UC Santa Barbara
UC Santa Barbara Electronic Theses and Dissertations

Title
Interconnect Architecture Design for Emerging Integration Technologies

Permalink
https://escholarship.org/uc/item/5pw9b3wv

Author
Akgun, Itir

Publication Date
2020

Peer reviewed|Thesis/dissertation

eScholarship.org Powered by the California Digital Library
University of California

University of California
Santa Barbara

Interconnect Architecture Design for Emerging Integration Technologies

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

Doctor of Philosophy
in
Electrical and Computer Engineering

by

Itir Akgun

Committee in charge:

Professor Yuan Xie, Chair
Professor Timothy Sherwood
Professor Dmitri B. Strukov
Professor Behrooz Parhami

September 2020

The Dissertation of Itir Akgun is approved.

Professor Timothy Sherwood

Professor Dmitri B. Strukov

Professor Behrooz Parhami

Professor Yuan Xie, Committee Chair

July 2020

Interconnect Architecture Design for Emerging Integration Technologies

Copyright © 2020

by

Itir Akgun

To my parents.

Acknowledgements

First and foremost, I would like to express my gratitude for my advisor Professor Yuan Xie for giving me this opportunity and for his patience and support throughout my PhD studies. Professor Xie's guidance throughout my PhD helped me grow as a researcher.

To follow, I would like to extend my appreciation for my dissertation committee, Professor Tim Sherwood, Professor Dmitri Strukov, and Professor Behrooz Parhami, and thank them for their guidance. I feel honored to not only have them in my committee but also have them as my teachers in UCSB. I want to thank Professor Tim Sherwood for always supporting me and including me as an honorary member of ArchLab. I also want to thank Professor Dmitri Strukov for always being open and welcoming. Finally, I want to thank Professor Behrooz Parhami for taking his time to provide feedback.

None of the work in this thesis would have been possible without my collaborators. I would like to extend special thanks to my collaborators in UCSB, especially, Jia Zhan, Dylan Stow, Weitao Li, Shuangchen Li, Xing Hu, Alvin Oliver Glova, Prashansa Mukim, Yang Zhao, Peng Gu.

I am grateful for the experiences I gathered during my summer internships at AMD Research and Hewlett Packard Labs. I would like to thank my mentors and collaborators at AMD Research, Matthew Poremba and Gabriel H. Loh, and at Hewlett Packard Labs, Cong Xu and Paolo Faraboschi for everything.

I would like to thank all the past and present members of SEAL Lab family for their friendship over the years: Jia Zhan, Shuangchen Li, Ping Chi, Hsiang-Yun Cheng, Xing Hu, Weitao Li, Fengbin Tu, Xin Ma, Lei Deng, Maohua Zhu, Peng Gu, Dylan Stow, Liu Liu, Wenqin Huangfu, Abanti Basak, Xinfeng Xie, Tianqi Tang, Bangyan Wang, Gushu Li, Ling Liang, Jilan Lin, Zheng Qu, Zhaodong Chen, and all the visiting students and

scholars.

Finally, none of this would have been possible without my friends and my support group. I cannot thank them enough for believing in me and adding joy to my life during my PhD years. I am grateful for the beautiful relationships that UCSB added to my life, especially, Russell Barnes, Shabnam Larimian, Deeksha Dangwal, Abhejit Rajagopal, Camille Endacott, Sydney Ma, Sebastiano Mariani, Onur Koksaldi, Alvin Oliver Glova, and many others. I owe a huge thanks for those friends who kept their support of me from afar, yet close at heart, over many years, Mehmet Kurt, Enver Candan, Onur Gur, Thibaut Detroux, Zeynep Cinar, Meric Altintas, Tolga Zeybek, Sinem Sayar.

Most importantly, I would like to present my deepest gratitude to my parents, Olcay and Tevfik, for their never-ending love and support.

Curriculum Vitæ

Itir Akgun

Education

2020	Ph.D. in Electrical and Computer Engineering (Expected), University of California, Santa Barbara.
2015	M.S. in Electrical and Computer Engineering, University of California, Santa Barbara.
2014	B.S. in Electrical and Computer Engineering, University of Illinois at Urbana-Champaign.

Publications

[1]. **Itir Akgun**, Dylan Stow, Yuan Xie, "Network-on-Chip Design Guidelines for Monolithic 3D Integration", in *IEEE Micro*, 2019.

[2]. Mingyu Yan, Xing Hu, Shuangchen Li, Abanti Basak, Han Li, Xin Ma, **Itir Akgun**, Yujing Feng, Peng Gu, Lei Deng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, Yuan Xie, "Alleviating Irregularity in Graph Analytics Acceleration: A Hardware/Software Co-Design Approach", in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2019.

[3]. Mingyu Yan, Xing Hu, Shuangchen Li, **Itir Akgun**, Han Li, Xin Ma, Lei Deng, Xiaochun Ye, Zhimin Zhang, Dongrui Fan, Yuan Xie, "Balancing Memory Accesses for Energy-Efficient Graph Analytics Accelerators", in *ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, 2019.

[4]. Dylan Stow, **Itir Akgun**, Yuan Xie, "Investigation of Cost-Optimal Network-on-Chip for Passive and Active Interposer Systems", in *ACM/IEEE System Level Interconnect Prediction Workshop (SLIP)*, 2019.

[5]. Dylan Stow, **Itir Akgun**, Wenqin Huangfu, Yuan Xie, Xueqi Li, Gabriel H. Loh, "Efficient System Architecture in the Era of Monolithic 3D: Dynamic Inter-tier Interconnect and Processing-in-Memory", in *ACM/IEEE Design Automation Conference (DAC)*, 2019. (Invited)

[6]. Alvin Oliver Glova, **Itir Akgun**, Shuangchen Li, Xing Hu, Yuan Xie, "Near-Data Acceleration of Privacy-Preserving Biomarker Search with 3D-Stacked Memory", *Design, Automation, and Test in Europe (DATE)*, 2019.

[7]. Matthew Poremba, **Itir Akgun**, Jieming Yin, Onur Kayiran, Yuan Xie, Gabriel H. Loh, "There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes", in *ACM/IEEE International Symposium on Computer Architecture (ISCA)*, 2017.

[8]. Dylan Stow, **Itir Akgun**, Russell Barnes, Peng Gu, Yuan Xie, "Cost Analysis and Cost-Driven IP Reuse Methodology for SoC Design Based on 2.5D/3D Integration",

in *IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2016. (Invited)

[9]. Jia Zhan, **Itir Akgun**, Jishen Zhao, Al Davis, Paolo Faraboschi, Yuangang Wang, Yuan Xie, "Unified Memory Network Architecture for In-Memory Computing in Commodity Servers", in *IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2016.

[10]. **Itir Akgun**, Jia Zhan, Yuangang Wang, Yuan Xie, "Scalable Memory Fabric for Silicon Interposer-Based Multi-Core Systems", in *IEEE International Conference on Computer Design (ICCD)*, 2016.

[11]. Dylan Stow, **Itir Akgun**, Russell Barnes, Peng Gu, Yuan Xie, "Cost and Thermal Analysis of High-Performance 2.5D and 3D Integrated Circuit Design Space", in *IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2016.

Abstract

Interconnect Architecture Design for Emerging Integration Technologies

by

Itir Akgun

As Moore's Law slows down, new integration technologies emerge, such as 3D integration, silicon interposer-based 2.5D integration, and most recently, monolithic 3D integration. In addition to offering performance, bandwidth, and energy improvements due to shorter wirelength, emerging integration technologies pose new opportunities, challenges, and targets for interconnect design. Meanwhile, interconnect has become an increasingly crucial design target due to hardware, such as data-centric architectures, and software trends, with memory-bound and data-intensive applications, putting more pressure on the communication system which significantly impacts the system performance. Due to these reasons, our work focuses on designing interconnect architectures for emerging integration technologies, as interconnects and communication fabric increasingly take the center stage in architecture design in post-Moore era.

In this thesis, we introduce interconnect architecture design for various emerging integration technologies, following the trends in hardware and software domains. First, targeting emerging data-intensive workloads with high memory capacity and bandwidth requirements, we propose scalable, low latency, high bandwidth, and low energy network-on-chip architecture design for 3D-stacked memories, called memory networks, on silicon interposer. Second, we evaluate memory network architectures for high performance computing and propose techniques to further improve the memory network latency. Third, following the advances in silicon interposer-based 2.5D integration, we propose a network-on-chip chiplet for intellectual property reuse, as a communication chiplet for

future chiplet-based heterogeneous SoCs. Finally, we provide design space exploration for interconnect architectures for monolithic 3D integration, to discover trade-offs and provide guidelines for network-on-chip design under unique interconnect characteristics.

Contents

Curriculum Vitae **vii**

Abstract **ix**

1 Introduction **1**
1.1 Motivation 1
1.2 Contributions 4

2 Background on Emerging Integration Technologies **5**
2.1 3D Integration Technology 5
2.2 2.5D Integration Technology 8
2.3 Monolithic 3D (M3D) Integration Technology 10

3 Scalable Memory Fabric for Silicon Interposer-Based Multi-Core Systems **12**
3.1 Introduction 13
3.2 Background 16
3.3 Target Architecture 18
3.4 Network Design 19
3.5 Experiments 28
3.6 Conclusion 35

4 There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes **36**
4.1 Introduction 37
4.2 Background 40
4.3 Addressing MN Latency Issues 53
4.4 Evaluation 59
4.5 Analysis 67
4.6 Related Work 71
4.7 Conclusions 73

5 NoC IP: NoC Chiplet for Scalable Parallel Computing Architecture **75**
5.1 Introduction 76
5.2 Interface Standards 77
5.3 NoC IP Router Design 89
5.4 NoC IP Use Cases 115

6 Network-on-Chip Design Guidelines for Monolithic 3D Integration **118**
6.1 Introduction 119
6.2 Background 121
6.3 M3D Interconnect Characteristics 124
6.4 M3D NoC Design Guidelines 128
6.5 Conclusion 133

7 Summary of Contributions **134**

Bibliography **137**

Chapter 1

Introduction

1.1 Motivation

Conventional computing systems mainly suffer from "Memory Wall", "Bandwidth Wall", and "Power Wall" challenges. Meanwhile, interconnects have become a first order design principle in computer architecture design as data movement increasingly affects the performance and power of a computing system.

"Memory Wall" is the performance gap caused by the diverging processor and off-chip memory speeds and is one of the main challenges of contemporary computing systems [1]. Traditional Von Neumann architecture design, where the compute and storage are separate, is one of the key contributors to the memory wall problem. Therefore, the interconnection between the processor and memory is an important factor that may exacerbate or mitigate the effects of memory wall.

Similar to memory wall is the "Bandwidth Wall", which is a result of performance gap caused by off-chip memory bandwidth compared to the increase in number of cores [2]. Contributors to bandwidth wall are pin limitations, power and cost constraints of memory systems [2] exacerbated by the architectural trends such as many core systems. The

physical interconnection between memory and processors directly impact the bandwidth wall.

With the end of Dennard scaling, the power reduction benefits from transistor scaling has ended, leading to a "Power Wall". After the end of Dennard scaling, as transistors scale, power density no longer stays constant, it increases. Therefore, since the end of Dennard scaling, power has become a first-order design target. Dark silicon became a reality, where due to power limitations not all logic can be utilized simultaneously [3]. Furthermore, interconnects are a significant source of energy consumption [4, 5] since they do not scale similarly to logic. Memory access energy is a significant contributor to energy consumption, to the point where data movement to an off-chip memory consumes two orders of magnitude more energy than a floating point operation [5]. Therefore, interconnect architecture is a key design target in order to reduce the energy challenges from data movement.

Meanwhile, as Moore's Law comes to an end, the performance and cost benefits of conventional CMOS technologies are not improving at the same rate as before. To overcome the challenges of the end of Moore's Law, research community has turned its attention to a few directions, including emerging integration technologies and data-centric architectures. Consequently, communication fabric is becoming an increasingly important design point in the post-Moore's Law era computing. Moreover, as the data we produce and process increases exponentially, the communication fabric is under even more pressure.

In order to reduce interconnect delays and achieve smaller area, innovations have been made in packaging and integration technology domains beyond two-dimensional (2D) integrated circuits (ICs). Through the emergence of three-dimensional (3D), silicon interposer-based 2.5D, and monolithic 3D integration (M3D), multiple dies can be integrated closer together. Emerging integration technologies offer benefits over 2D ICs,

such as smaller footprint, lower interconnect delays, and higher bandwidth. Smaller footprint can also result in cost benefits as it improves yield. Another approach to achieve cost benefits is to utilize 3D and 2.5D integration for intellectual property (IP) reuse. Through building smaller, higher yield, more cost-efficient chiplets that perform common operations, new architectures can be realized in cost-effective manner through the plug and play style of IP reuse that allows heterogeneous integration. Connections between these chiplets, however, should be through efficient interconnects.

Traditional Von Neumann compute-centric architectures, where the compute and memory are separate, inherently require costly data movement which is exacerbated with the increasing demands of emerging data-intensive applications. Recent work has advocated for data-centric architectures, where the compute and data are brought closer together through the improvements in emerging integration technologies. Memory-centric architectures are especially useful for memory-bound and data-heavy applications. Memory-centric architectures can take on few different styles. Processing-in-memory and near-data computing paradigms include addition of lightweight compute units in or near memory. For larger scale solutions, memory-centric architectures leverage 3D integration to interconnect 3D memory modules to create a memory network [6].

To sum up, interconnect has become an increasingly crucial design target as the computing systems suffer from memory, bandwidth, and power wall. Furthermore, architecture trends in post-Moore's Law era, towards emerging integration technologies and data-centric architectures, pose new opportunities, challenges, and targets for interconnect design. New interconnection architectures are needed due to innovations from bottom up, including new integration technologies and memory devices, as well as from top down, with new data-intensive applications. The communication fabric is key for not only providing sufficient bandwidth, but also to reduce latency and energy resulting from data movement.

1.2 Contributions

In this dissertation, our goal is to design interconnect architectures in post-Moore's Law era through use of various emerging integration technologies, following the trends in hardware and software domains. Specifically, the key idea of our research is to study the trade-offs and propose scalable, low latency, high bandwidth, and low energy on-package interconnect architecture designs for emerging integration technologies and emerging data-intensive applications. In Chapter 3, we propose a network-on-package for forming a memory network on silicon interposer. In Chapter 4, we propose a network-on-package for designing a memory network for high performance computing applications. In Chapter 5 we propose a network-on-chip chiplet for IP reuse through utilizing the advances in 2.5D integration. Lastly, in Chapter 6, we propose guidelines for designing a network-on-chip for monolithic 3D integration.

Work proposed in this dissertation cover network-on-chip and network-on-package evaluations for emerging integration technologies. We study trade-offs and design interconnection architectures for emerging technologies in 3D integration, 2.5D integration, and monolithic 3D integration, while targeting emerging workloads and adoption scenarios. In this dissertation, we provide analysis and insight into designing interconnection architectures in post-Moore's Law era in order to provide scalable, low latency, high bandwidth, low energy networks and tackle the challenges of data movement.

Chapter 2

Background on Emerging Integration Technologies

In this chapter, we provide background information on emerging integration technologies, in particular, on 3D integration, interposer-based 2.5D integration, and monolithic 3D integration. We present the characteristics, advantages, and drawbacks of each integration technique. Lastly, we provide example use cases and commercial uses, if applicable, for each integration technology.

2.1 3D Integration Technology

Characteristics. Three-dimensional (3D) integration technology integrates two or more active dies by stacking vertically, as shown in Figure 2.1(a) [8, 9]. Dies can be fabricated separately, and stacked together using wafer-to-wafer, die-to-wafer, or die-to-die techniques. Most common form of vertical connection between dies is through the use of through-silicon vias (TSVs). Bonding between two layers can follow face-to-face, top metal layers facing each other and connected through micro-bumps, or face-to-back,

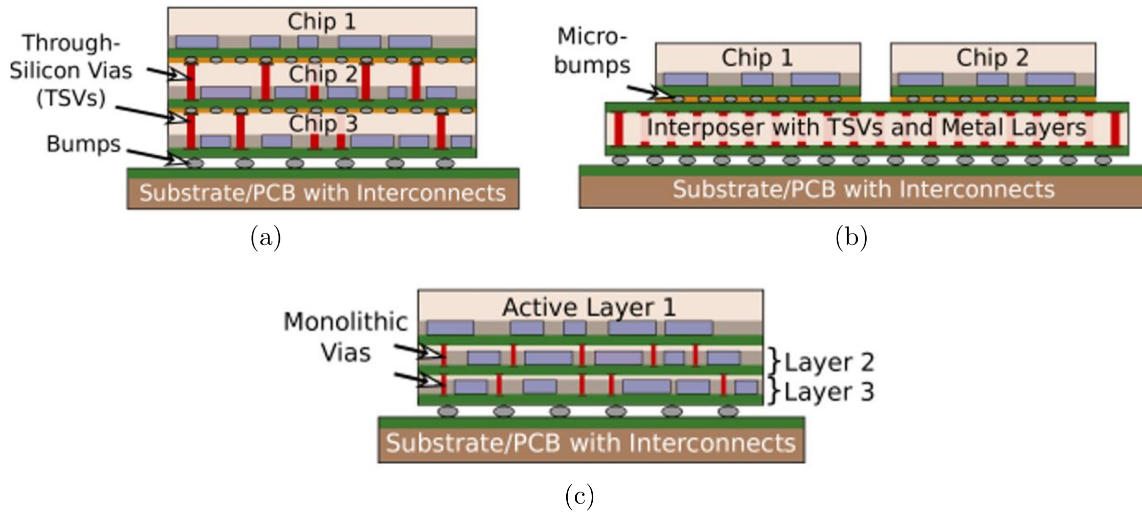

Figure 2.1: Emerging TSV-based 3D (a), silicon interposer-based 2.5D (b), and monolithic 3D (c) integration methods [7].

top metal layer of one layer facing the substrate of another and connected through TSVs.

Advantages. As the vertical distance between dies are closer than the horizontal distance, the main advantage of 3D integration is that, compared to conventional 2D integration, it reduces wire length and footprint. Reduction of the wire length also results in lower interconnect latency and lower interconnect power as the distance between dies are shorter. Higher interconnect density is achieved due to TSV integration, which does not follow the same I/O pin limitations as 2D integration. Cost benefits result from higher yield due to smaller footprint. Furthermore, 3D integration allows for heterogeneous integration, as the dies can be manufactured separately and then connected through TSVs.

Drawbacks. 3D integration also has some drawbacks over 2D integration. Main challenge of 3D integration is the thermal challenge. Due to stacking, the increased power density results in increased thermals and is exacerbated with more layers. Second challenge is the area needed for TSVs and their dedicated keep-out zones. Lastly, 3D integration and bonding comes with additional fabrication cost. Although smaller area

improves the yield, 3D integration introduces additional manufacturing costs for TSVs, bonding, and wafer thinning [10].

2.1.1 3D Memory Technologies

Although 3D integration of logic dies is not yet feasible due to thermal challenges, it is feasible to stack DRAM dies as their thermal dissipation is lower than logic dies. Therefore, 3D integration has reached commercial availability in terms of 3D memory technologies first. Next, we introduce two forms of 3D memory technologies that are developed by industry.

High Bandwidth Memory (HBM)

High Bandwidth Memory (HBM) is a 3D-stacked DRAM die JEDEC standard [11], started development in 2013. JEDEC standard for HBM defines the HBM DRAM operation details and DRAM DDR timings. HBM consists of a stack of multiple DRAM devices and independent memory channel interfaces. A base logic die is presented as vendor-optional and not specified in the standard. Early HBM devices demonstrated 256 GB/s bandwidth and up to 8 DRAM dies for a total of 8 GB memory capacity [12]. These numbers have since reached up to 640 GB/s bandwidth and 16 GB memory capacity under the latest HBM2E specification [13].

Commercial HBM devices are demonstrated by SK Hynix [12, 14, 15, 16] and Samsung [17, 13]. HBM technology is used in AMD's Fury [18, 19, 20], NVIDIA's Pascal [21], Volta [22], and Ampere [23] GPU architectures and Intel's Stratix 10 [24] and Xilinx's Virtex UltraScale+ FPGA [25] architectures.

HMC Config.	Max. Aggregate Link Bandwidth	Memory Capacity	# Vaults
HMC 1.0 (4-link) [27]	240 GB/s	4 GB	16
HMC 1.0 (8-link) [27]	320 GB/s	8 GB	32
HMC 2.1 (2-link) [28]	480 GB/s	4 GB	32
HMC 2.1 (4-link) [28]	480 GB/s	8 GB	32

Table 2.1: HMC Device Specifications

Hybrid Memory Cube (HMC)

Hybrid Memory Cube (HMC) is introduced by Micron back in 2011 and later developed by Hybrid Memory Cube Consortium [26, 27, 28]. HMC is a 3D-stacked DRAM package with multiple (eg. 4 or 8) DRAM dies stacked on top of a logic die. HMC device specifications are listed in Table 2.1. Memory is divided into partitions made up of multiple banks, where each vertical slice is called a vault. Each vault has a memory controller in the logic die, called vault controller, which handles memory accesses to that vault. Logic die consists of vault controllers, a crossbar switch and serial I/O links. Additionally, HMC proposes error detection, power management, built-in self test, and atomic operations. The packetized standard of HMC allows for memory accesses through read/write packets (and atomic operations in HMC 2.1 [28]). Therefore, multiple HMC devices can be chained together to form a memory network and communicate via a packetized interface over serial I/O links. Micron has led HMC production efforts, however HMC has since been discontinued due to low market adoption.

2.2 2.5D Integration Technology

Characteristics. A silicon interposer is a silicon die with metal layers as interconnects which allows for face-down integration of chips, as shown in Figure 2.1(b) [29]. Connection between the silicon interposer and the chip is through micro-bumps that

connect the metal layers. The name for 2.5D integration comes from using a silicon interposer to be able to integrate both 2D and 3D dies on top. Wire characteristics of interposer metal layers are similar to that of on-chip interconnects, with negligible impedance from micro-bumps [29].

Silicon interposers can have passive and active implementation. Passive interposer consists of only metal layers, providing interconnection through the interposer. Commercial products with passive interposers are currently available as GPU and FPGA systems that integrate HBM memory devices on package, as listed in Section 2.1.1. Active interposers can implement active devices in the interposer along with the metal layers at additional cost. However, to keep the interposer yield high and critical area low, "minimally active interposers" have been proposed [29]. Another way to achieve cost benefits is through using an older technology node for the interposer.

Advantages. 2.5D integration technology can provide higher bandwidth than 2D integration, as the interconnection density depends on the micro-bump pitch (e.g. 20 μm [30]) and perimeter of the stacked die [31]. Compared to 3D-stacking, 2.5D integration does not suffer from thermals as it is implemented as a passive or a minimally active interposer. Moreover, micro-bumps on the silicon interposer do not have the keep out zones as TSVs have. Through using 2.5D integration, per-die cost can be reduced when a large monolithic chip is divided into smaller chiplets integrated on an interposer.

Drawbacks. Compared to 3D-stacking, 2.5D integration provides less bandwidth as the 3D interconnect density depends on the chip area and not the perimeter as 2.5D integration does. Interconnect delay is also larger compared to 3D stacking as interconnects in silicon interposer travel horizontally. Silicon interposer introduces additional cost in terms of the interposer itself and the bonding [10].

2.3 Monolithic 3D (M3D) Integration Technology

Characteristics. Monolithic 3D integration technology (Figure 2.1(c)) uses sequential manufacturing techniques to integrate dies vertically using fine-pitched monolithic inter-tier vias (MIVs) to provide connection between tiers [32]. MIVs have 100× smaller pitch compared to TSVs, accounting for 10,000× vertical connection density [33]. Monolithic 3D enables inter-tier partitioning schemes at the granularity of transistors, gates, or blocks, as explained below.

Advantages. Compared to TSV-based 3D die stacking (see Sec. 2.1), monolithic 3D integration provides much higher interconnect density due to MIVs. MIVs do not have the same scalability challenges of TSVs of high area overhead and high parasitic capacitance [32], and therefore MIVs have smaller delay, power, and area overhead compared to TSVs.

Drawbacks. Drawbacks of monolithic 3D integration is that it is still in research phase with no commercial output so far. The sequential manufacturing technique of monolithic 3D results in higher wafer process costs [34]. EDA tools are still lacking for certain partitioning schemes of monolithic 3D integration. Although thermals are still an issue with monolithic 3D as it is with TSV-based 3D stacking, vertical heat transfer is better in monolithic 3D due to thinner inter-layer dielectric.

2.3.1 Transistor-Level M3D Partition Scheme

Transistor-level partitioning scheme partitions NMOS and PMOS transistors across two separate tiers. Transistor-level partitioning scheme is the finest granularity partitioning that is possible. Footprint reduction is limited as this partitioning scheme cannot allow partitioning over two tiers.

2.3.2 Gate-Level M3D Partition Scheme

Gate-level partitioning scheme partitions gates across two or more separate tiers. New EDA tools are required to allow gate-level partitioning across multiple layers. Footprint reduction is higher than transistor-level as the design can be partitioned into multiple tiers.

2.3.3 Block-Level M3D Partition Scheme

Block-level partitioning scheme partitions functional blocks across two or more separate tiers. Block-level partitioning scheme is the coarsest granularity partitioning. Footprint reduction depends on the block partitioning selection, which may result in block size imbalance.

Chapter 3

Scalable Memory Fabric for Silicon Interposer-Based Multi-Core Systems

Three-dimensional (3D) integration is considered as a solution to overcome capacity, bandwidth, and performance limitations of memories. However, due to thermal challenges and cost issues, industry embraced 2.5D implementation for integrating die-stacked memories with large-scale designs, which is enabled by silicon interposer technology that integrates processors and multiple modules of 3D-stacked memories in the same package. Previous work has adopted Network-on-Chip (NoC) concepts for the communication fabric of 3D designs, but the design of a scalable processor-memory interconnect for 2.5D integration remains elusive. Therefore, in this work, we first explore different network topologies for integrating CPUs and memories in a silicon interposer-based multi-core system and reveal that simple point-to-point connections cannot reach the full potential of the memory performance due to bandwidth limitations, especially as more and more memory modules are needed to enable emerging applications with high memory capacity

and bandwidth demand, such as in-memory computing. To overcome this scaling problem, we propose a memory network design to directly connect all the memory modules, utilizing the existing routing resource of silicon interposers in 2.5D designs. Observing the unique network traffic in our design, we present a design space exploration that evaluates network topologies and routing algorithms, taking process node and interposer technology design decisions into account. We implement an event-driven simulator to evaluate our proposed memory network in silicon interposer (MemNiSI) design with synthetic traffic as well as real in-memory computing workloads. Our experimental results show that compared to baseline designs, MemNiSI topology reduces the average packet latency by up to 15.3% and Choose Fastest Path (CFP) algorithm further reduces by up to 8.0%. Our scheme can utilize the potential of integrated stacked memory effectively while providing better scalability and infrastructure for large-scale silicon interposer-based 2.5D designs.

3.1 Introduction

Three-dimensional (3D) integration is a promising solution to overcome the memory wall problem industry faces nowadays. 3D-stacked memory such as High Bandwidth Memory (HBM) [12] provides larger memory capacity due to increased density, higher bandwidth due to Thermal Silicon Via (TSV) 3D integration technology, lower power consumption due to reduced interconnect, and better performance due to the integration of memory closer to the processor. Therefore, die-stacked memory is considered as a promising solution to enable applications, such as in-memory computing and High Performance Computing (HPC), that require high memory density, bandwidth and capacity.

Although memory technologies have matured enough to exploit 3D integration, stacking them on top of the processor chip exacerbates the thermal problems due to increased

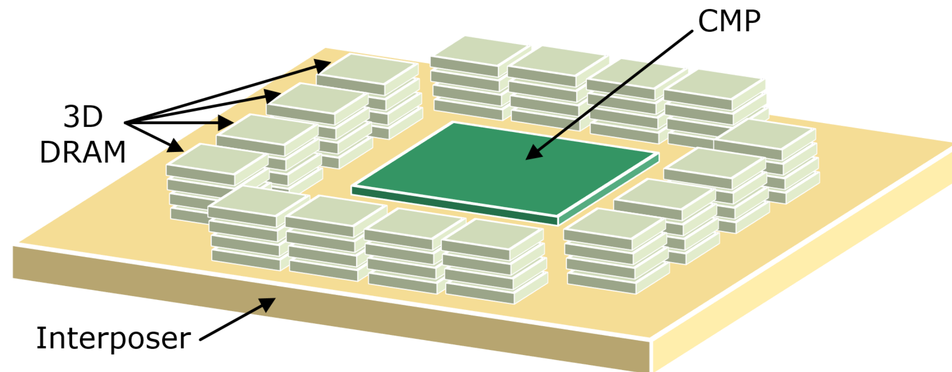

Figure 3.1: Silicon interposer-based 2.5D design

power density and therefore requires costly thermal solutions [35]. On the other hand, 2.5D implementation of integrating processors and 3D-stacked memory on silicon interposer, as shown in Figure 3.1, results in a lower power density and is a more cost-efficient option to enable such large-scale designs [10]. Industry confirms this trend with the emergence of 2.5D products such as AMD's Radeon 9 Series Fury X [36, 37] and NVIDIA's Pascal [38]. Furthermore, unlike 3D integration where the area of the processor is a limiting factor for the stacked memory capacity, the disaggragate nature of 2.5D integration enables integration of more on-package memory and better scalability.

Previous work [39] has adopted Network-on-Chip (NoC) concepts as the communication fabric for 3D integration of CMP systems with stacked and distributed L2 caches, which improves the performance by reducing the access latency by moving from 2D to 3D design and replacing long interconnects with vertical interconnects. With the advent of 2.5D designs, the notion of a hybrid network has emerged, which exploits the already-available resources of silicon interposer [40, 29, 41] to implement interconnect in silicon interposer, along with the NoC in the CPU die, so that CPU-to-CPU traffic and CPU-to-memory traffic can be directed into two separate networks. However, designing a hybrid network for a 2.5D design has its unique challenges compared to 3D. First of all, 2.5D is more bandwidth limited as the microbumps which provide connectivity between the

processor chip and the silicon interposer are limited by the perimeter of the chip [41]. As more memory modules are integrated, the per-memory stack bandwidth decreases even further since more modules would share the bandwidth as the total number of microbump connections of the CPU chip is fixed. Second, the layout of the memory modules on a silicon interposer is constrained by the CPU chip. The way memory and CPU modules are connected can therefore result in long wires. Lastly, silicon interposer can be manufactured in a different process technology than the CPU chip for cost benefits [41], which in turn would create a difference in maximum attainable frequency. Therefore, all of these constraints should be taken into account when designing an interconnection network for 2.5D designs.

Scalability is one major concern in large-scale system designs. For DIMM-based memory systems, as the data rate of DRAM increases, the number of DIMMs that can be supported on a conventional multi-drop bus decreases. Alternatively, multiple DIMMs can be connecting via point-to-point links [42], rings [43], or trees [44, 45], but all suffer from latency or power overhead, or are limited to specific RF interconnect or silicon photonics technologies. For 2.5D systems, Jerger et al. [40] propose using the already-available resource of the silicon interposers to implement a "duplicated" network for supplementing NoC bandwidth, but still deploys point-to-point connections for CPU-to-memory communication. However, as the need for integration of more and more memory modules increases due to workload demand of such applications as in-memory computing [46, 47, 48], such a design fails to support efficient integration of more modules while keeping the performance up to par. Recently, Kim et al. [6, 49] has proposed a memory-centric architecture for connecting multiple hybrid memory modules (HMCs) in a multi-socket server system, based on 3D stacking technologies. Zhan et al. [50] demonstrated how an inter-memory network helps provide scalability and proposed co-optimization of inter- and intra-memory network for in-memory computing. However,

there is still a lack of study on integrating multiple memory modules based on silicon interposer technologies.

In this work, we propose a memory-centric architecture based on silicon interposer to achieve high bandwidth and low latency in 2.5D integrations, taking advantage of the existing routing resources in order to better utilize the available memory performance. Observing the unique network traffic in 2.5D designs, we present a design space exploration that evaluates network design, topology, and routing algorithms, while taking process node and interposer technology design decisions into account. We implement an event-driven simulator and evaluate our processor-memory network designs with both synthetic traffic and real in-memory computing workloads. Experimental results presented in Section 3.5.3 show that proposed Memory Network in Silicon Interposer (MemNiSI) topology reduces the average packet latency by up to 15.3% and Choose Fastest Path (CFP) algorithm further reduces by up to 8.0% compared to baseline designs. Our scheme better utilizes the potential of integrated stacked memory to provide improved scalability and infrastructure for large-scale silicon interposer-based 2.5D designs.

3.2 Background

In this section, we introduce the silicon interposer technology which enables 2.5D integration. Then, we describe our target applications: in-memory computing, which drives the design of scalable memory systems.

3.2.1 Silicon Interposer

An interposer is a silicon die with metal layers on top that allows for face-down integration of chips as shown in Figure 3.1. Micro-bumps sit between the metal layers of the silicon interposer and the chip, providing electrical connection. Micro-bumps can be

manufactured with a pitch as small as 20 μm [30]. The bandwidth of the chip on silicon interposer therefore can be calculated given the micro-bump pitch and the perimeter of the stacked chip for interposer-based designs [41]. Furthermore, the wire characteristics of interposer metal layers are similar to on-chip interconnects with negligible impedance from micro-bumps [29].

Unlike 3D-stacked memory whose capacity is limited by the size of the processor chip, more memory nodes can be integrated in 2.5D systems as long as there is sufficient space on the silicon interposer. However, 3D stacking potentially provides more memory bandwidth because the number of TSV connections is basically proportional to the surface area of the processor chip, while the bandwidth between 2.5D-stacked chips is bounded by their perimeters. However, considering the additional area of TSVs, the bandwidth gap between 3D and 2.5D implementation is significantly reduced, and both technologies will provide substantial bandwidth improvement over traditional DIMM-based DRAM systems. Consider a large-scale interposer-based system similar to NVIDIA's Pascal or AMD's Fury X, with a processor die of size 600 mm^2. Assuming a square die and a microbump pitch of 20 μm, the die can have 4,900 microbumps in 2.5D implementation. Letting each signal operate at 1 Gbps, we get a total bandwidth of 612 GB/s. On the other hand, a 3D integration of the same die would result in 30,747 connections using 50 μm TSV pitch, reaching 3.8 TB/s of total bandwidth.

There are two implementation types of silicon interposers: passive and active. Passive interposer consists only of the metal layers and is the one that is manufacturable in the near future. Passive interposer integration can provide interconnection in the silicon layer. One downside of using passive interposer is the inability to implement repeaters on long wires to increase the interconnect speed. On the other hand, active interposer implements active devices in the interposer along with routing capability. Active interposers pose an extra cost challenge over passive interposers due to active devices increasing the critical

area and therefore leading to lower interposer yields. A way to achieve cost benefits is to implement the interposer in an older technology to achieve high yield.

3.2.2 In-Memory Computing

In-memory computing has become a widely adopted technology for real-time analytics applications in the industry. One important application is in-memory databases such as SAP HANA [46] and IBM DB2 BLU [51], which strive to *fit an entire database in memory*, potentially obviating the need of paging data to/from mechanical disks. The open-source community is also actively advancing big data processing frameworks that leverage in-memory computing. For example, Spark [47] is a cluster computing framework optimized for data reuse in iterative machine learning algorithms and interactive data analysis tools. Unlike MapReduce [52] which writes data into external storage system for data reuse between computations, Spark creates a new distributed memory abstraction and lets users explicitly *persist intermediate datasets in memory*, which can achieve up to 100× performance speedup [47].

As we can see, in-memory computing relies heavily on the memory capacity to keep the application's entire working dataset in memory for faster access. Therefore, we claim in-memory computing as one of the promising applications which can leverage a large-scale interposer-based system with significant in-package memory capacity and scalability.

3.3 Target Architecture

The main goal of this work is to build a scalable interposer-based network architecture for integrating a large number of memory modules in a multi-core system, in order to enable emerging applications with high memory capacity and bandwidth demand, such as in-memory computing. Current commercial products integrate four memory

stacks on a passive silicon interposer, connected to the CPU die with point-to-point links [37]. To study scalability, our model assumes a CPU chip and 16 individual die-stacked memory modules, integrated on a silicon interposer. The CPU chip consists of a chip multiprocessor (CMP) with 16 cores, connected via a Network-on-Chip (NoC) in a 2D mesh topology. Surrounding the CPU chip, there are 16 individual die-stacked memory modules (such as HBM), as shown in Figure 3.2. In this physical layout, we do not specify the how the memory modules are connected to the CPU chip as we will explore potential network topologies in Section 3.4.1. CPU and memory modules are integrated on a silicon interposer and connected to the metal layers of the silicon interposer via microbumps. The unique processor-memory traffic goes through the corner cores and corner memory modules—“pillar” nodes—creating a potential bottleneck at these routers. Therefore, in Section 3.4 we discuss optimizing the topology, router design, and routing algorithm to mitigate this bottleneck.

3.4 Network Design

While there exists multiple studies on NoC design in the CPU die, our focus is on the memory fabric design using the already available resources in silicon interposer. In this section, we will discuss the detailed network architecture design to integrate a large number of memory modules on the same silicon interposer-based system.

3.4.1 Topology

In this section, we consider three memory fabric topologies that provide connection between the CPU chip and memory modules using the available resources in silicon interposers for routing capabilities. All these topologies are implemented on the same physical layout shown in Figure 3.2, the only difference between them being the inter-

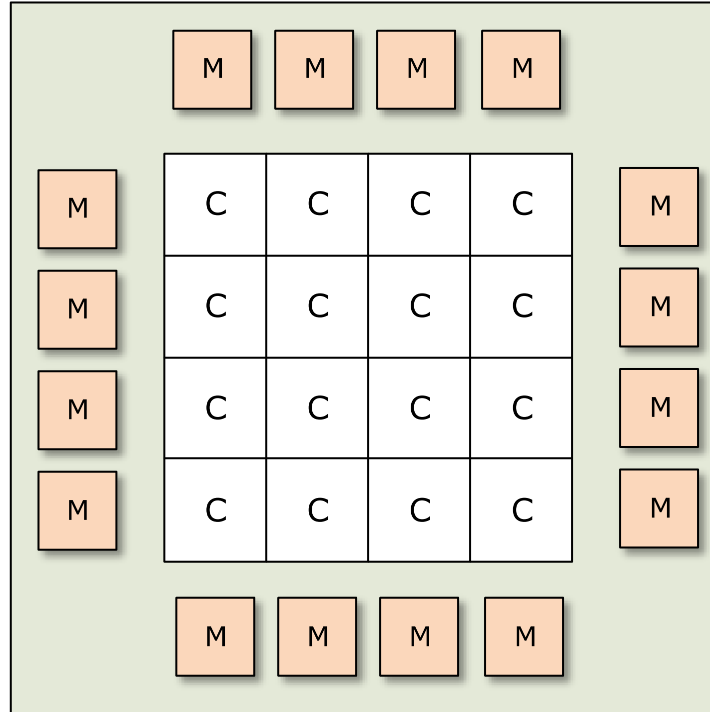

Figure 3.2: Physical layout of the 2.5D design

connections. The first design, as illustrated in Figure 3.3a, implements point-to-point connections between the memory modules and the CPU cores along the edges of the chip. Although each of the memory modules is only one hop away from its corresponding pillar CPU node, they all share the bandwidth of the CPU chip due to the fixed number of micro-bumps. As a result, the total memory bandwidth is statically partitioned to each memory module, limiting the available per-memory bandwidth. This problem will exacerbate as the number of memory modes increases.

The second baseline design, as shown in Figure 3.3b, quadruples the per-memory bandwidth compared to point-to-point design by connecting only four of the memory modules to the CPU chip and the rest to these pillar modules with a daisy chain. However, with this implementation, the number of hops to reach the memory modules is increased due to packets traversing the daisy chain and congestion might occur at the

pillar nodes. Moreover, the total memory address space is still partitioned into four independent regions, causing inefficiency in memory sharing. For example, if the dataset of an application resides in two memory modules separated into two diagonal regions, accesses between CPU and memory may have to traverse the entire NoC, causing long end-to-end delay. In this work, we do not evaluate a case that may exist between these two baselines, as scaling up would be eventually limited by either bandwidth or latency, whereas our goal is to reduce latency while keeping bandwidth the same.

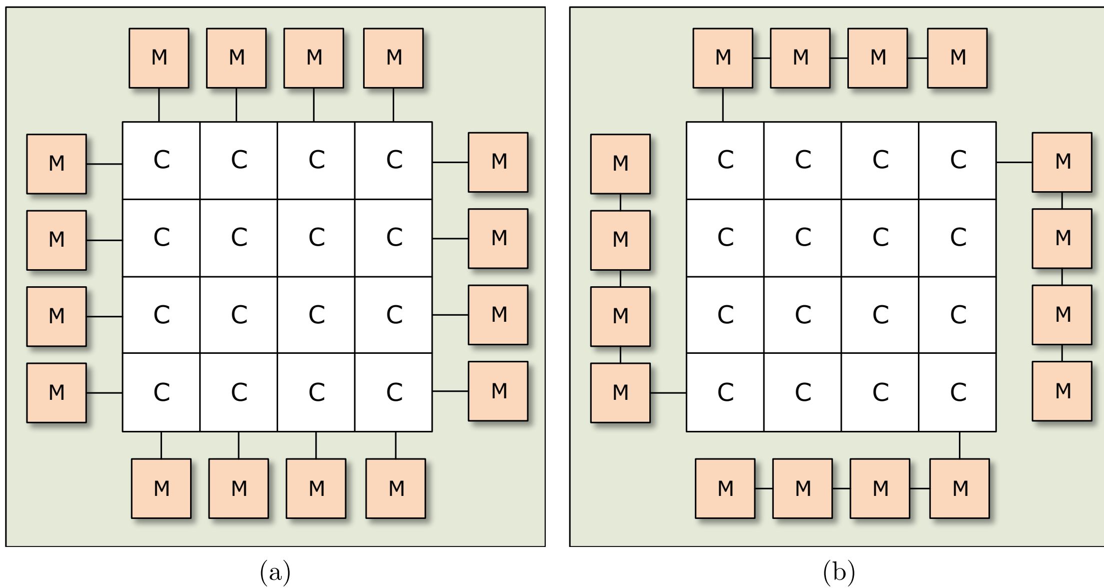

Figure 3.3: Baseline network topologies (a) Point-to-point, and (b) Daisy chain.

Notice that the prior two designs only use the interposer for edge-to-edge communication between adjacent chips (e.g. from CPU to stacked DRAMs). However, apart from the routing resources along the edge of the interposer, a vast majority of the interposer's area and routing resources are unused. Therefore, we leverage the otherwise wasted routing resources of the interposer to implement CPU-to-memory connections. Prior work [40] has proposed similar concepts but only has used the interposer to implement a duplicated NoC. In their case, CPU-to-CPU (cache coherence) traffic will use

the on-chip network, while the CPU-to-memory traffic will use the interposer network. While this design reduces interference of these two traffic classes, the benefit is limited because the NoC bisection bandwidth is already large enough to accommodate the traffic of both. For example, given 128-bit NoC links operating at the same frequency as CPU (3 GHz), a 4×4 mesh NoC would provide 384 GB/s bisection bandwidth.

Therefore, instead of using the interposer routing resources for only CPU-to-memory connections, we implement a memory network in the interposer to interconnect all the memory modules directly, forming a memory-centric architecture. Figure 3.4 shows the logical layout of our proposed memory network in silicon interposer (*MemNiSI*). Note that this scheme occupies the same area as the previous designs physically; however, the illustration depicts the logical connections for clarity.

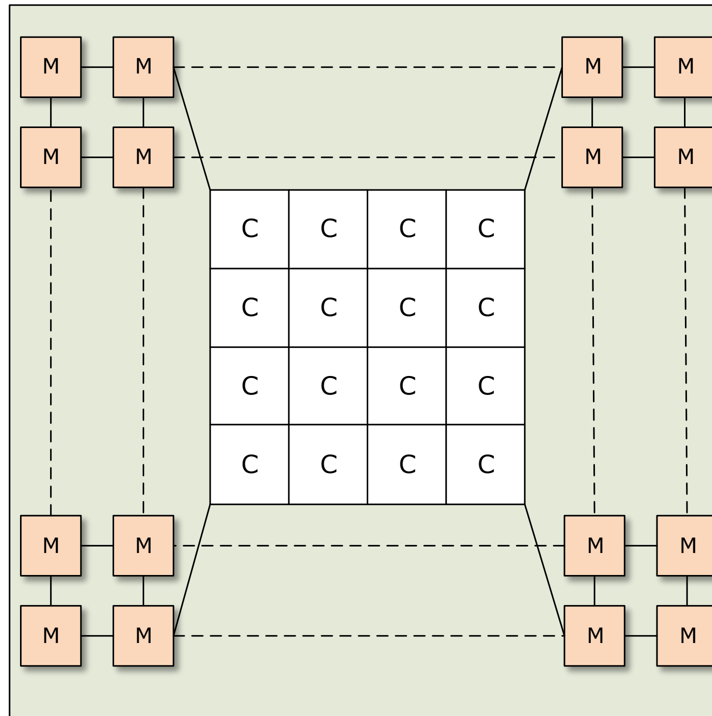

Figure 3.4: Logical layout of the proposed memory network topology, MemNiSI.

As we can see from this architecture, all of the memory nodes are connected using

a mesh network. Even though memories do not directly communicate with each other, the memory network provides additional routing flexibility which potentially reduces the end-to-end communication distances, without sacrificing the per-memory bandwidth. A detailed analysis of the routing algorithms will be provided in Section 3.4.3. In the meantime, a drawback of this design is the existance of longer wires connecting the memory modules across the CPU chip.

Since none of these designs is a clear winner, in Section 3.5.3 we evaluate all three topologies with the routing algorithms introduced in Section 3.4.3 under both synthetic traffic and real workloads.

3.4.2 Technology

Apart from the topology choices as discussed above, there are two important factors that influence the memory fabric design in silicon interposer-based systems: interposer type and process technology.

Silicon interposers can be either active or passive, meaning they can incorporate active devices or not, respectively, as described in Section 3.2.1. Our scheme is applicable to both of these design choices. With active interposer, the routers for the memory network can be implemented in the interposer itself. On the other hand, for passive interposer implementation, we assume the routers are implemented in a logic base under the stacked memory modules so that only wires go through the silicon interposer. Note that the choice between active and passive interposer does not affect the logical network topology. However, taking process technology into consideration, there are several variances that influences our design, as will be discussed next.

There are three outcomes to consider for the various process technology design decisions. The first case assumes the NoC and network in silicon interposer run at the same

frequency. This is a valid assumption because the silicon interposer interconnect speed can be as fast as on-chip communication [40]. On the other hand, the second case considers that the interposer is passive and is implemented with the same process technology as the processor chip. Since repeaters can be used for long wires within the CPU chip to increase the interconnect frequency but not in the passive interposer, NoC will be faster than network in silicon interposer. Finally, network in silicon interposer can be faster than NoC if it is manufactured with an older technology. Using an older technology has two advantages in this design: cost benefits due to higher yield and faster interconnect. The trends in process technology show that interconnect scaling exacerbates as we move to newer process nodes [53]. With these factors to consider, the actual frequency discrepancy between the silicon interposer network and the on-chip network may vary from technology to technology. Therefore, in Section 3.5.3, we provide a sensitivity analysis of our network design by sweeping the frequencies of the two networks.

3.4.3 Routing Algorithm

As mentioned in the previous topology design, memory network provides additional routing flexibilities for end-to-end memory accesses. In this section, we discuss the corresponding routing algorithms. Note that different topologies can be applied to our memory network design, such as torus, flattened-butterfly, etc. We use a generic mesh topology as a case study, while various topologies will cause different design tradeoffs and the search for the ideal network topology in silicon interposer is out of the scope of this work. Therefore, we can simply adopt dimension ordered routing which provides shortest distance routing while guaranteeing to be deadlock-free.

Baseline Designs

Point-to-point and daisy chain topologies implement a variety of XY routing algorithms that we call pillar router first (PRF) routing. PRF routing algorithm first routes the packets to the pillar router closest to the memory module using X-first Y-last routing algorithm and then to the destination router, again using X-first Y-last routing.

Memory Network in Silicon Interposer (*MemNiSI*) Design

Memory network topology allows more routing algorithms to be implemented. For CPU to memory traffic, the flow path is to route a packet from CPU to a pillar node through the NoC and then traverse the memory network to the destination. Reversely, for memory responses, the packets will traverse the memory network before being sent back on chip, and follow the NoC to reach the destined CPU. However, there are several ways to implement this algorithm as discussed below, all based on the XY routing algorithm.

Network in silicon interposer heavy (NiSIH) routing algorithm aims to utilize the silicon interposer more heavily by routing the packets to spend more hops in the silicon interposer network. For a fair comparison to the baseline cases, we take NoC routers in the corners to be the pillar routers. Although not evaluated in this work, there may exist other NoC pillar router candidates, such as the middle four, that may result in performance improvement over our assumption. For CPU to memory traffic, NiSIH first routes to the *closest pillar router to the CPU node* and from the pillar router to destination memory using XY routing via silicon interposer network. For memory to CPU traffic, however, NiSIH routes packets via silicon interposer network back to the *closest pillar router to the CPU* and from the pillar router to destination CPU using XY routing via NoC. Network-on-chip heavy (NoCH) routing algorithm, in contrast to NiSIH algorithm, utilizes the network-on-chip more heavily. CPU to memory traffic is routed

via the *closest pillar router to the memory node* through NoC first, using XY routing. Similarly, NoCH routes the memory to CPU traffic via the silicon interposer network to the *closest pillar router to the memory node*, and from the pillar node to the destination CPU via NoC. Figure 3.5 illustrates the NiSIH and NoCH routing algorithms. We expect NoCH routing algorithm to perform better than NiSIH in designs where NoC is faster and NiSIH to perform better than NoCH in designs where the silicon interposer network is faster.

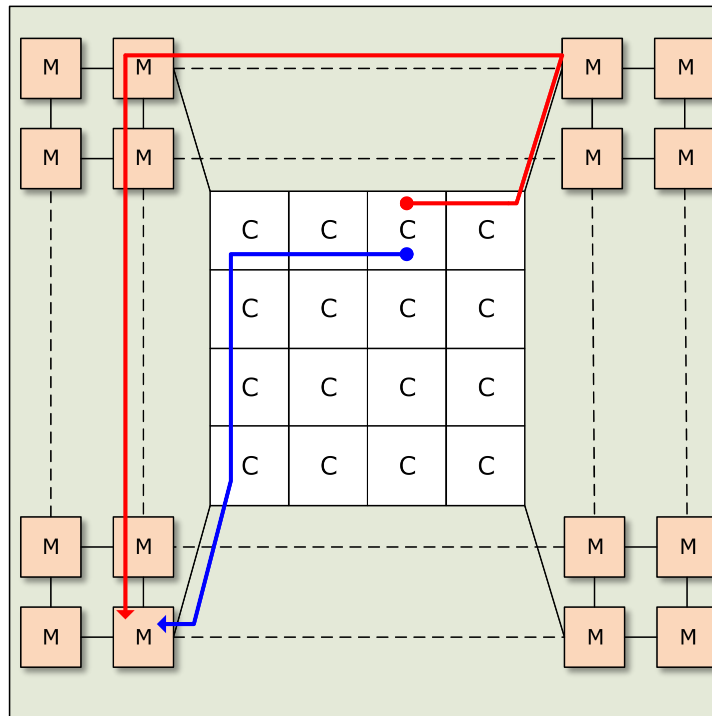

Figure 3.5: Illustration of NiSIH and NoCH routing algorithms on memory network topology, shown in red and blue respectively. Destination node is six hops away from the source node using either algorithm. Memory access response messages follow the corresponding request paths back.

Finally, considering the frequency discrepancy of on-chip network and silicon interposer network, we propose Choose-Faster-Path (CFP) routing algorithm, as shown in Algorithm 1, which estimates the time it takes from source to destination with NiSIH and NoCH given the number of hops from source to destination router and the network

frequencies. Specifically, CFP will dynamically select the routing path that yields the fastest expected route for each instance. Note that we can further improve the accuracy of our dynamic routing path selection process in a number of ways. In practice, not all the links between memory modules may be of the same length. CPF can be enhanced to take the heterogeneity between the link latencies into account while evaluating the fastest path. The exact implementation is dependent on the specific layout and topology considerations. Another improvement to CPF may leverage the congestion status of different routes. However, implementing this will incur significant overhead due to introduction of additional counters and synchronization logic to track the utilization and delay of individual routers. We leave it as future work to analyze the design trade-offs.

Algorithm 1 Choose Faster Path (CFP) Algorithm

Input: Current node, destination node
Output: Routing path

/* Determine pillar routers */
$close_pillar$:= closest pillar to current node
far_pillar := closest pillar to destination node

/* Calculate # of hops to destination */
$dest_close_NoC$:= # hops in NoC to/from $close_pillar$
$dest_close_NiSI$:= # hops in NiSI to/from $close_pillar$
$dest_far_NoC$:= # hops in NoC to/from far_pillar
$dest_far_NiSI$:= # hops in NiSI to/from far_pillar

/* Calculate total expected time to destination */
$total_close = (dest_close_NoC \times NoC_latency) + (dest_close_NiSI \times NiSI_latency)$
$total_far = (dest_far_NoC \times NoC_latency) + (dest_far_NiSI \times NiSI_latency)$

if $total_close \leq total_far$ **then**
  Route via $close_pillar$
else
  Route via far_pillar
end if

3.5 Experiments

In this section, we first describe the infrastructure and workloads setup to evaluate the 2.5D processor-memory interconnect design choices. Next, we present our results for topology study, sensitivity analysis, and routing algorithm study to determine the most optimal design for scalable interposer-based designs.

3.5.1 Simulator Setup

We implemented an event-driven simulator in C++ to model the hybrid network of the interposer-based design introduced in Section 3.3. The simulator can model an interconnect network that allows for specification of network design details such as the topology and routing algorithms. Each CPU core is implemented with an active router which can issue memory and cache coherence requests according to the traffic pattern. The individual die-stacked memory modules share the total available in-package memory capacity and are implemented with a passive router which can only receive and reply to the memory access requests issued by CPU cores. As explained in Section 3.3, the hybrid network consists of a 2D mesh network-on-chip that connects the CPU cores together and a network in silicon interposer that connects the memory modules to the processor chip via silicon interposer. We consider the design choices of network topology and routing algorithm, and distinct link frequencies for the network in silicon interposer. Table 3.1 shows the basic configurations of CPU, memory, and interconnect.

3.5.2 Workloads Setup

In our simulator we implement two synthetic traffic patterns, as described in Table 3.2, *uniform-random* and *hotspot*, with the option of additional CPU-to-CPU communication such as cache coherence to stress the NoC. *Uniform-random* traffic issues memory re-

Table 3.1: System (CPU, memory, and interconnect) configurations

CPU & Memory	16 cores, 2GHz; 64B cache-line size; 16 memory nodes; 4GB per node; 100 cycles memory latency
On-chip network	4-stage router pipeline; 4 VCs per port; 4 buffers per VC; flit size = 16B; maximum packet size = 4 flits
Baseline interposer network	4×4 mesh topology, same router configurations as the on-chip network.

Table 3.2: Different synthetic traffic patterns

Uniform-Random	Every processor sends an equal amount of traffic to every memory node.
Hotspot	Every processor sends a large portion (e.g. 50%) of traffic to a single memory node. The rest of the traffic distribution is uniform-random.

Table 3.3: Real in-memory computing workload characteristics

Pagerank	From the graph analysis benchmark in CloudSuite [54], which runs PageRank on a Twitter dataset with 11M vertices. GraphX is used to run PageRank in Spark.
Tunkrank	A measure of Twitter influence. From the graph analysis benchmark in CloudSuite [54]. Twitter dataset with 11M vertices. Data set size 1.3GB, and GraphLab requires 4GB heap memory.
Spark-grep	A ”grep” job running on Spark, which extracts matching strings from text files and counts how many times they occurred with the Wikipedia dataset provide in BigDataBench [55].
Spark-sort	A ”sort” job running on Spark, which sorts values by keys with the Wikipedia dataset provide in BigDataBench [55].
Memcached	From CloudSuite [54], which simulates the behavior of a Twitter caching server using the Twitter dataset with 8 client threads, 200 TCP/IP connections, and a get/set ratio of 0.8.
Redis	An in-memory database system which simulates running 50 clients at the same time sending 100,000 total queries [56].

quests that are uniformly distributed to all memory nodes. *Hotspot* traffic can issue a selected percentage of memory requests to one specified memory node to create a hotspot.

In order to evaluate real workloads, we extend our simulator to work with memory traces. We consider the following in-memory computing workloads and collect their instruction and memory traces use a Pin-based [57] functional simulator on a Dell PowerEdge T630 server: Pagerank and Memcached from CloudSuite benchmark [54], Redis benchmark [56], and Spark-Wordcount, Spark-Grep, and Spark-Sort from Wikipedia dataset provided in BigDataBench benchmark [55], using Spark-1.4.1 [58]. A detailed description of these workloads is provided in Table 3.3.

3.5.3 Results

In this section, we perform sensitivity analysis and study topology and routing algorithm choices for the processor-memory network in 2.5D designs using the event-driven simulator described in previous sections under both synthetic traffic and real in-memory computing workload traces.

Topology Study

We assume equal network frequencies while studying network topologies. PRF routing algorithm is used in point-to-point and daisy chain designs, whereas NiSIH routing algorithm is used in MemNiSI design. Figure 3.6 shows the mean packet latency of the network topologies with respect to the injection rate under *uniform-random* traffic. Point-to-point topology performs the best under low network load as each memory node is only one hop away from the NoC. Daisy chain, on the other hand, is the slowest topology due to the longer average distance between CPU and memory nodes and the congestion along the daisy chains. Under low network load, memory network design MemNiSI performs in between the two baseline designs as the average distance between the source and destination pairs heavily determine the packet latency. MemNiSI outperforms both

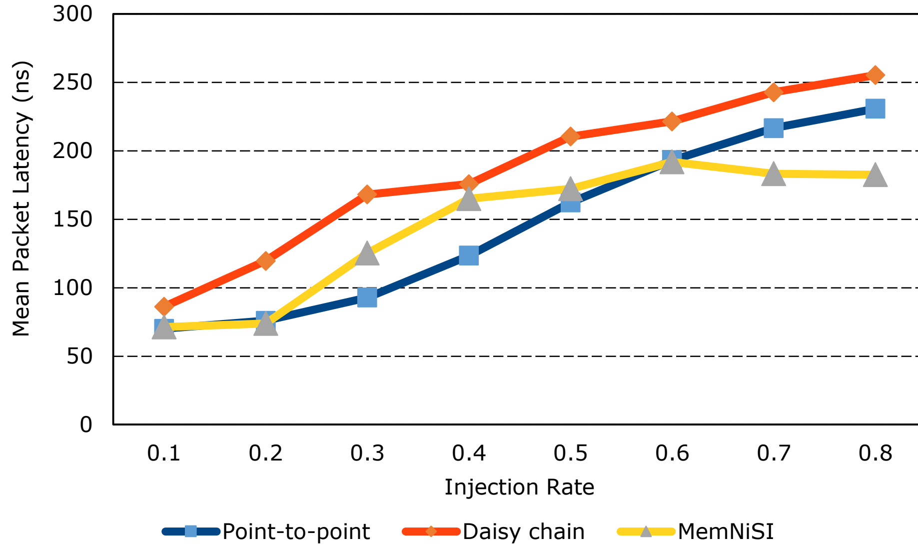

Figure 3.6: Average packet latency comparison of the network topologies under *uniform-random* traffic.

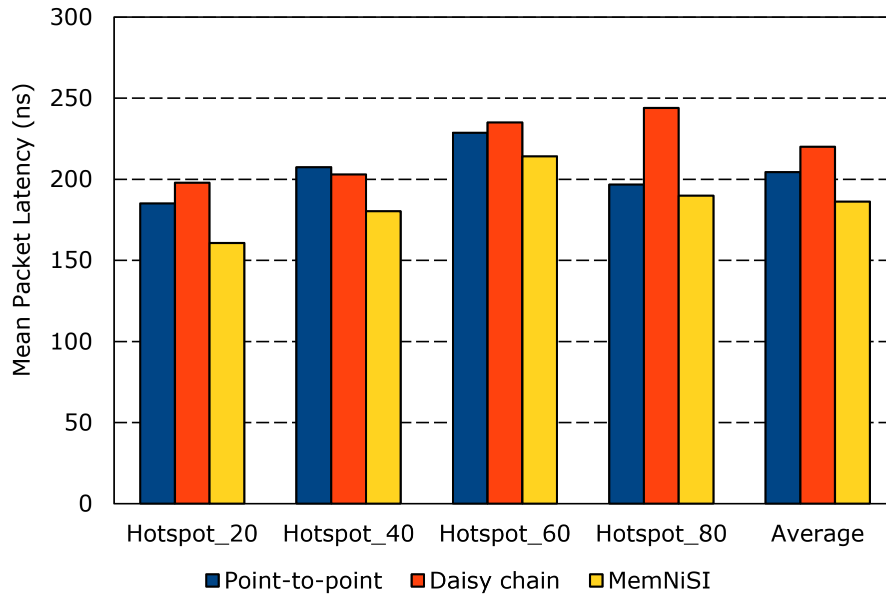

Figure 3.7: Average packet latency comparison of the network topologies under *hotspot* traffic. Lower is better.

designs under heavy network load since the network is not as heavily congested as memory network helps disperse the requests. Furthermore, we observe that while the average packet latency of point-to-point and daisy chain designs steadily increase as the network load increases, MemNiSI levels out, suggesting that it is a more scalable design under high network demand.

Figure 3.7 compares the average packet latency of the three network topologies under *hotspot* traffic which sends the specified percentage of traffic to a single memory node. On average, point-to-point performs slower than MemNiSI design since MemNiSI performs better under heavy network load of hotspot traffic. Daisy chain topology proves to be the least scalable design as the packet latency increases steadily with the hotspot traffic. MemNiSI is the fastest and most scalable design by being 8.92% and 15.33% faster compared to point-to-point and daisy chain topologies under *hotspot* traffic, on average.

Sensitivity Analysis

In order to evaluate our proposed network design better, we perform sensitivity analysis on relative network-on-chip and network in silicon interposer frequencies.

Figure 3.8 shows the network frequency sensitivity analysis under synthetic *uniform-random* traffic for CFP algorithm. As introduced in Section 3.4.3, CFP routing algorithm accounts for the number of hops between source and destination pairs for NiSIH and NoCH algorithms, and picks the one that would yield the fastest route, taking the network frequencies into account. By changing the network frequency ratio of network-on-chip to network in silicon interposer, we are able to observe the algorithm that is being favored. We can observe that for the configurations that NoC is faster than network in silicon interposer (NiSI), NoCH algorithm is favored over NiSIH, and vice versa, since it is faster overall if the packets are routed via the faster network for most of the distance. Another observation is that the algorithm choice difference stabilizes quickly and how much faster

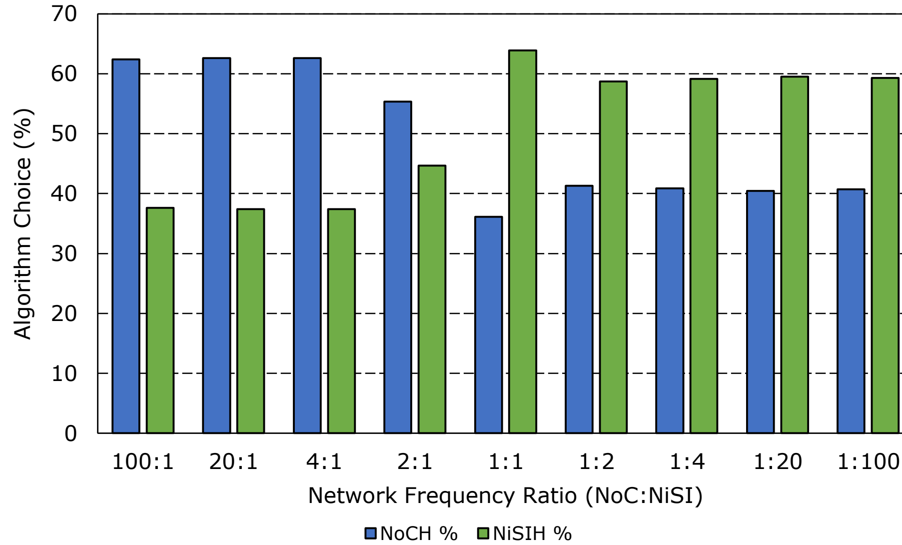

Figure 3.8: Sensitivity analysis for Choose Faster Path (CFP) algorithm which shows the algorithm choice between NiSIH and NoCH for various configurations of network frequency ratios of network-on-chip and network in silicon interposer.

one of the networks is, the packet latency is bounded by the other and so the relative picks of the two algorithms remain the same. Therefore, we pick 4:1 configuration for when NoC is faster and 1:4 for when silicon interposer network is faster for our following experiments.

Routing Algorithm Study

In this section, we evaluate our MemNiSI topology using three routing algorithms: NiSiH, NoCH, and CFP under synthetic traffic and real in-memory workloads.

Figure 3.9 provides a routing algorithm comparison under synthetic traffic for the three network frequency cases, normalized to NiSIH routing algorithm separately for each traffic. For the case where NoC and NiSI are equally as fast, NiSIH performs better than NoCH as also observed in Figure 3.8. This is due to the fact that memory network in silicon interposer reduces the average distance between CPU and memory nodes. For the cases where the two network frequencies differ, the algorithm that leverages the faster

network more yields the lower packet latency as expected. CFP always performs close to or better than the better performing one between NiSIH and NoCH in the particular case. The reason why CFP can perform better is that algorithm pick between NiSIH and NoCH is performed for every source-destination pair, greedily choosing the fastest for each packet. CFP can outperform up to 6.85%.

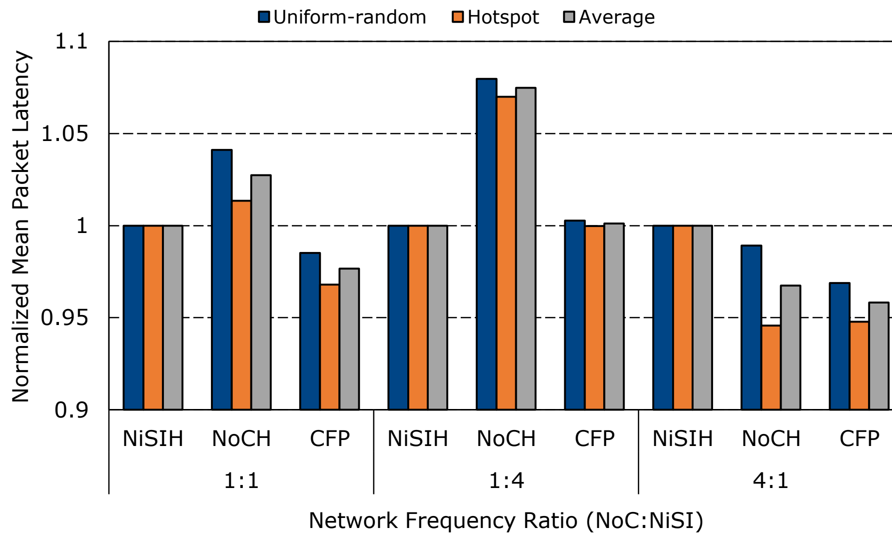

Figure 3.9: Routing algorithm comparison under *uniform-random* and *hotspot* traffic, normalized to NiSIH algorithm. Results show the various configurations of network frequency ratios of network-on-chip and network in silicon interposer.

Figure 3.10 compares routing algorithms for the in-memory computing workloads specified in Section 3.5.2. The results are normalized to NiSIH routing algorithm. On average, CFP outperforms NiSIH by up to 3.38% and NoCH by up to 10.03%. For the case where the frequencies of the two networks are equal, CPF performs better on average than NiSIH by 1.65% and NoCH by 7.99%. This shows that memory network in silicon interposer designs will benefit even from a naive CFP implementation.

Experimental results show that a memory network approach suits well to scalable interposer-based designs. In this work, we mainly focus on providing design considerations and are only able to explore a small part of the vast design space. Even though this

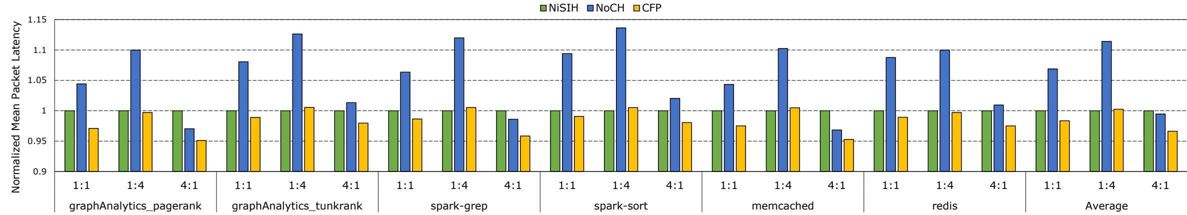

Figure 3.10: Routing algorithm comparison under real in-memory computing workloads for NoC to NiSI frequency ratios of 1:1, 1:4, and 4:1, normalized to NiSIH. Results show the various configurations of network frequency ratios of network-on-chip and network in silicon interposer.

work focuses on evaluating the network latency of a system of scalable memory, we plan to consider other metrics, such as power, area, cost, and reliability in order to compare processor-memory network designs in our future work.

3.6 Conclusion

One of the major challenges in large-scale interposer-based 2.5D designs is providing a scalable fabric for high-bandwidth, low-latency communication between the processor chip and stacked memory nodes. In this work, we demonstrate that the existing processor-memory network of 2.5D designs cannot provide the full potential of the memory performance and memory-centric network design in silicon interposer is a promising scalable solution. We present a design space exploration that evaluates network design, topology, and routing algorithms, taking process node and interposer technology design decisions into account. Our scheme achieves better utilization of integrated stacked memory while allowing better scalability and infrastructure for large-scale silicon interposer-based 2.5D designs.

Chapter 4

There and Back Again: Optimizing the Interconnect in Networks of Memory Cubes

High-performance computing, enterprise, and datacenter servers are driving demands for higher total memory capacity as well as memory performance. Memory "cubes" with high per-package capacity (from 3D integration) along with high-speed point-to-point interconnects provide a scalable memory system architecture with the potential to deliver both capacity and performance. Multiple such cubes connected together can form a "Memory Network" (MN), but the design space for such MNs is quite vast, including multiple topology types and multiple memory technologies per memory cube.

In this work, we first analyze several MN topologies with different mixes of memory package technologies to understand the key tradeoffs and bottlenecks for such systems. We find that most of a MN's performance challenges arise from the interconnection network that binds the memory cubes together. In particular, arbitration schemes used to route through MNs, ratio of NVM to DRAM, and specific topologies used have dramatic

impact on performance and energy results. Our initial analysis indicates that introducing non-volatile memory to the MN presents a unique tradeoff between memory array latency and network latency. We observe that placing NVM cubes in a specific order in the MN improves performance by reducing the network size/diameter up to a certain NVM to DRAM ratio. Novel MN topologies and arbitration schemes also provide performance and energy deltas by reducing the hop count of requests and response in the MN. Based on our analyses, we introduce three techniques to address MN latency issues: (1) Distance-based arbitration scheme to improve queuing latencies throughout the network, (2) skip-list topology, derived from the classic data structure, to improve network latency and link usage, and (3) the MetaCube, a denser memory cube that leverages advanced packaging technologies to improve latency by reducing MN size.

4.1 Introduction

Computing trends in multiple areas are placing increasing demands on the memory systems of modern servers, supercomputers, and even high-end workstations. Greater total memory capacity is required to solve huge scientific problems, tackle massive data analytics challenges, and host larger numbers of virtualized servers. For conventional memory systems (e.g., DDR4), increasing memory capacity comes with a performance penalty. A typical server processor package has a limited number of memory channels. If a single dual-inline memory module (DIMM) is placed on a channel, the bus speed can typically be operated at its maximum speed. However, as the number of DIMMs per channel increases, the electrical loading often requires the bus to be operated at a lower rate, trading memory bandwidth of the slower bus for greater memory capacity of extra DIMMs. Furthermore, the ability to increase capacity is typically limited to a few (e.g., three) DIMMs per channel.

The other approach to increase memory capacity is to increase the number of memory channels per processor package. However, this is very expensive in terms of the number of pins required and the subsequent impact on package design. The DDR4 interface requires 288 pins per channel; e.g., a typical four-channel server processor would need 1,152 pins for memory interfaces alone. We do not further explore increasing memory capacity by adding more channels in this work.

One recent alternative approach is to combine 3D-stacking technology for higher per-package capacity with narrower high-speed serial links, internal switches, and an abstracted interface (as opposed to low-level DRAM commands). The primary demonstrator of this is embodied in Micron's Hybrid Memory Cube (HMC) [59], but the concept is being espoused by others [60, 61].

Current HMC capacity is 2-4GB per cube [59], although other 3D-stacked memory technologies, such as JEDEC's HBM2, are projected to provide up to 8GB per DRAM stack [62]. Furthermore, the memory capacity and bandwidth are expected to double with HBM Gen3 [63], so we expect that a memory cube could support similar capacities. By chaining (or arranging in other topologies) multiple memory cubes, system capacity can be scaled while maintaining high bandwidth through the use of high-speed point-to-point links (i.e., unlike DDR, link speed does not fall off with more packages). The tradeoff is the interconnect latency to reach a memory cube can become non-trivial, especially if the destination memory cube requires multiple hops to reach.

In this work, we start off with a performance analysis of a variety of different topologies of memory networks (MNs). From this, we identify several bottlenecks in building large, terabyte-scale memory systems. There is no single Achilles Heel for MNs, but the performance challenges come from a combination of the latency of the underlying memory technology, the multi-hop interconnect latency, and queuing/arbitration issues in the MN. Based on these observations, we recommend several improvements to MNs to

increase performance for large-capacity memory systems. These include a new MN topology inspired from classic "skip list" data structures, a distributed arbitration mechanism to improve overall MN throughput, and the mixing in of non-volatile memory technologies to achieve different tradeoffs between interconnect latency (MN size/diameter) and memory array latency. We also introduce a "cube-of-cubes" concept that can further improve performance, especially for systems with very large capacities.

To further increase per-package capacity, another approach would be to employ non-DRAM storage devices. In particular over the last several years, there has been tremendous interest in a variety of non-volatile memory (NVM) devices (e.g., phase-change memory, STT-MRAM, memristors, 3D-XPoint) that are promising as either replacements or augmentations to DRAM. NVM-only or hybrid (NVM and DRAM) cubes provide another dimension to further increase system capacity, but here the tradeoff is that NVM latencies are typically integral factors slower (e.g., 2-8$\times$ depending on the exact device) than DRAM.

While the memory-cube approach is quite exciting for its ability to significantly increase memory system capacity *and* performance, the memory system architect is faced with a very large design space. To help navigate this large space, we propose a simple taxonomy for this class of memory systems that we call a *Network of Memories* (NOM). The different quadrants of the taxonomy are based on different possible mixes of memory technologies (i.e., DRAM vs. NVM), but the choice has direct implications on the organization and topology on the NOM. Based on our observations and analysis, we propose a *MetaCube* (cube of memory cubes) structure that combines memory cube concepts with silicon interposer packaging technology. The MetaCube enables overall memory organizations that significantly reduce interconnect latencies (hop counts) while still providing high total memory capacity.

4.2 Background

In this section, we briefly discuss the limitations of the conventional DDR interfaces and introduce recent memory technologies that we use to form MNs, particularly, memory cubes and non-volatile memories.

4.2.1 Limitations of DDR

As discussed in the introduction, bus-based memory interfaces such as DDR4 typically take a performance hit as memory capacity is increased. As more DIMMs are added to the bus, the increase in electrical loading makes it harder to run the interface at the highest speeds. The exact supported bus speeds for a given number of DIMMs per channel (DPC) varies by manufacturer and server part. Table 4.1 shows example maximum bus speeds for DDR3 and DDR4 memories given different numbers of DPC. For DDR3, to operate at the higher bus speed of 1333 MHz, only one DPC can be populated. To achieve higher capacity with two DPC, the bus speed drops to 1066 MHz, and three DPC goes all the way down to 800 MHz.[1] DDR4 has some improvements (especially if registered (RDIMM) or load-reduced DIMMs are used), but the same general phenomena apply. When going from one DPC to two, there is no drop-off in bus speed, but adding the third DPC still incurs the performance penalty. Attempts to support even more DPC to further increase capacity exacerbate the electrical challenges of high-speed signaling due to the additional impedance on the bus. For this reason in many servers, three DPC is the limit supported. These examples illustrate the fundamental tradeoff between memory capacity and memory performance in conventional bus-based, multi-drop memory interfaces.

[1] There are variations where multi-rank DIMMs can cause the bus speed to drop compared to a single-rank DIMM even for the same number of DPC, but such details are not key to the current discussion.

Number of DPC	DDR3	DDR4
1	1333 MHz	2133 MHz
2	1066 MHz	2133 MHz
3	800 MHz	1866 MHz

Table 4.1: Maximum memory interface speeds given different numbers of DIMMs per channel for DDR3 [64] and DDR4 [65].

4.2.2 Memory Cubes

The maturity of 3D die-stacking technology along with advancements in short-range, high-speed signaling, have enabled a new memory package organization. The pioneering example of this approach is embodied in the Hybrid Memory Cube (HMC) [27, 28]. Figure 4.1(a) shows a conceptual depiction of a memory cube. 3D stacking is used to integrate multiple layers of DRAM on top of a base logic die. The logic die implements an array of memory controllers tightly coupled to the vertical DRAM channels ("vaults") through arrays of through-silicon vias (TSVs) that has additional performance and energy benefits from having the controllers physically close to the DRAM with a fixed amount of electrical loading (a single controller-to-vault interface is represented by the vertical dashed line in the figure). The logic die also implements high-speed serial links to the outside of the package. These links can be used to connect to a host processor or to other memory cubes. A switch on the logic die connects the external links to the internal memory controllers. HMC's logic die also implements other functionality such as built-in self-test and error correction, but we do not explore these additional types of features in this work.

The external links of a memory cube can be operated at very high speeds in part due to the point-to-point nature of the memory organization. Figure 4.1(b) shows a processor connected to four chains with four memory cubes per chain. Each link starts at one package and ends a short distance away at the next package; this is in contrast to

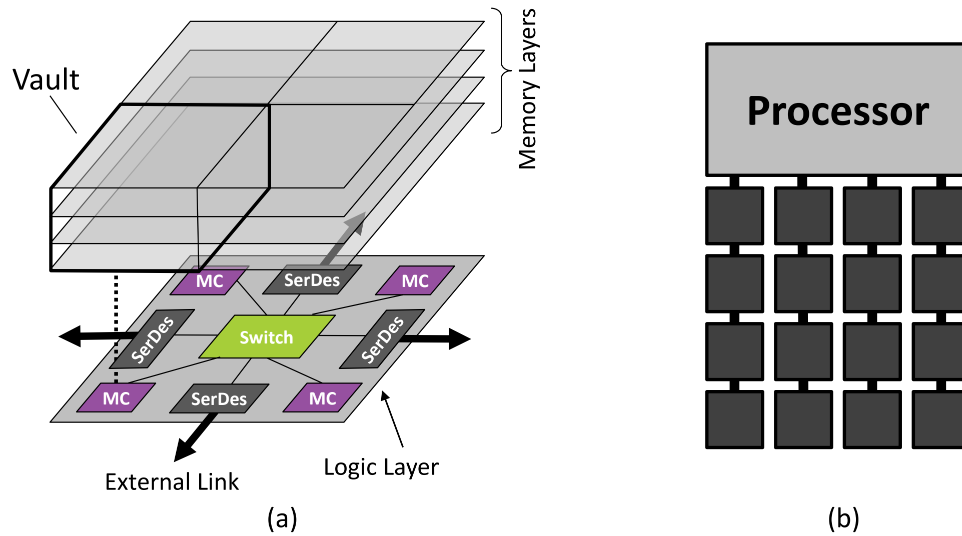

Figure 4.1: (a) Conceptual view of a Hybrid Memory Cube with a logic layer and four memory layers, and (b) an example processor with multiple HMCs arranged as four parallel chains.

a DDR bus that traverses a greater distance on the motherboard and may have a variable number of DIMMs. HMC 1.0 offered links speeds up to 15 Gbps, for a peak aggregate bandwidth of 240 GB/s [27]. HMC 2.0 (short-range version) supports link speeds up to 30 Gbps (peak aggregate bandwidth of 320 GB/s/link) [28]. In contrast, the fastest current DDR4 bus speeds are only 1.6 Gbps (25.6 GB/s/channel). Furthermore, the DDR4 interface requires 288 pins per channel, while HMC 2.0 only needs 66 pins per link; i.e., for the same pin cost on the processor package, one could have over four times the number of HMC 2.0 links (this is a simplistic comparison as there are other costs apart from pins).

Another advantage of the memory cube approach is that the package-to-package links make use of a much more abstract interface than DDR [60]. DDR requires the host processor's memory controller to send a sequence of explicit, low-level, carefully-timed commands (e.g., precharge, row activation, column read). With an abstract interface, an asynchronous protocol is instead employed, where the host processor can, for example,

issue a "read" command to the memory cube that then internally can decide how (and with what timing) to read the data from the DRAM. Whenever the data access has completed, the memory cube's logic die formulates a response packet to be sent back to the host processor. Such an abstracted interface decouples the semantic actions (e.g., read) from the low-level device behaviors (e.g., row activation).

Baseline Memory Cube: We briefly describe our baseline assumptions for the organization of the memory cubes assumed throughout the rest of this work.[2] The logic die at the bottom of the stack contains the SerDes (serialization-deserialization) interfaces for the external high-speed links. The SerDes are coupled to a switch that either forwards requests out of the appropriate external link, or to one of the internal memory controllers. The memory stacked on top of the logic die provides multiple channels for independent and concurrent accessing of the memory resources. We do not assume any specific memory layout; vertical "vaults" like in HMC could be used, or each layer could implement one or more independent channels as in HBM. The memory controller receives a request, and then handles any further processing (e.g., issuing of device-level sub-commands) required to complete the request.

While memory cubes could support additional functionality, this work does not consider any further capabilities beyond routing requests and performing reads and writes of memory (along with any basic "maintenance" operations required for proper operation, e.g., refresh). These could include but are not limited to built-in self-test, ECC, repair, thermal sensing and management, or processing in-memory features. The switch within the logic layer could be a single monolithic multi-ported switch, or it could consist of multiple switches or routers arranged as a small network on chip.

[2]We use the term "Memory Cube" instead of Hybrid Memory Cube or HMC, because HMC is a definition for specific family of memory cubes. As this work considers future cube organizations, the more general term Memory Cube was deemed to be more appropriate (as well as to not be misleading about HMC).

4.2.3 Memory Networks

Industry efforts from HP [66] and IBM [67] demonstrate the trend toward interconnecting memory modules to enable efficient main memory expansion. In academia, several key prior works have already started the exploration of Memory Networks (MNs). Kim et al. [68] introduced the term Memory-Centric Network (MCN) and then later shortened it to just Memory Network (MN) [69]. We follow the more succinct MN terminology here.

The past works have considered a variety of computing scenarios including multiple CPU packages and CPUs with multiple discrete GPUs. Examples of these are shown in Figure 4.2(a) and (b). A key assumption in these past systems is that all memory cubes are connected to all other memory cubes. This may be appropriate for the multi-GPU systems where the MN not only provides the memory storage, but also the datapath for the disparate GPU units to access all of the shared memory.

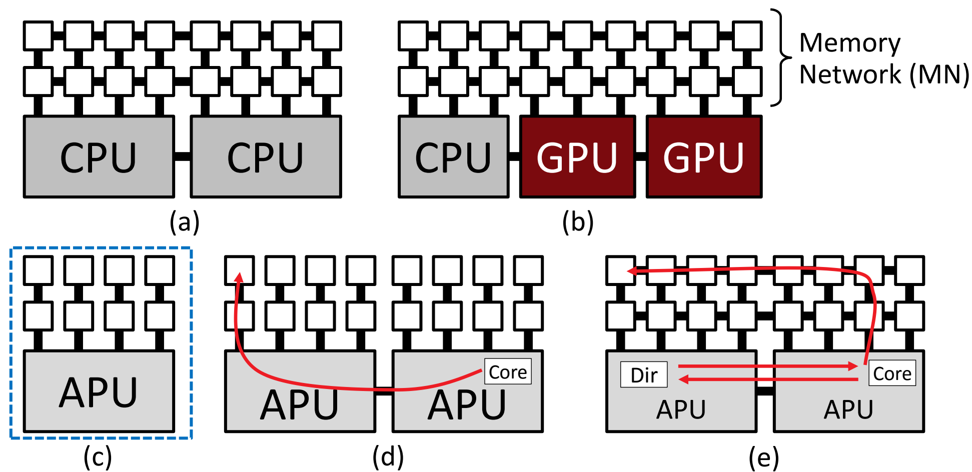

Figure 4.2: Memory Networks (MNs) connecting (a) multiple CPUs and (b) a CPU with multiple GPUs. Disjoint per-port MNs for (c) one APU and (d) two APUs. (e) Example showing the ineffectiveness of fully-connected MNs in a cache-coherent multi-package system.

In this work, our baseline system consists of a high-performance heterogeneous processor consisting of CPU and GPU resources integrated in a single chip (sometimes referred

to as an accelerated processing unit or APU). The APU supports multiple memory ports, but we make a key assumption that simplifies the MNs. In particular, as is typically done for systems today, the physical memory address space is interleaved across the APU's memory ports (this applies for CPU-only configurations as well). As a result, each memory port serves a *disjoint* subset of the memory space, which means that there is no need for memory cubes from one port to be connected to memory cubes on another port.[3]

Figure 4.2(c) and (d) shows examples of such MN organizations for one and two APU packages, respectively. Note that in particular for the dual-package configuration in Figure 4.2(d), requests from the right APU package destined for the left set of memory cubes must first route to the left APU package. At first glance, it would seem desirable to route such requests directly to the left memory cubes through a MN connecting all cubes, which would leverage the greater interconnect bandwidth of the MN and possibly reduce the end-to-end request latency. However in a cache-coherent system, requests would likely first need to route to the left APU anyway to consult cache coherence structures (e.g., directories); once a request has been routed to the left APU for coherence reasons, there is no point in sending the request *back* to the right APU only to re-route through the whole MN as shown in Figure 4.2(e). As a result, we focus on MN organizations similar to those shown (highlighted with a dashed box) in Figure 4.2(c).

4.2.4 Non-Volatile Memories

Various emerging memory technologies (e.g., phase-change memory, STT-MRAM, memristors, 3D-XPoint) are gaining traction to be used as replacement for, or together with existing technologies, in caches and main memories. This trend is primarily due

[3]The exception is to support reliability schemes where a hardware failure at one memory port can be compensated for by routing requests around through another port. As this work is not focused on RAS issues and our proposals can be generalized to include some of this type of redundancy, we maintain this simplifying assumption of disjoint per-port address spaces through the rest of this work.

to their advantages such as non-volatility, density, and decoupled sensing and buffering. DRAM can entirely be replaced with NVM [70, 71], or can be reorganized in a multi-level hierarchy with different memory technologies. In this work, we consider mixing DRAM and NVM memory cubes within the same MN. Such an organization requires management of data migration across different technologies/levels of memory, and metadata management. Prior works have attempted to address these issues [72, 73, 74, 75]; in this work we rely on the existence of appropriate heterogeneous management mechanisms (we do not try to re-invent anything on this particular topic as that is not the focus of the current work).

4.2.5 Baseline MN Performance

We begin with a performance analysis and deconstruction of a few simple MN systems. Figure 4.3(a) shows our baseline APU system. The APU has eight memory ports, each connected to an independent MN serving a disjoint slice of the global physical memory address space, as discussed in Section 4.2. The full details of the APU micro-architecture and other evaluation details can be found in Section 4.4. Connected to each of the APU's memory ports is a set of memory cubes interconnected in identical topologies. We study a system with a total memory capacity of 2TB, where each memory cube has a 16GB capacity (technology assumptions are also explained in Section 4.4). This leads to a total count of 128 memory cubes for the entire system, or 16 cubes per memory port. In terms of memory management, we make a conservative assumption that memory requests are uniformly interleaved based on their *addresses*. This means that for a system with 50% capacity from NVM, half of the memory requests will go to the NVM cubes. Although this work does not focus on the management of heterogeneous memory, prior work [76, 73, 74, 75, 77] exemplify how one can make use of the underlying MN more

effectively.

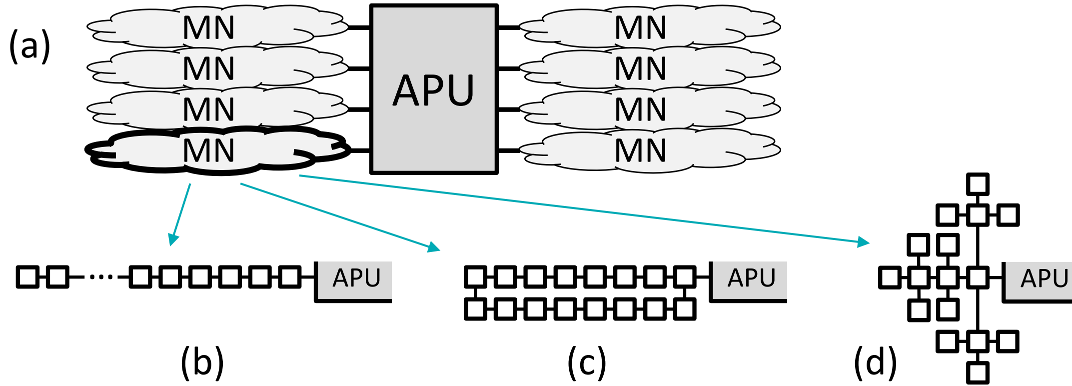

Figure 4.3: (a) APU with eight memory ports, each with a disjoint MN. The MN topologies evaluated in this section include (b) chain, (c) ring, and (d) tree topologies.

We evaluate three different MN topologies. The first is a basic chain of memory cubes, shown in Figure 4.3(b). This baseline minimizes the number of ports required per memory cube and has similarities with buffered DRAM topologies [78], but also suffers from large hop counts to reach the further memory cubes in the chain. The second is a ring topology, shown in Figure 4.3(c), that halves the average hop count compared to the chain as requests can take the shorter of the upper or lower branches of the ring. The last topology is a tree, shown in Figure 4.3(d), that in theory makes the most effective use of the memory cube's multiple external links. This results in a MN that has a worst-case hop count that increases only logarithmically with the number of memory cubes in the network. We do not consider a mesh in this work as the average hop count is larger than a tree no matter which memory cube is connected to the host, and for the reasons discussed in Section 4.2.3. Furthermore, we model HMC-like memory packages with 4 ports per package and only create networks utilizing the routers on the packages rather than adding specialized intermediate routing devices to the PCB. This means that high radix networks and networks requiring intermediate routers with no external link, such as flattened butterflies, distributed networks, all-to-all, etc. [68] are not considered.

4.2.6 Speedup Results

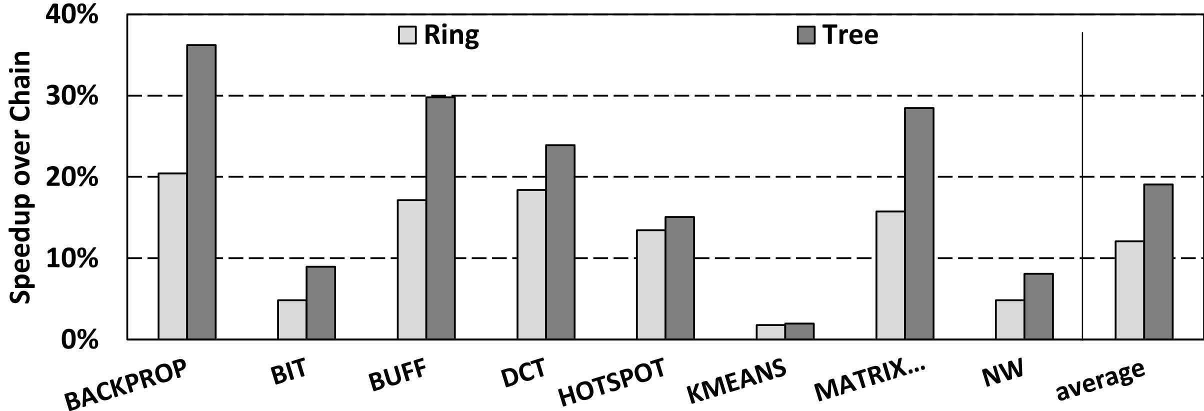

Figure 4.4: Speedup comparison of DRAM memory networks normalized to a chain topology.

We first compare the speedup of MNs consisting of all-DRAM cubes. Figure 4.4 compares the speedup of the topologies normalized to the chain, which always provides the lowest performance. As expected, the topology with the fewest hops (i.e., tree) provides the best performance.

To further reveal potential bottlenecks in the topologies, we look at the breakdown of network latencies in both directions (to and from the destination cube) compared to the core access latency of the memory array.

4.2.7 Latency Breakdown

Figure 4.5 shows the breakdown of latency to memory, latency in memory (i.e., memory array access), and latency back from the memory cube. We make three key observations revealing the potential bottlenecks that we later address in Section 4.3.

First, in most workloads, the latency spent in the network is significantly higher than latency of the memory core. This is especially true during periods of high network load. The discrepancy between latencies to and from memory occurs due to the network's

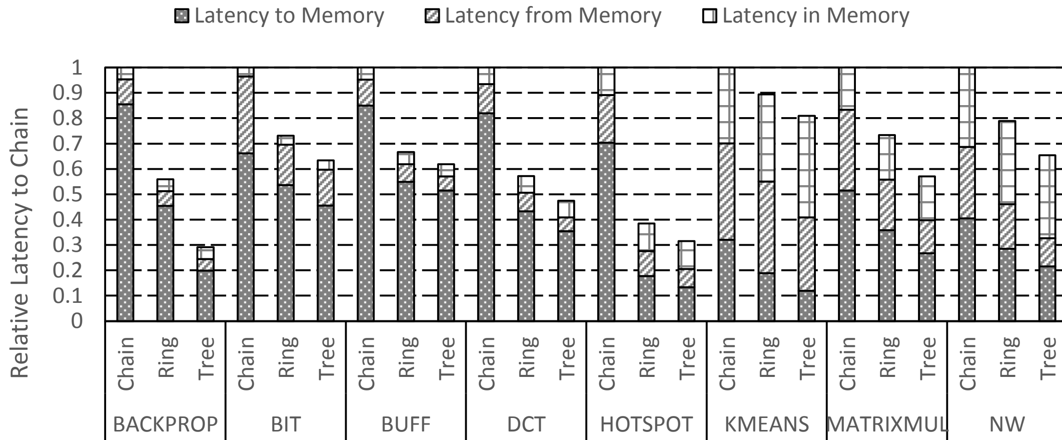

Figure 4.5: Breakdown of memory request latency in DRAM memory networks normalized to chain topology.

single link that by default prioritizes responses (i.e., read data or write acknowledgment) over requests in order to prevent deadlocks from older responses being blocked by newer requests. This causes higher queuing latency at the outgoing memory port as requests are backed up waiting for responses to free up buffer slots. Overall, we observe that the topologies with fewer hops greatly reduce the network latency while the latency in memory remains relatively constant.

Second, differences between latency breakdowns across workloads is caused by the ratio of reads to writes in workload. We assume read responses and write requests (i.e., packets with data) are 5 times larger than control packets (i.e., read request and write response). For example, the workloads KMEANS, MATRIXMUL, and NW all have at least two reads for every one write on average. The BACKPROP workload has significantly more writes than reads, and the remaining workloads have very similar amounts of read and write requests. The NW workload also has the lowest network load of all the workloads plotted, and therefore has the highest percentage of latency spent performing the memory array access due to lower contention in the network.

Last, to gain additional insights into where the cycles are being spent, we analyzed the

waiting times observed in the various structures in the per-cube routers. In particular, we observed that the queuing latencies for the router input-ports were highly unbalanced, with the cubes closer to the processor showing more problems. This is because the default router implementation uses a locally-fair round-robin arbitration scheme that picks from each input queue in a uniform manner. However, four of the input queues come from the cube's local memory vaults, while the remaining input queues serve return traffic from other cubes. As an extreme example, the chain topology would pick the local input queues four out of five times (80% service) while picking the port connecting to the rest of the chain only one out of five times (20% service). Such unfairness in multi-stage arbitration has been previously observed for a two-stage 3D router [79], but with up to n stages in a MN, the effects are much worse. Hence, in arbitration we propose a globally fair arbitration technique to overcome the unbalanced queuing latencies.

4.2.8 Heterogeneous Mix of Cubes

The somewhat unsurprising observation that having network latency scale with the number of hops leads us to the conclusion that introducing higher density memory cubes can allow us to reduce the number of hops to decrease network latency. Using NVMs in memory cubes can provide this density benefit at the cost of higher latency. However, because of the magnitude of the network latency, the increase in access latency may be worthwhile to save on network latency while still reducing the overall round-trip latency. The heterogeneity in the memory network introduces non-uniformity in memory access, in terms of both topology and memory technology. Therefore approaches addressing non-uniform memory access (NUMA) systems can also be considered for MNs.

In the limit, given n memory cubes and a bound on the number of ports/links per cube (as well as motherboard routability concerns), topology selection can only go so far to

reduce the number of hops through the MN interconnect. Another approach is to attempt to reduce n directly. To do this, we take advantage of emerging non-volatile memory technologies that provide greater storage density compared to conventional DRAM. For a memory technology such as phase-change memory (which is what we assume in our evaluations), we project that a memory cube can provide 4× as much capacity (e.g., 64GB per NVM cube compared to our baseline 16GB per DRAM cube). If all cubes are replaced by NVM, then the MN size can be reduced to only $\frac{n}{4}$ cubes, with a corresponding reduction in MN hop counts. However, this is not free, because the latency and energy to read, and especially to write, for NVM is worse than of DRAM.

Instead of implementing a MN with only DRAM (minimizing memory array latency, shown in Figure 4.6(a)) or with only NVM (minimizing interconnect latency, shown in Figure 4.6(b)), we propose to build the MN with a mix of memory technologies. By varying what fraction of a MN's capacity is provided by DRAM versus NVM, we can vary how much of a request's latency (on average) comes from interconnect versus memory technology. Figure 4.6(c) shows a MN where half of its capacity is provided by DRAM and the other half from NVM. This results in two NVM cubes and eight DRAM cubes, for a total MN size of n=10 cubes (compared to n=16 for the all-DRAM case). The examples in Figure 4.6 show chain topologies simply for ease of illustration; other topologies can be constructed.

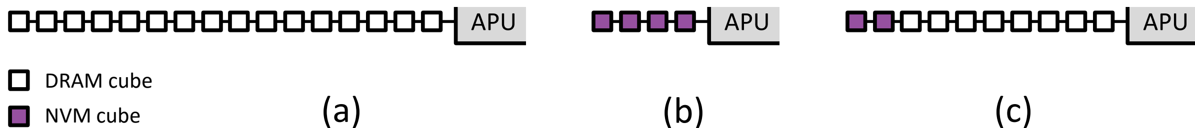

Figure 4.6: Example MN topologies using (a) only DRAM memory cubes, (b) only NVM memory cubes with 4× the capacity per cube, and (c) a mix of both DRAM and NVM where each *type* provides half of the MN's total capacity. The figure only shows a single memory port, configuration is repeated across all ports.

We performed a similar performance analysis using different ratios of NVM to DRAM. Throughout this work, the different ratios of NVM to DRAM are labeled by the percentage of DRAM in the MN by capacity. For any MN where multiple technologies exist, it is possible to change the locations of the memory cubes in the network. In other words, a MN with both NVM and DRAM can choose to place the NVM further away from or closer to the system port. These are differentiated using the suffixes -L (last) and -F (first), respectively. This location refers to the position of the NVM cubes. The topologies are differentiated using their first letter as the suffix. For example, 50%-C (NVM-L) is a MN designed with a chain topology with 50% DRAM and 50% NVM, where the NVM cubes are placed further away from the processor. A 50%-C (NVM-L) configuration is depicted in Figure 4.6(c).

Figure 4.7 shows the speedup results for the tree topology only. These are again normalized to the performance of a 100% Chain. The key takeaways from this figure are that it is beneficial to use some amount of NVM in the MN. There is no clear best topology in this case, however, the 50%-T (NVM-L) topology performs the best on average. The 0%-Tree varies highly with the workload. Workloads with the lowest network contention tend to show degradation, for example NW. This happens because the latency spent in the network is not as prominent as in other workloads, where the dramatic reduction in hop count begins to reduce latency more than the NVM cubes increase latency.

The overall latency breakdown is very similar to the all-DRAM case. In particular, there is very unbalanced request and response latency, whereby most workloads exhibit behavior where the request path is significantly longer than the response path. Additionally, workloads with higher read-to-write ratios typically have more balanced latency breakdowns. Due to this similarity, we focus on network latency optimizations over memory core latency.

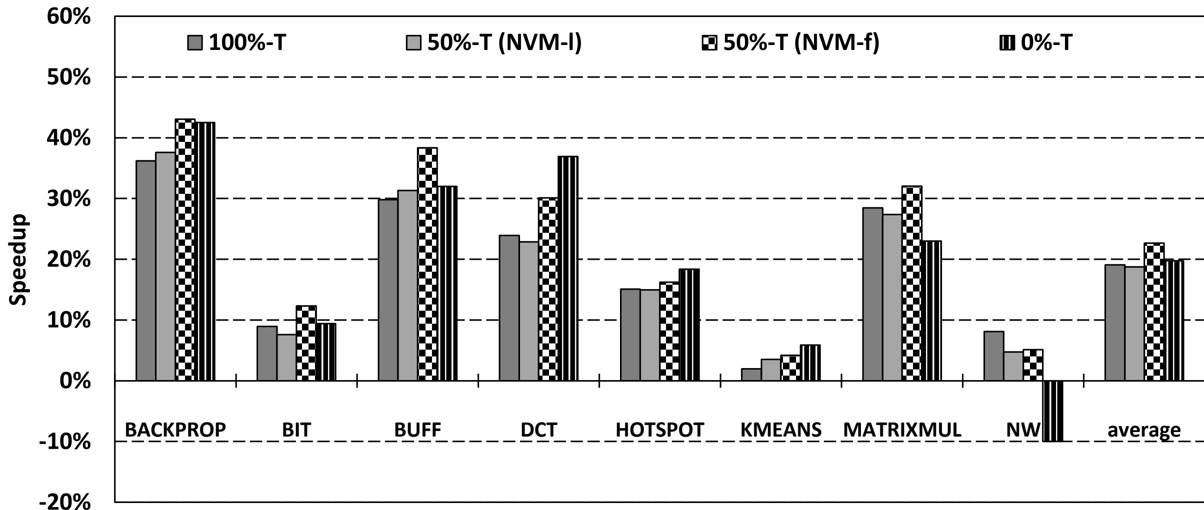

Figure 4.7: Performance analysis of a Tree-based topology with different ratios of DRAM to NVM. 100% implies all DRAM while 0% implies all NVM. NVM-L places NVM cubes last (further away from the processor) while NVM-F places NVM cubes first (closer to the processor).

4.3 Addressing MN Latency Issues

The analysis from the previous section showed that various factors of the MN interconnection are responsible for the majority of a request's end-to-end latency. In particular, low hop-count topologies such as the tree topology performed significantly better than a ring or chain topology. Additionally, we observed a large increase in latency within the network due to arbitration techniques and packet prioritization.

In this section, we first introduce a distance-based arbitration technique that reduces unbalanced input port selection by attempting to service older requests first. We then propose two topologies that attempt to exceed the performance of a tree topology. The first topology exploits the fact that memory traffic is kept coherent beyond the system port and uses this to provide more paths based on request type. The second is a cluster-like topology that leverages die-stacking further in order to reduce the hop count beyond that of a tree topology.

4.3.1 Fair, Distributed Arbitration

Section 4.2.5 showed how a locally fair round-robin arbitration on a memory cube's router can lead to global unfairness by introducing significant queuing delays for the ports connected to more distant cubes, also referred to as the "parking lot problem" [80]. If the router was made aware of how many requests were pending not only at a particular input port, but also downstream at any additional memory cubes that have to eventually flow back through this port, then the router arbitration could use a weighted round-robin approach where the probability of picking a port is weighted by the number of downstream requests that are trying to make their way back to the host. While this approach would work well, it requires routers to maintain global knowledge of all other downstream memory cubes, which would be difficult to implement in practice.

Another alternative is to use age-based prioritization. If the router arbiter knows which requests are the oldest, then no request will be starved for very long. In particular, if all requests are injected into the MN at approximately the same time (subject to the injection bandwidth limit on the first MN link), then having each router favor the oldest requests will tend to cause all of these requests to be returned to the host ahead of any later requests. This approach also performs well to alleviate high-contention scenarios, but unfortunately it also has a significant implementation challenge. In particular, each message would have to have some form of timestamp embedded in it. However, typical flit header formats do not provide very many (if any) unused bits that could be used to store a timestamp.

We make the observation that in general, requests that are destined for cubes that are farther away (larger hop count) will have longer end-to-end latencies, and therefore are more likely to be among the oldest requests seeking arbitration at a router. Therefore, we propose a router arbitration scheme that uses a message's *distance* as a proxy for

its age. The distance can be derived directly from a message's header flit; the header encodes source and destination information, which combined with knowledge of the MN topology, can be translated into a hop count. This information can be pre-computed and stored in a very small hardware lookup table for the router arbitration logic. In this case, only about 8 bytes of data are needed so hardware cost is negligible. Using this information, our arbitration performs a weighted round-robin, where the input port selection is weighted by the distance to the memory cube from where the request came from.

4.3.2 The Skip-List Topology

The results from Section 4.2.5 showed that the tree topology consistently provides the best overall results. This in of itself is not very surprising as the average hop count increases at a much slower rate than the chain or ring (for increasing MN size). One observation about the tree topology is that the majority of its links, apart from those near the "root," tend to be under-utilized. This is because the total throughput of the MN is ultimately still limited by the single link connecting the MN back to the host memory port.

Because we are evaluating a CPU-centric MN, memory requests and responses are past the coherence ordering point in the system. In directory-based systems, reads to an address with an outstanding write must wait until a write acknowledgment is received at the directory. We can take advantage of this by utilizing "non-coherent" routing. This allows for several interesting network routing policies that are not possible when coherence is required. For example, we are allowed to reorder requests in the network, which allows for request priorities to be specified. When injecting to the network, the router output queue could skip over sending write requests at the head of the queue

in favor of read requests. Another possibility is to defer non-critical-path writes to be routed through lesser used non-minimal routes to free up link utilization for reads. We examine one such topology employing the last technique mentioned.

Taking inspiration from classic data structures, we propose organizing the memory cubes as a "Skip List" [81]. The traditional skip list consists of a regular linked list augmented with additional pointers that provide a path to bypass or short-cut around large sections of the linked list. The "length" of these bypass pointers (i.e., the number of nodes skipped) can be chosen such that the average number of hops to reach a node is logarithmic in the number of nodes in the entire list (same as a tree). Our implementation of the skip-list topology is similar to that of express cubes [82, 83, 84, 85] that proposes additional express channels to reduce network diameter. Figure 4.8 shows a 16-node MN organized as a skip list. There is a central sequential chain, analogous to a conventional linked list. The additional memory cube ports are used to implement the bypass or "skip" links. The set of highlighted (bold) links show how the farthest cube can be reached in only five hops (i.e., logarithmic in the number of cubes).

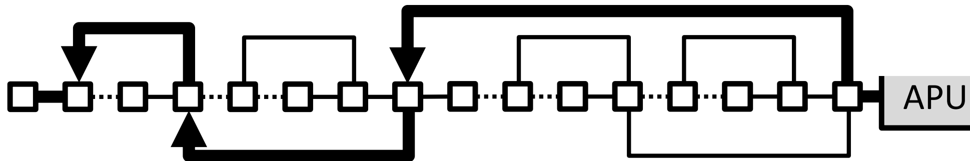

Figure 4.8: Example skip-list topology for 16 memory cubes. The bold path shows the route to the farthest memory cube, and the dashed lines indicate links that are only ever used by write requests.

Using the skip-list topology as is results in a topology with a lower branching factor than the baseline tree shown in Figure 4.3(d), which is a ternary (base-3) tree.[4] Some of the links (shown dashed) would not ever be used because they are not part of the shortest path to any of the cubes. What we instead propose is to differentiate the MN

[4]The skip-list ends up with an average branching factor somewhere between 2 and 3.

traffic based on read and write requests. For most workloads, read latency directly impacts performance, but writes can typically be handled off of a program's critical path so long as enough aggregate bandwidth is provided. Our skip-list topology routes read requests along the shortest paths to the destination memory cubes, fully utilizing the skip links where possible, different from the routing on express channels. All write requests are routed along the slower, central set of sequential "chain" links. This increases the latency per write request, but removes them from the more performance-sensitive skip links for reads. We effectively trade off some of a tree's fan out (by backing off from a full ternary tree) to shunt off lower-priority write traffic away from the more critical read requests.

As reads and writes to the same address can now take different paths through the MN, there exists the possibility that a read dependent on a write could reach the destination memory cube first, introducing the possibility of a consistency violation (i.e., the read receives a stale value from memory). It is therefore sufficient that the requirement of a directory stalling a read request until a write acknowledgment is received holds true for a skip-list MN.

4.3.3 The MetaCube

The last, most aggressive optimization is to condense the MN into as few cubes as possible. A similar concept to reduce average network latency was adopted by earlier works on bristled [86] and concentrated networks [87]. In our case, we propose leveraging advanced packaging technology to effectively build a "cube of memory cubes" that we call a MetaCube. The MetaCube can use a passive silicon interposer with an interface chip for the router or slightly smaller active interposer to integrate multiple memory cubes into the same package. Figure 4.3.3 shows a MetaCube with four DRAM-based

memory cubes and an interface chip stacked on a passive silicon interposer.[5]

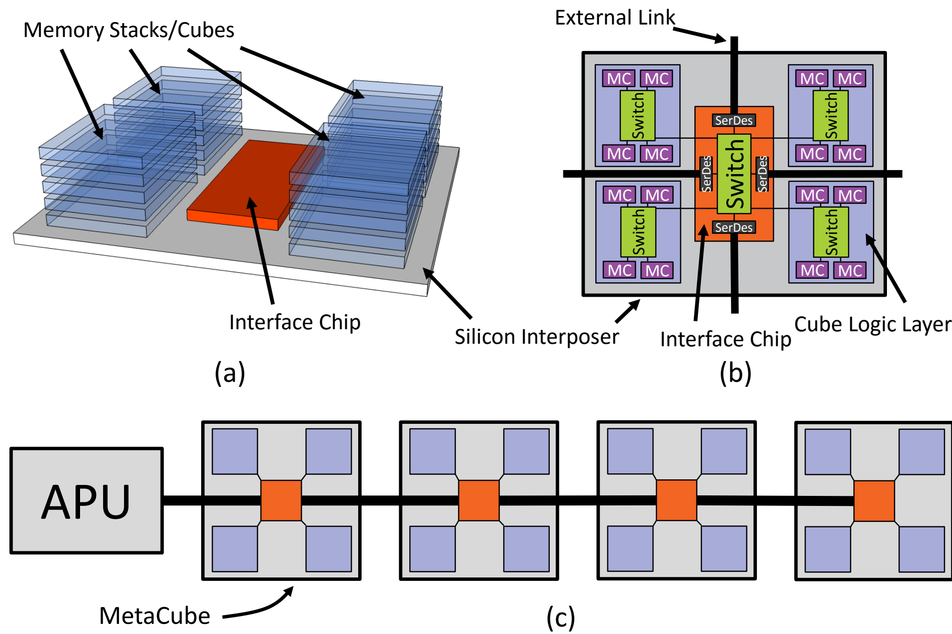

Figure 4.9: Example MetaCube organizations. Structural view of a 4-DRAM stack MetaCube, and (b) the block-diagram view of the logic layers.

The external links come from the package pins to the central interface chip in a similar manner to a normal memory cube. The router on the interface chip connects to the different memory stacks through direct point-to-point links across the interposer. Routers on the logic layer of the memory cubes then forward requests to the corresponding memory controllers. Figure 4.3.3(c) shows an example MetaCube MN consisting of four MetaCubes, each internally providing four DRAM memory cubes. This relieves the limitation of 4 ports per memory package by being able to design a high-radix router on an interposer. The maximum number of memory packages in a MetaCube is still limited by the maximum size of an interposer, however, so it is not always possible to create a single MetaCube with an all-to-all network on the interposer. Figure 4.3.3(c) shows an example providing 16 cubes' worth of capacity with a greatly reduced worst-case hop

[5]The same general topology can be implemented on an MCM substrate, with the primary tradeoff being the width of the links between the central interface chips and the individual cubes.

count to the furthest memory package.

The MetaCube is not without its own costs. In particular, additional costs must be paid for in terms of the silicon interposer or MCM and interface chips. The package size will also be larger than a conventional memory cube, which will also be more expensive. The power-per-package also increases, which could further increase package-related expenses if more sophisticated cooling solutions are required (although the reduced hop count will at least help reduce interconnect-related power consumption).

4.4 Evaluation

We explore the design space of MNs using a heterogeneous computing platform consisting of CPU and GPU resources executing a mix of scientific computing workload proxies and high-bandwidth GPGPU workloads. We choose these workloads as they provide a high range of memory footprint sizes and bandwidth usages that expose solvable issues with memory networks. Workloads chosen are taken from the AMD SDK [88] and Rodinia [89, 90] suites. Other large footprint workloads such as big data and cloud applications likely exhibit similar issues solvable by the techniques we describe.

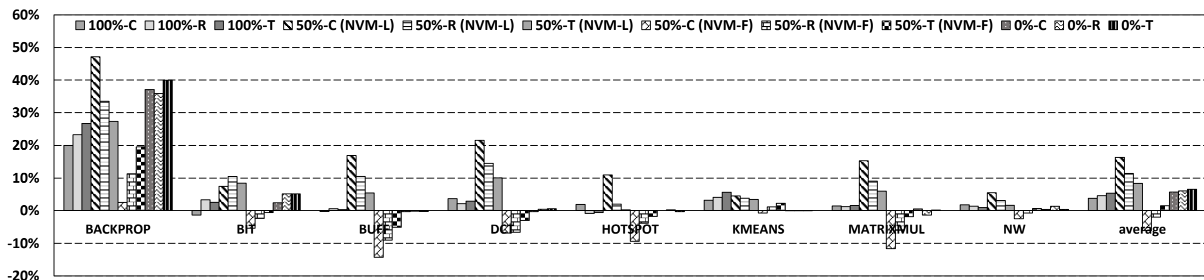

Figure 4.10: Speedup comparison using only distance-based arbitration technique. Only baseline topologies are compared to show impact of distance-based arbitration alone.

The GPU compute units are provisioned similarly to those described for AMD's GCN

Parameter	Value
Compute Units	32
Clock Speeds	1GHz (GPU), 3GHz (CPU)
GPU L1I$	32kB 8-way
GPU L1D$	16kB 16-way
GPU L2$	2MB 16-way
Memory Ports	8
Total Memory	2TB
Stack Capacity	16GB (DRAM), 64GB (NVM)
Banks / Stack	256
DRAM Timings (Major)	tRCD=12ns, tCL=6ns tRP=14ns, tRAS=33ns
NVM Timings (Major)	tRCD=40ns, tCL=10ns tWR=320ns, 500MHz
DRAM Read/Write NVM Read/Write	12 pJ/bit [59] 12 pJ/bit / 120 pJ/bit
Network Energy	5 pJ/bit/hop

Table 4.2: List of Parameters in Evaluated System.

pipelines [91]. Our system is configured as a 4x8 mesh of GCN pipelines, with a CPU mainly used to perform operations such as loading input data and dispatching kernels to the GPU. The key system configuration parameters are listed in Table 4.2.

Our evaluation implements MNs within the gem5 simulator as part of the Garnet network [92]. We utilize Garnet's default routing implementation to find shortest-path routes to each memory package and quadrant of the specific memory controller within the package. This network is modeled as high-speed SerDes links to pipeline packetized data through the MN. For the memory cubes, we evaluate assuming HBM-like memories stacked on a base die with a 4-link router, 4 quadrant design similar to HMC. This allows us to simulate the intra-cube memory timings (i.e., the turnaround time from when a request packet arrives at a cube to when a response packet is sent) using memory timing parameters from HBM datasheets.

The internal configuration of each memory cube distributes the memory banks evenly

across four quadrants, also similar to HMC. Requests that arrive to a link in the "wrong" quadrant are imposed with a 1ns latency to model intra-cube routing to the correct quadrant and back. Links connecting memory packages are assumed to be narrow 16-bit links running at a frequency of 15Gbps [59]. We assume an additional 2ns latency to account for time needed for serialization, scrambling, descrambling, and deserialization circuitry when traversing each SerDes link because these circuits have a non-zero latency even if they are pipelined. We experimented with modifying this parameter and found that 2ns made little difference compared to no latency, however larger values (e.g., 10ns) have a large impact on network latency.

Each memory package has a single link connection between another memory package or the host, with four links per memory package. Requests that must route through packages are assumed to be descrambled and deserialized in order to read packet routing information and traverse the internal package switch and therefore incur the additional 2ns latency per hop. Each memory package contains a centralized switch that uses round-robin arbitration to select between internal quadrants and external links to the MN.

Internally within the system processor, we assume there are eight ports connected to an external link of a memory cube. Addresses are mapped to a port from within the system processor distributed using address interleaving and hashing techniques for MN load balancing. Requests leave and return through the same port to simplify cache coherence. In order to provide some level of memory parallelism, and in order to stress TB-sized memory in simulation within a reasonable turnaround time, addresses are mapped to ports at a 256 byte granularity interleaving. This size was chosen empirically based on a sweep of various mapping sizes. In the presence of spatial locality, larger mapping granularities (e.g., 1024 bytes) caused increases in network latency large enough for performance degradation. The smallest size, 64 bytes, caused reduction in row-buffer hits

within the memory cubes.

We estimate the dynamic energy used in the network using average picojoule-per-bit numbers to provide a fairer comparison of NVM and DRAM energies. We assume that network traffic incurs an energy cost at each hop in the network, and we break up NVM values into read and write energies. For network energy, we assume 5pJ/bit for each hop. We assume 12pJ/bit for DRAM [59] and 12pJ/bit for reads 120pJ/bit (10x read energy) for writes with a PCM-based NVM cube. These results do not include static energy, as the static power savings is highly dependent on the underlying process management assumptions (e.g., race-to-idle), although of course NVM provides lower (near-zero) standby power. Note, however, that the memory cube SerDes links will likely still consume non-trivial amounts of power even during "idle" periods because of performance concerns related to the long latencies required to retrain the SerDes links.

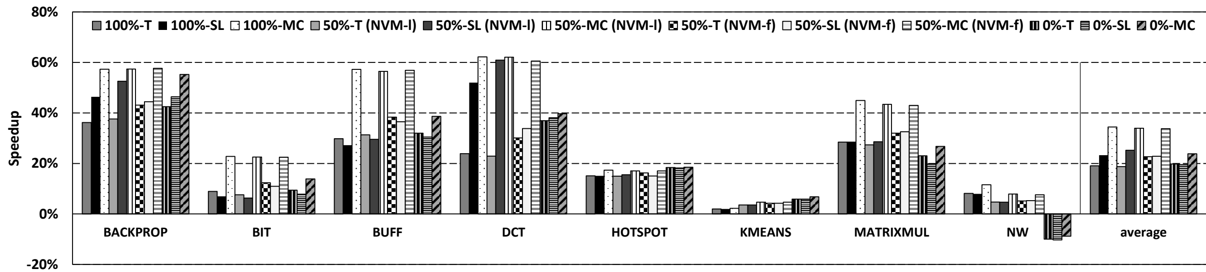

Figure 4.11: Comparison of Tree topology compared to our SkipList- and MetaCube-based topologies. The default localized round-robin arbitration is used in these results. All results normalized to 100% Chain.

4.4.1 Distance-Based Arbitration

We compare our distance-based arbitration approach against the baseline localized round-robin arbitration scheme in each memory cube, with the speedup over round-robin shown in Figure 4.10. In order to highlight the impact of the arbitration technique alone, we only compare against the original baseline topologies shown in Figure 4.10.

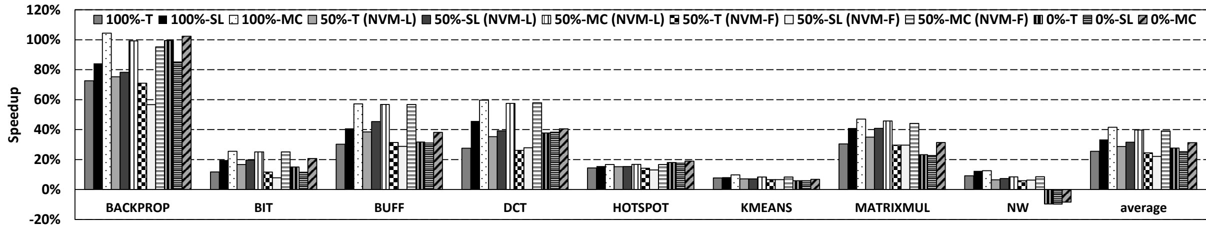

Figure 4.12: Overall speedup with all concepts proposed in this work applied Tree topology, SkipList, and MetaCube using distance-based arbitration. All results normalized to 100% Chain.

Originally, the study yielded mixed results. In particular, 50% Chain, Ring, and Tree NVM-L topologies show opposite trends compared to 50% Chain, Ring, and Tree NVM-F topologies. One insight from this is that while we attempt to use distance as a proxy for age, the NVM-F responses will typically be older due to the array latency of NVM. On the other hand, when NVM is placed further from the host processor, the responses returning upstream are nearly guaranteed to be the oldest responses and are much more likely to be selected. In the cases of all-DRAM and all-NVM the average results are very similar. Due to the memory packages having the same latency, the performance results only vary on the distribution of the requests latencies within the cubes (i.e., refresh occurring or unbalanced queuing latency within a specific cube).

In most topologies our distance-based arbitration provides benefits preventing local memory controllers from taking too much weight over request and response packets passing through the memory cube as well as reducing the discrepancies between to- and from-memory latency. Although performance generally increases when using distance-based arbitration, we observed the technique can have adverse effects on topologies such as skip-list and in networks with different mixes of technologies within memory cubes.

In a naive implementation applied to a skip-list, older write requests and responses can be selected over younger read requests and responses due to the longer write path in the network. Similarly, as evidenced by Figure 4.10, the technique can have detrimental

impact on MNs with NVM placed first, as older responses are predicted to be from the incoming external link, when requests serviced by NVM are actually older.

To combat these issues, we augment our original implementation discussed in Section 4.10 to be aware of both the network topology as well as the memory technology type of the response's source. This allows our distance-based arbitration to better estimate the age of a response. This implementation is applied to our final results shown in Figure 4.12.

4.4.2 Skip-List and MetaCube Topologies

Here we compare the tree against the Skip-List and MetaCube topologies by themselves without the usage of distance-based arbitration. The skip-list is effectively a slightly lower arity tree (compared to the default ternary tree), with links to provide extra paths for lower priority requests. The MetaCube is a cluster-like topology that similarly aims to reduce hop distance without breaking the constraints of the memory cube package. Results of both topologies are shown in Figure 4.11 and Figure 4.12

Because the average hop count of skip-list is similar to that of a tree topology, we expect it will perform similarly to that of a tree. On average, the skip-list topology provided only slight benefit over the corresponding tree topology for each DRAM:NVM ratio. For most workloads, contention of read requests due to writes was not great enough to cause the additional write path to make a large impact. The largest average benefit of the skip-list topology is exhibited in the NVM-L topologies. In these configurations, writes that may otherwise fill the queues of local memory vaults are pushed downstream through the network, limiting the amount of performance critical reads blocked in the network at a memory cube's external input port. This phenomenon is not exhibited in most other configurations.

One potential problem with a skip-list topology is always routing writes through the longer path. In some cases, it is beneficial to allow writes to use the skip-paths in presence of very few reads. This allows workloads with either large bursts of write requests or heavy usage of read-modify-writes, for example, to regain some of the performance loss compared to tree topologies. This concept is integrated with our final results in Section 4.4.3.

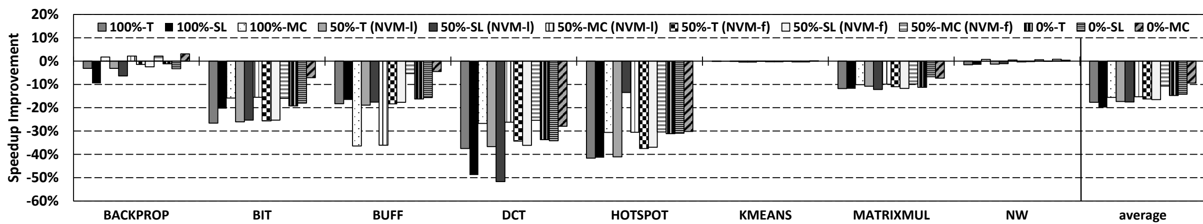

Figure 4.13: Average total memory access latency when the host processor has four or eight (baseline) port with a fixed 2TB of memory capacity. Note that the 4-ported system therefore has twice as many packages per port.

Next, a MetaCube topology has the potential to provide an even smaller hop count than either tree or skip-list. This allows for the largest speedup potential of any other given topology across all workloads as the cost of additional manufacturing. Because most of the memory round-trip latency consisted of to-memory latency in the majority of workloads, the low hop count of the MetaCube provides significant speedup improvements.

Figure 4.11 demonstrates that MetaCubes outperform other types of topology in each simulated run. Different from the previous topologies, the MetaCube is the first topology in which the 100% configurations provided better performance than all topology mixes containing NVM. This occurs because the hop count of a MetaCube configuration is near the threshold where memory array latency, which is higher in NVM, begins to become dominant over both to- and from-memory latency.

However, although Figure 4.11 indicates that all-DRAM topologies provide the best

performance overall, static power savings from incorporating NVM into the memory system are beneficial in terms of power while yielding nearly the same performance as in the all DRAM case. From our results less than 1% performance degradation was incurred compared to all DRAM MetaCubes.

4.4.3 Combining All Techniques

In this work so far we have evaluated a distance-based arbitration technique, skip-list topology, and MetaCube-based topology. Here we evaluate the combinations of each proposed topology while using distance-based arbitration. Using insights gained from the evaluations throughout this section, we were able to further improve each of the topologies.

In particular, several additions were made to the distance-based arbitration to provide better average performance to topologies. First, our table used to calculate distance was augmented with knowledge of request type priority, allowing writes to be further delayed. This was especially important in the skip-list topology. Next, the same table was augment with knowledge of the memory cube types at each source/destination node allowing it to further weight requests coming from, for example, NVM. Last, we were able to monitor periods of high write traffic at the system port with some hysteresis in order to allow writes to route through shorter network paths in the presence of a skip-list configuration.

The values used to calculate approximate age using distance as a proxy were determined empirically using both average network hop latency and average memory access latency for each cube technology type. Figure 4.12 shows the results of this tuning. Compared to the previous results in Figure 4.11 we were able to obtain much better average speedup for skip-list in not only 100% MNs, but in 50% NVM-L networks as well (e.g., in BIT and BUFF). This is primarily an artifact of distance-based arbitration's ability

to push slower NVM writes further downstream and utilize shorter skip-paths for writes.

As demonstrated in these final results and across many of the previous results, the BACKPROP workload saw the most benefit from each of our experiments. This workload, compared to the others, was by far the most write intensive workload in our suite. In contrast, KMEANS was the most read intensive workload in our suite. However overall each workload in isolation exhibited a respectable increase in performance by employing the methods in this work.

4.5 Analysis

In our evaluations so far, we have seen that MNs are highly sensitive to various configuration parameters, topologies, and workloads. In order to get a better handle of the different parameters that can cause a large impact, we vary the size of the MN in two dimensions. Until now we have simulated a system with 2TB of memory. This is approximately the inflection point where memory cubes begin to overcome the limitations of DDR as discussed in Section 4.2.1. We reduce the capacity to 1TB in order to decrease the "length" of the MN. Also discussed in Section 4.2.1 is the limitation on the number of pins available to the host processor package. We study decreasing the number of system ports to decrease the "height" of the MN. Here we examine both of these "dimensions" in order to construct a short sensitivity study.

4.5.1 Number of System Ports

Although eight ports is a reasonable estimate for the available number of pins the host processor could allocate to memory connections[6], some pin-constrained systems may only

[6]While current systems only have on the order of four DDR channels per package, recall that HMC-style interfaces are significantly narrower and so eight ports is very reasonable.

support fewer ports. Here we investigate the impact of allocating fewer pins to memory by having only four system interfaces to the MNs.

When the number of ports is reduced, the number of memory cube packages per port must be increased in order to provide the same amount of memory capacity. This will generally lead to larger networks, a higher number of average hops, and therefore a performance degradation. Figure 4.13 shows the performance loss when decreasing the number of system ports from eight to four. In addition to higher network latency, reducing the number of ports also requires more requests to contend for the same output ports on the processor.

For topologies that grow in a linear fashion (e.g., chain and ring), hop counts double each time the number of ports is reduced by half. This causes the fastest rise in MN latency when compared to other topologies. In the case of NVM configurations, 50% NVM-L has the highest degradation. This is again caused by hop count as 50% of requests must be routed to the furthest hop. All-NVM topologies exhibit the lowest degradation on average because these configurations are bound more by memory latency than any other configuration.

Across some workloads, (e.g., BACKPROP and NW) there is a negligible increase (1-3%) in the performance of the MetaCube topologies while the remaining topologies have degraded performance. In these cases the MetaCube provides similar average hop count as the eight-port system with only a marginal increase in latency. The difference in performance for MetaCubes is largely dependent on differences in internal routing within the host processor compared to the eight-port case. Furthermore, workloads such as KMEANS and NW show very little difference in performance.

4.5.2 System Capacity

The total memory capacity of a system has a large impact on network performance and the overall design because the number of memory cubes is dependent upon it. In systems where one specific topology performed the best, smaller-sized systems may have different characteristics. We reduce the capacity of the system by half while keeping the number of memory cubes the same. For the purpose of this study, we assume the workload's memory footprint is just under the total memory capacity such that it *needs* to access all cubes.

Figure 4.14 shows the speedup of a 1TB system over the 2TB baseline. While the reduction in the number of ports intuitively caused the system performance to decrease on average, decreasing the capacity yielded mixed results for each configuration. In the case of 100% topologies, the network latency was reduced while the memory latency remained relatively constant despite each cube servicing twice as many requests. For 50%, the results surprisingly showed a decrease in performance. While the network latency increased by roughly the same amount as the 100% topologies, there was significant increase in the memory's core latency due to the reduction in memory level parallelism and increased queuing latency in the memory cubes. The largest drop was in the 0% case, as expected, which exhibits performance degradation similar to 50% topologies. This behavior is similar to the conclusion in Section 4.4.2 where there is an inflection point where memory core latency begins to dominate to- and from-network latency.

4.5.3 Energy Analysis

An energy analysis of the MN paints a different picture of the optimal MN. Because NVMs, especially PCM, typically have higher write energy, write-heavy applications consume a large portion of the total energy in the MN. Figure 4.15 shows the breakdown

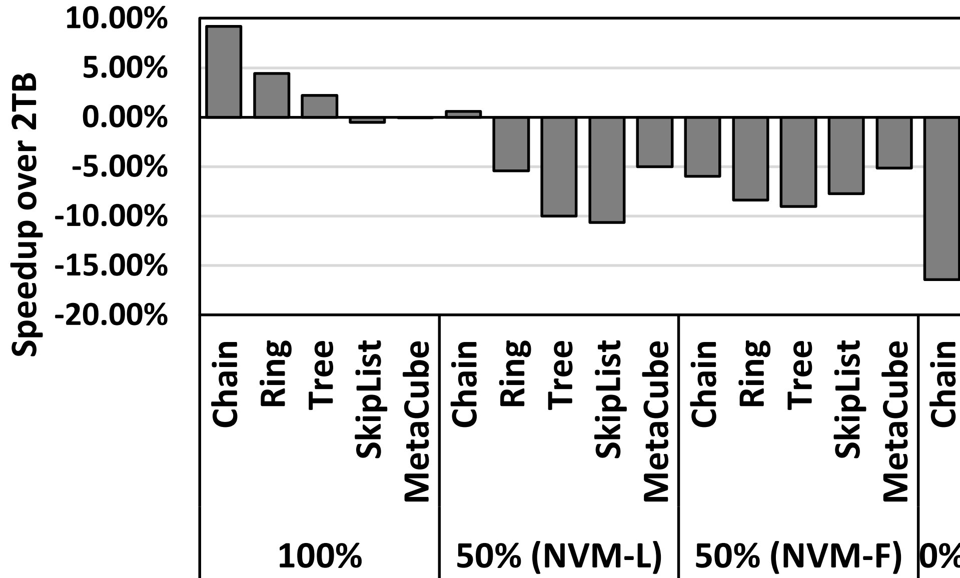

Figure 4.14: Average system speedup when moving from 2TB to 1TB.

of energy usage for network/data movement energy and read/write or memory access energy within the memory package. This breakdown is the average energy utilization of all workloads for each type of MN.

The figure shows there is a significant tradeoff between network energy and memory access energy. For larger networks, the amount of energy required to move data increases linearly with the hop count. This means that DRAM network energy is the largest fraction of total energy in the 100% MNs. On the other end of the spectrum, 0%-C MN reduces network energy by nearly 3x, but the increase in write energy is enough to push the total energy above that of the baseline 100%-C MN. In between these two boundaries, there is more balanced usage in terms of energy.

Topology-wise the tree topology general uses the least amount of energy. Although the skip-list topology emulates a tree-like topology and has a lower average hop count for reads, the additional energy required to route write requests through non-optimal paths increases the network energy by enough to consume more energy than a tree-topology. However, these energy values are highly sensitive to the write energy values used for

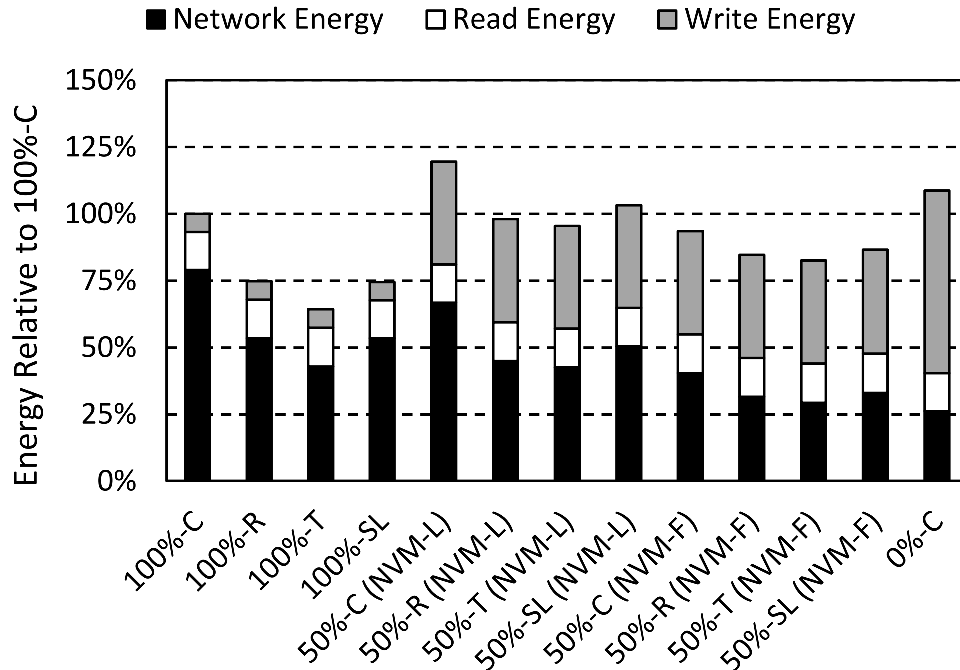

Figure 4.15: Breakdown of network (transport) energy and read/write (memory access) energy normalized to a 100%-C MN.

NVMs. Over time if NVM write energy is reduced, NVM-based MNs may be able to overtake DRAM in both energy usages and performance.

4.6 Related Work

In addition to the work outlined in Section 4.2.3, Kim et al. also studied the design of MNs in the context of tying together multiple memory cubes with processing-in-memory (PIM) capabilities [93]. As any memory cube/PIM could potentially reference memory locations in any other cube, a full any-to-any MN would be desirable to avoid having to route such requests back through the host processor(s).

Beyond Micron's seminal Hybrid Memory Cube work [59], others have also advocated for moving future memory systems to adopt more abstract, asynchronous, packet-based interfaces. Resnick and Ignatowski [60] discuss some of the motivations and possible

directions for such future interfaces, particularly packetized abstract interfaces. In our work, we assume a packetized memory system over a more abstract interface compared to that of DRAM, which is in line with the future systems they project. Roberts et al. [61] propose a new memory interface (NMI) that supports point-to-point networks of NMI nodes, nodes with diverse technologies, compatibility for heterogeneous computing (supporting the heterogeneous systems architecture (HSA) specifically), fine-grain logical memory region management, etc. Our proposed architecture can make use of such an interface, as we investigate incorporating different memory technologies, and connect them via point-to-point links.

Cooper-Balis et al. [94] target the problem related to capacity/bandwidth trade-off in traditional DDR-based memory systems by proposing a buffer-on-board design. Although we target a similar problem, our approach is different as we employ MNs to achieve both high bandwidth and high capacity for the system. Zhan et al. [95] demonstrate employing MNs to support in-memory computing and provide latency and power optimizations to their proposed unified MN, targeting an all-to-all connected MN. In contrast, our work does not consider a MN with full connectivity due to reasons explained earlier, and therefore provide non-complementary techniques to tackle the MN latency issues. Azarkhish et al. [96] propose a network for the logic base dies of HMCs. They make use of HMC's base die for employing processing-in-memory (PIM), and their proposed interconnect can provide additional bandwidth to the PIM. While they consider the interconnect design within a cube, we consider the interconnect design for the overall MN. Furthermore, there are multiple efforts that employ near-data computing in the logic base die of memory cubes that are interconnected together [97, 98, 99]. PIM and NDC studies are therefore orthogonal to MNs and both approaches can be complementary. On another front, Akgun et al. [100] propose employing MNs in the silicon interposer of 2.5D integration to enhance the scalability and performance of large capacity designs.

Although we propose using silicon interposers for the MetaCube implementation, we only integrate four memory cubes on the interposer and simply use an interface chip to provide direct point-to-point links to achieve low latency. If more memory cubes are to be integrated on the silicon interposer, alternatively, one can decide to implement MNs in lieu of a costly high-radix switch. Rosenfeld [101] analyzes how multiple HMCs perform when they are connected using chain or ring topologies, with various routing algorithms. In our work, we consider other topologies such as skip lists. We also identify performance bottlenecks for such systems, propose arbitration techniques to improve overall MN throughput, and incorporate non-volatile memories to achieve better balance between interconnect and memory array latencies.

4.7 Conclusions

In this work, we examined the design of MNs and analyzed sources of performance degradation. Based on our analysis, we proposed several mechanisms and MN organizations to address the performance issues. We targeted globally-fair arbitration through distance-based arbitration that estimates the age of a request and provided several insights behind the implementation of such a scheme. We introduced a non-coherent skip-list-based topology able to provide performance matching or exceeding, on average, that of the lowest hop count tree topology. To reduce MN size/diameter, we considered two approaches: the first uses the higher density of NVM and the second leverages advanced packaging technologies to create very high capacity MetaCubes, both of which increase per-package capacity and reduce MN size. These provide effective techniques to improve the performance of future MNs to support modern servers, supercomputers, and high-end workstations with terabyte-scale memory capacities.

While this work focused on MNs for a particular family of system organizations

(specifically where each of the host processor's memory ports covers a disjoint portion of the physical address space), the insights learned and techniques proposed should extend to other types of MNs. For example, distributed arbitration to increase fairness/throughput and read-write differentiated routing could both be promising for multi-GPU systems [69] or networks of PIMs [93], optimizations for future non-coherent topologies, and the ability to cluster memory network-like networks into MetaCube-like devices.

Chapter 5

NoC IP: NoC Chiplet for Scalable Parallel Computing Architecture

The goal of this chapter is to create the logical and physical standard of Network-on-Chip (NoC) chiplets for intellectual property (IP) reuse. NoC chiplets are envisioned to be used for providing an on-package interconnection network between any chiplets. In order to realize seamless integration and communication between chiplets, the NoC chiplet has to ensure the implementation of common chiplet interfaces. Another requirement for the NoC chiplet is to support both streaming and transactional communication streams. In this chapter, we present two main aspects of the NoC chiplet design, interface integration and router design.

For the NoC chiplet design, our primary focus is to meet the needs for low-latency communication of streaming data. To this end, a physical serial interface standard is proposed. Routing solutions and their use for communication of streaming data as well as transactional data over Micron's CPI protocol are also included.

5.1 Introduction

Modular integration of IP blocks, also called chiplets, are being considered for cost-efficient custom designs. Heterogeneous chiplet-based systems are enabled by the innovations in interposer-based 2.5D integration technology. Modular integration of chiplets on top of an interposer allows heterogeneity, scalability, and flexibility, given the superior properties of interposers over more traditional package technologies. Cost-efficiency of the chiplet-based systems comes from the fact that smaller chiplets having higher yield. Therefore, for custom designs where the manufacturing volume is expected to be low, can achieve cost benefits through modular use of chiplets that implement common functionalities, also known as IP reuse.

Common goals of IP reuse include the following properties, which we also take into consideration in this work. Scalability is a key consideration, however chiplet interface efforts so far consider point-to-point connection between chiplets. Through its modularity, IP reuse aims to improve chip turnaround time and design flexibility. Furthermore, as IP reuse is a promising effort for custom designs, some of the applications that are interested in IP reuse may require transactional as well as streaming data flow for real-time use cases, as outlined by DARPA's CHIPS program [102].

Although modular chiplet-based systems have their advantages in terms of cost and flexibility, modularity also introduces overheads. Therefore, one of the main goals for heterogeneous chiplet-based systems is interconnecting chiplets with close-to-monolithic performance. In this chapter, we tackle this issue by proposing a network-on-chip IP chiplet for IP reuse. The goals for our NoC IP design are listed as follows:

- **On-package network between chiplets.** We propose a NoC chiplet design and describe the router design and interface integration. We propose forming a network of NoC chiplets for improving scalability.

- **Logical/physical standard of NoC chiplets.** We adopt chiplet interface standards to ensure compatibility with the chiplet ecosystem. For physical standard we consider Intel's AIB, and for logical standard we consider Micron's CPI standards.
- **Support for transactional and streaming dataflow.** We support the two dataflow cases with our hybrid packet- and circuit-switched router design.

In this chapter, we first introduce chiplet interface standards that we conform to (Section 5.2) and to follow we introduce the details of our proposed NoC IP router (Section 5.3). We conclude the chapter by listing the use cases of our NoC IP proposal as part of the chiplet ecosystem.

5.2 Interface Standards

This section presents the interface standards proposed to implement in the NoC chiplets for seamless communication between chiplets. Interface standard candidates for, chiplet based systems include Intel's AIB, Micron's CPI, MoSys's GCI. Next, we present a NoC-to-NoC phyisal interface standard for low latency and high throughput long range communication. The use of most of the topology choices, addressing and routing algorithms for the core NoC router described in Section 5.3, is independent of the physical interface specifics.

5.2.1 Intel – AIB (Advanced Interface Bus)

Intel introduced a physical parallel interface standard called Advanced Interface Bus (AIB) [103, 104], initially proposed for the 55 μm pitch. The AIB standard is to be used to interface the NoC chiplet to endpoint chiplets, enabling communication between devices

Interface standard	Max. # of data lines (/mm)	# of wires (/mm)	Voltage (V)	Max. clock freq. (MHz)	Bit error rate	Max. data rate/lane (Gbps)	Max. BW density (Gbps /mm)	Energy (pJ /bit)
AIB LR Legacy	128	250	≤ 0.9	1000	E-23	2	500	1
AIB LR GEN1	480	500	≤ 0.9	1000	E-23	2	1000	1
AIB LR GEN2	900	1000	≤ 0.9	1000	E-23	2	2000	0.6
AIB LR GEN3	900	1000	0.7	2000	E-23	4	4000	0.4
AIB SR GEN1	1920	2000	≤ 0.9	2000	E-23	4	8000	0.2
AIB SR GEN2	2880	3000	0.7	2000	E-23	4	12000	0.12

Table 5.1: AIB roadmap.

from different vendors. The AIB interface addresses communication between chiplets via an interposer within the same package, and not chiplet interposer-to-external reference.

There are six different AIB generations with bandwidth specifications that increase from one generation to another. The increase is attributed to factors such as lower bump pitch, lower package pitch, improved data rate and reduced power consumption. Specifications of the different generations are summarized in Table 5.1.

Other than the AIB-Legacy generation, all other generations have two configurations: (i) a base configuration with no DLL (Delay-locked-loop) or DCC (Duty-cycle correction) support, having clock being forwarded with data, maximum data rates of 800 Mbps in DDR (double data-rate) mode and 1 Gbps in SDR (single data-rate) mode, (ii) a base-plus configuration with DLL and DCC support, additional link calibration process and higher data rates.

At the physical layer, AIB is organized into channels, comprising of basic I/O cells, where each I/O cell can be configured as an input or output buffer and operate in either

SDR, DDR or asynchronous bypass modes. Further, operating modes to configure input/output drivers to be ON or OFF are available. DCC and DLL blocks are to be used for over 400 MHz DDR data transfer. I/O cells are partitioned by an AIB adapter into different groups based on their functionality.

AIB data transfer is based on a simple clock forwarding architecture where the clock is forwarded along with the transferred data, with DCC at the transmitting end and DLL at the receiving end. Implementation details of the AIB I/O cell, clock buffers, clock distribution, DLL and DCC circuits are detailed in AIB standard.

The AIB adapter supports three kinds of transfer mechanisms categorized as:

- Synchronous data transfer
- Variable latency synchronous data transfer using serial shift register chains to multiplex multiple side band control signals through a single connection
- Direct asynchronous signal transfer

For the synchronous data transfer mode, FIFOs are used to provide safe clock domain transfers which can also be implemented as elastic FIFOs to overcome frequency offsets and perturbations on clocks. Two clock and FIFO configurations are shown in Figure 5.1 and Figure 5.2. In option 1, the slave chiplet (chiplet B) does not have its own system clock and relies on its master (chiplet A) for a clock. In option 2, both chiplets are referenced to the same external clock source.

Other features of the AIB standard includes the AIBAUX module used to hold redundancy information in a non-volatile memory. This redundancy information is used at power up to repair applicable I/O microbumps. AIB boundary scan operations are also supported using JTAG instructions. The AIB standard embodies more details on power on and reset sequences as well as timing diagrams and state machines for AIB operation.

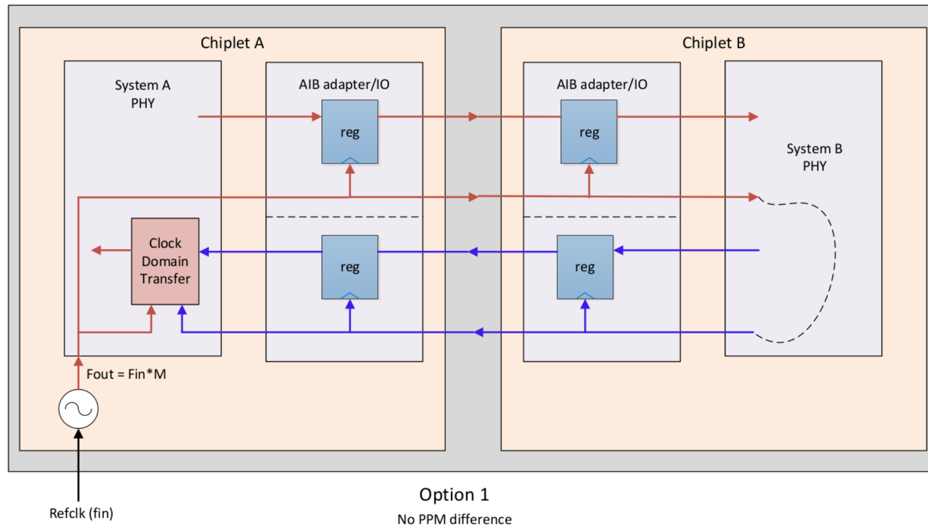

Figure 5.1: Clock and FIFO configuration (option 1).

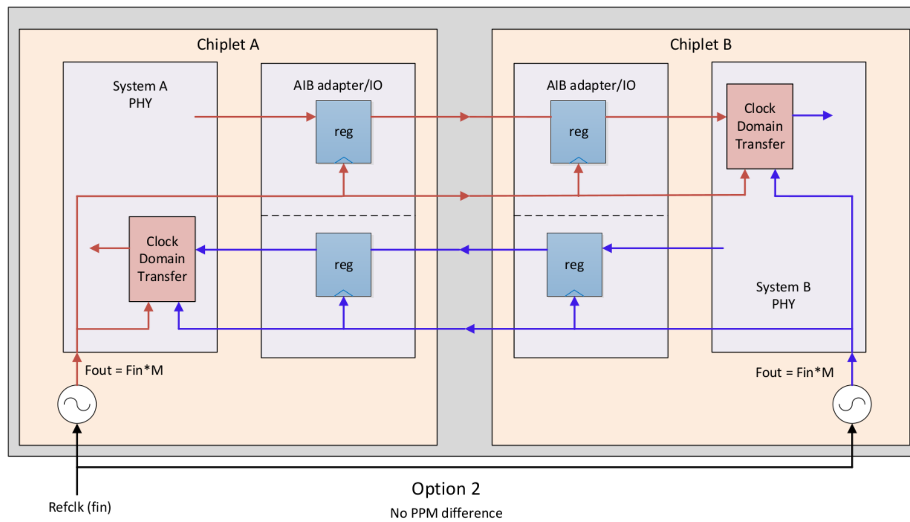

Figure 5.2: Clock and FIFO configuration (option 2).

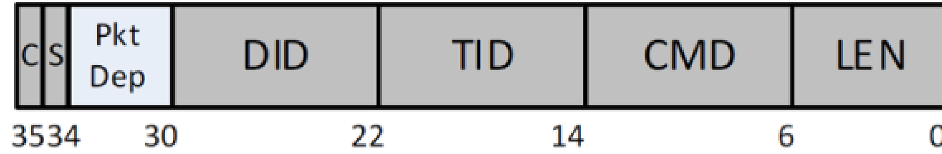

Figure 5.3: CPI header flit.

LEN [5:0] Packet Length Encoding	Defines the number of flits to route in this packet. Also can be used to determine if it is an idle flit.
CMD [13:6] Packet Command	Provides assigned virtual channel info and special cases. (eg. extended command)
DID [29:22] Destination Endpoint ID	Defines 8-bit destination ID for endpoint addressing. Can inherently define the topology (eg. a 2D mesh with 4-bit vertical ID and 4-bit horizontal ID) and be used to determine routing.
S [34] Sequential Packet	When set, the packet is a part of indivisible multi-packet transfer.
C [35] Credit Present	Credit bit is used in credit-based flow control scheme for avoiding network contention. When bit is set, credit bits are present in 3rd – n^{th} flits.

Table 5.2: CPI header flit fields.

5.2.2 Micron – CPI (Common Protocol Interface)

Micron proposed a logical interface standard called common protocol interface (CPI), targeting the transactional communication between chiplets over interposer [105]. In our NoC chiplet, we implement CPI for the transactional logical communication standard, mainly used for routing decisions. In most cases, only the first flit of the CPI packet, shown in Figure 5.3, is required to handle routing.

Table 5.2 explains how the specific fields within CPI header flit is used by NoC chiplet for routing purposes. The information header flit contains for routing include: packet length, packet command, destination endpoint ID. The flow control scheme uses the sequential and credit bits.

Figure 5.4 demonstrates the routing flow in NoC chiplet using CPI standard. Upon

receipt, a flit is identified as a header flit or a body flit via the router's packet length counter. If the flit is a header flit, the routing information is extracted with respect to the fields as explained above. The packet length counter is set so the body flits would take the same path. If the flit is a body flit, it is routed along the path the header flit set, and upon transmission, the packet length counter is decremented and the credit information in the downstream counter has to be updated. The flow control scheme checks whether there are enough credits in the downstream router's input buffer to receive this packet. When there are enough credits to do so, the packet can be transmitted downstream, as determined by the flow-control scheme of CPI.

Optionally, NoC will report ECC errors to external interface, but will not act on them. Depending on the specific application requirement, error in ECC computation or accumulated error rate may reset the link by raising the ERROR line. This decision is made outside of the NoC.

Below is a list of opportunities using CPI protocol on NoC IP that we have identified:

- *Inefficiency of Idle Packets:* CPI assumes idle packets are sent on CPI if no other packets are pending. We find this approach to be inefficient in terms of power and performance. Instead, we propose reducing the transmission of idle packets. One proposal we made is link power-gating instead of transmitting idle packets, with the exception of idle packets that carry credit information. Another approach is to drop idle packets in NoC instead of transmitting, again with the exception of idle packets carrying credit information.

- *Prioritizing Virtual Channels:* CPI allows up to eight virtual channels (VCs) while requiring only two, VC0 for requests and VC1 for replies, for deadlock avoidance purposes. Other ways to utilize the virtual channels inside NoC chiplet is by using for performance or quality of service prioritization. VC[2-3] are reserved for high

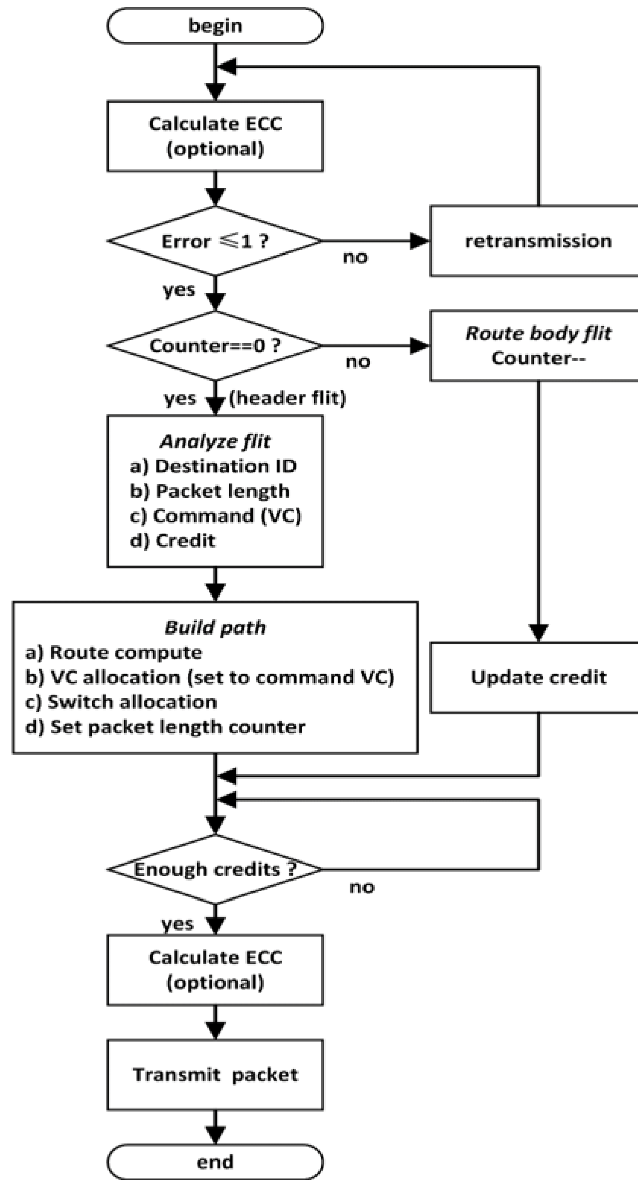

Figure 5.4: Routing flow in NoC chiplet using CPI standard.

priority packets for memory access, however the specific uses of VC[4-7] are not mandated by CPI specification, so NoC architecture proposes to further leverage these available virtual channels for performance or quality of service prioritization.

- *Link Layer Configurations:* As CPI has multiple link layer configurations, NoC chiplet can explore the trade-off between narrower data lanes and faster link clock frequencies on the physical link layer.
- *Topology and Routing:* Since CPI specification does not mandate a specific topology or routing, the NoC chiplet can explore and implement various regular/irregular topologies or dynamic/static routing algorithms depending on the use case.
- *Reliability:* Reliability mechanisms such as end-to-end or per-hop ECC schemes may be explored. Another reliability aspect that can be implemented in the NoC network is fault tolerant routing that routes around failed or power gated links or chiplets.
- *Circuit-Switched Layer:* As will be mentioned later in detail, the circuit-switched layer of our two-layer design can do much more than just routing streaming data. This layer can be chosen for routing very large transactional packets or provide guaranteed services for quality-of-service prioritization.

5.2.3 GigaChips Interface (GCI) Protocol

The GCI protocol by MoSys [106, 107] is proposed to be adopted on top of CPI for error detection and recovery for the NoC-NoC high speed serial interface. GCI is partitioned into three layers with a four-layer stack as follows:

- *Transactional Layer:* This layer is not defined by GCI. We plan to use the CPI protocol for the transaction layer.

- *Data Link Layer:* This layer appends Cyclic Redundancy Check (CRC) code to data words and supports two kinds of error recovery modes:
  - Automatic Error Recovery (AER): This mode provides a replay mechanism for losslessly recovering from errors. Frames are retransmitted when the error threshold crosses a predefined value.
  - Host Error Recover (HER): This mode simply discards the bad frames.
- *Physical Coding Sublayer (PCS):* This layer scrambles data with a PRBS sequence, providing sufficient transition density and DC balance without the overhead of 8b/10b encoding.
- *Physical Medium Attachment Layer (PMA):* This layer is meant to perform electrical and timing functions, equalization, clock and data recovery, serialization and deserialization.

Our proposal is to use the AIB and NoC-to-NoC interface for performing all functions associated with the PMA layer. GCI's data link and PCS layers are proposed for performing functions such as line coding, character alignment, computation of CRC codes and error recovery.

5.2.4 NoC-to-NoC Physical Interface Proposal

In this section, we present our proposal on the physical interface standard between interconnected NoC IPs. The goal of this interface is to achieve low latency and high throughput while providing long range communication for distances above 10mm. The physical interface is to support both streaming and transactional communication.

For the NoC-NoC interface to be able to support very large bandwidths (100-400 Gbps) due to incoming data from a massively parallel interface like AIB, we need a bus

Metal Width (μm)	Loss (dB/mm)
2	0.246
5	0.211
10	0.163
15	0.128
20	0.093

Table 5.3: Channel characteristics.

of high speed serial links. The motivation for using serial links stems from the fact that it is very hard to move data on a large number of parallel wires for distances longer than 0.3 mm, without the possibility of losing synchronization due to wire-length mismatches or encountering errors due to crosstalk. We need fewer wires running at a higher per wire bandwidth, and hence we need a serial interface. Our analysis also showed that having an I/O pitch of 10 μm is extremely beneficial for supporting large bandwidths.

In order to estimate the highest possible data-rates for a high-speed link, a study was carried to characterize channel properties of 10x thick wires in 130 nm technology. Loss figures for different metal widths are summarized in Table 5.3.

The conclusion of this study was that it is possible to transmit over distances as long as 15 mm without considerable amplitude loss and reasonable real-estate due to wires, for data rates as high as 20 Gbps. We propose using n parallel links with serial data at 20 Gbps, where the choice of n is based on total system bandwidth requirements. Practical circuit design constraints, such as fan-in and fan-out of gates, limit the value of n to 6 wires. We propose a three-level differential encoding with complete common-mode cancellation between group of 6 wires for intersymbol interference (ISI) and crosstalk mitigation [108]. For system bandwidths larger than what can be supported on 6 wires, multiple shielded groups of 6 wires can be used. Three-level encoding also allows code families where data can be encoded in signal transitions, enabling source synchronous links that exchange data without transmission of clock on a separate wire and also do

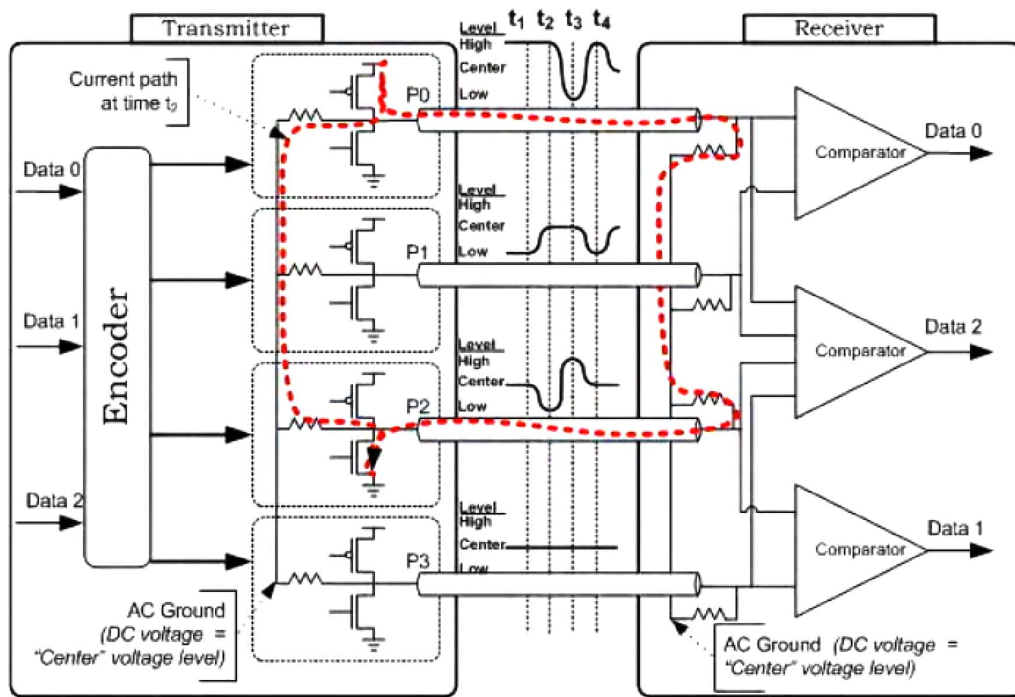

Figure 5.5: 3-level encoding transmitter/receiver architecture for 3 bits on 4 wires [108].

not need PLLs or DLLs for data synchronization.

Circuit implementations for 3-level encoding with 3 bits on 4 wires are shown in Figure 5.5. It uses three levels: Low (L), High (H) and Centre (C) such that for each symbol, the common mode voltage is 0. The truth table for this encoding is shown in Figure 5.6. Due to the zero common-mode level of this encoding, crosstalk and ISI are minimized, and a high noise margin can be achieved. A similar encoding can also be used to transmit 5 bits over 6 wires, resulting in a larger density of code transmission. For 6 wires, if we use 48 of the 90 possible symbols, circuit complexity doesn't grow drastically, and decoding of data can be done with only 6 comparators.

In summary, benefits of the proposed NoC-NoC serial bus interface are as follows:

- *Long-distance communication:* Data can be transmitted for distances up to 15 mm.

	Data to be transmitted (Data[0:2])							
Pins	000	001	010	011	100	101	110	111
P0	H	C	H	C	C	L	C	L
P1	C	L	C	L	H	C	H	C
P2	C	H	L	C	C	H	L	C
P3	L	C	C	H	L	C	C	H
	*L = Low, C = Center, H=High							

Figure 5.6: 3-level encoding for 3 bits on 4 wires [108].

The signaling scheme allows for common-mode voltage cancellation and voltage-mode drivers with resistive termination can be used for better signal integrity and data reception.

- *Area benefits:* Using a serial interface results in fewer wires and pins, along with a higher wire density.

- *Reduced latency:* With the proposed encoding, it is possible to encode data into signal transitions, enabling instantaneous asynchronous detection at the receiving end and no cycles are spent for PLL/DLL locking. Moreover, longer is the transmission distance for the link, fewer will be the number of NoC-NoC hops, and lower will be the latency.

- *High throughput and energy efficiency:* Throughputs as high as 100-400 Gbps can be supported due to the 6-wire combination of high-speed serial links. The absence of PLL/DLL blocks reduces the idle power dissipation. Further energy efficiency for streaming data can be achieved by carrying out serial / parallel conversion only at the source and destination chiplets, and not at the intermediate NoC chiplets.

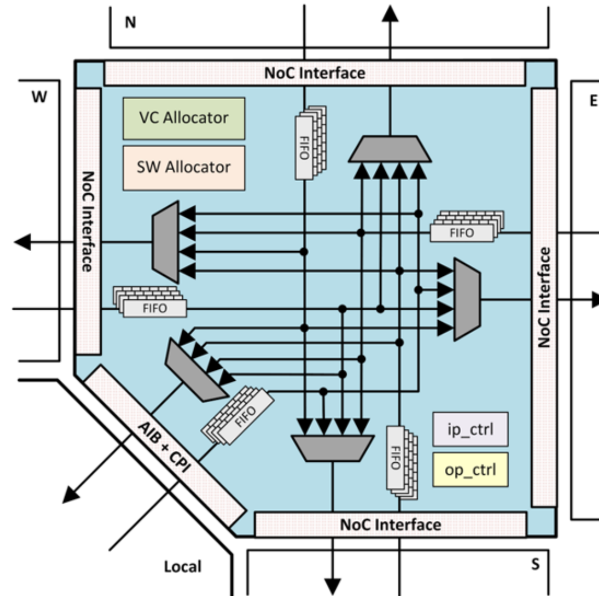

Figure 5.7: Packet-switched router: Transactional data.

5.3 NoC IP Router Design

NoC router is used to direct transactional and streaming data in a network of interconnected chips or chiplets. In this section we present the proposed two-layer router design, implementation decisions, topology, routing, and additional features for supporting a network of chiplets.

5.3.1 Two-Layer Packet-/Circuit-Switched Router Design

One of the requirements for NoC chiplet is that it should have routing support for both streaming and transactional data. Following this requirement, we propose using a two-layer packet-/circuit-switched NoC router, as shown in Figure 5.7 and Figure 5.8, for supporting packetized and streaming data, as previously demonstrated [109]. Our implementation of the two-layer router design aims at supporting chiplet interface standards and constraints.

Packet-switched layer is aimed for transactional communication using CPI packets,

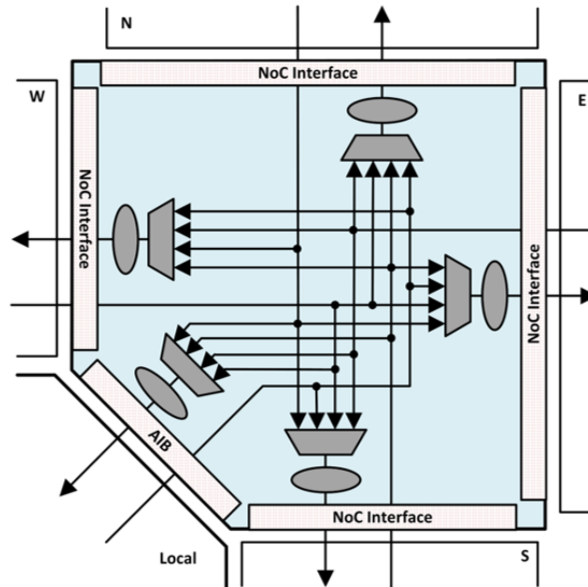

Figure 5.8: Circuit-switched router: Streaming data.

shown in Figure 5.7. Packet-switched layer is preferred for small amount of data and provides flexibility in terms of features supported. Latency of the packet-switched router includes NoC pipeline and deserialization/serialization that take place at each hop. Assuming conventional NoC pipeline takes 5 cycles and deserialization/serialization take 3 cycles each the final estimation of the latency at packet-switched layer is around 11 cycles, not including additional link latencies such as retiming.

On the other hand, circuit-switched layer, shown in Figure 5.8 is aimed for streaming communication due to its high efficiency for transferring large amount of data faster and with greater energy efficiency. Circuit-switched layer can be further improved in order to accelerate specific communication patterns that are known ahead of time. As explained in Section 5.3.4, transactional packets are responsible for the configure and release of streaming path. Streaming data after initial circuit-switched layer setup takes only 1 cycle in the NoC router, not including additional link latencies, proving our NoC IP solution is an efficient case for streaming communication. Furthermore, streaming data

transmission can be made even more efficient if the data can bet transmitted without deserialization/serialization at the bypass routers.

The two-layer router design is parameterizable and fully configurable in terms of:

- **Topology:** Mesh / Tree / Irregular (can be extended for Torus / Flattened Butterfly / Ring / etc.)
- **Sizes:** Number of ports, virtual channels, flits per virtual channel, flit width, endpoints per router, address widths
- **Packet Format:** CPI or any custom packet format
- **Control Mechanism:** Flow-control, routing and arbitration control

Following a common chiplet protocol, we fix some of these design choices, such as CPI as the packet format, credit-based flow-control scheme. However, decisions such as topology and routing are left to the NoC IP designer and we discuss these design decisions in the following sections.

A network of NoC IPs enables high interconnectivity between chiplets and is essential for designing scalable systems with many endpoints. Figure 5.9 and Figure 5.10 show example communication over two-layer routers in a network of NoCs. In this example, each NoC router is connected to a single local endpoint via a parallel interface (eg. AIB) and is connected to other NoC routers via a low-latency and high-throughput interface (eg. NoC interface, SerDes, etc.). Streaming communication utilizes the circuit-switching layer of the router as in Figure 5.8, whereas transactional communication utilizes the packet-switching layer of the router that implements the CPI protocol as in Figure 5.7. The physical interface is used for both types of data, while streaming communication does not necessitate a logical interface whereas transactional communication follows CPI protocol.

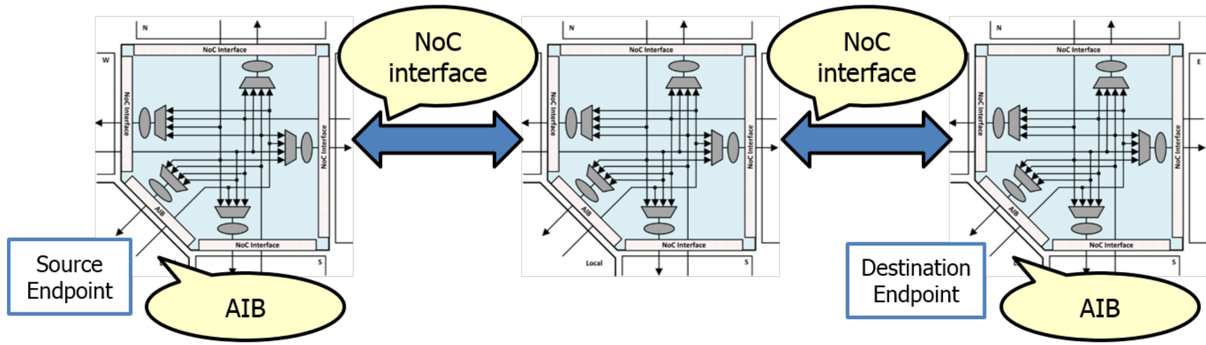

Figure 5.9: Streaming communication over circuit-switching NoC IP layer.

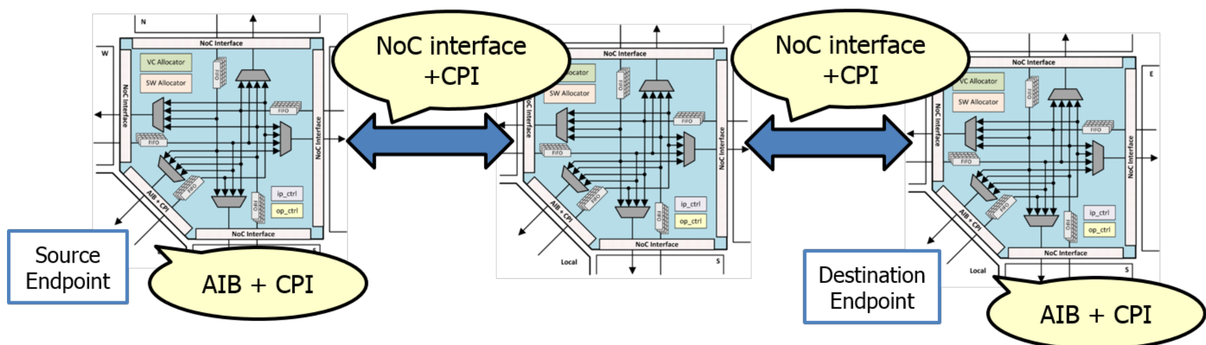

Figure 5.10: Transactional communication over packet-switching NoC IP layer.

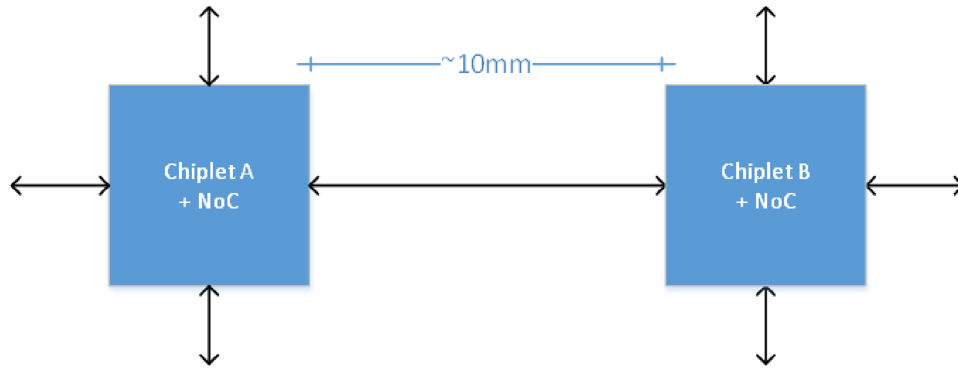

Figure 5.11: NoC router integrated implementation.

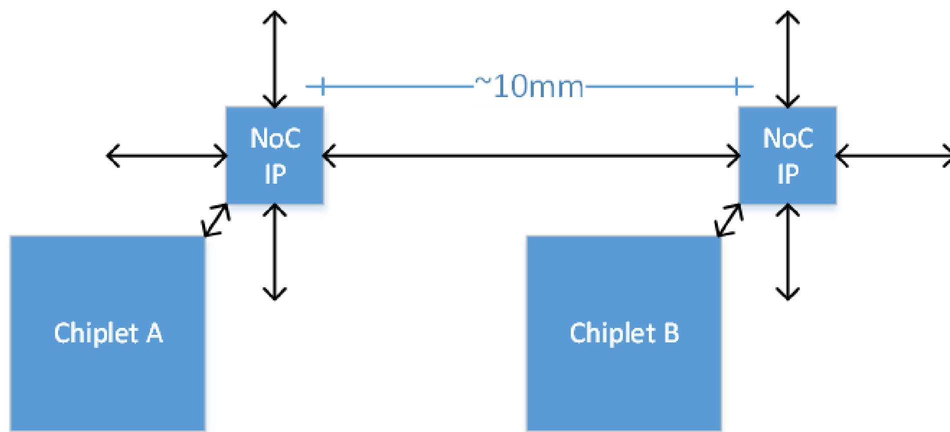

Figure 5.12: NoC router standalone implementation.

5.3.2 Integrated vs. Standalone NoC IP Study

In this section, we present our findings for the trade-offs between integrated and standalone NoC router implementations. There are two main implementation decisions for the NoC router: integrated within another chiplet as shown in Figure 5.11 or as a standalone IP as shown in Figure 5.12. We assume all links go through silicon interposer and the distances between NoC routers are on the order of tens of millimeters. We assume the connection between endpoint chiplet and NoC chiplet in case shown in Figure 5.12 is a parallel connection whereas the $10mm$ distance is supplied by high speed serial links.

Table 5.4 presents a brief qualitative analysis between the two implementation tech-

Metric	Integrated NoC	Standalone NoC
Latency	-	+ Few cycles
Area	+ Area overhead (NoC-NoC interface)	+ Area overhead (extra I/O)
Power	+ Power overhead (NoC-NoC interface)	+ Power overhead (extra I/O)
IP Reusability	Comparable if hard IP is provided	-
Heterogeneity	Comparable if soft IP is provided	-

Table 5.4: Qualitative analysis of integrated vs. standalone NoC implementation.

niques. In terms of latency, prior work claims routing on-interposer has identical characteristics to routing on-chip [29]. Few cycles of overhead at each endpoint to router is due to synchronization over the parallel interface. Synchronization may lead to a larger overhead for streaming communication due to lower latency circuit switched routing. Area and power overheads of integrated NoC are due to additional high-speed serial links. On the other hand, area and power overheads of standalone NoC are due to additional parallel links. NoC-NoC interface needs to be hard IP because of the analog circuitry in order to be flexible and reusable. Hard NoC IP may lead to integration issues if chiplets have different technology node than NoC IP. Soft IP would be synthesized for the chiplet technology instead.

In response to the overhead of standalone NoC implementation, we propose a concentrated standalone approach as shown in Figure 5.13. The reasoning behind this scheme is that if a single NoC chiplet can be connected to multiple endpoint chiplets, the overhead can be mitigated. This approach leads to a trade-off between latency and area, as more endpoint connections require fewer number of routers with larger area. However, adopting this approach would depend on the feasibility of the physical layout.

Overall, the qualitative comparison leads to two observations. First, integrated NoC implementation seems to be more efficient, especially for streaming applications. Second, concentration for standalone NoC implementation can offset the overheads and provide an efficient case for transactional applications. As a next step, we evaluate the integrated

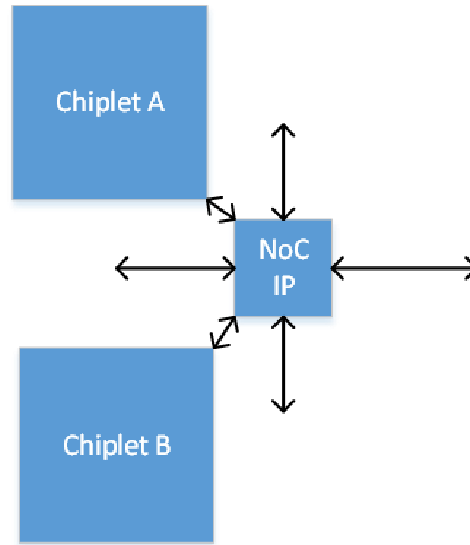

Figure 5.13: NoC router concentrated standalone implementation.

	Overhead
Latency (Streaming)	$18 - 20\%$
Latency (Transactional)	$10 - 15\%$
Area ($10\mu m$ pitch)	$11200\mu m^2$
Area ($55\mu m$ pitch)	$338800\mu m^2$
Power	$80mW$

Table 5.5: Overheads of a standalone NoC chiplet.

and standalone trade-off quantitatively and compare latency, area, and power, shown in Table 5.5.

In terms of latency, for the integrated NoC link latencies we assume 2 cycles for scramble/descramble and serializer/deserializer operations between endpoint and SerDes links. We assume 4 cycles for retiming and phase encoding/decoding on the SerDes links. For the standalone NoC link latencies we assume 3 cycles between endpoint and SerDes links, including retiming. We assume 5 cycles on SerDes links for a conservative case. NoC router latencies take 1 cycle for streaming common case communication via circuit-switched layer and 11 cycles for transactional communication via packet-switched layer assuming 5-cycle NoC pipeline and serializer/deserializer operations at each hop.

Considering all, the latency overhead of standalone NoC over integrated NoC would incur a $18-20\%$ overhead for streaming communication and $10-15\%$ overhead for transactional communication.

In terms of area, we calculate standalone chiplet area overhead by additional parallel I/O required. Table 5.5 shows the area overhead of a parallel interface of 40 data wires for a 6×7 grid for $10\mu m$ and $55\mu m$ pitch. Similarly, in terms of power, we calculate standalone power overhead by additional parallel I/O required. Assuming a parallel interface of 40 data wires with $1mW$ of I/O power each, total parallel I/O power overhead would incur $80mW$.

Overall, the quantitative analysis follows our observations on qualitative analysis in that especially for streaming communication heavy use cases, integrating NoC within other chiplets would be preferred for latency, area, and power reasons.

5.3.3 Topology and Endpoint Addressing

In this section, we will go over three topology implementations for the proposed network of interconnected chiplets: mesh, tree, and irregular topology. From here on, we assume an integrated chiplet approach where each endpoint represents a chiplet with an NoC router containing an endpoint address and that lines between chiplets represent SerDes links.

Mesh Interconnect Topology

A notional 2D mesh arrangement of links is shown in Figure 5.14 for simplified 4-bit address. A CPI flit (Figure 5.3, Table 5.2) is defined with an 8-bit address field, which provides 256 endpoint IDs. First flit of the packet is used to determine routing. For 2D mesh addressing, the address would correspond to a 4-bit X location and a

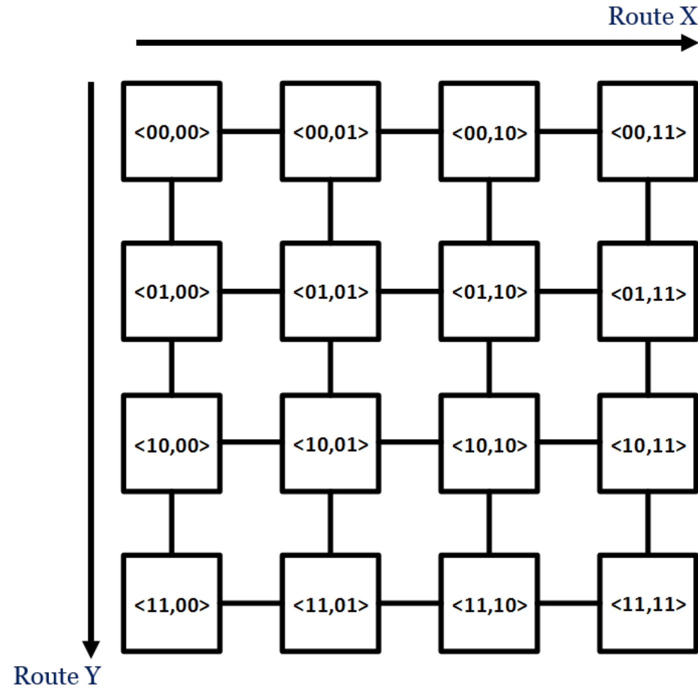

Figure 5.14: Mesh arrangement of 16 chiplets containing NoC routers.

4-bit Y location. CPI also provides optional 12-bit address field using the extended headers, which provides for 4096 endpoint IDs. The scheme described below also takes the extended address bits into account and for a mesh it would correspond to 6-bit X location and 6-bit Y location.

In mesh, each chiplet may contain up to 4 SerDes ports, identified as North, East, South, and West. Each point knows its address ("local" node endpoint ID), which is assigned with Endpoint Address Initialization procedure within CPI protocol described in Section 5.3.7. Routing for mesh topology is described in Section 5.3.4.

Tree Interconnect Topology

A notional binary tree arrangement of links is shown in Figure 5.15 for simplified 4-bit address. The endpoint addressing follows a binary search tree notation, where all

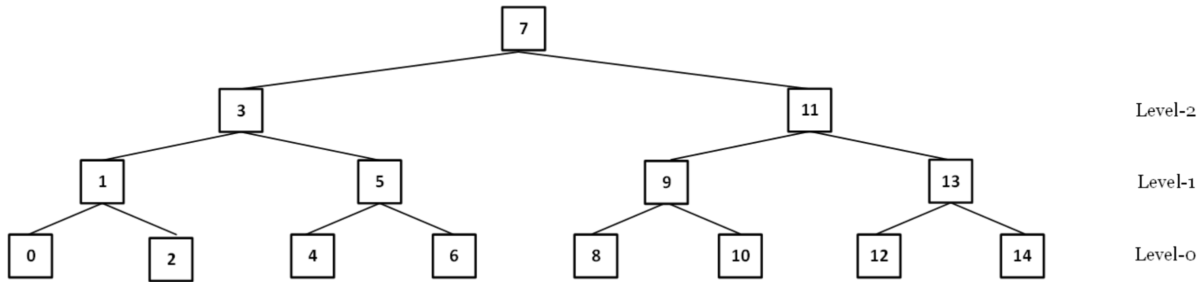

Figure 5.15: Tree arrangement of 15 chiplets containing NoC routers.

the child nodes to the left of a node has a value less than and all the child nodes to the right of a node has a value greater than the node itself. 8-bit endpoint address can be used to implement a height-7 binary tree for 255 chiplets in total. Similarly, for 12-bit endpoint address, a height-11 binary tree topology can implement 4095 chiplets in total.

In tree topology, each chiplet may contain up to 3 SerDes ports, identified as Parent, Left, and Right. Each point knows its address ("local" node endpoint ID) as well as the address of its parent node (parent endpoint ID). Routing for tree topology is described in Section 5.3.4.

Irregular Topology

Any regular or irregular topology can be implemented in the NoC by using the routing table implementation. The endpoint addressing can be assigned arbitrarily as long as it follows endpoint initialization methods. The trade-off with this implementation is between area and flexibility. With this scheme, all NoC routers will have to store a routing table that includes the information for which output port to take for any of the possible destination addresses, which can be quite large for many number of endpoints. Furthermore, the system designer would need to come up with all routing decisions ahead of time. On the other hand, this scheme has an inherent flexibility in terms of arbitrary topologies and routing scenarios.

5.3.4 Routing Algorithm

Routing in NoC router requires differentiating between transactional and streaming communication. In this section, we present routing in NoC over the described interface standards where we require SerDes frame to include a data type bit, sent as the first bit. Data type bit zero (0) indicates transactional data and data type one (1) indicates streaming data.

Routing for Transactional Data Type

When a NoC router receives a SerDes frame and identifies it as transactional (DATA TYPE=0), it goes through a sequence shown in Figure 5.4 to implement the received CPI command. The header flit of a packet contains the routing information that needs to be extracted for the body flits to follow. The flits can be transmitted to the downstream chiplet determined by the routing algorithm only when it has enough credits for the entire packet, as determined by the flow-control scheme of CPI.

Routing Algorithm for Mesh Topology. Given the destination endpoint ID (DID) within the CPI header flit, a route is computed for transactional transfer within the mesh using X-first routing described as shown in Algorithm 2, where HID stands for 4-bit horizontal ID and VID stands for 4-bit vertical ID. This algorithm first routes in X-direction until the horizontal IDs match and then in Y-direction until the vertical IDs match.

Routing Algorithm for Tree Topology. Given the destination endpoint ID (DID) within the CPI header flit, a route is computed for transactional transfer within the tree using a modified binary search tree routing described as shown in Algorithm 3, where local and parent endpoint IDs are assumed to be stored locally during initialization. This routing algorithm compares the destination ID to self and parent IDs and determine

Algorithm 2 Routing algorithm for mesh topology.

```
Compare 8-bit destination address with own address
if destination ID = source ID then
  consume data locally
else
  Decode address into 4-bit HID and 4-bit VID
  if destination HID > self HID then
    route EAST
  else if destination HID < self HID then
    route WEST
  else if destination VID > self VID then
    route SOUTH
  else if destination VID < self VID then
    route NORTH
  end if
end if
```

whether the packet is routed to one of its left and right children or backtracked to its parent before being routed in the other subtree.

Routing for Irregular Topology. As mentioned in Section 5.3.3, routing using routing tables is very straight-forward. Each router, instead of implementing a routing algorithm, stores a routing table. The table consists of the following information, the output port to route for each destination IDs. The size of this table may be of concern if the system incorporates many endpoints. The complexity of this implementation lies with the system designer at initialization time.

Routing for Streaming Data Type

A routing path for streaming data is preset using transactional communication. In this section, we present proposed CPI commands to configure and release circuit-switched NoC layer for streaming communication. With this method, streaming communication efficiency will be very high, since the transactional packet overheads to configure the circuit-switched layer is fixed for any length of streaming communication.

Algorithm 3 Routing algorithm for tree topology.

```
Compare 8-bit destination address with own address
if destination ID = source ID then
  consume data locally
else
  if destination ID < self ID then
    if parent ID < self ID then
      route PARENT
    else if parent ID > self ID then
      route LEFT
    end if
  else if destination ID > self ID then
    if parent ID < self ID then
      route RIGHT
    else if parent ID > self ID then
      route PARENT
    end if
  end if
end if
```

By default, the streaming data (DATA TYPE=1) is consumed at the destination chiplet when received, without having its body data analyzed by the routers. However, if a streaming path is specified, the NOC router checks DATA TYPE bit, and if it is set to 1, the deserialized frame is routed, without any additional processing along a preset path, from deserializer of one SerDes port to serializer of another SerDes port.

The routing set up consumes the involved SerDes ports of all nodes along the route to destination chiplet and makes them unavailable for transactional transfers for all flits except port release (NSPR) defined below. For example, in Figure 14, a streaming path is shown from node at simplified 4-bit address $< 01, 00 >$ to node $< 10, 10 >$. The EAST port of $< 01, 00 >$ and WEST port of $< 01, 01 >$ are blocked for a transactional transfer occurring between $< 00, 00 >$ and $< 01, 01 >$. In the absence of collision avoidance in implementation, transactional packets that encountered a "blocked" path are dropped.

Path calculation for a streaming route is same as that for transactional. To en-

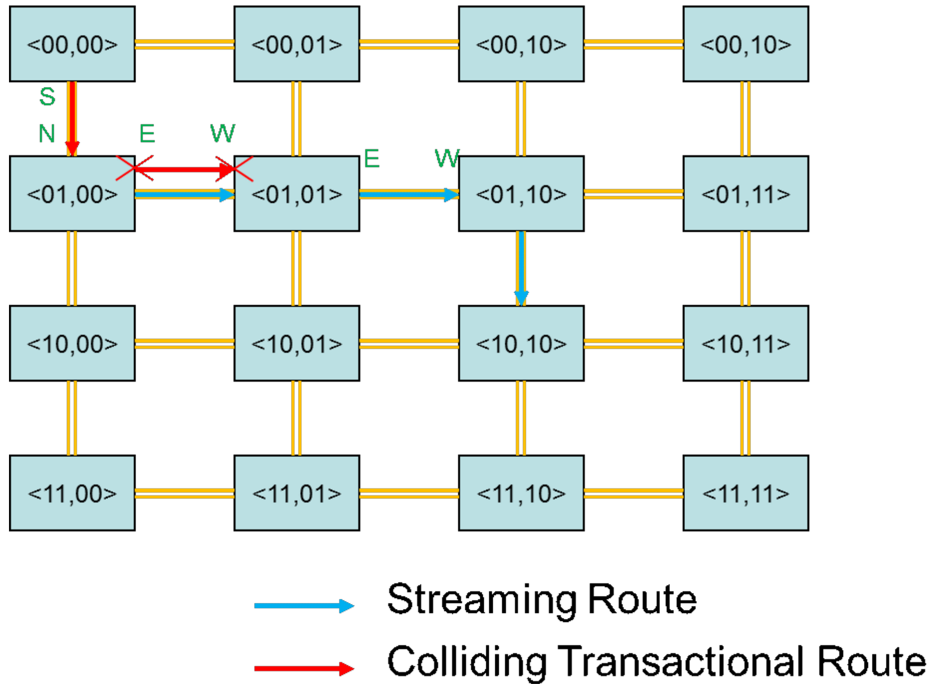

Figure 5.16: Example of streaming path route with colliding path.

able streaming path data transfer, two new commands are proposed in the CPI interface (flit bits 13:6). First command is, NoC Streaming Ports Setup Configure/Release (NSPC/NSPR), shown in Figure 5.17 and Table 5.6. Second command is, NoC Streaming Ports Acknowledge Configure/Release (NSPCA/NSPRA), shown in Figure 5.18 and Table 5.7.

Streaming route is initialized as follows:

Figure 5.17: NoC Streaming Ports Setup CPI command.

Field	Usage
LEN [5:0]	Encodes the packet length, LEN = 2 for this case (000010)
CMD [13:16]	Encodes the command, uses Vendor VC0 CMD#80 (01010000)
TID [21:14]	Encodes the transaction ID
DID [29:22]	Encodes the destination ID [7:0]
C/R [30]	Command Setup (0) / Release Setup (1)
S [34]	Sequential packet, S = 0 in this case
C [35]	Credit bit
SID [7:0]	Encodes the source ID [7:0]
DID Extd [11:8]	Destination ID Extended ([11:8] - optional for 12-bit addressing)
SID Extd [15:12]	Source ID Extended ([11:8] - optional for 12-bit addressing)

Table 5.6: NoC Streaming Ports Setup CPI command fields.

Figure 5.18: NoC Streaming Ports Acknowledge CPI command.

Field	Usage
LEN [5:0]	Encodes the packet length, LEN = 2 for this case (000010)
CMD [13:16]	Encodes the command, uses Vendor VC0 CMD#152 (10011000)
TID [21:14]	Encodes the transaction ID
DID [29:22]	Encodes the destination ID [7:0]
C/R [30]	Command Ack (0) / Release Ack (1)
S [34]	Sequential packet, S = 0 in this case
C [35]	Credit bit
SID [7:0]	Encodes the source ID [7:0]
DID Extd [11:8]	Destination ID Extended ([11:8] - optional for 12-bit addressing)
SID Extd [15:12]	Source ID Extended ([11:8] - optional for 12-bit addressing)

Table 5.7: NoC Streaming Ports Acknowledge CPI command fields.

- The source chiplet requesting a streaming route issues a transactional NSPC command containing address of the destination node (DID), address of the source node (SID), and configure bit [30]. For 12-bit case, DID extended and SID extended bits can are used.

- As the command is routed to the destination, it is decoded as a streaming setup request for the bi-directional links along that route.

- The destination chiplet issues an NSPCA command back to the source. The NSPCA command routing rules are, however, routed in reverse relative to the standard transactional routing ("Y-first"). This causes NSPCA flit to cover same nodes NSPC command arrived through. For example, in route setup of Figure 5.16:
  - NSPC command travels: $< 01, 00 >$ (src) $\rightarrow < 01, 01 > \rightarrow < 01, 10 > \rightarrow < 10, 10 >$ (dest)
  - NSPCA command travels: $< 10, 10 > \rightarrow < 01, 10 > \rightarrow < 01, 01 > \rightarrow < 01, 00 >$

- Upon NSPCA command leaving the destination node, destination node locks that port for all transactional transfers in either direction, with exception of NSPR command.

- Each NoC node that routes the NSPCA command locks the port on which NSPCA arrived and locks the port on which NSPCA travels back to requesting node. Both Tx and Rx directions for a port are locked for streaming.

- Once NSPCA arrives to requesting node, route setup is acknowledged and streaming transfer may commence with DATA TYPE is set to 1. A failure of acknowledgement

may be detected with a watchdog at the requesting chiplet, which can reissue the NSPC command at a later time. The watchdog will report a timeout externally.

- At the end of the streaming communication, requesting chiplet issues a NSPR (port release) command, containing DID, SID, and release bit [30], following the same path as NSPC.
- Similar to NSPCA, a release acknowledge packet (NSPRA) is sent back to the requesting chiplet, while releasing the bi-directional port locks for transactional operation. A failure of acknowledgement may be detected with a watchdog at the requesting endpoint chiplet, which would report the timeout and may resend the NSPR command.

5.3.5 Collision Avoidance

In this section we introduce methods to avoid routing collisions between various communication streams. We define collision as when more than one data stream or packet are contending for the same output ports on the same path. Specifically, three collision scenarios may exist: transactional vs. transactional, streaming vs. streaming, and streaming vs. transactional. For the general case, we assume streaming communication has priority over transactional communication, as streaming communication tends to be for real-time data streams.

Transactional vs. Transactional Collisions

NoC implements collision resolution for transactional data within its arbiter. The arbiter schedules contending transactional requests to a shared output port in a given order. A simple reference arbitration algorithm is round robin arbitration, which allows access to requests in the input buffers in circular prioritization.

Figure 5.19 shows the round robin arbitration algorithm for a single output port. First, all input ports contending for the same output are checked for any incoming requests. Requests for all input ports are set for the given timestep. Then, as per CPI protocol, all flits of a packet need to be issued consecutively, so the packet length counter is checked. Only those packets that contain all their flits within the input buffer are allowed to be scheduled. Last, the flow-control credits are checked to make sure there are enough credits for the size of the packet to be sent downstream. Only if all three of these conditions are satisfied, the request is granted and queued in a statically given order (of W, N, E, S). Similarly, we follow the round robin approach within the virtual channels of input ports and perform a circular prioritization. We check for similar conditions, that all flits in a packet are available and there are enough credits for the corresponding virtual channel's input buffer of the downstream router.

Overall, through the functionality of NoC router and credit-based flow-control scheme of CPI, transactional buffer occupancy collisions would inherently be solved by waiting for one another given some priority scheme as the round robin arbitration explained above.

Streaming vs. Streaming Collisions

Due to the real-time nature of streaming communication, the communication streams cannot wait for another in the event of collisions. Therefore, the goal is to avoid collisions by leveraging the path diversity of the topologies, when possible. We mainly consider minimal deterministic and oblivious routing schemes in order to keep the blocked path to be minimal (shortest path).

- **Streaming data flow known ahead of time**

  In the case when streaming data flow is deterministic and known ahead of the time,

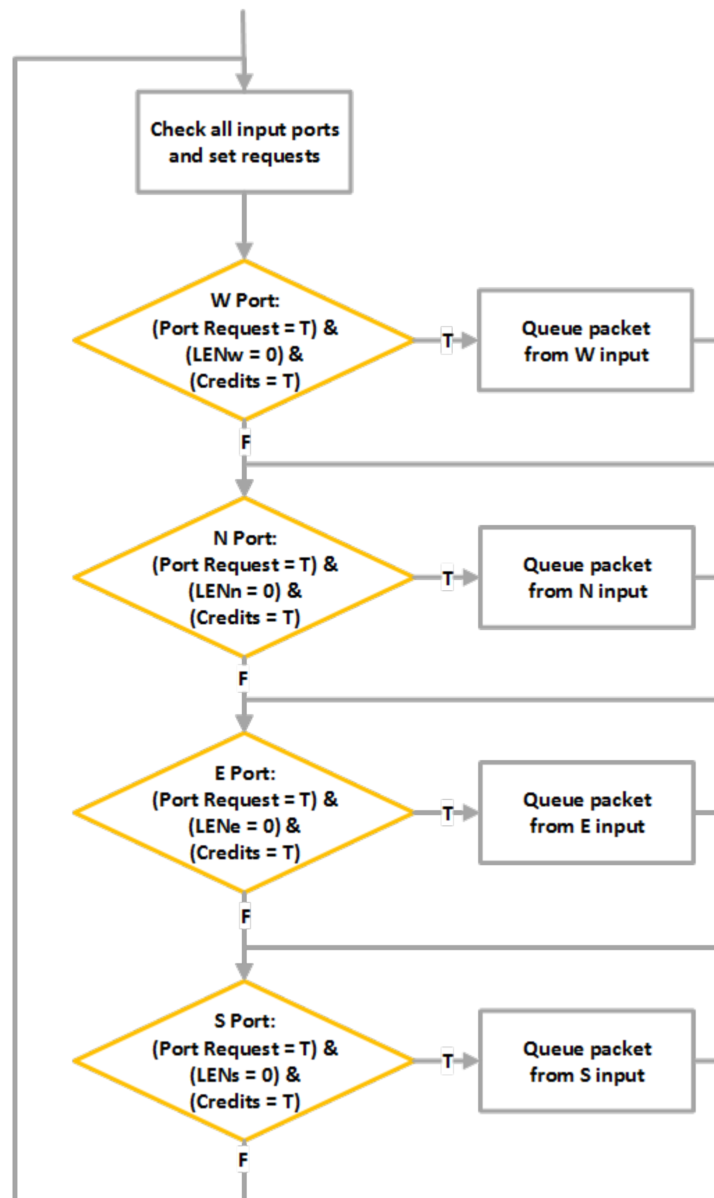

Figure 5.19: Round robin algorithm in NoC.

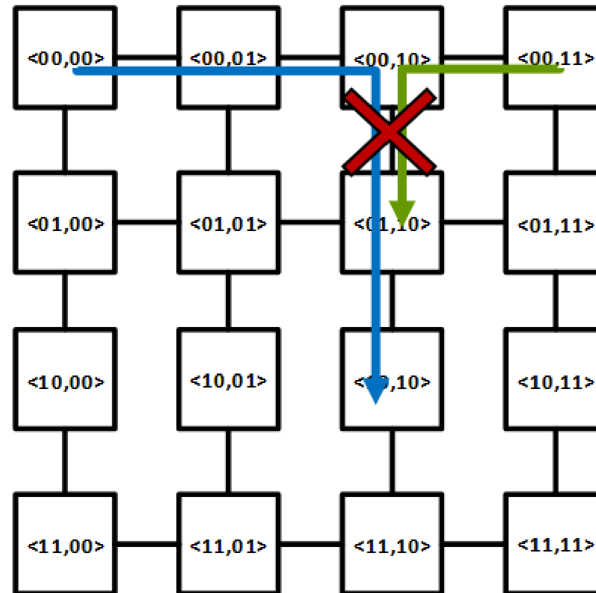

Figure 5.20: XY routing example for streaming collisions.

table-based deterministic routing can be implemented to ensure paths are determined ahead of time so that no streaming communication collisions are expected.

Figure 5.20 and Figure 5.21 show a use case of deterministic table-based routing example on mesh topology. Consider blue and green streams that would incur collision under X-first routing algorithm. However, if it is known that blue and green streams flow at the same time, the system designer can initialize the table-based routing scheme so that the streams communicate via paths that do not lead to any collisions but are still minimal.

- **Streaming data flow *not* known ahead of time**

  If the streaming data flow is not known ahead of time table-based routing cannot necessarily avoid collisions. In this case, NoC router can employ oblivious routing algorithms which increase the path diversity and still minimal.

  One example is choosing between X-first (XY) or Y-first (YX) routing for mesh topology. As shown in Figure 5.22, with XY routing there is only one deterministic

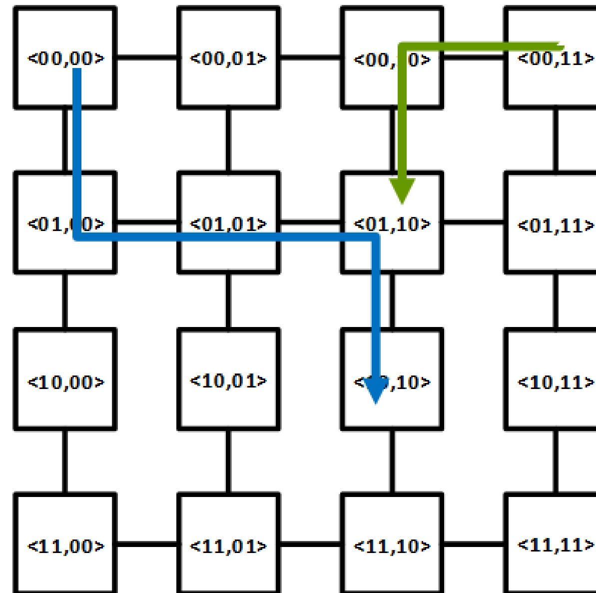

Figure 5.21: Deterministic table-based routing example for streaming collisions.

path that results in collision. However, with XY/YX routing the increased path diversity can be utilized to route the green stream to take YX routing and therefore avoid collisions.

Streaming vs. Transactional Collisions

In the event of transactional packets colliding with a path blocked by a streaming communication, we assume that streaming communication always has priority over transactional communication. As mentioned above, streaming communication is preferred to take the shortest path in order to minimize the blocked links. Therefore, a way to avoid collisions is to implement adaptive routing for transactional communication. Adaptive routing considers current network state, such as link availability, buffer occupancy, or channel load, to adaptively select the available path to route. In the event of streaming vs. transactional collisions, link availability can determine the adaptiveness since streaming communication blocks the links routed upon deeming them unavailable for

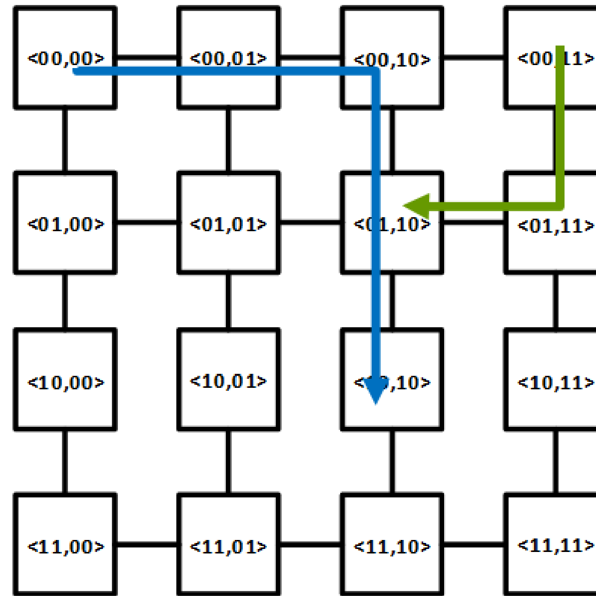

Figure 5.22: Oblivious XY/YX routing example for streaming collisions.

transactional communication.

In the unavailability of a link due to streaming communication blockage, we apply an easy to implement heuristic for redirecting the packets to take a non-minimal path to the destination. The reason we do not let transactional packets wait for streaming communication is because there is no way of determining how long of a stream is being transmitted, so non-minimal routing is introduced to avoid starvation. Algorithm 4 depicts the generalized adaptive routing scheme for mesh topology.

First, the X port of the minimal path to destination is checked. If blocked, Y port of the minimal path to destination is checked. If both are blocked, the router picks from available ports for non-minimal "hot potato" routing. The heuristics to make this decision can either (a) prioritizes larger buffer-space, or (b) statically prioritize N, E, S, W ports. By applying the heuristics, "hot potato" routing deflects the packet when it encounters blocked path. The next-hop router then would again go through the same adaptive routing scheme, which applies XY / YX / hot potato routing ordering. Due to

Algorithm 4 Adaptive routing scheme for mesh topology.

```
Check X port availability
if X port blocked then
  Check Y port availability
  if Y port blocked then
    if Available ports exist then
      "Hot potato" pick port using a heuristic
    else
      Wait at buffer for retry later
    end if
  else
    Route Y port
  end if
else
  Route X port
end if
```

prioritizing XY/YX minimal routing first, we can provide a guarantee that the packet will not deadlock or livelock and eventually get to the destination unless many streams are blocking the paths in which case the packets would stall otherwise.

Figure 5.23 shows an example of our routing algorithm heuristic. Source and destination nodes for a transactional packet are marked on the network. Red arrows represent streaming communication streams, blocking the ports marked with an "X". The packet therefore has to go (East, North) from the source chiplet. First, East port is checked and found blocked by a stream. Next, North port is checked and found blocked by another stream. After this point it is determined that a non-minimal route will have to be taken via hot-potato routing, which chooses one of the two available ports, South and West, according to a predetermined heuristic. At the next router all ports are available, either one of the XY/YX routing can be taken, as long as the packet doesn't backtrace to previous chiplet.

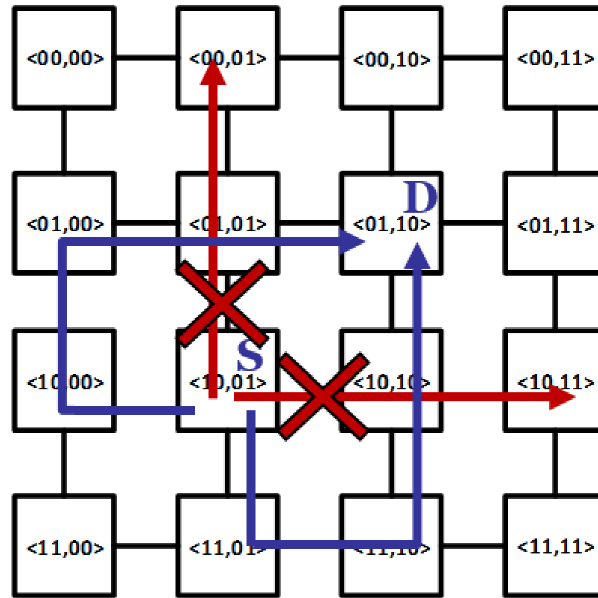

Figure 5.23: “Hot potato” routing of transactional packets.

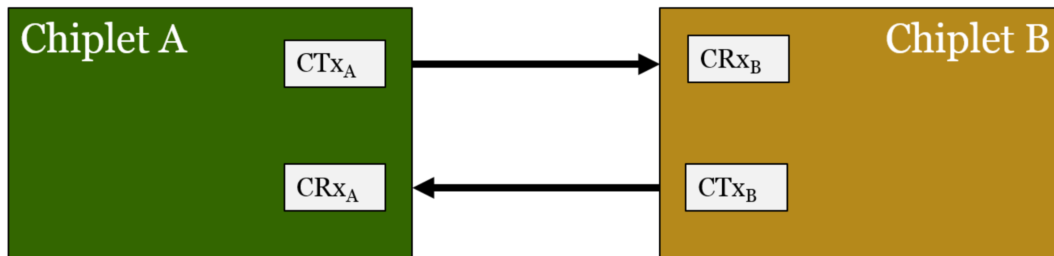

Figure 5.24: Heartbeat monitor components.

5.3.6 Optional Heartbeat Monitor

This section proposes an optional heartbeat monitor for periodic link verification by defining a new CPI command. For a single bi-directional link between two chiplets, four counters are needed for the heartbeat monitor, two for each direction (CTx, CRx), as shown in Figure 5.24. Periodically, CPI Heartbeat command will be sent with a determined period, eg. every 1000 flits.

Following diagrams show the flow chart of the heartbeat functionality. Figure 5.25 shows the transceiver side, a counter CTx, set to zero initially, which down-counts from

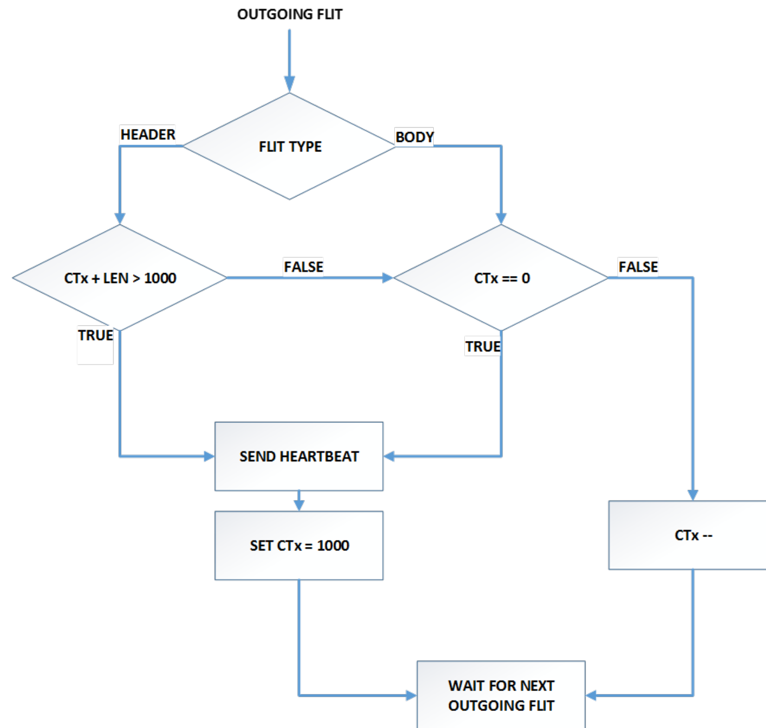

Figure 5.25: Heartbeat Monitor flow chart (transceiver side).

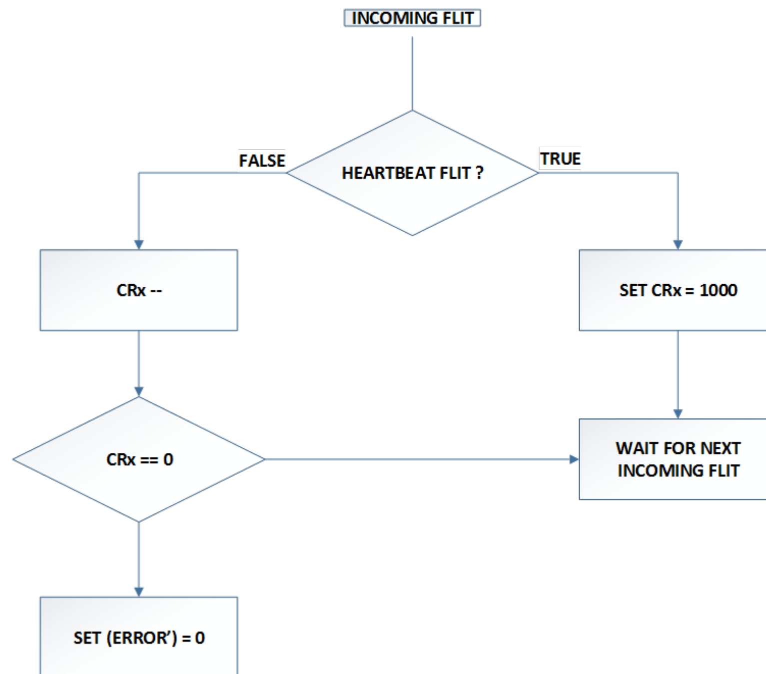

Figure 5.26: Heartbeat Monitor flow chart (receiver side).

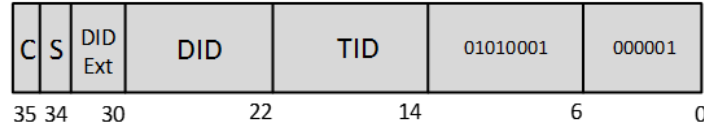

Figure 5.27: CPI Heartbeat command.

Field	Usage
LEN [5:0]	Encodes the packet length, LEN = 1 for this case (000001)
CMD [13:16]	Encodes the command, uses Vendor VC0 CMD#81 (01010001)
TID [21:14]	Encodes the transaction ID
DID [29:22]	Encodes the destination ID [7:0].
DID Extd [11:8]	Destination ID Extended ([11:8])
S [34]	Sequential packet, S = 0 in this case
C [35]	Credit bit

Table 5.8: CPI Heartbeat command fields.

the threshold upon every outgoing flit and issues a heartbeat command when the counter reaches zero and sets the counter back to the threshold. With every outgoing header flit for CPI packets, the length of the packet is checked to make sure the CTx will not go over the threshold (since all flits in a packet should be transmitted consecutively without interruption). If an overflow is expected, then the heartbeat is prioritized before the CPI header flit.

Figure 5.26 shows the receiver side, a counter CRx, set to zero initially, which down-counts upon every incoming flit. The CRx is set to the threshold whenever a heartbeat command is received. If the counter is zero, the chiplet initiates a link reset through setting ERROR' equal to zero. This way soft errors that affect the links can be overcome.

CPI Heartbeat command is defined as shown in Figure 5.27 and Table 5.8.

5.3.7 Endpoint Initialization

Assignment of endpoint addresses is performed using a "master chiplet" or an external SerDes link that is connected to neighboring chiplets through the SerDes interface.

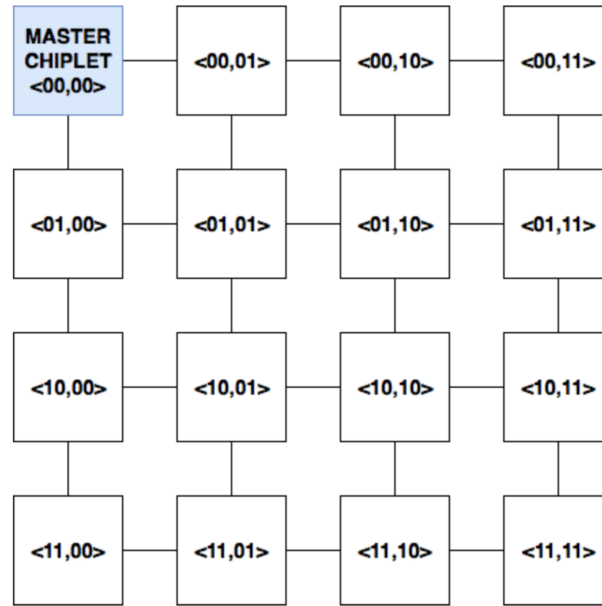

Figure 5.28: Master chiplet connected to Mesh Network

The master chiplet can be programmed to send CPI packets to other chiplets that configure connections between chiplets, as well as assign addresses to chiplets. Examples of connections of a master chiplet to other chiplets are shown in Figure 5.28.

In order to perform endpoint initialization, we can simply leverage existing CPI Configuration Write and CPI Configuration Read operations issued by the master chiplet in order to assign address information to the endpoints.

5.4 NoC IP Use Cases

To conclude, we discuss some use cases of our NoC IP proposal as listed below:

- **Enables Building Mega Chips:**

  A network of NoC IPs enables scalability, especially going for mega chips scale. In such a large-scale integration with long distances, point to point or bus based connections will not suffice the amount of communication between many chiplets.

Network of NoC IPs can give a scalable, low latency and high bandwidth solution.

A network of NoC IPs also provides a fault tolerant interconnection around failed links and chiplets when implemented with adaptive routing algorithms. The possibility of chiplet or link failure increases significantly with the number of components, such as in the mega chips concept. Therefore, a NoC IP interconnection network adoption is essential for the reliability and feasibility of the mega chips concept.

- **Solves Data Converter Bandwidth Bottleneck:**

  NoC IPs provide a scalable high bandwidth data transmission for long range, given the high-speed serial signaling between the routers. After the streaming route has been set, as explained above, streaming data takes only a few cycles per hop, improving the communication bandwidth.

  Unlike point-to-point connections between chiplets, a network of NoC IPs enables multi-input multi-output switching which can handle the communication of multiple simultaneous streams over the network.

  Support for dynamic streaming data flow is enabled by our proposal of routing streaming data, as explained in Section 5.3.4. Unlike fixing connections in hardware between few chiplets for specific streaming connections, our NoC IP solution can adapt to dynamic changes within or between various applications. Therefore, changing streaming communication needs do not require changes to the system hardware.

- **System Development Cost/Speed/Flexibility:**

  Proposed NoC IP enables faster and more flexible system-level integration by potentially supporting many topologies, routing algorithms, communication patterns, and application scenarios. Therefore, adopting a network of NoC IP interface stan-

dard removes the need for system development or network design from scratch for each system, thereby improving development cost, speed, and flexibility.

- **Infinite Flexibility Fabric ("Lego" Vision):**

  A network of NoC IPs can enable the Lego vision of providing interconnect standard to integrate many IP cores together on a single scalable platform. NoC IP as a standalone chiplet would further contribute to the Lego vision by separating the routing functionality.

  NoC IP supports both streaming and transactional data communication, providing the flexibility for the Lego vision irregardless of the chiplets the NoC IP is connected to.

Chapter 6

Network-on-Chip Design Guidelines for Monolithic 3D Integration

Monolithic three-dimensional (M3D) integration is viewed as a promising improvement over through-silicon via based 3D integration due to its greater inter-tier connectivity, higher circuit density, and lower parasitic capacitance. With M3D integration, network-on-chip (NoC) communication fabric can benefit from reduced link distances and improved intra-router efficiency. However, the sequential fabrication methods utilized for M3D integration impose unique interconnect requirements for each of the possible partitioning schemes at transistor, gate, and block granularities. Further, increased cell density introduces contention of available routing resources. Prior work on M3D network-on-chips has focused on the benefits of reduced distances, but has not considered these process-imposed circuit complications. In this work, NoC topology decisions are analyzed in conjunction with these M3D interconnect requirements to provide an equivalent architectural comparison between M3D partitioning schemes.

6.1 Introduction

As progress has slowed in traditional transistor and interconnect scaling, three-dimensional (3D) integration is becoming an increasingly promising solution to improve efficiency through wirelength reduction. However, current through-silicon via (TSV) integration has scalability challenges due to high area overhead and high parasitic capacitance, which leads to power and delay overheads [32]. Instead, monolithic 3D (M3D) [1] is a promising integration technology that utilizes sequential manufacturing techniques to provide fine-pitched monolithic inter-tier vias (MIVs) as connectivity between tiers. M3D integration reduces interconnect distances while increasing the vertical connection density versus TSV integration by 10,000× [33].

With fine-pitched, low-overhead vertical MIV interconnects, M3D enables inter-tier partitioning at the granularity of either transistors, gates, or blocks. Each partition strategy is able to improve power efficiency and performance through wirelength reduction, but each strategy also introduces unique manufacturing and circuit complexities. For example, increased cell density in transistor-level partitioning can introduce route congestion [32], while the manufacturing temperature requirements for gate-level and block-level partitioning can result in performance degradation of the bottom-tier interconnect or top-tier transistors [33]. Further, M3D integration often achieves the greatest benefits only after additional metal layers have been provided, but these layers further increase process costs. When designing modern Network-on-Chip (NoC) systems, it is necessary to consider not only the beneficial distance reductions of M3D integration, but also these critical process complexities.

Most prior research on 3D NoC designs utilize TSV-based 3D integration. While this technology can reduce link distances, vertical TSV connections introduce area and

[1] In this work, we consider the MIV-based integration for the M3D technology, as detailed per prior work [32, 33, 110, 111].

delay overheads that must be carefully managed. To limit the number of TSVs, *Li et al.* proposed a NoC-bus hybrid router design that uses a 2D mesh NoC for horizontal (intra-tier) communication and tDMA busses for vertical (inter-tier) communication [112]. Although this router design enables single-cycle vertical transmission and reduces crossbar area compared to a symmetric 3D NoC router, the bus does not support concurrent vertical communication. *Kim et al.* proposed a dimensionally-decomposed NoC which decomposes the X, Y, Z dimensions to reduce crossbar complexity [113]. Although this router design reduces crossbar area, this architecture does not fully leverage the short vertical distance between tiers, as a hop is still required between each tier. Instead of having separate routers between tiers, a true 3D router implementation can partition router components between tiers to reduce intra-router distances while maintaining a two-dimensional NoC topology [114]. Prior work proposes partitioning the router across multiple layers with block granularity [115], however TSV area and density are limiting factors for this design.

Monolithic 3D integration enables a true 3D router design by leveraging the greater inter-tier vertical connection density and smaller MIV area and delay. Accordingly, NoC architectures need to be redesigned for the specific integration and interconnect characteristics of these developing M3D technologies [114]. Prior work has focused on leveraging the available M3D vertical interconnect density, but without consideration of the manufacturing and routing challenges that accompany M3D integration [116]. On the other hand, results from M3D EDA studies [32, 117] suggest that additional metal layers may be necessary to improve routability to maximize M3D efficiency improvement. However, these additional layers introduce a cost overhead.

Unlike prior M3D NoC research, this work addresses the manufacturing and interconnect challenges for each of the different M3D partitioning schemes. Based on these process features, an M3D NoC delay and topology analysis is conducted under a resource-

equivalent comparison. These evaluations showcase a balanced design space exploration across M3D partitioning techniques and NoC architecture, providing design guidance on the interconnection related trade-offs for M3D-integrated NoC.

6.2 Background

This section provides an overview, benefits, and challenges of MIV-based M3D integration, which can be partitioned at the level of transistors, gates, or blocks.

Transistor-Level Partitioning. NMOS and PMOS transistors are partitioned across two separate tiers in the transistor-level partitioning scheme (*TR-M3D*). Standard cells can be created with multiple sub-100nm MIVs in each gate, resulting in total footprint reductions of about 40% compared to 2D integration. By reducing inter-gate distances, efficiency can improve through reduction in wirelength, average gate size, and buffer count. However, ideal footprint reduction is not achieved by *TR-M3D* due to NMOS/PMOS mismatch and MIV overhead [117]. One drawback of *TR-M3D* is that increased pin density results in increased route congestion. Accordingly, additional metal layers, at additional cost, are necessary to achieve optimal efficiency [117]. Despite this, digital *TR-M3D* circuits can be designed in a similar manner as a 2D process technology with standard cell place and route methodology, as all inter-tier connections are encapsulated within the standard cells.

Gate-Level Partitioning. In this scheme, gates are partitioned across two or more separate layers, resulting in a coarser granularity than *TR-M3D*. Gate-level partitioning (*G-M3D*) can reduce interconnect distance, gate size, and buffer count by increasing intra-block gate density, resulting in consistent footprint reduction of 50% or more compared to 2D integration [33]. *G-M3D* can use existing 2D standard cells, however it requires 3D EDA tools [33].

Block-Level Partitioning. In this scheme, functional blocks are partitioned into separate tiers at a coarse granularity. Block-level partitioning (*B-M3D*) has the benefit of utilizing existing layouts and macros. Inter-block distance reduction may be hindered by block imbalance when partitioning [118]. Therefore, *B-M3D* achieves less efficiency improvement than *G-M3D* (and converges to *G-M3D* as block granularity becomes finer). Accordingly, this work focuses on *G-M3D* over *B-M3D* for optimal NoC-based M3D systems. However, block-level partitioning may be useful for heterogeneous M3D process integration to provide connectivity between different technologies.

Beyond these partition strategies, M3D integration strategies also introduce important technology considerations that influence the circuit and system design.

MIV Overhead: The monolithic inter-tier vias introduce very little delay and area overhead, especially compared to TSVs. MIV diameters are similar to interconnect vias and smaller than standard cells. When driven by a $45nm$ 4x inverter, MIV delay is only $40ps$, more than $18\times$ less than a $5\mu m$ TSV [33].

Metal Stack: The partitioning strategy influences the optimal metal stack. Gate-level and block-level partitioning require metal stacks on each tier. Because the MIV pitch is determined by the widest metal pitch on the tier below, lower tiers are limited to intermediate-width interconnect, and global interconnect is only available to the top tier [32]. These process stacks are visualized in Figure 6.1. All partitioning strategies, especially transistor-level partitioning, increase route congestion due to increased gate or pin density [32]. More metal layers may be necessary to achieve optimal benefits.

Process Complications: Due to the additional processing steps of sequential integration, as well as the potential increase in the number of metal layers, M3D integration is likely to increase the per wafer process costs. Until M3D processes develop further, temperature requirements for sequential manufacturing may require either slower tungsten interconnect on the lower tiers or weaker transistors on the upper tiers for *G-M3D*

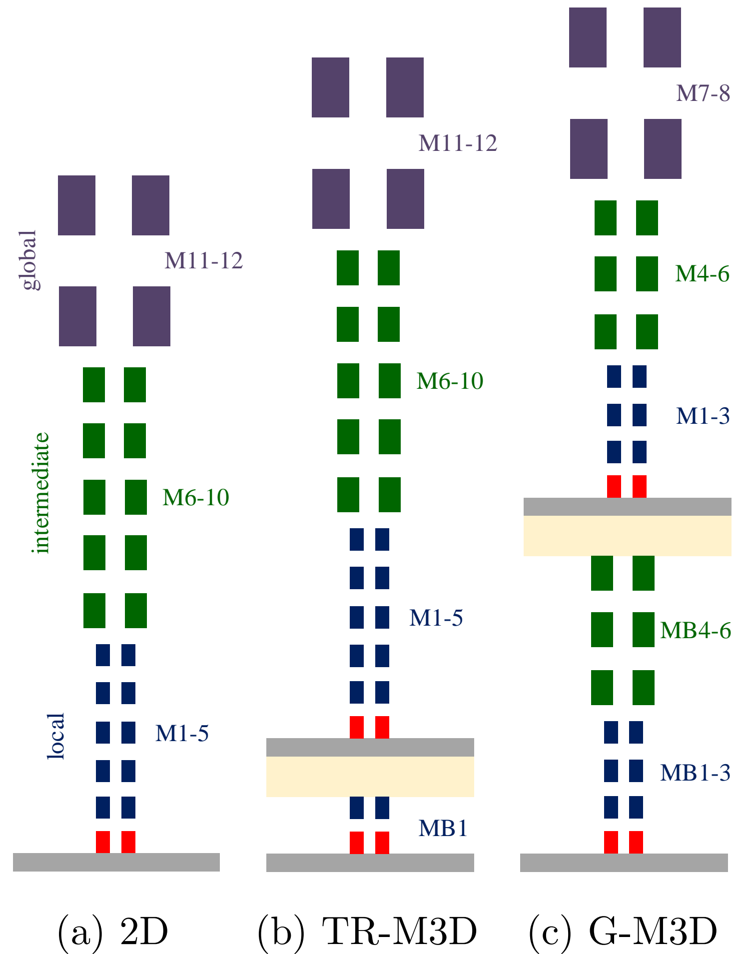

Figure 6.1: Metal layer stack diagrams.

and *B-M3D* [118]. Although EDA solutions can reduce this performance degradation for some logic through selective partitioning, lengthy NoC links are sensitive to degradation in interconnect resistance and drive strength.

Thermal Characteristics: A key challenge with M3D integration is management of the increased thermal density. However, unlike TSV-based 3D, there is little thermal resistance between tiers in M3D due to the thin intra-layer dielectric, which aides vertical heat transfer [119]. On the other hand, lateral thermal conductivity is reduced in M3D [119]. MIV placement is not a concern for thermal management, as MIVs do not contribute to heat conduction, while TSVs do [119]. Consequently, prior work considers

these thermal characteristics of M3D to propose a thermal-aware M3D NoC design [120].

6.3 M3D Interconnect Characteristics

In this section, the interconnection of each M3D partitioning schemes is characterized by evaluating their metal layer stacks and modeling the route delays for the NoC links.

6.3.1 M3D Partitioning Comparison

Below is a list of assumptions for the footprint and wirelength comparison between M3D partitioning techniques, based on recent M3D EDA results [117][33][121].

2D Baseline. 2D integration is selected as the baseline for footprint and wirelength comparison.

***TR-M3D* Partitioning.** Transistor-level partitioning can provide a 40% footprint reduction, as explained in Section 6.2. The *TR-M3D* footprint is 0.60× that of 2D. This translates to wirelength of 0.77× of 2D.

***G-M3D* Partitioning.** Gate-level partitioning can provide at least a 50% footprint reduction, as explained in Section 6.2. The *G-M3D* footprint is 0.50× that of 2D. This translates to wirelength of 0.71× of 2D.

6.3.2 Metal Layer Stack Characterization

In this section, the metal layer stack of each M3D partitioning scheme are characterized. Figure 6.1 shows the metal layer stack diagrams for 2D integration, *TR-M3D* partitioning, and *G-M3D* partitioning schemes.

2D Baseline. The 2D baseline has 5 local metal layers, 5 intermediate metal layers, and 2 global metal layers.

***TR-M3D* Partitioning.** The top tier of the *TR-M3D* scheme identical to the 2D metal layer stack. The bottom tier only has one local metal layer used for intra-cell connectivity. Increased cell density ($1.7 - 2.0\times$ compared to 2D) results in route congestion [117]. Previous proposals add more local and intermediate metal layers [117], however this is not possible without cost overheads. Therefore, no additional metal layers are supplied for this comparison.

***G-M3D* Partitioning.** The *G-M3D* scheme requires multiple metal layers in both tiers. The bottom tier lacks global interconnects, as the MIV pitch is determined by the widest metal pitch of the bottom tier, which causes per-cell contention for the global metal resources. Due to sequential manufacturing requirements, the bottom tier may need to replace copper interconnect with slower, resistive tungsten interconnect. Both copper and tungsten interconnect are analyzed in this study.

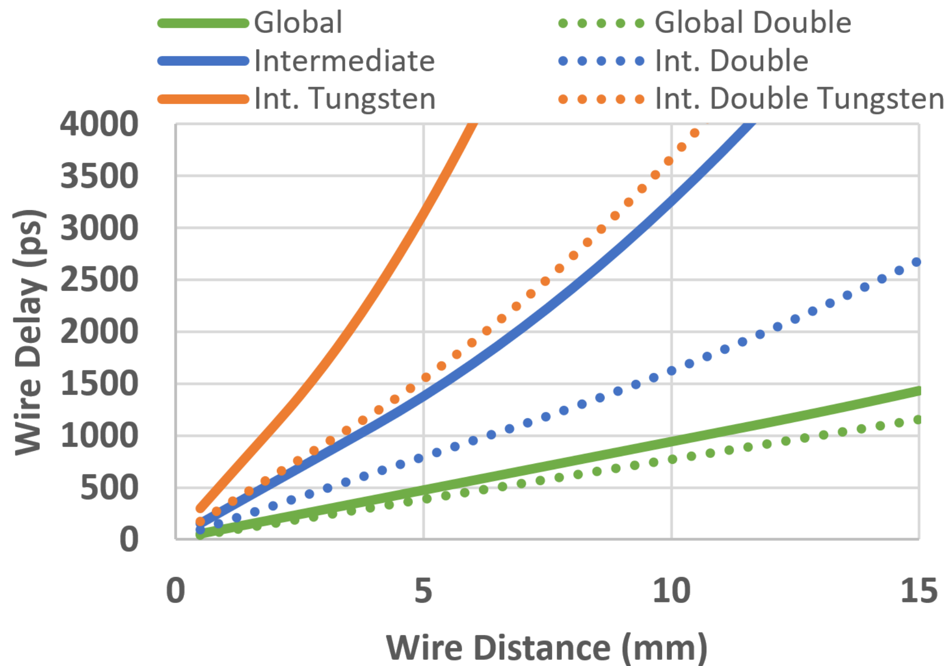

Figure 6.2: Metal layer link delay characterization versus distance. ”Double” indicates double-width, double-spacing.

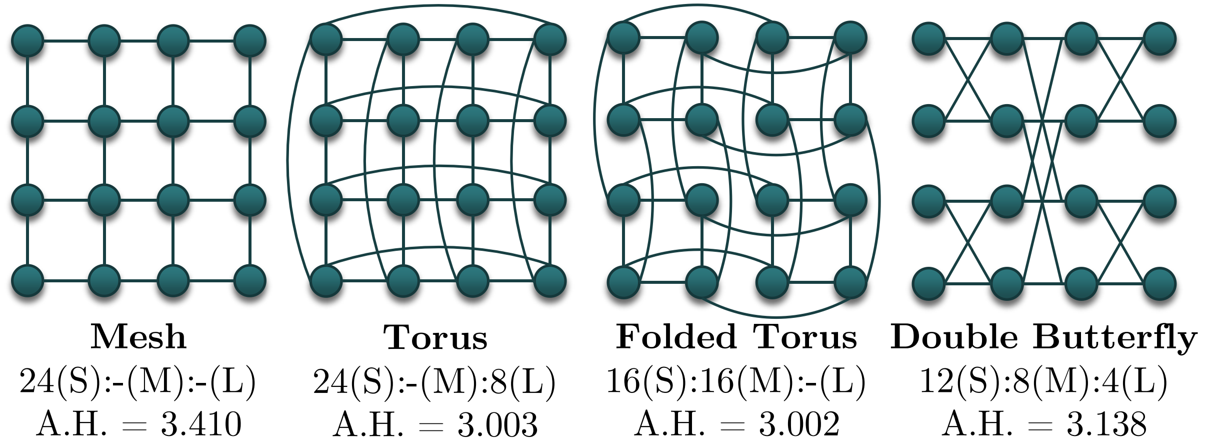

Figure 6.3: Evaluated topologies and characteristics: Number of short (S), medium (M), and long (L) links and average hops (A.H.)

6.3.3 Interconnect Characterization

In this section, the interconnect resources of each M3D scheme are characterized for select NoC topologies.

Methodology. HSPICE was used to compute the delay of optimally-driven, optimally-repeated links for each metal category in the ASAP7 7nm PDK. A three-segment pi-model interconnect was used to model each segment between repeaters. Worst-case parallel-neighbor parasitic capacitance is modeled to determine maximum link delay. Delay results are for copper interconnect RC values unless specified as tungsten.

Metal Layer Performance. Figure 6.2 shows link delays for each metal layer category plotted against link distances for the repeated links. "Double" corresponds to double-width, double-spaced interconnect, which utilizes twice the amount of available interconnect in order to reduce interconnect resistance.

NoC Baseline. As a feasible case study, a NoC-based system is selected with 16 chips that are $5mm \times 5mm$ in size (for the 2D baseline), arranged in a 4×4 layout. The design space is constrained to synchronous NoCs with single-cycle links. To maintain

M3D Type	Mesh	Torus	FTorus	DBFly
2D	1492	621	875	621
G-M3D	1857	837	1157	837
G-M3D (Tungsten)	1857	763†	763†	763†
TR-M3D	3005*	1694	1752†	1694

Table 6.1: Maximum frequency (in MHz) for M3D types and topologies. ($\star$) Maximum frequency is capped at 2000 MHz. ($\dagger$) Limited by intermediate metal layer.

a fair comparison between topology resources, routers are limited to a maximum of 5 ports[2], including the local connection. Accordingly, the following topologies shown in Figure 6.3 are analyzed: mesh, torus, folded torus (FTorus), and double butterfly (DBFly). The number of links in each topology for each relative length are listed in Figure 6.3. Note that medium links are twice as long as short links, and long links are three times as long. Diagonal distances for the double butterfly topology are Manhattan distances due to X-Y track routing.

M3D NoC Frequencies. Using the metal layer characteristics, Table 6.1 shows the maximum frequency calculations for each M3D partitioning scheme for the link distances of the four topologies that are considered. A 200ps overhead for skew and sequential capture, based on ASAP7 HSPICE timing, are added to the link delay to convert from delay to synchronous frequency. The maximum network frequency, independent of link delay, is capped at 2000 MHz based on the selected router microarchitecture. All of the longest links across topologies utilize the available global metal layers, but, to limit global interconnect utilization, the short links utilize intermediate metal for all but the mesh topology.

[2]While not evaluated in this work, high-radix routers are starting to have a wide-spread adoption. In M3D, it is possible to implement a NoC in a true 3D approach in order to avoid hops in the vertical dimension. However, higher radix demands more routing resources. Routing resources are already limited in M3D, but narrower channels can be used if radix increases. Trade-off between higher radix and narrower channels translates to a trade-off between the number of hops and number of flits in a packet. However, maximum link length is another limiting factor, as our evaluations will show that long links often limit achievable NoC frequencies.

The mesh topology achieves the fastest interconnect due to the short links between all router connections. The torus generally has slower links, limited by the long wrap-around links. The folded torus achieves higher frequency as the longest links are shorter. The double butterfly has equally long links as the torus, and with both, all diagonal links utilize the global interconnect.

To have a resource-equivalent comparison of the available interconnect between each manufacturing scheme, the 2D baseline has sufficient global interconnect for double-width, double-spaced links, but it suffers from the relatively long wirelength compared to M3D integration. *G-M3D* can take advantage of the multiple tiers of intermediate links to offset the limited global interconnect on the top layer. Two *G-M3D* technology options were considered: a baseline case without low-temperature process consideration and a case with tungsten interconnect on the bottom intermediate tier. For *G-M3D*, as indicated in Table 6.1, long links in global metal limit the NoC frequency, so it is safe to assign shorter links to the intermediate metal layers. *TR-M3D*, can take advantage of a reduced M3D wirelength, although the congestion from increased cell density poses a limit to interconnect resource availability.

6.4 M3D NoC Design Guidelines

In this section, the selected NoC topologies are compared according to the interconnect resources and characteristics of each M3D partitioning scheme.

6.4.1 NoC Topologies

In this section, NoC topologies with maximum radix of 5, including the local connection, are analyzed for each M3D partition scheme. The mesh, torus, folded torus, and double butterfly, shown in Figure 6.3, are evaluated. Figure 6.3 also presents a

comparison of these topologies in terms of average hops and the number of links of each distance.

Methodology. SynFull [122] integrated with BookSim [123] is used to evaluate the NoC topologies. Router microarchitecture is a 4-cycle pipeline. The default flit width (link width) is 32 bits, with 64 bits for the double-wide case. Networks are evaluated on PARSEC and SPLASH-2 benchmarks using SynFull.

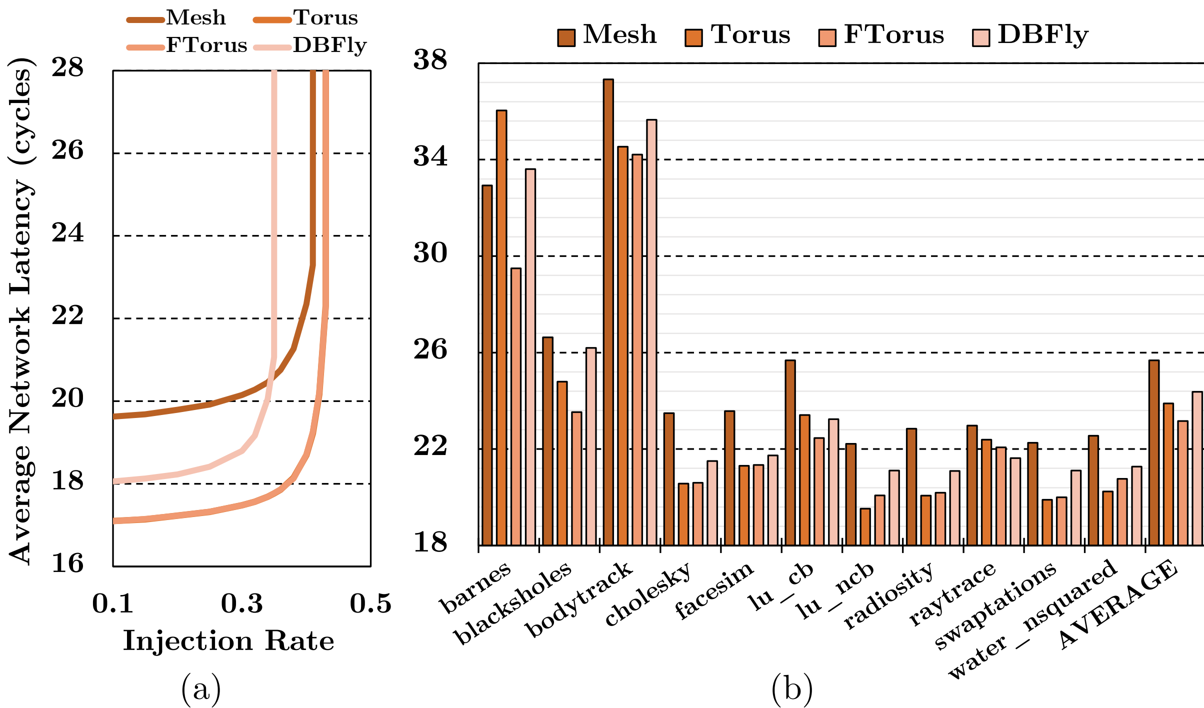

Figure 6.4: Topology comparison in terms of average network latency in cycles for: (a) uniform random synthetic traffic and (b) PARSEC and SPLASH-2 benchmarks

Topology Comparison. Figure 6.4(a) compares the four topologies in terms of average network latency in cycles by sweeping the injection rate for uniform random synthetic traffic. As expected, the mesh has the highest zero-load network latency, due to the high average hop count. The torus and folded torus perform similarly and have lower latencies and higher saturation throughputs than mesh. Although the double butterfly topology has a lower network latency than the mesh, we observe that it is not as scalable

and has a lower saturation throughput.

Figure 6.4(b) shows topology comparison results in terms of average network latency in cycles using SynFull, for the double-wide case, for PARSEC and SPLASH-2 benchmarks. The slowest topology is the mesh, while the fastest topology is the folded torus, which corresponds with their average hop counts. The average network latency changes by up to 12.5%.

These experiments were repeated for the default 32-bit link width, as well as half-wide (16-bit) links, in order to demonstrate the impact of routing congestion. Using 32-bit links increases average network latency by up to 34.2% on average over double-wide links. A pessimistic comparison is assumed with *TR-M3D* having 50% less routing resources. On average, reducing the routing resources by half increases average network latency further by up to 44.4% on average, over the default link width. For each configuration, available interconnect can either be utilized for link bit width (cycles) or physical interconnect width (frequency), and the lowest-latency configuration is selected.

6.4.2 Network Latency Comparison

This section presents the network latency results for the different partitioning strategies on the four topologies by combining interconnect characterization and topology comparison for PARSEC and SPLASH-2 benchmarks.

Figure 6.5 shows the average network latency of the four topologies for 2D, *G-M3D*, and *TR-M3D* partitioning schemes.

2D integration favors topologies with shorter links as the main source of latency reduction comes from higher frequency links. Due to lower cell density, 2D can utilize double-wide 64-bit links to reduce NoC latency.

G-M3D, for all but the mesh topology, uses intermediate metal across both tiers for

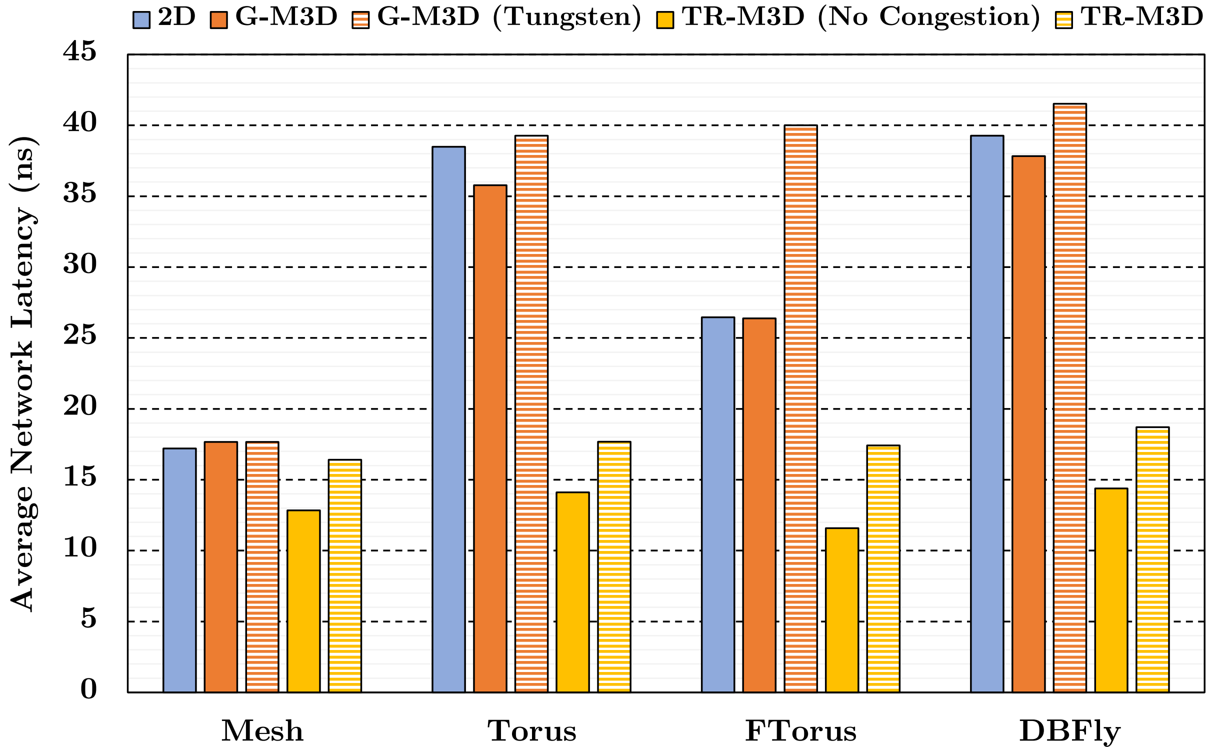

Figure 6.5: M3D NoC latency comparison

short links and global metal in the top tier for long links. *G-M3D* latency is comparable to 2D and within 10% lower latency except for mesh, without consideration of the low-temperature manufacturing requirements. When slow tungsten interconnect in the bottom tier is instead utilized, the latency significantly degrades by up to 1.51× that of 2D. The low latency of the mesh comes from using only global metal layer in the top tier, but due to global metal resource contention, *G-M3D* is constrained to default-width 32-bit links.

TR-M3D using global metal on the top tier can achieve a significant latency advantage from the reduced wirelength. Without taking routing congestion into account, and utilizing double-wide 64-bit links, *TR-M3D* latency for a given topology (FTorus) can be as low as 0.37× of 2D. When routing congestion is considered by constraining to only half

the routing resources, the latency increases but still stays below that of 2D integration, improving by up to 0.46×.

6.4.3 Discussion

This section presents M3D NoC guidelines based on the results and observations above.

- Frequency plays a significant role in NoC performance and it is not possible to select a topology by only considering the average hops. Accordingly, the mesh frequently performs best, despite a greater hop count, because it can achieve faster link frequencies.
- Despite the greater distance reduction, *G-M3D* may not be better than 2D because of the limited global interconnect.
- The temperature manufacturing requirements for *G-M3D* can result in even higher NoC latency than 2D. Except for the mesh, frequency is limited by the short links routed in intermediate tungsten metal in the bottom tier. Therefore, *G-M3D* favors topologies with short links to better utilize the limited global metal in top tier.
- *TR-M3D* can achieve significant improvement, due to shorter wirelength and the availability of global metal layers.
- With routing congestion taken into account, *TR-M3D* can still provide significant improvement over 2D for most topologies. Mesh performs the best but achieves latency similar to 2D. *TR-M3D* introduces a trade-off between extra routing resources and cost.

While we do not compare thermal characteristics in this work, the NoC frequency, number of links, and link width are expected to impact the thermal characteristics of the M3D NoC.

6.5 Conclusion

Monolithic 3D integration is a promising sequential manufacturing technique over TSV-based 3D integration. This work identifies the manufacturing and interconnect challenges for various M3D partitioning schemes and analyzes M3D NoC delay and topology under a resource-equivalent comparison. Our evaluations showcase a design space exploration of M3D partitioning techniques and NoC architectures to provide design guidance on the interconnection related trade-offs for M3D-integrated NoC.

Chapter 7

Summary of Contributions

Interconnect architecture has taken a center stage in computer architecture design given the memory wall, bandwidth wall, and power wall challenges of data movement in traditional Von Neumann architectures and the end of Moore's Law. Recent integration and packaging techniques emerge, such as 3D integration, silicon interposer-based 2.5D integration, and monolithic 3D integration, which have unique characteristics, challenges, and opportunities for the interconnect architecture. In this thesis, after presenting a background on these emerging integration technologies, we then evaluate and design interconnect architectures for them.

In Chapter 3, we propose a scalable memory network in silicon interposer interconnect architecture on 2.5D packaging. In this work, we consider a chip multi-processor system on silicon interposer-based 2.5D integration, with on-package 3D memory modules. In order to provide scalability without compromising latency or bandwidth, we propose interconnecting the 3D memory modules to create a memory network by using the routing resources of the silicon interposer package. We then provide topology and routing for the two layer network, comprised of the network-on-chip on the chip multiprocessor and the memory network in silicon interposer. We demonstrate that a memory network approach

can utilize the potential of the on-package memory effectively and provide scalability.

In Chapter 4, we propose a memory network with on-package 3D memory modules for high performance computing applications that require high memory capacity and performance. We first evaluate memory network topologies with DRAM and non-volatile memory technologies to understand the key trade-offs and bottlenecks. Our observations show that interconnection network contributes significantly to memory network latency. We then propose three techniques to reduce the memory network latency. First, we propose a globally fair distance-based arbitration scheme to reduce memory network diameter. Second, we propose a skip-list topology to reduce the memory network diameter. Finally, we leverage silicon interposer 2.5D integration technology and propose an advanced packaging technology called MetaCube to further reduce the memory network diameter. Our techniques provide effective improvements to the performance of future memory networks to support large systems with terabyte-scale memory capacities.

In Chapter 5, we propose a network-on-chip IP chiplet design to achieve scalable communication and close-to-monolithic performance for chiplet-based IP reuse systems integrated on silicon interposers. We design a NoC IP router to match the physical and logical standards for the chiplet ecosystem and to support both transactional and streaming dataflow to cover custom chip design usecases. We propose a two layer NoC router with circuit- and packet-switched layers and demonstrate topology and routing algorithm decisions over the common chiplet interfaces. We further evaluate collision avoidance scenarios and provide comparison for integrated and standalone implementations of the NoC IP. Overall, proposed NoC IP can be used to provide scalable communication for heterogeneous chiplet-based architectures over silicon interposer.

In Chapter 6, we evaluate network-on-chip implementations for various monolithic 3D integration partitioning schemes in order to provide NoC design guidelines. Different monolithic 3D partitioning schemes such as transistor-level partitioning or gate-level

partitioning have different manufacturing and interconnect characteristics. By identifying the unique interconnect characteristics, we perform a design space exploration of NoC delay over common NoC topologies. Finally, we present design guidelines and trade-offs to guide NoC design for monolithic 3D integration technology schemes while providing an equivalent architectural comparison between M3D partitioning schemes.

We hope the work in this thesis would provide insight and help guide interconnection architecture research in the era of emerging integration technologies, data-centric architectures, and modular chiplet-based systems.

Bibliography

[1] W. A. Wulf and S. A. McKee, *Hitting the memory wall: Implications of the obvious*, *SIGARCH Comput. Archit. News* **23** (Mar., 1995) 20–24.

[2] B. M. Rogers, A. Krishna, G. B. Bell, K. Vu, X. Jiang, and Y. Solihin, *Scaling the bandwidth wall: Challenges in and avenues for cmp scaling*, in *Proceedings of the 36th Annual International Symposium on Computer Architecture*, ISCA '09, (New York, NY, USA), p. 371–382, Association for Computing Machinery, 2009.

[3] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, *Dark silicon and the end of multicore scaling*, in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, pp. 365–376, 2011.

[4] H. Cheng, J. Zhan, J. Zhao, Y. Xie, J. Sampson, and M. J. Irwin, *Core vs. uncore: The heart of darkness*, in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, 2015.

[5] S. W. Keckler, W. J. Dally, B. Khailany, M. Garland, and D. Glasco, *Gpus and the future of parallel computing*, *IEEE Micro* **31** (2011), no. 5 7–17.

[6] G. Kim *et. al.*, *Memory-centric system interconnect design with Hybrid Memory Cubes*, in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, pp. 145–155, Sept, 2013.

[7] J. Knechtel, S. Patnaik, and O. Sinanoglu, *3d integration: Another dimension toward hardware security*, in *2019 IEEE 25th International Symposium on On-Line Testing and Robust System Design (IOLTS)*, pp. 147–150, 2019.

[8] G. Sun, Y. Chen, X. Dong, J. Ouyang, and Y. Xie, *Three-dimensional Integrated Circuits: Design, EDA, and Architecture*. 2011.

[9] J. Zhao, Q. Zou, and Y. Xie, *Overview of 3-d architecture design opportunities and techniques*, *IEEE Design Test* **34** (2017), no. 4 60–68.

[10] D. Stow, I. Akgun, R. Barnes, P. Gu, and Y. Xie, *Cost and thermal analysis of high-performance 2.5d and 3d integrated circuit design space*, in *2016 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pp. 637–642, 2016.

[11] JEDEC, "High Bandwidth Memory (HBM) DRAM." https://www.jedec.org/standards-documents/docs/jesd235a.

[12] J. Kim and Y. Kim, *Hbm: Memory solution for bandwidth-hungry processors*, in *Hot Chips*, vol. 26, 2014.

[13] C. Oh, K. C. Chun, Y. Byun, Y. Kim, S. Kim, Y. Ryu, J. Park, S. Kim, S. Cha, D. Shin, J. Lee, J. Son, B. Ho, S. Cho, B. Kil, S. Ahn, B. Lim, Y. Park, K. Lee, M. Lee, S. Baek, J. Noh, J. Lee, S. Lee, S. Kim, B. Lim, S. Choi, J. Kim, H. Choi, H. Kwon, J. J. Kong, K. Sohn, N. S. Kim, K. Park, and J. Lee, *22.1 a 1.1v 16gb 640gb/s hbm2e dram with a data-bus window-extension technique and a synergetic on-die ecc scheme*, in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 330–332, 2020.

[14] D. U. Lee, K. W. Kim, K. W. Kim, H. Kim, J. Y. Kim, Y. J. Park, J. H. Kim, D. S. Kim, H. B. Park, J. W. Shin, J. H. Cho, K. H. Kwon, M. J. Kim, J. Lee, K. W. Park, B. Chung, and S. Hong, *25.2 a 1.2v 8gb 8-channel 128gb/s high-bandwidth memory (hbm) stacked dram with effective microbump i/o test methods using 29nm process and tsv*, in *2014 IEEE International Solid-State Circuits Conference Digest of Technical Papers (ISSCC)*, pp. 432–433, 2014.

[15] J. C. Lee, J. Kim, K. W. Kim, Y. J. Ku, D. S. Kim, C. Jeong, T. S. Yun, H. Kim, H. S. Cho, Y. O. Kim, J. H. Kim, J. H. Kim, S. Oh, H. S. Lee, K. H. Kwon, D. B. Lee, Y. J. Choi, J. Lee, H. G. Kim, J. H. Chun, J. Oh, and S. H. Lee, *18.3 a 1.2v 64gb 8-channel 256gb/s hbm dram with peripheral-base-die architecture and small-swing technique on heavy load interface*, in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 318–319, 2016.

[16] D. U. Lee, H. S. Cho, J. Kim, Y. J. Ku, S. Oh, C. Dae Kim, H. W. Kim, W. Y. Lee, T. K. Kim, T. S. Yun, M. J. Kim, S. Lim, S. H. Lee, B. K. Yun, J. I. Moon, J. H. Park, S. Choi, Y. J. Park, C. K. Lee, C. Jeong, J. Lee, S. H. Lee, W. S. We, J. C. Yun, D. Lee, J. Shin, S. Kim, J. Lee, J. Choi, Y. Ju, M. Park, K. S. Lee, Y. Hur, D. Shim, S. Lee, J. Chun, and K. Jin, *22.3 a 128gb 8-high 512gb/s hbm2e dram with a pseudo quarter bank structure, power dispersion and an instruction-based at-speed pmbist*, in *2020 IEEE International Solid- State Circuits Conference - (ISSCC)*, pp. 334–336, 2020.

[17] K. Sohn, W. Yun, R. Oh, C. Oh, S. Seo, M. Park, D. Shin, W. Jung, S. Shin, J. Ryu, H. Yu, J. Jung, K. Nam, S. Choi, J. Lee, U. Kang, Y. Sohn, J. Choi, C. Kim, S. Jang, and G. Jin, *18.2 a 1.2v 20nm 307gb/s hbm dram with at-speed wafer-level i/o test scheme and adaptive refresh considering temperature distribution*, in *2016 IEEE International Solid-State Circuits Conference (ISSCC)*, pp. 316–317, 2016.

[18] J. Macri, R. Koduri, M. Mantor, and B. Black, *AMD's Next Generation GPU and Memory Architecture*, in *2015 IEEE Hot Chips 27 Symposium (HCS)*, IEEE, 2015.

[19] A. Rush, *Memory technology and applications*, in *2016 IEEE Hot Chips 28 Symposium (HCS)*, pp. 1–14, IEEE, 2016.

[20] M. Mantor and B. Sander, *Amd's radeon next generation gpu architecture*, in *2017 IEEE Hot Chips 29 Symposium (HCS)*, IEEE, 2017.

[21] J. Danskin and D. Foley, *Ultra-performance pascal gpu and nvlink interconnect*, in *2016 IEEE Hot Chips 28 Symposium (HCS)*, pp. 1–14, IEEE, 2016.

[22] J. Choquette, *Nvidia's volta gpu: Programmability and performance for gpu computing*, in *2017 IEEE Hot Chips 29 Symposium (HCS)*, IEEE, 2017.

[23] HMCC, "NVIDIA Ampere." https://www.anandtech.com/show/15801/nvidia-announces-ampere-architecture-and-a100-products, 2020.
Online.

[24] S. Shumarayev, *Stratix 10: Intel's 14nm Heterogeneous FPGA System-in-Package (SiP) Platform*, in *2017 IEEE Hot Chips 29 Symposium (HCS)*, IEEE, 2017.

[25] G. Singh and S. Ahmad, *Xilinx 16nm Datacenter Device Family with In-Package HBM and CCIX Interconnect*, in *2017 IEEE Hot Chips 29 Symposium (HCS)*, IEEE, 2017.

[26] J. T. Pawlowski, *Hybrid Memory Cube (HMC)*, in *2011 IEEE Hot Chips 23 Symposium (HCS)*, IEEE, 2011.

[27] HMCC, "Hybrid Memory Cube Specification 1.0." http://hybridmemorycube.org/specification-download/.

[28] HMCC, "Hybrid Memory Cube Specification 2.1." http://www.hybridmemorycube.org/specification-v2-download-form/, 2014.
Online.

[29] A. Kannan *et. al.*, *Enabling Interposer-based Disintegration of Multi-core Processors*, in *Proceedings of the 48th International Symposium on Microarchitecture*, MICRO-48, (New York, NY, USA), pp. 546–558, ACM, 2015.

[30] R. Huemoeller, "Through Silicon Via (TSV) Product Technology." Amkor Technology, Technical Report. Presented to IMAPS North Carolina Chapter, 2012.

[31] G. H. Loh, N. E. Jerger, A. Kannan, and Y. Eckert, *Interconnect-Memory Challenges for Multi-chip, Silicon Interposer Systems*, in *2015 Intl. Symp. on Memory Systems*, (Washington DC), pp. 3–10, October, 2015.

[32] Y. Lee and S. K. Lim, *Ultrahigh Density Logic Designs Using Monolithic 3-D Integration*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **32** (Dec, 2013) 1892–1905.

[33] C. Liu and S. K. Lim, *A Design Tradeoff Study with Monolithic 3D Integration*, in *Thirteenth International Symposium on Quality Electronic Design (ISQED)*, pp. 529–536, March, 2012.

[34] D. Stow, I. Akgun, W. Huangfu, Y. Xie, X. Li, and G. H. Loh, *Efficient system architecture in the era of monolithic 3d: Dynamic inter-tier interconnect and processing-in-memory*, in *Proceedings of the 56th Annual Design Automation Conference 2019*, DAC '19, (New York, NY, USA), Association for Computing Machinery, 2019.

[35] X. Wu *et. al.*, *Thermal-aware 3d ic designs*, *3D Integration for VLSI Systems* (2011) 313.

[36] "AMD Radeon™ R9 Series Graphics Cards with High-Bandwidth Memory." `http://www.amd.com/en-us/products/graphics/desktop/r9`.

[37] J. Macri *et. al.*, *AMD's next Generation GPU and Memory Architecture*, in *Hot Chips*, 2015.

[38] "NVLink, Pascal and Stacked Memory: Feeding the Appetite for Big Data." `http://devblogs.nvidia.com/parallelforall/nvlink-pascal-stacked-memory-feeding-appetite-big-data/`.

[39] F. Li *et. al.*, *Design and Management of 3D Chip Multiprocessors Using Network-in-Memory*, in *33rd International Symposium on Computer Architecture (ISCA)*, pp. 130–141, 2006.

[40] N. E. Jerger *et. al.*, *NoC Architectures for Silicon Interposer Systems: Why Pay for more Wires when you Can Get them (from your interposer) for Free?*, in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 458–470, Dec, 2014.

[41]

[42] B. Ganesh *et. al.*, *Fully-buffered DIMM memory architectures: Understanding mechanisms, overheads and scaling*, in *IEEE 13th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 109–120, IEEE, 2007.

[43] A. N. Udipi *et. al.*, *Combining Memory and a Controller with Photonics Through 3D-stacking to Enable Scalable and Energy-efficient Systems*, in *Proceedings of the 38th Annual International Symposium on Computer Architecture (ISCA)*, pp. 425–436, ACM, 2011.

[44] K. Therdsteerasukdi *et. al.*, *The DIMM tree architecture: A high bandwidth and scalable memory system*, in *IEEE 29th International Conference on Computer Design (ICCD)*, pp. 388–395, IEEE, 2011.

[45] T. J. Ham *et. al.*, *Disintegrated Control for Energy-efficient and Heterogeneous Memory Systems*, in *Proceedings of the 2013 IEEE 19th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 424–435, IEEE, 2013.

[46] F. Färber *et. al.*, *SAP HANA database: data management for modern business applications*, *ACM Sigmod Record* **40** (2012), no. 4 45–51.

[47] M. Zaharia *et. al.*, *Spark: cluster computing with working sets*, in *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, vol. 10, p. 10, 2010.

[48] A. Daglis *et. al.*, *Manycore Network Interfaces for In-Memory Rack-Scale Computing*, in *ACM/IEEE 42nd International Symposium on Computer Architecture (ISCA)*, pp. 567–579, ACM, 2015.

[49] G. Kim *et. al.*, *Multi-gpu system design with memory networks*, in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture*, pp. 484–495, IEEE Computer Society, 2014.

[50] J. Zhan *et. al.*, *A Unified Memory Network Architecture for In-Memory Computing in Commodity Servers*, in *Proceedings of the 49th ACM/IEEE International Symposum on Microarchitecture (MICRO)*, 2016.

[51] V. Raman *et. al.*, *DB2 with BLU acceleration: So much more than just a column store*, *Proceedings of the VLDB Endowment* **6** (2013), no. 11 1080–1091.

[52] J. Dean and S. Ghemawat, *MapReduce: simplified data processing on large clusters*, *Communications of the ACM* **51** (2008), no. 1 107–113.

[53] R. Ho, K. Mai, and M. Horowitz, *Managing wire scaling: a circuit perspective*, in *Interconnect Technology Conference, 2003. Proceedings of the IEEE 2003 International*, pp. 177–179, June, 2003.

[54] M. Ferdman *et. al.*, *Clearing the clouds: a study of emerging scale-out workloads on modern hardware*, *ACM SIGARCH Computer Architecture News* **40** (2012), no. 1 37–48.

[55] L. Wang *et. al.*, *Bigdatabench: a big data benchmark suite from internet services*, in *IEEE 20th International Symposium on High Performance Computer Architecture (HPCA)*, pp. 488–499, IEEE, 2014.

[56] "Redis Benchmark." `http://redis.io/topics/benchmarks`.

[57] H. Patil *et. al.*, *Pinpointing representative portions of large Intel® Itanium® programs with dynamic instrumentation*, in *IEEE/ACM 37th International Symposium on Microarchitecture (MICRO)*, pp. 81–92, IEEE Computer Society, 2004.

[58] "Spark 1.4.1." `http://spark.apache.org/downloads.html`.

[59] J. T. Pawlowski, *Hybrid Memory Cube: Breakthrough DRAM Performance with a Fundamentally Re-Architected DRAM Subsystem*, in *Hot Chips 23*, 2011.

[60] D. R. Resnick and M. Ignatowski, *Proposing an Abstracted Interface and Protocol for Computer Systems*, SAND2014 15795, Sandia National Laboratories, July, 2014.

[61] D. Roberts, A. Farmahini-Farahani, K. Cheng, N. Hu, D. Mayhew, and M. Ignatowski, *NMI: A New Memory Interface to Enable Innovation*, in *Hot Chips*, (Aspen, Colorado), January, 2015.

[62] Samsung Electronics Corp., "Samsung Begins Mass Producing World's Fastest DRAM - Based on Newest High Bandwidth Memory (HBM) Interface." http://news.samsung.com, Jan. 19,, 2016.

[63] Arstechnica, "HBM3: Cheaper, up to 64GB on-package, and terabytes-per-second bandwidth." http://arstechnica.com/gadgets/2016/08/hbm3-details-price-bandwidth/.

[64] P. Benson, *Dell^{TM} PowerEdge^{TM} Servers 2009 - Memory*, *Dell Technical Whitepaper* (February, 2009).

[65] Dell Corporation, *Memory Speeds and Population*, tech. rep., 2016. http://www.dell.com/learn/us/en/2684/campaigns/memory-speeds-and-population.

[66] "The Machine: A new kind of computer." https://www.labs.hpe.com/the-machine.

[67] R. Nair, S. Antao, C. Bertolli, P. Bose, J. Brunheroto, T. Chen, C. Cher, C. Costa, J. Doi, C. Evangelinos, B. Fleischer, T. Fox, D. Gallo, L. Grinberg, J. Gunnels, A. Jacob, P. Jacob, H. Jacobson, T. Karkhanis, C. Kim, J. Moreno, J. O'Brien, M. Ohmacht, Y. Park, D. Prener, B. Rosenburg, K. Ryu,

O. Sallenave, M. Serrano, P. Siegl, K. Sugavanam, and Z. Sura, *Active Memory Cube: A processing-in-memory architecture for exascale systems*, *IBM Journal of Research and Development* **59** (2015), no. 2/3 17–1.

[68] G. Kim, J. Kim, J.-H. Ahn, and J. Kim, *Memory-centric system interconnect design with hybrid memory cubes*, in *Intl. Conf. on Parallel Architectures and Compilation Techniques*, 2013.

[69] G. Kim, M. Lee, J. Jeong, and J. Kim, *Multi-GPU System Design with Memory Networks*, in *47th Intl. Symp. on Microarchitecture*, (Cambridge, UK), pp. 484–495, December, 2014.

[70] E. Kultursay, M. Kandemir, A. Sivasubramaniam, and O. Mutlu, *Evaluating STT-RAM as an Energy-efficient Main Memory Alternative*, in *2013 Intl. Symp. on Performance Analysis of Systems and Software*, pp. 256–267, IEEE, 2013.

[71] B. C. Lee, E. Ipek, O. Mutlu, and D. Burger, *Architecting Phase Change Memory As a Scalable Dram Alternative*, in *2009 Intl. Symp. on Computer Architecture*, ISCA '09, (New York, NY, USA), pp. 2–13, ACM, 2009.

[72] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, *Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support*, in *SC*, 2010.

[73] J. Meza, J. Chang, H. Yoon, O. Mutlu, and P. Ranganathan, *Enabling Efficient and Scalable Hybrid Memories Using Fine-granularity DRAM Cache Management*, *Computer Architecture Letters* **11** (2012), no. 2 61–64.

[74] M. Meswani, S. Balgodurov, D. Roberts, J. Slice, M. Ignatowski, and G. Loh, *Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories*, in *21st Intl. Symp. on High Performance Computer Architecture*, (San Francisco, CA), pp. 126–136, February, 2015.

[75] C. Sun, E. A. Leon, G. H. Loh, D. Roberts, K. Cameron, D. S. Nikolopoulos, and B. S. de Supinkski, *HpMC: An Energy-aware Management System of Multi-level Memory Architectures*, in *2015 Intl. Symp. on Memory Systems*, (Washington DC), October, 2015.

[76] X. Dong, Y. Xie, N. Muralimanohar, and N. P. Jouppi, *Simple but Effective Heterogeneous Main Memory with On-Chip Memory Controller Support*, in *2010 ACM/IEEE Int'l Conf. for High Performance Computing, Networking, Storage and Analysis*, SC '10, (Washington, DC, USA), pp. 1–11, IEEE Computer Society, 2010.

[77] X. Jiang, N. Madan, L. Zhao, M. Upton, R. Iyer, S. Makineni, D. Newell, Y. Solihin, and R. Balasubramonian, *CHOP: Adaptive Filter-Based DRAM Caching for CMP Server Platforms*, in *HPCA-16*, 2010.

[78] B. Ganesh, A. Jaleel, D. Wang, and B. Jacob, *Fully-buffered dimm memory architectures: Understanding mechanisms, overheads and scaling*, in *Proceedings of the 2007 IEEE 13th International Symposium on High Performance Computer Architecture*, HPCA '07, (Washington, DC, USA), pp. 109–120, IEEE Computer Society, 2007.

[79] S. Jeloka, R. Das, R. G. Dreslinski, T. Mudge, and D. Blaauw, *Hi-Rise: A high-radix switch for 3D integration with single-cycle arbitration*, in *47th Intl. Symp. on Microarchitecture*, (Cambridge, UK), pp. 471–483, December, 2014.

[80] W. Song, H. J. Jung, J. Ahn, J. Lee, and J. Kim, *Evaluation of Performance Unfairness in NUMA System Architecture*, *IEEE Computer Architecture Letters* **PP** (2016), no. 99 1–1.

[81] W. Pugh, *Skip lists: A probabilistic alternative to balanced trees*, *Communications of the Association for Computing Machinery* **33** (June, 1990) 668–0676.

[82] W. J. Dally, "Express cubes: improving the performance of k-ary n -cube interconnection networks." `https://apps.dtic.mil/sti/pdfs/ADA217117.pdf`, 1989.

[83] W. J. Dally, *Express cubes: improving the performance of k-ary n -cube interconnection networks*, *IEEE Transactions on Computers* **40** (Sep, 1991) 1016–1023.

[84] W. J. Dally, *Express channels for diminishing latency and increasing throughput in an interconnection network*, Dec. 12, 1995. US Patent 5,475,857.

[85] B. Grot, J. Hestness, S. W. Keckler, and O. Mutlu, *Express cube topologies for on-chip interconnects*, in *2009 IEEE 15th International Symposium on High Performance Computer Architecture*, pp. 163–174, 2009.

[86] J. F. Martínez, J. Torrellas, and J. Duato, *Improving the Performance of Bristled CC-NUMA Systems Using Virtual Channels and Adaptivity*, in *1999 Intl. Conf. on Supercomputing*, ICS '99, (New York, NY, USA), pp. 202–209, ACM, 1999.

[87] J. Balfour and W. J. Dally, *Design Tradeoffs for Tiled CMP On-chip Networks*, in *2006 Intl. Conf. on Supercomputing*, ICS '06, (New York, NY, USA), pp. 187–198, ACM, 2006.

[88] I. Advanced Micro Devices, "AMD SDK." http://developer.amd.com/tools-and-sdks.

[89] S. Che, M. Boyer, J. Meng, D. Tarjan, J. W. Sheaffer, S.-H. Lee, and K. Skadron, *Rodinia: A benchmark suite for heterogeneous computing*, in *IEEE Intl. Symp. on Workload Characterization*, 2009.

[90] S. Che, J. W. Sheaffer, M. Boyer, L. G. Szafaryn, L. Wang, and K. Skadron, *A characterization of the rodinia benchmark suite with comparison to contemporary cmp workloads*, in *IEEE Intl. Symp. on Workload Characterization*, 2010.

[91] AMD Advanced Micro Devices, Inc., "AMD Graphics Core Next (GCN) Architecture." https://www.amd.com/Documents/GCN_Architecture_whitepaper.pdf, June, 2012.

[92] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, *Garnet: A detailed on-chip network model inside a full-system simulator*, in *Performance Analysis of Systems and Software, 2009. ISPASS 2009. IEEE International Symposium on*, pp. 33–42, IEEE, 2009.

[93] G. Kim, J. Kim, J. H. Ahn, and Y. Kwon, *Memory Network: Enabling Technology for Scalable Near-Data Computing*, in *2nd Workshop on Near-Data Processing*, (Cambridge, UK), 2014.

[94] E. Cooper-Balis, P. Rosenfeld, and B. Jacob, *Buffer-on-board Memory Systems*, in *39th Intl. Symp. on Computer Architecture*, (Portland, OR), pp. 392–403, June, 2012.

[95] J. Zhan, I. Akgun, J. Zhao, A. Davis, P. Faraboschi, Y. Wang, and Y. Xie, *A Unified Memory Network Architecture for In-Memory Computing in Commodity Servers*, in *2016 Intl. Symp. on Microarchitecture*, IEEE, 2016.

[96] E. Azarkhish, D. Rossi, I. Loi, and L. Benini, *A Logic-base Interconnect for Supporting Near Memory Computation in the Hybrid Memory Cube*, in *2nd Workshop on Near-Data Processing*, (Cambridge, UK), 2014.

[97] S. Pugsley, J. Jestes, H. Zhang, R. Balasubramonian, V. Srinivasan, A. Buyuktosunoglu, A. Davis, and F. Li, *NDC: Analyzing the impact of 3D-stacked memory+logic devices on MapReduce workloads*, in *2014 Intl. Symp. on Performance Analysis of Systems and Software*, pp. 190–200, IEEE, 2014.

[98] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, *A scalable processing-in-memory accelerator for parallel graph processing*, in *2015 Intl. Symp. on Computer Architecture*, pp. 105–117, ACM, 2015.

[99] M. Gao, G. Ayers, and C. Kozyrakis, *Practical Near-Data Processing for In-Memory Analytics Frameworks*, in *2015 Intl. Conf. on Parallel Architectures and Compilation Techniques*, pp. 113–124, Oct, 2015.

[100] I. Akgun, J. Zhan, Y. Wang, and Y. Xie, *Scalable Memory Fabric for Silicon Interposer-Based Multi-Core Systems*, in *2016 Intl. Conf. on Computer Design*, 2016.

[101] P. Rosenfeld, *Performance Exploration of the Hybrid Memory Cube*. PhD thesis, University of Maryland, College Park, 2014.

[102] DARPA, "Common Heterogeneous Integration and IP Reuse Strategies (CHIPS)." `https://www.darpa.mil/program/common-heterogeneous-integration-and-ip-reuse-strategies`, 2017.

[103] Intel, *Advanced Interface Bus Specification*, 2018. `https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/manual/advanced-interface-bus-specification.pdf`.

[104] D. Kehlet, *Accelerating Innovation Through A Standard Chiplet Interface: The Advanced Interface Bus (AIB)*. `https://www.intel.com/content/dam/www/public/us/en/documents/white-papers/accelerating-innovation-through-aib-whitepaper.pdf`.

[105] T. M. Brewer, *Interface for data communication between chiplets or other integrated circuits on an interposer*, Aug. 8, 2019. US Patent App. 16/266,033.

[106] MoSys, *GigaChip Interface (IP)*. `https://mosys.com/technology/gigachip-interface/`.

[107] G. Interface, *GigaChip Interface*. `http://www.gigachipinterface.com/index.html`.

[108] S. Zogopoulos and W. Namgoong, *High-speed single-ended parallel link based on three-level differential encoding*, *IEEE Journal of Solid-State Circuits* **44** (2009), no. 2 549–557.

[109] P. Ou, J. Zhang, H. Quan, Y. Li, M. He, Z. Yu, X. Yu, S. Cui, J. Feng, S. Zhu, J. Lin, M. Jing, X. Zeng, and Z. Yu, *A 65nm 39gops/w 24-core processor with 11tb/s/w packet-controlled circuit-switched double-layer network-on-chip and heterogeneous execution array*, in *2013 IEEE International Solid-State Circuits Conference Digest of Technical Papers*, pp. 56–57, 2013.

[110] S. K. Samal, D. Nayak, M. Ichihashi, S. Banna, and S. K. Lim, *Monolithic 3d ic vs. tsv-based 3d ic in 14nm finfet technology*, in *2016 IEEE SOI-3D-Subthreshold Microelectronics Technology Unified Conference (S3S)*, pp. 1–2, 2016.

[111] S. Panth, K. Samadi, Y. Du, and S. K. Lim, *Shrunk-2-d: A physical design methodology to build commercial-quality monolithic 3-d ics*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **36** (2017), no. 10 1716–1724.

[112] F. Li, C. Nicopoulos, T. Richardson, Y. Xie, V. Narayanan, and M. Kandemir, *Design and Management of 3D Chip Multiprocessors Using Network-in-Memory*, in *Proceedings of the 33rd Annual International Symposium on Computer Architecture*, (Washington, DC, USA), pp. 130–141, IEEE Computer Society, 2006.

[113] J. Kim, C. Nicopoulos, D. Park, R. Das, Y. Xie, V. Narayanan, M. S. Yousif, and C. R. Das, *A Novel Dimensionally-decomposed Router for On-chip Communication in 3D Architectures*, in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, (New York, NY, USA), pp. 138–149, ACM, 2007.

[114] D. Stow, I. Akgun, W. Huangfu, Y. Xie, X. Li, and G. H. Loh, *Efficient System Architecture in the Era of Monolithic 3D: Dynamic Inter-tier Interconnect and Processing-in-Memory*, in *Proceedings of the 56th Annual Design Automation Conference (DAC)*, (New York, NY, USA), pp. 100:1–100:4, ACM, 2019.

[115] D. Park, S. Eachempati, R. Das, A. K. Mishra, Y. Xie, N. Vijaykrishnan, and C. R. Das, *MIRA: A Multi-layered On-Chip Interconnect Router Architecture*, in *International Symposium on Computer Architecture (ISCA)*, pp. 251–261, June, 2008.

[116] S. Das, J. R. Doppa, P. P. Pande, and K. Chakrabarty, *Monolithic 3D-Enabled High Performance and Energy Efficient Network-on-Chip*, in *IEEE International Conference on Computer Design (ICCD)*, pp. 233–240, Nov, 2017.

[117] Y. Lee, D. Limbrick, and S. K. Lim, *Power Benefit Study for Ultra-High Density Transistor-Level Monolithic 3D ICs*, in *50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–10, May, 2013.

[118] S. Panth, K. Samadi, Y. Du, and S. K. Lim, *Power-Performance Study of Block-Level Monolithic 3D-ICs Considering Inter-Tier Performance Variations*, in *51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pp. 1–6, June, 2014.

[119] S. K. Samal, S. Panth, K. Samadi, M. Saedi, Y. Du, and S. K. Lim, *Fast and Accurate Thermal Modeling and Optimization for Monolithic 3D ICs*, in *Proceedings of the 51st Annual Design Automation Conference (DAC)*, (New York, NY, USA), pp. 206:1–206:6, ACM, 2014.

[120] D. Lee, S. Das, J. R. Doppa, P. P. Pande, and K. Chakrabarty, *Performance and Thermal Tradeoffs for Energy-Efficient Monolithic 3D Network-on-Chip*, *ACM Trans. Des. Autom. Electron. Syst.* **23** (Aug., 2018) 60:1–60:25.

[121] S. Panth, K. Samadi, Y. Du, and S. K. Lim, *Shrunk-2-D: A Physical Design Methodology to Build Commercial-Quality Monolithic 3-D ICs*, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* **36** (Oct, 2017) 1716–1724.

[122] M. Badr and N. E. Jerger, *SynFull: Synthetic Traffic Models Capturing Cache Coherent Behaviour*, in *ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pp. 109–120, June, 2014.

[123] Nan Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J. Kim, and W. J. Dally, *A Detailed and Flexible Cycle-Accurate Network-on-Chip Simulator*, in *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 86–96, April, 2013.