(m) \) will remove the frame region boundary between frame \((m - 1)\) and frame \( m \). Therefore the injection frame pointers originally in frame region \( FR(0) \) can now advance to \( FR(m) \), and schedule data flits in \( FR(m) \). We denote the increase of \( B^{FR}(m) \) caused by merging as \( \Delta B^{FR}(m) \).

With condition (A.2), we have \( \text{skipped}(m) \geq B^{FR}(0) - U^{FR}(0) \) and \( \Delta B^{FR}(m) \leq R^{FR}(0) - \text{skipped}(m) \leq R^{FR}(0) - B^{FR}(0) + U^{FR}(0) \)

Therefore \( B^{FR'}(0), U^{FR'}(0), \) and \( R^{FR'}(0) \) of the merged frame region \( FR'(0) \) can be calculated as follows:

\[
\begin{align*}
B^{FR'}(0) &= B^{FR}(0) + B^{FR}(m) + \Delta B^{FR}(m) \\
&\leq B^{FR}(m) + R^{FR}(0) + U^{FR}(0) \\
U^{FR'}(0) &= U^{FR}(0) + U^{FR}(m) \\
R^{FR'}(0) &= R^{FR}(0) + R^{FR}(m)
\end{align*}
\]

From the equations above, we have \( R^{FR'}(0) \geq B^{FR'}(0) - U^{FR'}(0) \) and the proof of Lemma I is completed.

Now we are ready to prove Theorem I with the help of Lemma I.

\[ \text{Proof of Theorem I} - \text{Take an arbitrary entry } (X,a) \text{ in the output scheduling table. We construct a frame region } FR(0) \text{ containing frame 0 to frame } (X - 1), \text{ and a frame region } FR(X) \text{ containing only frame } X. \text{ From Lemma I, we know that } R^{FR}(0) \geq B^{FR}(0) - U^{FR}(0). \text{ In addition, it is straightforward to see that } \sum_{j=0}^{a} b(X, j) \leq B(X) \leq R^{FR}(X) + R^{FR}(0) - \text{skipped}(X). \text{ Recall that with condition (A.2), we have } \text{skipped}(X) \geq B^{FR}(0) - U^{FR}(0). \text{ In summary, the virtual credit count of slot } (X,a) \text{ can be calculated as follows.}
\]

\[
\text{out\_table}(X,a).\text{virtual\_credit} = F - B^{FR}(0) + U^{FR}(0) - \sum_{j=0}^{a} b(X, j) + \sum_{j=0}^{a} u(X, j) \\
\geq F - R^{FR}(0) - R^{FR}(X)
\]

Since by construction \( R^{FR}(0) + R^{FR}(X) \leq \sum_{i,j} R_{ij} \leq F \),
out_table(X, a).virtual_credit ≥ 0 is true for any valid X and a and Theorem I is proven.

In addition, condition (A.2) implies that aggressive flows cannot inject into a frame if the virtual credits consumed in previous frames have not been returned; instead, the aggressive flow flow_{ij} will advance its injection frame pointer and thus skipped(IF_{ij}) is increased by C_{ij}. Effectively, aggressive flows voluntarily yield buffer space to moderate flows when condition (4.1)/(A.2) is not satisfied.
Bibliography

URL http://www.itrs.net/


URL http://www.tilera.com/productsprocessors.php

URL http://dx.doi.org/10.1109/MICRO.2006.19


URL http://doi.acm.org/10.1145/1183401.1183430

URL http://doi.acm.org/10.1145/1250662.1250679

URL http://dx.doi.org/10.1109/MICRO.2007.15


URL http://doi.acm.org/10.1145/1815961.1815976


URL http://doi.acm.org/10.1145/2000064.2000113


[103] “NoX NoC simulator,” . URL http://www.cse.psu.edu/~dpark/nox/


Jin Ouyang
Department of Computer Science and Engineering
Pennsylvania State University
351 IST Building, University Park, PA, 16802
(814) 3213411
jouyang@cse.psu.edu
http://www.cse.psu.edu/~jouyang/

Education
08/2007 – 05/2012
Pennsylvania State University University Park, PA
Ph.D. in Computer Science and Engineering, Advisor: Prof. Yuan Xie
Thesis: Architecting On-Chip Interconnection Networks for Future Many-Core Chip-Multiprocessors
GPA: 3.85/4.0
Certificate in SoC Design from the adjunct program of the CSE department, PSU and the Technology Collaborate

Peking University Beijing, P.R.China
B.S. in Microelectronics
GPA: 3.5/4.0

Summary and Qualification

Research Interests
- Network-on-chip (NoC), on-chip interconnects
- Quality-of-service, real-time embedded systems
- 3D-IC
- 11 technical papers published on top IEEE/ACM conferences since 2007
- Served as an external reviewer of multiple IEEE/ACM conferences/journals

Experience and skills
- 3 years of experience in FPGA and ASIC flow
- 3 years of experience in performance evaluation and modeling of processor architectures
- Experienced in SoC and embedded system design, familiar with common bus technologies (AMBA, Wishbone, CoreConnect) and processor/cache coherence architectures
- 5 years of C/C++ programming, and Linux/Unix scripting experience
- Experienced in parallel programming and parallel processing environment

Selected Publications (for the full publication list please visit my homepage)


Yaoyao Ye, Lian Duan, Jiang Xu, Jin Ouyang, Mo Kwai Hung, Yuan Xie, “3D Optical Network-on-Chip for Multiprocessor Systems-on-Chip (MPSoC)”, International 3D System Integration Conference (3DSiC ’09), Sep. 2009, San Francisco, USA

Tools and Skills
Programming
- C/C++, CUDA, OpenMP, MPI, VHDL, Verilog HDL, Matlab, SPARC Assembly, Unix/Linux scripting environment, Python/TCL.

Tools
- Design Compiler, PrimeTime, HSPICE, Virtuoso, Encounter, Calibre, Xilinx ISE and EDK, ModelSim, Simics