BOSTON UNIVERSITY

COLLEGE OF ENGINEERING

Dissertation

RESOURCE AND THERMAL MANAGEMENT IN 3D-STACKED MULTI-/MANY-CORE SYSTEMS

by

TIANSHENG ZHANG

B.S., Harbin Institute of Technology, 2010

Submitted in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy

2017

ProQuest Number: 10254654

All rights reserved

INFORMATION TO ALL USERS
The quality of this reproduction is dependent upon the quality of the copy submitted.

In the unlikely event that the author did not send a complete manuscript
and there are missing pages, these will be noted. Also, if material had to be removed,
a note will indicate the deletion.

ProQuest 10254654

Published by ProQuest LLC (2017). Copyright of the Dissertation is held by the Author.

All rights reserved.
This work is protected against unauthorized copying under Title 17, United States Code
Microform Edition © ProQuest LLC.

ProQuest LLC.
789 East Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106 – 1346

© 2017 by
TIANSHENG ZHANG
All rights reserved

Approved by

First Reader

Ayse K. Coskun, Ph.D.
Associate Professor of Electrical and Computer Engineering

Second Reader

Ajay Joshi, Ph.D.
Associate Professor of Electrical and Computer Engineering

Third Reader

Manuel Egele, Ph.D.
Assistant Professor of Electrical and Computer Engineering

Fourth Reader

Michel Kinsy, Ph.D.
Assistant Professor of Electrical and Computer Engineering

If little labour, little are our gains:
Man's fortunes are according to his pains.

Robert Herrick

Acknowledgments

First and foremost, I want to express my deepest gratitude to my PhD advisor, Prof. Ayse K. Coskun. She has been a tremendous mentor for me. I would like to thank her for her inspirational guidance, endless support and encouragement, as well as her patience throughout my PhD study. I appreciate all her contributions, her time, ideas, and funding that have made my PhD experience productive and stimulating. Her advice on both research and my future career has been priceless.

In addition to my advisor, I am also grateful to Prof. Ajay Joshi for his valuable advice and feedback. Discussions with him have always been enlightening and productive. I feel very fortunate that I have worked with him.

I would also like to thank the rest of my thesis committee, Prof. Manuel Egele and Prof. Michel Kinsy, for their precious time and insightful comments.

I extend special thanks to Prof. Andrew B. Kahng, whose research advice and feedback not only substantially helped improve quality of my papers, but also provided guidelines for my future career. I must acknowledge Prof. Satish Narayanasamy for his suggestions and instructions during the heterogeneous memory management project. I also thank Prof. Jonathan Klamkin for guiding me with his vast knowledge on silicon-photonic technology.

I sincerely thank Prof. Yusuf Leblebici for the summer internship opportunity at EPFL, which led to my very first publication in my PhD study. I would also like to acknowledge Dr. Alessandro Cevrero, Dr. Giulia Beanato, and the other Microelectronic Systems Lab members for their enormous support during my stay in Lausanne, Switzerland.

I would like to express my appreciation to Dr. Yuan Lin and Dr. Henry Cox for their guidance and valuable feedback during my internship at MediaTek, Inc. Discussions with them were always inspiring and extremely helpful. I would also like

to thank Mr. Darrin Johnson for the summer internship opportunity at Oracle Corp., Santa Clara, CA.

I am grateful to my collaborators and co-authors: Prof. Jose Abellan at Catholic University of Murcia, John Recchio, Vaishnav Srinivas, Anjun Gu at the University of California, San Diego, Shaizeen Aga at University of Michigan, and Raphael Landaverde at Microsoft, for productive collaborations and all the helpful discussions.

I am also grateful to those who made my PhD study at Boston University pleasant and joyful, including Prof. Martin Herbordt, Prof. Peter Galvin and Ms. Cali Stephens.

I thank the members of the PEACLab research group, the ICSG research group and the CAAD research group. They have been great sources of friendship as well as good advice and collaborations. I would like to especially acknowledge Dr. Jie Meng, Dr. Chao Chen, Dr. Schuyler Eldridge, Fulya Kaplan, Onur Sahin, and Yenai Ma for their significant help and encouragement during my time at Boston University.

Finally, I would like to thank my family for their unconditional support and understanding during my PhD career. Especially, I want to thank my fiance, Liang Li, for her continuous love, company, and compassion. I would not have been able to overcome the stress and difficulties without their help and support. I would like to dedicate this dissertation to all my family.

The research that forms the basis of this dissertation has been partially funded by NSF CAREER grant CNS-1149703.

The contents of Chapter 3 are in part reprints of the material from the papers, *Jie Meng, Tiansheng Zhang, and Ayse K. Coskun, "Dynamic Cache Pooling for Improving Energy Efficiency in 3D Stacked Multicore Processors"*, in Proceedings of the IFIP/IEEE International Conference on Very Large Scale Integration, 2013, and *Tiansheng Zhang, Jie Meng, and Ayse K. Coskun, "Dynamic Cache Pooling in*

3D Multicore Processors", in ACM Journal on Emerging Technologies in Computing Systems, 2015.

The contents of Chapter 4 are in part reprints of the material from the papers, *Tiansheng Zhang, Alessandro Cevrero, Giulia Beanato, Panagiotis Athanasopoulos, Ayse K. Coskun, and Yusuf Leblebici, "3D-MMC: A Modular 3D Multi-Core Architecture with Efficient Resource Pooling"*, in Proceedings of Design, Automation and Test in Europe (DATE), 2013, and *Tiansheng Zhang, Shaizeen Aga, Satish Narayanasamy, and Ayse K. Coskun, "MOCA: Memory Object Classification and Allocation in Heterogeneous Memory Systems"* (in review).

The contents of Chapter 5 are in part reprints of the material from the papers, *Tiansheng Zhang, Jose L. Abellan, Ajay Joshi, and Ayse K. Coskun, "Thermal Management of Manycore Systems with Silicon-Photonic Networks"*, in Proceedings of DATE, 2014, *Jose L. Abellan, Ayse K. Coskun, Anjun Gu, Warren Jin, Ajay Joshi, Andrew B. Kahng, Jonathan Klamkin, Cristian Morales, John Recchio, Vaishnav Srinivas, and Tiansheng Zhang, "Adaptive Tuning of Photonic Devices in a Photonic NoC Through Dynamic Workload Allocation"*, to appear in IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems, 2016, *Chao Chen, Tiansheng Zhang, Pietro Contu, Jonathan Klamkin, Ayse K. Coskun, and Ajay Joshi, "Sharing and Placement of On-chip Laser Sources in Silicon-Photonic NoCs"*, in Proceedings of International Symposium on Networks-on-Chip, 2014, and *Ayse K. Coskun, Anjun Gu, Warren Jin, Ajay Joshi, Andrew B. Kahng, Jonathan Klamkin, Yenai Ma, John Recchio, Vaishnav Srinivas, and Tiansheng Zhang, "Cross-layer Floorplan Optimization for Silicon Photonic NoCs In Many-core Systems"*, in Proceedings of DATE, 2016.

RESOURCE AND THERMAL MANAGEMENT IN 3D-STACKED MULTI-/MANY-CORE SYSTEMS

TIANSHENG ZHANG

Boston University, College of Engineering, 2017

Major Professor: Ayse K. Coskun, Ph.D.
Associate Professor of Electrical and Computer
Engineering

ABSTRACT

Continuous semiconductor technology scaling and the rapid increase in computational needs have stimulated the emergence of multi-/many-core processors. While up to hundreds of cores can be placed on a single chip, the performance capacity of the cores cannot be fully exploited due to high latencies of interconnects and memory, high power consumption, and low manufacturing yield in traditional (2D) chips. 3D stacking is an emerging technology that aims to overcome these limitations of 2D designs by stacking processor dies over each other and using through-silicon-vias (TSVs) for on-chip communication, and thus, provides a large amount of on-chip resources and shortens communication latency. These benefits, however, are limited by challenges in high power densities and temperatures.

3D stacking also enables integrating heterogeneous technologies into a single chip. One example of heterogeneous integration is building many-core systems with silicon-photonic network-on-chip (PNoC), which reduces on-chip communication latency significantly and provides higher bandwidth compared to electrical links. However, silicon-photonic links are vulnerable to on-chip thermal and process variations. These variations can be countered by actively tuning the temperatures of optical devices through micro-heaters, but at the cost of substantial power overhead.

This thesis claims that unearthing the energy efficiency potential of 3D-stacked systems requires intelligent and application-aware resource management. Specifically, the thesis improves energy efficiency of 3D-stacked systems via three major components of computing systems: cache, memory, and on-chip communication. We analyze characteristics of workloads in computation, memory usage, and communication, and present techniques that leverage these characteristics for energy-efficient computing.

This thesis introduces 3D cache resource pooling, a cache design that allows for *flexible heterogeneity* in cache configuration across a 3D-stacked system and improves cache utilization and system energy efficiency. We also demonstrate the impact of resource pooling on a real prototype 3D system with scratchpad memory.

At the main memory level, we claim that utilizing heterogeneous memory modules and memory object level management significantly helps with energy efficiency. This thesis proposes a memory management scheme at a finer granularity: memory object level, and a page allocation policy to leverage the heterogeneity of available memory modules and cater to the diverse memory requirements of workloads.

On the on-chip communication side, we introduce an approach to limit the power overhead of PNoC in (3D) many-core systems through cross-layer thermal management. Our proposed thermally-aware workload allocation policies coupled with an adaptive thermal tuning policy minimize the required thermal tuning power for PNoC, and in this way, help broader integration of PNoC. The thesis also introduces techniques in placement and floorplanning of optical devices to reduce optical loss and, thus, laser source power consumption.

Contents

1 Introduction **1**

1.1 Problem Statement 3

1.2 Thesis Contribution 6

1.3 Organization 8

2 Background and Related Work **10**

2.1 3D Stacking Technology 10

2.2 On-Chip Resource Management 12

2.3 Heterogeneous Memory Systems 14

2.4 Silicon-Photonic Network-on-Chips 16

2.4.1 Silicon-Photonic Link 17

2.4.2 Thermal Sensitivity of Silicon-Photonic Devices 18

2.4.3 Thermal Management in PNoCs 20

2.5 Distinguishing Aspects from Prior Work 22

3 Cache Resource Management in 3D Multi-core Systems **25**

3.1 Overview 25

3.2 Cache Resource Pooling in 3D Stacked Systems 28

3.2.1 Cache Resource Pooling Architecture 28

3.2.2 Cache Resource Management Policy 33

3.2.3 Experimental Methodology 41

3.2.4 Evaluation 44

3.3 Summary 50

4 Memory Resource Management in 3D Multi-core Systems 51

4.1 Memory Access Scheduling in 3D-MMC 51

4.1.1 3D Modular Multi-core Architecture Design 52

4.1.2 Resource Pooling For Efficient Memory Access 58

4.1.3 Performance evaluation 63

4.2 Object-Level Memory Management in Heterogeneous Memory Systems 66

4.2.1 Motivation 68

4.2.2 MOCA: Memory Object Classification and Allocation 69

4.2.3 Implementation 73

4.2.4 Experimental Results 76

4.3 Summary 79

5 Thermal Management of 3D Stacked Many-core Systems with PNoC 81

5.1 Many-core Systems with PNoC 81

5.1.1 Many-core System Architecture 82

5.1.2 PNoC Design Flow 85

5.2 Experimental Methodology 87

5.2.1 Performance and Power Simulation 88

5.2.2 Thermal Simulation 90

5.3 Runtime Thermal Management Through Workload Allocation 92

5.3.1 Workload Allocation Policies 92

5.3.2 Frequency Tuning Methods 97

5.3.3 Experimental Results 101

5.3.4 Comparison with Optimal Workload Allocation 112

5.4 Design-time Power Management Through Laser Source Placement . . 113

5.4.1 Laser source power model 114

5.4.2 Laser source power, efficiency and temperature tradeoffs . . . 115
5.4.3 On-chip laser source sharing and placement strategy 118
5.4.4 Case Studies 119
5.5 Design-time Thermal Management Through PNoC Floorplanning . . 123
5.5.1 MILP-Based PNoC Floorplan Optimization 124
5.5.2 Experimental Results and Discussion 132
5.6 Summary 136
6 Conclusions and Future Directions 138
6.1 Summary of Major Contributions 138
6.2 Future Research Directions 140
6.2.1 Heterogeneous Memory Architecture in 3D Systems 140
6.2.2 Thermal/Power Management for PNoC with On-chip Laser Sources and 2.5D Integration 142
References 144
Curriculum Vitae 156

List of Tables

3.1 Core Architecture Parameters 41

3.2 Main Memory Access Latency for the 3D CRP System 42

3.3 Benchmark classification according to memory-intensiveness and cache-hungriness 44

3.4 4-core system workload sets 44

3.5 16-core system workload sets. *nch, lch, mch, hch,ach* represent non-cache-hungry, low-cache-hungry, med-cache-hungry, high-cache-hungry and all-cache-hungry, respectively. 45

4.1 Power Consumption and Thermal Properties of 3D-MMC 57

4.2 Execution time of different shared memory access scenarios when all eight cores are active. 65

4.3 Timing and architectural parameters for various memory modules used in this work. 75

5.1 Notations used in this work. 83

5.2 Classification of applications. 89

5.3 Properties of the materials in our target system. 91

5.4 Workload combinations. HP: High-Power; MP: Medium-Power; LP: Low-Power. 103

5.5 Running times of jobs (Unit: *ms*). 103

5.6 Architectural-level parameters for three tested PNoCs. U-shaped and W-shaped layouts are shown in Fig. 5·1. 122

5.7 Experimental configurations studied. 132
5.8 Losses in PNoCs.(Joshi et al., 2009) 133

List of Figures

1·1 40 years of microprocessor trend data (Rupp, 2015). 2

2·1 (a) An example of 3D face-to-back stacked system using TSVs. (b) Micrograph of TSVs structure (Courtesy of LSM, EPFL). 10

2·2 A silicon photonic link. 17

2·3 Impact of resonant frequency mismatch. Case 1: Small mismatch reduces the filtered optical power; Case 2: Large mismatch may result in a ring to filter the data of its neighboring ring in the frequency domain. 19

3·1 IPC of SPEC CPU 2006 benchmarks for increasing L2 cache size. The IPC values are normalized with respect to using a 256 KB L2 cache. 25

3·2 Power consumption of SPEC CPU 2006 benchmarks under cache sizes from 256 KB to 2048 KB. 26

3·3 IPC improvement of SPEC CPU 2006 benchmarks using stacked DRAM in comparison to off-chip DRAM. 26

3·4 L2 MPKI of SPEC CPU 2006 benchmarks under cache sizes from 256 KB to 2048 KB. 27

3·5 Proposed 3D system with cache resource pooling versus 3D systems with fixed 1 MB caches. In (a), cores are able to access caches on the adjacent layers through the TSVs. 29

3·6 Cache resource pooling implementation. 30

3·7 (a) The cross-section view of large 3D-CRP system; (b) An example showing cache resource pooling within a column. 32

3·8 A flow chart illustrating our runtime job allocation and cache resource pooling policy. (*) p_i represents the predicted IPC improvement for each job when running with 4 cache partitions compared to running with 1 partition. (**) Condition is checked only if J_i and J_j are competing for the same partition. 34

3·9 The 2-stage intra-column runtime job allocation and cache resource pooling policy. 36

3·10 The illustration of inter-column job allocation algorithm. 38

3·11 An example of job allocation in a 16-core 3D system. 39

3·12 Time overhead (μs) of cache cold start effect for all applications under cache sizes from 256 KB to 2048 KB 40

3·13 The relationship between memory access arrival rate and memory controller queuing delay. The data points from left to right represent the memory access arrival rate of one bzip2, two instances of bzip2 and four instances of bzip2, respectively. 43

3·14 Normalized IPC, EDP and EDAP of low-power 3D-CRP system and the 3D baseline systems with 1 MB fixed caches, 2 MB fixed caches and 1 MB caches with selective cache way. 46

3·15 Normalized IPC, EDP and EDAP of high-performance 3D-CRP system and the 3D baseline systems with 1 MB fixed caches, 2 MB fixed caches and 1 MB caches with selective cache way. 47

3·16 Normalized IPC, EDP and EDAP of 16-core low-power 3D-CRP system and the 3D baseline systems with 1 MB fixed caches, 2 MB fixed caches and 1 MB caches with selective cache way. ICA refers to Inter-column job allocation. 48

4·1 Overview of the 3D-MMC built with stacking identical layers. The figure does not show the data TSVs for clarity. 52
4·2 (a) PE internal architecture; (b) PS internal architecture. 54
4·3 (a) Clock distribution and propagation between two layers using three redundant TSVs; (b) LayerID generation and propagation between two layers using three redundant TSVs. 55
4·4 The figure demonstrates the peak temperatures at steady state for a single layer as well as 2-, 4-, and 8-layered stack. On the right, we show the thermal map of the top layer for the 2-layered stacks. Thermal variations are similarly low (limited to a few degrees only) for 4- and 8-layered stacks. 58
4·5 Comparison of execution time of *Memory Stress* benchmark when all cores access local memory, all cores access remote memory, and when memory resource pooling is applied. 59
4·6 Performance under different workload allocation and scheduling combinations. (a) is the baseline, (b) has the same schedule with (a) but has fewer remote memory accesses, while (c) has the same number of remote memory accesses but has a different schedule. 60
4·7 Workload schedules for task level resource pooling. (a). Four threads accessing remote shared memory at the same time–*4-thread-RSM*; (b). (1)-(3): Two threads accessing remote shared memory at the same time–*2-thread-RSM*; (c). (1)-(3): One thread accessing remote shared memory–*1-thread-RSM*. 61
4·8 Test results of different memory resource pooling schedules and the optimal schedule's curve based on Eqn. (1). 63
4·9 Performance improvement compared to single core. 64

4·10 Memory access characteristics of memory objects for selected SPEC CPU2006 applications. The x-axis shows different memory objects in the profiled applications, and each object's footprint is denoted on the bar (in MB). For *milc* and *h264ref*, we only mark the objects larger than 1 MB. 67
4·11 The workflow of MOCA. The profiling stage uniquely names objects and profiles their memory access behavior. Classification stage uses this information to classify objects. At runtime, each memory object is allocated pages from its best-fitting memory module based on its classification. . . . 70
4·12 An example of memory object naming convention. 70
4·13 Classification of memory objects. 71
4·14 Virtual and physical memory space separation for MOCA support in real systems. 73
4·15 Memory access statistics (average ROB head stall cycles per L2 miss and L2 MPKI) of selected SPEC applications. The letter in each bar represents the category of that application. L: latency-sensitive; B: bandwidth-sensitive; P: neither (placed in low-power memory). 76
4·16 Performance of homogeneous memory systems and heterogeneous memory systems with different page allocation algorithms. 77
4·17 System ED^{2}P of homogeneous memory systems and heterogeneous memory systems with different page allocation algorithms. 78
4·18 Memory ED^{2}P of homogeneous memory systems and heterogeneous memory systems with different page allocation algorithms. 79

5·1 Target many-core system with a PNoC (a), many-core systems with 8-ary 3-stage Clos topology and shifted physical layouts (b)-(c), and many-core systems with different logical topology and physical layout combinations (d)-(e). (d) is designed with 16-ary 3-stage Clos topology and W-shape physical layout; (e) is designed with 8-ary 3-stage Clos topology and chain-shape physical layout. 82
5·2 PNoC design flow chart. 85
5·3 Our performance (Carlson et al., 2011), power (Thoziyoor et al., 2008; Li et al., 2009) and thermal (Skadron et al., 2003) simulation setup for modeling many-core systems with a PNoC. 87
5·4 Thermal dependence between workload distribution and optical device frequency control power. 91
5·5 (a) Classification of $RD0$ cores and (b) an example of *RingAware* allocation in a 64-core system. 94
5·6 Illustration of *FreqAlign* workload allocation policy and adaptive frequency tuning AFT policy. Every thread allocated by *FreqAlign* increases the temperatures of ring groups and causes a downward shift in their frequencies. When all threads are allocated, thermal tuning is used to bring all ring groups to the lowest common resonant frequency. Above, ring groups 1 and 3, as well as the laser source, are tuned to match the resonant frequency of ring group 2. 99
5·7 Average resonant frequency differences when using *Clustered*, *RingAware* and *FreqAlign* workload allocation policy for U-shape layout with 8-ary 3-stage Clos topology shown in Fig. 5·1(a). Each bar represents a workload and utilization combination case. 103

5·8 Average optical tuning power when using localized tuning for *Clustered*, *RingAware* and *FreqAlign* workload allocation policy for 8-ary 3-stage Clos topology with U-shape layout in Fig. 5·1(a). *TFT*: Target Frequency Tuning; *AFT*: Adaptive Frequency Tuning. 104

5·9 Average resonant frequency difference when comparing *RingAware* and *FreqAlign* workload allocation policy for U-shape layout with 8-ary 3-stage Clos topology and a wavelength variation of 400 *pm/cm* in multiple directions due to process variations. The process variation cases and tuning power reduction of *FreqAlign+AFT* compared to *RingAware+AFT* are shown in the captions of subfigures. 106

5·9 Average resonant frequency difference when comparing *RingAware* and *FreqAlign* workload allocation policy for U-shape layout with 8-ary 3-stage Clos topology and a wavelength variation of 400 *pm/cm* in multiple directions due to process variations. The process variation cases and tuning power reduction of *FreqAlign+AFT* compared to *RingAware+AFT* are shown in the captions of subfigures. 107

5·10 Process variation directions considered for case study. 108

5·11 Comparison of average resonant frequency difference among ring groups for different PNoC logical topology and physical layout combinations between *RingAware* and *FreqAlign*. The PNoC cases and tuning power reduction of *FreqAlign+AFT* compared to *RingAware+AFT* are shown in captions of subfigures. 109

5·11 Comparison of average resonant frequency difference among ring groups for different PNoC logical topology and physical layout combinations between *RingAware* and *FreqAlign*. The PNoC cases and tuning power reduction of *FreqAlign*+AFT compared to *RingAware*+AFT are shown in captions of subfigures. 110

5·12 Runtime optical frequency difference trace for transient case with large temporal on-chip temperature gradients prior to applying tuning methods. 111

5·13 Floorplan of a 2× 4 system. 112

5·14 Ranking percentile of *FreqAlign*, *RingAware* and *Clustered* that indicates the ranking of these workload allocation algorithm among all possible workload allocation solutions for ten randomly generated power profiles (varying from 0.4 W to 2.8 W) for cores. 113

5·15 (a) P-I characteristics of a laser source, (b) WPE vs. input current, and (c) WPE vs. laser source lengths at various temperatures. 116

5·16 Laser source temperature vs. electrical input power for a 300 μm × 50 μm laser source. 117

5·17 Wall-plug efficiency vs. optical output power by the laser source for different granularities of sharing while a background logical layer operates at 0.4 W per core (a) and 0.7 W per core (b). 117

5·18 Total laser power vs. waveguide loss for various sharing scenarios and placements of on-chip laser sources. (a) and (b) compare various topologies with U-shaped layout. (b) and (c) compare various layouts for 16-ary 3-stage Clos topology. We assume each core in the logical layer consumes a power of 0.7 W. 120

5·19 (a) Example of chip floorplan to illustrate our terminology. (b) A vertex and its surrounding edges in the routing graph. (c) 3-stage Clos topology with 8 router groups per stage. 126

5·20 The floorplan optimization flow. 130

5·21 Core impact matrix generation: (a) illustrative floorplan with 16 tiles (64 cores) and nine potential router group positions; (b) sample core impact calculation for router group (1,3); (c) sample core impact calculation for router group (2,2); (d) a 1x9 core impact array generated for the floorplan. 131

5·22 Six power profiles studied. Darker tiles indicate higher power cores. . 133

5·23 Accumulated thermal weight profile and optimal floorplan vs. N_{cores}. 134

5·24 Accumulated thermal weight profile and optimal floorplan vs. AR. . . 135

5·25 Accumulated thermal weight profile on the first row, and optimal floorplan with WPE of 5% and 15% on the second and third row respectively for power profiles (a) - (f) in Figure 5·22. 136

6·1 Thermal maps of single chip case and 16-chiplet case with different spacings. 143

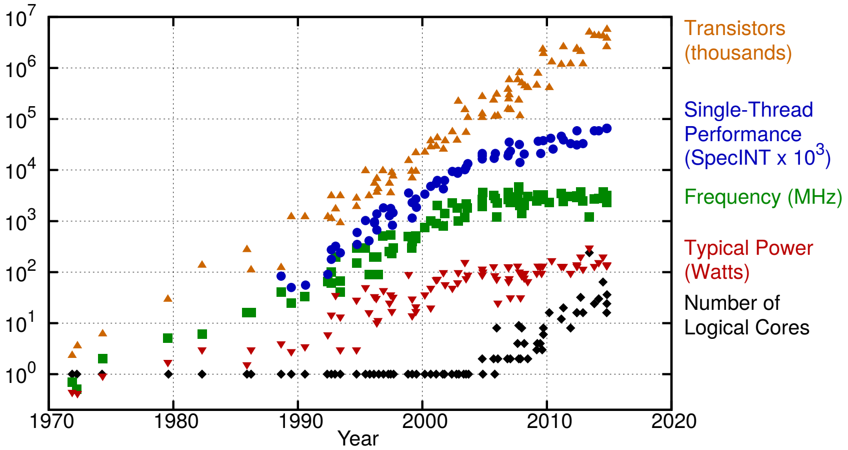
List of Abbreviations

CMOS	....	Complementary Metal-Oxide-Semiconductor
DRAM	....	Dynamic Random Access Memory
DVFS	....	Dynamic Voltage-Frequency Scaling
EDAP	....	Energy Delay Area Product
EDP	....	Energy Delay Product
ENoC	....	Electrical Network-on-Chip
FSR	....	Finite Spectral Range
FWHM	....	Full Width at Half Maximum
GPU	....	Graphics Processing Unit
IPC	....	Instructions Per Cycle
LLC	....	Last Level Cache
LPDRAM	....	Low Power DRAM
MILP	....	Mixed-Integer Linear Programming
MPKI	....	Misses Per Kilo Instructions
NoC	....	Network-on-Chip
OS	....	Operating System
P & R	....	Place and Route
PCM	....	Phase Change Memory
PE	....	Processing Element
PNoC	....	Silicon-photonic Network-on-Chip
PS	....	Peripheral Subsystem
PTE	....	Page Table Entry
RLDRAM	....	Reduced Latency DRAM
ROB	....	Reorder Buffer
SCC	....	Single-chip Cloud Computer
SOI	....	Silicon on Insulator
TLB	....	Translation Lookaside Buffer
TOC	....	Thermo-Optic Coefficient
TSVs	....	Through-Silicon-Vias
WDM	....	Wavelength-Division Multiplexing
WID	....	With-In Die
WPE	....	Wall-Plug Efficiency

Chapter 1

Introduction

Continuous semiconductor technology scaling has led to a transition from single-core to multi-core processors, and the trend is now moving towards many-core architectures (Owens et al., 2007), as shown in Fig. 1·1. Using traditional (2D) chip design methods, as the number of cores per chip grows, chip area increases, which worsens manufacturing yield in return. In addition, the communication latency among on-chip resources also increases along with chip area and restricts the performance of many-core systems. 3D stacking is an emerging integration technology that has the potential of overcoming these limitations of 2D chip design (Loh, 2008). Instead of expanding chip size horizontally, 3D stacking utilizes the vertical dimension to integrate more resources by stacking processor dies on top of each other. Since every die is manufactured separately, yield does not decrease as the number of on-chip cores grows. Interconnects between dies in 3D stacking are implemented using through-silicon-vias (TSVs), which provide lower effective communication latency compared to electrical links in 2D designs owing to the shorter length of TSVs and larger potential bandwidth (i.e., as a large number of TSVs can connect stacked chips without the pinout restrictions). One other benefit of 3D stacking is the ability of integrating dies with heterogeneous technologies in a single chip, e.g., multi-core systems with DRAM or many-core systems with silicon-photonic network-on-chip (PNoC). In summary, compared to 2D designs, 3D stacking allows for lower communication latency, better yield, and higher efficiency due to integration of heterogeneous technologies.



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten. New plot and data collected for 2010-2015 by K. Rupp.

Figure 1·1: 40 years of microprocessor trend data (Rupp, 2015).

In tandem with the benefits above, 3D stacking also introduces challenges in on-chip resource and thermal management. First, due to the diverse resource requirements of workloads, statically optimized on-chip resources may be under-utilized, which leaves potential performance improvement on the table. Second, as a larger number of on-chip resources are integrated vertically, the power density per footprint of 3D-stacked systems is much higher than 2D systems, which often leads to thermal violations. Last but not least, technologies such as PNoC are highly sensitive to on-chip thermal and process variations. These variations may result in PNoC malfunction (Mohamed et al., 2010) or high power overhead to compensate for the variations, which limit PNoC's wide adoption in computing systems.

This thesis claims that application-aware, cross-layer design and resource management techniques are essential for energy-efficient 3D-stacked multi-/many-core systems. Specifically, the thesis improves the performance and energy efficiency of 3D-stacked multi-/many-core systems by identifying workloads' resource requirements, constructing systems with flexible heterogeneity that is capable of catering to these various requirements, and intelligently managing the available resources under power and thermal constraints.

1.1 Problem Statement

3D stacking enables integrating more on-chip resources compared to 2D designs. At the same time, resource management becomes even more essential, especially for memory and communication resources, due to the increasing number of cores. 3D stacking also allows for heterogeneous technology integration (DRAM + logic, PNoC + logic, etc.). Considering the complexity and heterogeneity, design of efficient architectures and resource management policies in 3D systems, especially in data access and transmission, can make a tremendous difference in system energy efficiency.

Recent research explores performance and energy efficiency benefits of 3D-stacked systems by considering fixed computational and memory resources (Black et al., 2006; Loh, 2008), where a core cannot dynamically change its hardware resources. Different types of workloads, however, exhibit different needs for hardware resources (e.g., cache/memory usage or computation intensity), where a fixed homogeneous architecture (where each core has the same amount of resources) cannot always meet these needs. Heterogeneous designs, such as systems with various core types or heterogeneous memory architecture designs, have been proposed as a solution to this challenge. However, such systems also typically have fixed resources that are not reconfigurable at runtime and do not provide flexibility for diverse resource requirements of applications. In addition, heterogeneous systems are more complex and expensive to design compared to homogeneous systems.

Resource pooling, where components of a core can also be used by other cores, enables "*flexible heterogeneity*" (i.e., each core can adjust its hardware resources flexibly) in a multi-core processor. The *flexible heterogeneity* provided by resource pooling can address various resource needs of workloads by reconfiguring resources among cores according to the demand (Ipek et al., 2007; Ponomarev et al., 2006). However, in conventional 2D chips, resource sharing is strongly limited by the long access latency of remote resources in the horizontal dimension (Homayoun et al., 2012). Sharing resources across a chip becomes particularly inefficient for a large number of cores and large chip sizes. In 3D-stacked systems, TSVs can be used for pooling resources among multiple layers due to its short length and low latency. A recently proposed resource pooling design for 3D systems with homogeneous layers enables pooling performance-critical microarchitectural components such as register files and load/store buffers vertically (Homayoun et al., 2012). This approach demonstrates considerable energy efficiency improvement in 3D systems using resource pooling. However, the performance and energy efficiency potentials in pooling on-chip memory resources (e.g., caches or scratchpad) in 3D-stacked systems has not been studied prior to our work.

Diversity of workloads in resource requirements does not only reflect in cache usage, but also in main memory. There are various types of memories provided by memory vendors optimized for latency, bandwidth, or power, targeting a wide range of system requirements. For example, Reduced Latency DRAM (RLDRAM) is a type of memory optimized for low latency, which makes it ideal for network switch and router applications (MICRON, 2016). RLDRAM's bandwidth is smaller and power consumption is significantly higher compared to DDR3. On the other hand, Low Power DRAM (LPDRAM) reduces power consumption substantially, but has higher access latency and lower bandwidth (MICRON, 2013); thus, it is attractive for power-constrained systems such as mobile platforms. However, a single memory module cannot provide the lowest latency, highest bandwidth, and lowest power consumption at the same time, making a homogeneous memory system sub-optimal in terms of energy efficiency for serving a diverse set of applications. A heterogeneous memory system, which contains diverse memory modules, is able to cater to a wide range of workloads with higher energy efficiency, compared to homogeneous memory systems (Phadke and Narayanasamy, 2011; Chatterjee et al., 2012). A major challenge, however, lies in how to efficiently manage the memory allocation/reallocation among different memory modules based on workloads' memory access patterns (Agarwal et al., 2015; Tran et al., 2013).

In many-core systems, the on-chip communication among cores, caches, and on-chip memory controllers plays an important role in system performance and energy efficiency. The ever-increasing thread-level parallelism exhibited by multi-threaded workloads requires network-on-chip (NoC) bandwidth to increase correspondingly. Emerging application domains (e.g., cognitive or big data applications) are expected to require even larger network bandwidths. Thus, using electrical links to provide low latency and large bandwidth while staying within limited power budgets is becoming increasingly difficult. Silicon-photonic links have been proposed as potential replacements for electrical links in NoC designs because they provide significantly higher bandwidth density ($Gb \cdot s^{-1} \cdot \mu m^{-1}$), lower global communication latency, and lower data-dependent power (Joshi et al., 2009; Ramini et al., 2012). However, the thermal tuning power consumption required to compensate the impact of thermal and process variations on optical devices (such as ring resonators) and the power consumption of laser sources restrict the wide adoption of PNoC, especially in many-core systems due to the large chip area. For example, every ring modulator/filter resonates at a fixed optical frequency to modulate/filter optical signals. With the presence of thermal and process variations between a sender and a receiver, optical frequencies of these devices may shift in different amounts and fail to match each other, leading to potential data loss or errors. Research work that has been carried out in the area of PNoC thermal management ranges from device-level techniques (DeRose et al., 2010; Djordjevic et al., 2013) to chip floorplanning techniques (Ding et al., 2012; Coskun et al., 2016). The most common approach is to integrate a micro-heater under each optical transmitter (ring modulator or ring filter) and control temperatures of optical devices by heating them up (DeRose et al., 2010). However, this approach is energy-inefficient when optical frequency variation is high among optical devices or when a system is underutilized. Thus, there is an essential need for low-power techniques that can align optical frequencies for on-chip optical devices.

In summary, albeit the benefits of 3D stacking on integrating more on-chip resources and heterogeneous technologies in a single chip, its true potential in performance and energy efficiency remains unearthed without application-aware, cross-layer resource and thermal management techniques.

1.2 Thesis Contribution

This thesis improves energy efficiency of 3D multi-/many-core systems through (1) resource management techniques and (2) thermal management techniques. Our research does not only show the performance and energy efficiency improvement of providing heterogeneity in memory resources and conducting thermal management of 3D multi-/many-core systems, but also provides a reusable cross-layer simulation framework to enable future research. **Specifically, the contributions of this thesis are as follows:**

- We demonstrate that cache requirement varies significantly among modern single-thread workloads. Motivated by this observation, we introduce a 3D Cache Resource Pooling architecture (**3D-CRP**) (Meng et al., 2013; Zhang et al., 2015) that requires minimal architectural modifications compared to conventional (fixed) cache. We propose a novel application-aware job allocation and cache pooling policy for efficient sharing of cache partitions across workloads. We demonstrate that 3D-CRP achieves 19% improvement on average in system energy efficiency (Meng et al., 2013) and show the potential performance benefits when using 3D-stacked DRAM as well (Zhang et al., 2015).

- We introduce a novel low-power 3D Modular Multi-Core system (**3D-MMC**) (Zhang et al., 2013), composed of homogeneous layers to augment system performance with minimal design cost compared to traditional 2D design. We propose a shared memory resource pooling technique to improve system performance in 3D-MMC. We demonstrate that memory resource pooling brings up to 48% runtime reduction when memory is highly stressed. 3D-MMC also provides a demonstration of SW applications running on a 3D multi-core system.

- We argue that designing a system with heterogeneous memory components has significant potential to improve energy efficiency, and we propose a technique for memory object classification and allocation (**MOCA**). MOCA creates a profile of each object's memory access behavior to determine which memory module is the most suitable for this object in a given heterogeneous memory system (consisting of latency-, bandwidth-, and power-optimized memory modules). We also design a page allocation algorithm cognizant of memory object-level performance profiles. MOCA improves the energy delay square product compared to an equivalent homogeneous memory system with DDR3 modules by 17.3% on average for memory-intensive workloads and by 19% on average compared to application-level page allocation for workloads with memory object diversity.

- As the number of cores integrated on a chip increases, PNoC provides potential benefits. To be able to accurately model performance, power, and temperature of many-core systems with PNoC, we design a cross-layer modeling stack, which considers device properties and layout in thermal simulations and temperature-dependent laser source power (Abellan et al., 2016).

- This thesis, for the first time, introduces a thermally-aware workload allocation method to balance temperatures of optical devices in many-core systems with

PNoC (i.e., a "**RingAware**" policy) (Zhang et al., 2014). We demonstrate that RingAware outperforms existing thermally-aware job allocation policies in balancing temperatures of on-chip optical devices.

- We enhance **RingAware** with awareness of thermal and process variations and design **FreqAlign** (Abellan et al., 2016), to further minimize optical frequency difference among optical devices. Coupled with FreqAlign, we also propose an Adaptive Frequency Tuning policy, (**AFT**) (Abellan et al., 2016), to control optical frequencies of optical devices adaptively based on their temperatures at runtime to reduce tuning power of many-core systems with PNoC. FreqAlign combined with AFT reduces thermal tuning power in many-core systems with PNoC from 20 W on average to lower than 1 W.

- As design-time decisions greatly affect on-chip thermal conditions, we also investigate the impact of place-and-route (P&R) on on-chip thermal conditions, provide on-chip laser source placement and sharing schemes under various PNoC logical topology and physical layout combinations (Chen et al., 2014), and propose a cross-layer framework to optimize P&R solutions (Coskun et al., 2016).

1.3 Organization

The rest of this thesis starts with a review of the background and related work on 3D-stacked architectures, state-of-the-art cache/memory resource management techniques, as well as thermal management approaches for PNoC in Chapter 2. Chapter 3 studies cache requirements of modern workloads. Then, we introduce the proposed 3D-CRP architecture and the corresponding workload allocation policy. In Chapter 4, we introduce our work on memory access scheduling in 3D-MMC followed by a proposed memory management technique, MOCA, for heterogeneous memory systems.

We present a full modeling stack of performance, power, and thermal simulations for many-core systems with PNoC at the start of Chapter 5. Then, we introduce the proposed runtime thermal management techniques and the design-time P & R optimizations. Chapter 6 concludes the thesis and discusses future research directions.

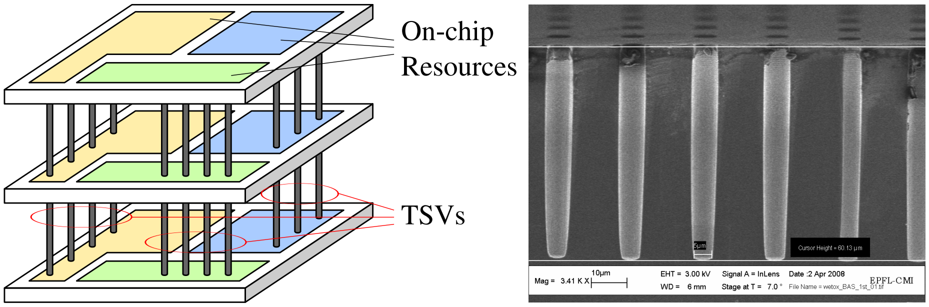
Chapter 2

Background and Related Work

This thesis aims to improve the performance and energy efficiency of 3D-stacked systems through intelligent resource and thermal management techniques. To this end, this chapter starts with a discussion of the state-of-the-art 3D-stacked architectures and previous work in memory resource management in multi-core systems. Then, we introduce the PNoC background and existing work in thermal management techniques for many-core systems with PNoC. We conclude the chapter by highlighting the novel aspects of our work compared to existing work.

2.1 3D Stacking Technology

3D stacking is a technique to stack silicon dies vertically to make them a single device. Manufacturers perform 3D die stacking in the following two major approaches: face-to-face and face-to-back stacking (Noia and Chakrabarty, 2014). Face-to-face



(a) (b)

Figure 2·1: (a) An example of 3D face-to-back stacked system using TSVs. (b) Micrograph of TSVs structure (Courtesy of LSM, EPFL).

approach connects the resources on two dies through micro-bumps and uses TSVs for I/O and power supply. This approach is limited to stacking two dies together. Face-to-back approach uses either only TSVs or TSVs + micro-bumps for the interconnects between dies. A TSV is a vertical electrical link that goes through a silicon wafer. Compared to micro-bumps, TSVs have substantial higher integration density. Figure 2·1 shows an example of a 3D system based on face-to-back stacking approach and TSV connections in (a) and a micrograph of TSV structures in (b). 3D stacking enables integrating a larger amount of on-chip resources compared to traditional 2D IC design and integration of dies with different technologies, which provides increased efficiency by limiting/avoiding off-chip communication.

3D system architectures can be broadly classified into two categories: logic + logic systems and logic + other systems. The former includes heterogeneous die stacking and homogeneous die stacking. Heterogeneous die stacking involves splitting a planar design's logic area into two or more layers (e.g., the 3D version of an Intel Pentium 4 family processor (Garrou et al., 2008)), and vertically stacking additional blocks (e.g., more caches, reservation station, etc.) to target different market segments (Loh, 2008) to improve processor performance. The latter generally refers to stacked systems that include other technologies over the logic die. For example, stacked DRAM on the logic die provides shorter memory access latency and higher bandwidth compared to off-chip DRAM chips (Garrou et al., 2008). Other examples in this category, utilizing GlobalFoundries' 130 nm process and Tezzaron's FaStack technology, respectively, are 3D-MAPS (Kim et al., 2012), where the logic die consists of 64 cores operating at 277 MHz and the stacked memory die contains 256KB SRAM, and Centip3De (Fick et al., 2012), a configurable near-threshold 3D-stacked system with 64 ARM Cortex-M3 cores. In this thesis, we investigate the resource/thermal management techniques for 3D systems to improve their performance and energy efficiency.

2.2 On-Chip Resource Management

Although 3D stacking allows integration of more on-chip resources, it does not automatically bring more efficient computing compared to 2D designs. Workloads usually have very diverse needs for chip resources. If the extra resources are only private (e.g., private cache), the resources might be under-utilized for most of the time. An architecture that can support sharing resources among dies together with a management policy to decide the resources allocation among dies are essential to unveil the real performance and energy efficiency potential in 3D-stacked systems. In this section, we give an introduction on on-chip memory resource management in 2D and 3D systems and discuss relevant work on management of other microarchitectural units.

Resource Pooling

Resource pooling is a design technique that allows for components of a core to be shared with other cores, which enables *flexible heterogeneity* (each core can adjust its hardware resources flexibly) in a multi-core processor. Prior work on resource pooling has mainly focused on 2D multi-core systems. Ipek et al. propose a reconfigurable architecture to combine the resources of simple cores into more powerful processors (Ipek et al., 2007). Ponomarev et al. introduce a technique to dynamically adjust the sizes of the performance-critical microarchitectural components, such as reorder buffer or instruction queue (Ponomarev et al., 2006). However, as the number of on-chip cores increases, the long access latency among resources on 2D chips makes it difficult to get fast response from the pooled resources. In 3D systems, stacking the layers vertically and using TSVs for connection enable short access latency among on-chip resources. Homayoun et al. are the first to explore microarchitectural resource pooling in 3D-stacked processors for sharing resources at a fine granularity (Homayoun et al., 2012). As memory is also a performance-critical component in computer systems, an architecture with memory resource pooling in 3D system can bring substantial performance improvements to the system. However, none of the prior work investigates cache or memory resource pooling in 3D systems.

Cache Partitioning and Reconfiguration

Cache resources are important to a processor's performance, thus, cache sharing and partitioning have been well studied in 2D multi-core systems. For example, a cache architecture named molecular caches is proposed to create dynamic heterogeneous cache regions (Varadarajan et al., 2006). One other technique partitions caches between multiple applications based on their cache miss rate during runtime (Qureshi and Patt, 2006). Chiou et al. propose a dynamic cache partitioning method though fine-grain control of cache placement policy (Chiou et al., 2000). However, the benefits of cache sharing in 2D systems are highly limited by the on-chip interconnect latency, and sharing the L2 cache among multiple cores is significantly less attractive when the interconnect overheads are taken into account (Kumar et al., 2005).

Cache design and management in 3D-stacked systems have been investigated recently. Sun et al. explore the energy efficiency benefits of 3D-stacked MRAM L2 caches (Sun et al., 2009). Prior work on 3D caches and memories either considers integrating heterogeneous SRAM or DRAM layers into 3D architectures (e.g., (Meng et al., 2012; Jung et al., 2011)), or involves major modifications to conventional cache design (e.g., (Sun et al., 2009)). Compared to such heterogeneous 3D systems, a 3D system with homogeneous layers and resource pooling features is more scalable to a larger number of cores and a wider range of workloads (Zhang et al., 2015).

Runtime Management of Multi-core Systems

To manage on-chip resources for improved performance and energy efficiency, an intelligent runtime policy is required. Recent research on runtime policies in 2D systems generally focuses on improving performance and reducing the communication, power, or cooling cost through job allocation. For example, Snavely et al. present a mechanism that allows the scheduler to exploit the workload characteristics for improving processor performance (Snavely and Tullsen, 2000). Das et al. propose an application-to-core mapping algorithm to maximize system performance (Das et al., 2012). Bogdan et al. propose a novel dynamic power management approach based on the dynamics of queue utilizations and fractional differential equations to avoid inefficient communication and high power density in NoC (Bogdan et al., 2012).

Dynamic job allocation in 3D systems mostly addresses power and thermal challenges induced by vertical stacking. For example, dynamic thermally-aware job scheduling techniques use thermal history of cores to balance the temperature and reduce hot spots (Coskun et al., 2009a; Zhu et al., 2008). Zhou et al. propose an OS-level workload scheduling algorithm for optimizing 3D system temperature (Zhou et al., 2008). Another technique is proposed to dynamically adapt core resources based on application needs and thermal behavior to boost performance while maintaining thermal safety (Hameed et al., 2011). However, such methods do not perform a detailed performance analysis of workloads or investigate *flexible heterogeneity* in memory resources, and thus, leave potential energy efficiency unclaimed.

2.3 Heterogeneous Memory Systems

3D stacking enables on-chip DRAM for short access latency and high bandwidth. However, the capacity of stacked DRAM die is highly restricted by the area of logic die, which limits the performance of applications with large input work sets. Using off-chip memory together with on-chip memory to construct heterogeneous memory systems can effectively counter such issues. However, data allocation within heterogeneous memory systems needs to be carefully managed to improve performance.

There has been extensive study in heterogeneous memory system architecture and management. We first discuss prior work that exploits heterogeneity in DRAM to construct a heterogeneous memory system. We then discuss other methods that use on-chip scratchpad memory, 3D-stacked DRAM or phase change memory (PCM) to construct heterogeneous memory systems.

Exploiting Heterogeneity in DRAM

Phadke et al. (Phadke and Narayanasamy, 2011) employ latency, bandwidth, and power optimized memory modules and choose a single optimal memory module for an application based on offline profiling. As shown in Section 5.5.2, their method leaves substantial performance and energy savings at the table. Chatterjee et al. (Chatterjee et al., 2012) place critical words in a cache line in latency-optimized memory module and rest of the cache line in power-optimized modules. For a heterogeneous system comprising of bandwidth optimized and capacity optimized memories, Agarwal et al. (Agarwal et al., 2015) propose a page placement policy, which places highly accessed pages in bandwidth-optimized memory. This is efficient for graphics processing unit (GPU) programs, which hide memory latency well.

Heterogeneous Memory Architecture

Prior work constructs a heterogeneous memory system comprising of either on-chip scratchpad memory (Shen et al., 2016; Peón-quirós et al., 2015) or 3D-stacked memory (Meswani et al., 2015; Tran et al., 2013; Dong et al., 2010b; Lee et al., 2013) or non-volatile memory such as PCM (Dulloor et al., 2016; Pavlovic et al., 2013) and traditional DRAM. Many of these employ page-level policies to utilize the lowest latency memory module available in the system optimally. To do so, they either track frequently accessed pages and move them to this module (Meswani et al., 2015; Dong et al., 2010b; Lee et al., 2013; Pavlovic et al., 2013) or control the amount of memory mapped to such a module based on bandwidth utilization (Tran et al., 2013).

Several prior approaches employ an application profile based scheme to guide data placement. Shen et al. (Shen et al., 2016) use PIN-based (Luk et al., 2005) profiling to track array allocations to place frequently accessed and low-locality arrays in on-chip scratchpad. They also decompose larger arrays into smaller chunks for fine grain data placement. Dulloor et al. (Dulloor et al., 2016) profile memory access patterns of data structures as either sequential, random, or involving pointer chasing. Data structures exhibiting latency-sensitive patterns like pointer chasing are then placed in DRAM and the rest are placed in PCM. Peon-Quiros et al. (Peón-quirós et al., 2015) profile dynamically allocated data structures for embedded system applications. They track frequency of access per byte and changing memory footprint over time of these structures to place them in either on-chip SRAM or off-chip DRAM modules.

While future memory systems will employ such varied forms of memory technologies as envisioned in these prior works, our work aims to highlight the benefits of utilizing the heterogeneity in DRAM modules and managing workloads' data in memory object level.

2.4 Silicon-Photonic Network-on-Chips

As the number of cores per chip increases and there is higher parallelism in emerging workloads (especially in emerging cognitive or big data applications), the requirement for on-chip network bandwidth also raises. Silicon photonics is a promising technology to support this increasing demand for high-bandwidth and energy-efficient on-chip communication in future many-core systems. Compared to an Electrical NoC (ENoC), a PNoC tends to have higher bandwidth density with lower data-dependent power dissipation. Thus, designing an energy-efficient PNoC has been widely explored (Shacham et al., 2007; Joshi et al., 2009; Pan et al., 2009; Vantrease et al., 2008; Cianchetti et al., 2009; Kirman et al., 2006; Ramini et al., 2012). We provide the relevant background of PNoC and the challenges in its adoption in this section.

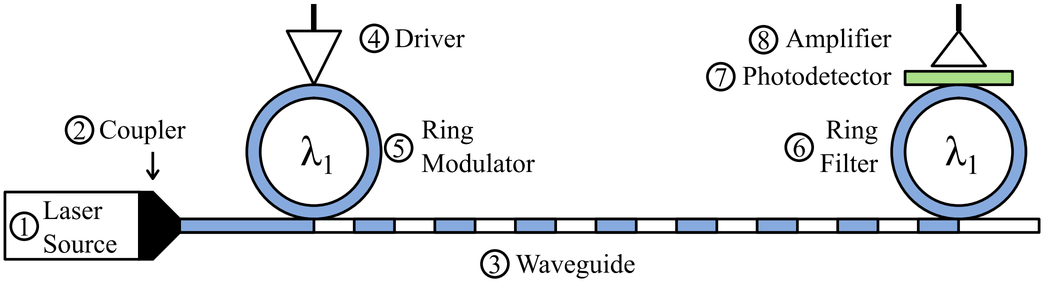


Figure 2·2: A silicon photonic link.

2.4.1 Silicon-Photonic Link

A PNoC is composed of silicon-photonic links, each of which contains the following devices: (1) a laser source that emits optical waves, (2) a coupler that couples optical waves from the laser source to a waveguide, (3) a waveguide that carries optical waves, (4) a driver that receives electrical signals from the circuit side, (5) a ring modulator that modulates optical waves at the transmitter side based on the electrical signals of the driver, (6) a ring filter that filters optical waves at the receiver side, (7) a photodetector that converts optical signals into electrical signals, and (8) an amplifier that amplifies electrical signals, as shown in Fig. 2·2. To transmit data without errors or loss, the ring modulator and filter must resonate at the same frequency as the optical frequency of the corresponding laser source.

There are two major factors that affect the optical frequencies of silicon-photonic devices and can lead to optical frequency mismatch: with-in-die (WID) process variations and thermal variations. The WID process variations means that due to the manufacturing process limitations, the actual physical dimensions of silicon-photonic devices (e.g., height, width, or radius) may differ from the designed values. Such variations depend on the quality of manufacturing process and the locations of these devices on a wafer. The mismatch in physical dimensions of silicon-photonic devices caused by such variations results in a mismatch between the designed and the actual optical frequencies of these devices. Since silicon-photonic devices are spread across the whole chip for a system with a PNoC, they may have different amounts of process variations, which causes optical frequency mismatch. Such process variations are measurable after the chip manufacture and their impact can be compensated through thermal management. Thermal variations have significant impact on optical frequencies of silicon-photonic devices such as ring resonators, due to the thermo-optic effect (thermal modulation of the refractive index of a material). We explain the details of the thermal sensitivity of silicon-photonic devices in the next subsection.

2.4.2 Thermal Sensitivity of Silicon-Photonic Devices

The refractive index of a material changes as its temperature changes, which is known as thermo-optic effect. Optical devices manufactured using silicon are extremely sensitive to thermal changes because silicon possesses a relatively large thermo-optic coefficient (the refractive index changes significantly under a given temperature change). For example, the ring resonators for PNoC are commonly designed around a center wavelength (λ_0) of 1550 nm, and they have a thermal sensitivity ($\Delta\lambda_R$) of 78 pm/K (Orcutt et al., 2012). This translates to a 9.7 GHz frequency shift per degree (Δf_R), based on the following equations:

$$F_0 = \frac{c}{\lambda_0} = 193\ THz, \tag{2.1}$$

$$\frac{\Delta\lambda_R}{\lambda_0} = \frac{\Delta f_R}{F_0} \tag{2.2}$$

Thus, for every degree of temperature difference between a ring modulator and a ring filter in a silicon-photonic link, there is a resonant frequency mismatch of 9.7 GHz.

The impact of thermal variations among silicon-photonic devices, depends on the corresponding resonant frequency difference as well as the frequency spacing between two adjacent wavelengths in a waveguide. In PNoC, every waveguide is multiplexed

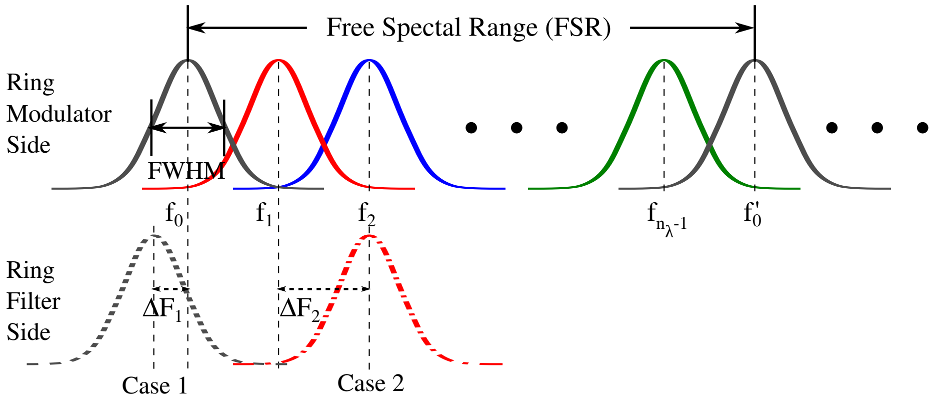


Figure 2·3: Impact of resonant frequency mismatch. Case 1: Small mismatch reduces the filtered optical power; Case 2: Large mismatch may result in a ring to filter the data of its neighboring ring in the frequency domain.

by a number of optical waves in different wavelengths, i.e., wavelength-division multiplexing (WDM). The spacing between adjacent wavelengths in each waveguide depends on the free spectral range (FSR) of a ring resonator design and the number of wavelengths multiplexed onto this waveguide (n_λ), as shown below:

$$FSR = \frac{c}{2\pi r n_g} \tag{2.3}$$

$$F_{spacing} = \frac{FSR}{n_\lambda} \tag{2.4}$$

where n_g is the group index, c is the speed of light, and $F_{spacing}$ is the spacing in resonant frequency for two adjacent wavelengths in a waveguide. The impact of resonant frequency mismatch is shown in Fig. 2·3, where $FWHM$ represents full width at half maximum. When the mismatch is small, a ring filter receives only a portion of the signal power, resulting in less current from the photodetector and causing data loss (Case 1). As the mismatch increases, a ring filter may even filter the optical waves corresponding to its neighboring resonant frequency (Case 2).

The laser sources are also sensitive to temperature variations (Kimoto et al., 2003). There are two major ways of integrating laser sources with the chip package: off-chip integration and on-chip integration (Heck and Bowers, 2014). For off-chip laser sources, their temperatures are typically controlled to guarantee the frequencies of emitted optical waves, or a frequency locking circuit is employed. The operation of these off-chip laser sources is agnostic to on-chip temperatures. On the other hand, on-chip laser sources' temperatures are affected by chip thermal conditions due to their close proximity to the computational components. Thus, for a PNoC with on-chip laser sources, one must control the optical frequencies of both ring resonators and laser sources for reliable silicon-photonic link operation.

2.4.3 Thermal Management in PNoCs

To counter the impacts of WID process and thermal variations, there have been techniques proposed from device level to system design and management level, most of which utilizes thermo-optic effect.

At the device level, there are two common ways to protect the silicon-photonic devices from temperature variations: (1) actively control the temperatures of these devices; (2) choose/design silicon-photonic devices that are less thermally-sensitive. Active temperature control (or localized thermal tuning) is carried out by integrating micro-heaters with silicon-photonic devices (DeRose et al., 2010). During PNoC operation, the micro-heaters can heat up each silicon-photonic device to a fixed temperature, which usually is the maximum temperature allowed for on-chip logic components. Process variations can be measured after chip manufacture and their impact on optical frequency can be countered using thermo-optic effect by heating up ring resonators with optical frequency shift. As for the athermalization techniques, one can either clad silicon with materials with negative thermo-optic coefficient (TOC) (Djordjevic et al., 2013) or couple the silicon-photonic devices with other athermal devices such as Mach-Zehnder interferometers (MZI) (Guha et al., 2010). Generally speaking, active thermal control techniques are easier to implement, are able to provide flexibility in runtime thermal management, and are more mature. However, it is very energy-inefficient when the optical frequency difference among silicon-photonic devices is high. On the other hand, passive techniques that make silicon-photonic devices athermal do not require extra energy for thermal management, however, they are usually not compatible with traditional CMOS manufacturing process and may not correct for process variations. Thus, using micro-heaters is a more common way for PNoC thermal management.

One of the main challenges of designing an energy-efficient PNoC is the large thermal tuning power overhead, which negatively affects the system energy efficiency. There has been extensive research conducted on chip design techniques for preemptive PNoC thermal management. One approach (Nitta et al., 2011) integrates redundant ring resonators in order to provide higher tolerable temperature gradients among ring modulators and filters within a given thermal tuning budget. In this technique, a ring group is defined as a collection of co-located ring resonators used to implement a communication interface. Usually, every ring modulator has only one associated ring filter in a silicon-photonic link and they are in different ring groups. If there is a temperature gradient between these two ring groups, optical signals are either lost or filtered by neighboring ring filters. Adding extra ring filters creates a sliding ring window, which takes advantages of the fact that the resonant frequencies of ring resonators in one ring group shift the same amount during temperature changes and allowing the optical signals to be all filtered out by the neighboring ring filters. Another chip design thermally decouples the processor die from the photonics die by inserting an insulator layer in between to make active thermal tuning more efficient (Demir and Hardavellas, 2015). This design assumes that each ring resonator is only integrated with a micro heater (without a temperature sensor) and all the ring resonators work at a fixed temperature (90 ^oC) using localized thermal tuning. Normally, the thermal tuning power is mostly wasted as it dissipates through the processor stack and also heats up the processor die. To prevent such power waste, this design adds an insulator layer between the processor die and the photonics die, which keeps the temperatures for silicon-photonic devices more stable and minimizes the spatial and temporal thermal coupling between logic components and silicon-photonic devices.

As for the runtime PNoC thermal management techniques, one method, named Aurora (Li et al., 2015b), leverages localized tuning and workload allocation techniques and embodies a cross-layer approach at the device, architecture and OS levels. At the device level, Aurora controls small temperature variations by applying a bias current through the ring resonators (Manipatruni et al., 2008). For larger temperature changes, packets are rerouted away from hot regions, and dynamic voltage and frequency scaling (DVFS) reduces temperature of hot areas. At the OS level, a job allocation policy prioritizes jobs to the outer cores of the chip. Since workload allocation decides the power profiles of systems, it directly impact the thermal conditions of silicon-photonic devices. However, there has not been research focusing PNoC thermal management through workload allocation techniques until our work.

2.5 Distinguishing Aspects from Prior Work

The novel aspects of our work compared to the existing research are as follows:

Our 3D-CRP work (Meng et al., 2013; Zhang et al., 2015) in Chapter 3 is the first to propose a cache resource pooling architecture complemented with a novel dynamic job allocation and cache pooling policy in 3D multi-core systems, which requires minimal hardware modifications. Our dynamic job allocation and cache pooling policy differentiates itself from prior work as it partitions the available cache resources from adjacent layers in the 3D-stacked system in an application-aware manner and utilizes the existing cache resources to the maximum extent. We also evaluate the performance, energy efficiency, and thermal behavior of multi-core 3D systems with and without DRAM stacking. In addition, our work improves the performance model of the 3D system with stacked DRAM by introducing a detailed, accurate memory access latency model for on-chip memory controllers.

Our 3D-MMC work (Zhang et al., 2013) in Chapter 3 introduces both the hardware architecture and software implementation for a novel low-power 3D system. We focus on exploiting resource pooling at fine granularity and provide a practical implementation. We apply homogeneous stacking, which results in lower wafer and 3D bonding costs compared to heterogeneous partitioning.

MOCA (Zhang et al., 2017) in Chapter 4 is our proposed technique to name and allocate the memory objects instantiated during workload execution for better performance and energy efficiency. MOCA creates a detailed profile of each object's memory access behavior so as to determine which memory module is the most suitable for this object in a given heterogeneous memory system. We design a page allocation algorithm cognizant of the memory module best suited for an object to improve system energy efficiency.

Our runtime thermal management of 3D many-core systems with PNoC (Zhang et al., 2014; Abellan et al., 2016) in Chapter 5, for the first time, addresses matching the optical frequencies of both ring resonators and on-chip laser sources, at a low power cost. In our work, we propose a workload allocation policy that considers both on-chip thermal and process variations coupled with a thermal tuning policy that sets target frequency for silicon-photonic devices adaptively based on system's thermal condition. Our management policies require much lower thermal tuning power for system, and they are effective also in presence of process variations.

Our laser source sharing and placement methodology (Chen et al., 2014) and cross-layer floorplan optimizer (Coskun et al., 2016) for 3D many-core systems with PNoC in Chapter 5 simultaneously consider NoC bandwidth constraints, thermal constraints and physical layout constraints to determine on-chip silicon-photonic devices' P & R solution to reduce PNoC power consumption.

Chapter 3

Cache Resource Management in 3D Multi-core Systems

3.1 Overview

Modern processors get significant performance improvement from caches. It is expected that CPU performance generally increases along with larger cache sizes. However, larger caches consume higher power, and bring varying performance improvements for different applications due to their varying cache usage. Thus, depending on the applications, the optimal cache size to achieve the best energy efficiency may differ. In this work, we use energy-delay-product (EDP) to represent the energy efficiency. In homogeneous 3D-stacked systems, each core on each layer has a fixed size private last-level cache (LLC), which potentially restrains the system's performance and energy efficiency. In this section, we investigate the impact of LLC cache (L2

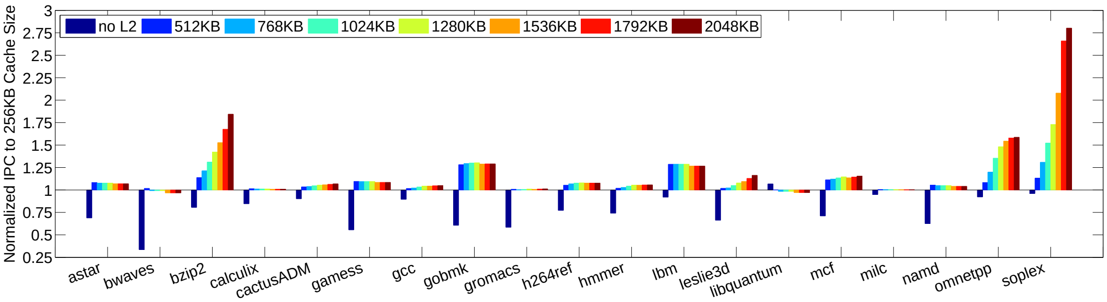


Figure 3·1: IPC of SPEC CPU 2006 benchmarks for increasing L2 cache size. The IPC values are normalized with respect to using a 256 KB L2 cache.

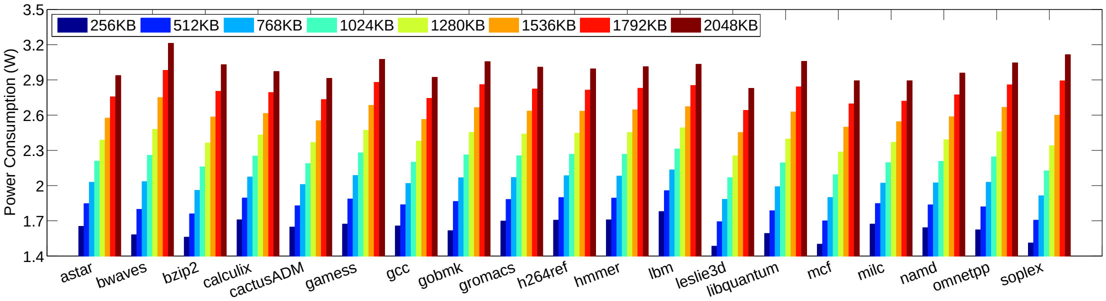


Figure 3·2: Power consumption of SPEC CPU 2006 benchmarks under cache sizes from 256 KB to 2048 KB.

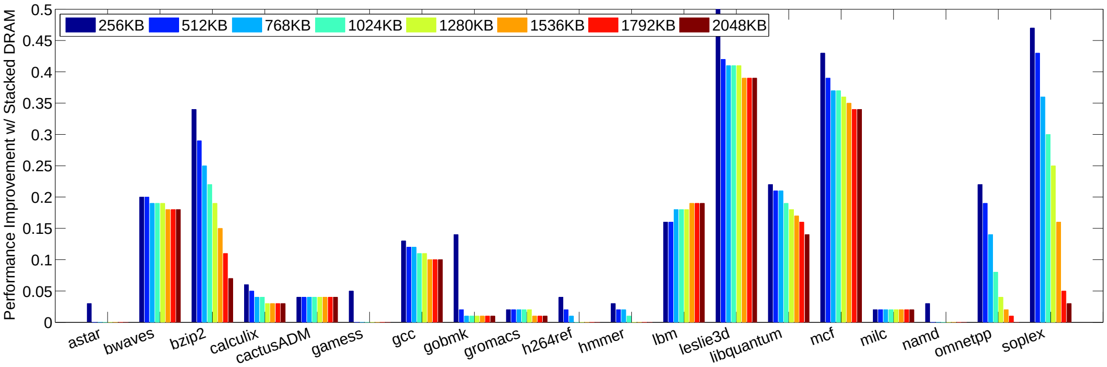


Figure 3·3: IPC improvement of SPEC CPU 2006 benchmarks using stacked DRAM in comparison to off-chip DRAM.

cache, in our target systems) sizes on performance and energy efficiency of various applications. Although we focus on L2 cache in this work, our proposed technique can also be applied to the other levels of caches in the multi-core systems.

To quantify the impact of L2 cache, we simulate a single core with varying L2 cache sizes (from 0 KB to 2048 KB with a step of 256 KB) and compare the performance and power consumption of the applications in SPEC CPU 2006 benchmark suite. We use Gem5 (Binkert et al., 2011) for performance simulation, McPAT (Li et al., 2009) and CACTI 5.3 (Thoziyoor et al., 2008) for core and cache power consumption, respectively (details of our simulation methodology are presented in Section 3.2.3).

Figure 3·1 shows the normalized IPC of all applications under different L2 cache sizes. As shown in Fig. 3·1, among all applications, soplex, omnetpp, and bzip2 have

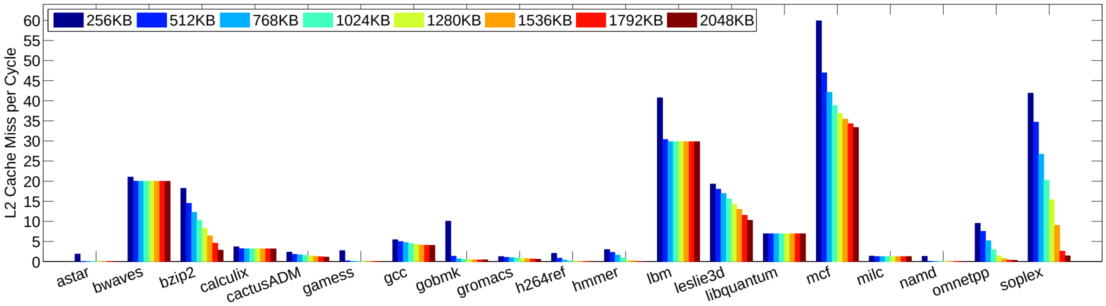


Figure 3·4: L2 MPKI of SPEC CPU 2006 benchmarks under cache sizes from 256 KB to 2048 KB.

significant performance improvement at large L2 cache sizes of up to 1.8x. We call such applications *cache-hungry* applications. On the other hand, applications such as bwaves barely benefit from an L2 cache larger than 256 KB. Figure 3·2 shows the core + cache power consumption for all applications under different L2 cache sizes. As shown in the figure, the power consumption increases with L2 cache size in general. Figure 3·1 and Fig. 3·2 indicate that while some applications' EDP values strongly benefit from large caches, others have marginal or no benefits. Such variances of IPC and power motivates tuning L2 cache size to optimize system EDP. Thus, we propose a homogeneous 3D architecture that enables vertical cache resource pooling (CRP).

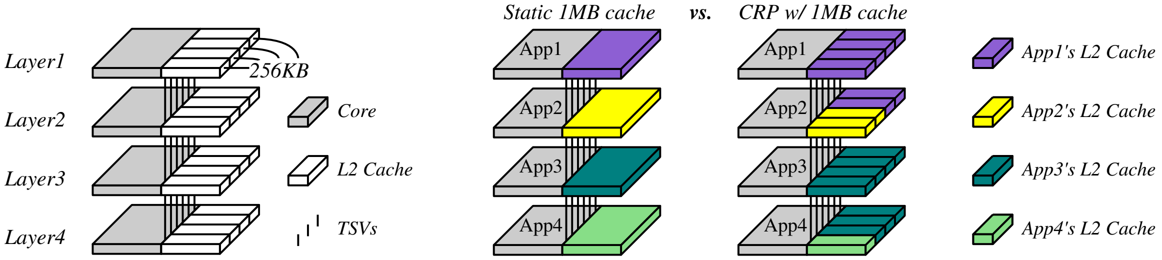
In addition to the requirement of LLC size, the memory access behavior also differs among applications; thus, memory system architecture is also affecting performance and energy efficiency. Owing to the small area cost of TSVs, on-chip 3D-stacked DRAM can have higher bandwidth and more parallelism in memory access, which leads to a lower average queuing latency in memory controllers and higher system performance (Loh, 2009). Figure 3·3 shows the performance improvement of on-chip stacked memory over off-chip memory and Fig. 3·4 shows the L2 miss per kilo-instruction (MPKI) of the applications under different L2 cache sizes with off-chip memory. These results demonstrate that there is a strong correlation between performance improvement of using on-chip memory and the L2 MPKI of an application.

Applications such as astar, calculix, and hmmer do not show obvious performance improvement on a system with on-chip DRAM (compared to the same system with off-chip DRAM) memory while some other applications' performance is significantly improved (e.g., bwaves, gcc, libquantum). As for bzip2, omnetpp, and soplex, they benefit more from stacked memory when they have a small amount of cache resources. This is because memory access rate directly affects the queuing delay in memory controllers. If all cores attached to a memory controller have high memory access rates, the memory controller queuing delay increases dramatically. Therefore, in systems with stacked memory, we also need to consider the memory access intensity for each memory controller when allocating workloads.

3.2 Cache Resource Pooling in 3D Stacked Systems

3.2.1 Cache Resource Pooling Architecture

In this section, we describe the proposed CRP architecture (Meng et al., 2013; Zhang et al., 2015). The CRP architecture is demonstrated using a four-layer 3D system that has one core with a private 1 MB L2 cache on each layer. The vertically adjacent caches are connected via TSVs for cache pooling, as shown in Fig. 3·5 (a). On-chip communication is performed through shared memory. Thus, all L2 caches are connected to a shared memory controller. Figure 3·5 (b) shows an example of the differences in cache resource allocation between the systems with fixed L2 caches and CRP. In this example, applications 1 and 3 require larger caches than the other two applications, and thus, acquire extra cache resources from their adjacent layers in the proposed CRP architecture. In contrast, in a system with fixed L2 caches, an application can only work with a fixed amount of cache.



(a) 1MB Cache Pooling

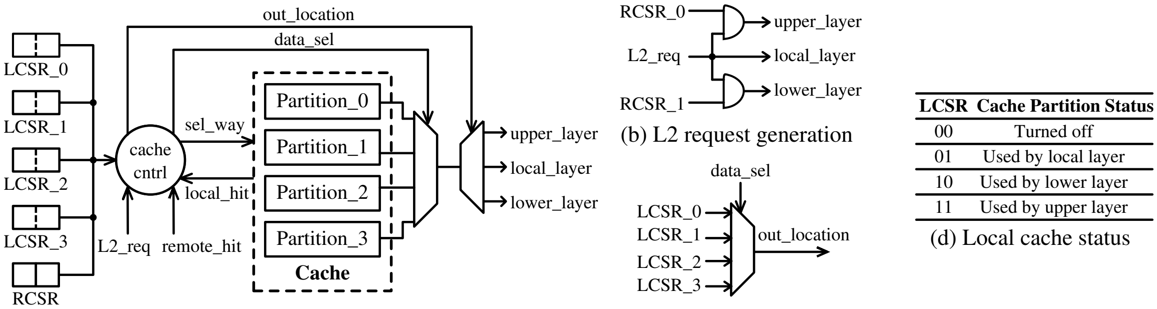
(b) Cache resource allocation using static cache and CRP

Figure 3·5: Proposed 3D system with cache resource pooling versus 3D systems with fixed 1 MB caches. In (a), cores are able to access caches on the adjacent layers through the TSVs.

3D-CRP Design Overview

Enabling cache resource pooling in 3D systems requires some modifications to the conventional cache architecture. The modified cache architecture allows cores in the homogeneous 3D-stacked system to increase their private L2 cache sizes by pooling the cache resources from the other layers at negligible access latency penalty. The objective of our design is to improve the system energy efficiency, which can be divided into two aspects: (1) to improve the performance by increasing cache size for cache-hungry applications, and (2) to save power by turning off unused cache partitions for non-cache-hungry applications. We focus on pooling L2 caches because L2 cache usage varies significantly across applications as shown in Section 3.1. It is possible to extend the strategy to other levels of caches.

Cache size is determined by cache line size, number of sets, and level of associativity. We adjust cache size by changing the cache associativity with the other two parameters fixed. We base our architecture on the selective way cache architecture proposed in prior work (Albonesi, 1999), which aims at turning off unnecessary cache ways for saving power in 2D systems. We call each cache way a *cache partition* in our design. Each partition is independently poolable to one of its adjacent layers. In order to maintain scalability of the design and provide equivalent access time to different



(a) Cache partition management implementation

(b) L2 request generation

(c) Output location generation

LCSR	Cache Partition Status
00	Turned off
01	Used by local layer
10	Used by lower layer
11	Used by upper layer

(d) Local cache status

Figure 3·6: Cache resource pooling implementation.

partitions, we do not allow cores in non-adjacent layers to share cache resources. We also do not allow a core to pool cache partitions from both upper and lower layers at the same time to limit the design complexity. In fact, we observe that for most of the applications in our experiments pooling cache resources from two adjacent layers at the same time would not bring considerable performance improvement.

3D Cache Partition Management Implementation

To implement cache resource pooling in 3D systems, we introduce additional hardware components to the conventional cache architecture. As shown in Fig. 3·6, we make modifications to both cache status registers and cache control logic.

In 3D-CRP, the cores need to access cache partitions from both the local layer and remote layers. First, we add a Local Cache Status Register (LCSR) for each local L2 cache partition (e.g., there are four partitions in a 1 MB cache in our design) to record the status of local cache partitions. There are four possible statuses for each local cache partition: *used by local layer, used by upper layer, used by lower layer, and turned off*. Each LCSR keeps two bits to indicate the current status of the corresponding partition as listed in Fig. 3·6 (d). The status of local cache partitions is used for deciding the destination of the output data and hit signals. Second, we introduce Remote Cache Status Registers (RCSR) for the L1 cache so that L1 cache is aware of its remote L2 cache partitions when sending L2 access requests. We maintain two 1-bit RCSRs in L1 caches for each core. If both RCSRs of an L1 cache are set to 0, there is no remote cache partition in use. In contrast, an RCSR bit is set to 1 if the core is using cache partitions from the corresponding adjacent layer. RCSR_0 and RCSR_1 denote the upper layer and the lower layer, respectively. The values of these registers are set by the runtime management policy in Section 3.2.2.

Through LCSRs and RCSRs, the cores are able to communicate with cache partitions from multiple layers. When there is an L1 miss, the core sends a request and the requested address based on RCSRs, as shown in Fig. 3·6 (b). Once the requests and addresses arrive at the cache controller, the tag from the requested address is compared with the tag array. At the same time, the entries of each way are chosen according to the index. The destinations of data and hit signals are determined by LCSR of the corresponding cache partition after a cache hit. We add a multiplexer to select the destination, as shown in Fig. 3·6 (c). When there is an L2 cache hit, the hit signal is sent back to the cache at the destination based on LCSR. When both the local hit signal and the remote hit signal are 0, thus indicates an L2 miss.

As the cache partitions can be dynamically re-assigned by the runtime policy, we need to maintain the data integrity of all the caches. In case of a cache partition re-allocation (e.g., a partition servicing a remote layer is selected to service the local core), we write back all the dirty blocks from a cache way before it is re-allocated. When a cache line is invalidated, both LCSRs and RCSRs are reset to 0 to disable the access from remote layers.

Larger 3D-CRP Systems with On-Chip DRAM

We name all the cores vertically stacked in the 3D architecture as a *column*. In the single-column system, we use off-chip DRAM because the chip area is not sufficiently large to include a reasonable DRAM size such as 1 GB. For a larger system with more

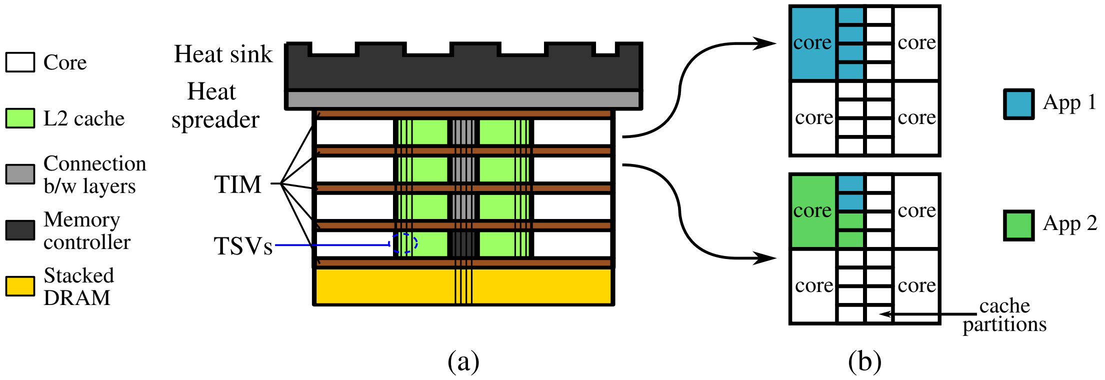


Figure 3·7: (a) The cross-section view of large 3D-CRP system; (b) An example showing cache resource pooling within a column.

cores organized in columns, the memory access rate increases as the number of cores increases, which results in longer memory access latency. When multiple memory controllers are used with stacked DRAM, it helps reduce the average memory access latency for larger 3D systems. Figure 3·7 shows the cross-section of a 16-core 3D-CRP system and an example of cache resource pooling within a column, respectively in (a) and (b). Stacked DRAM layers are located at the bottom of the 3D-CRP system and there are four memory controllers on the bottom logic layer, one for each column. However, the workloads of each column may have a different memory access rate, which results different memory access latencies. The column with workloads that all have a high memory access rate suffers from long memory access latency while the column with non-memory-intensive workloads does not. Therefore, a policy to monitor and adjust the job allocation in the aspect of memory accesses is necessary for such designs. Through job allocation, we balance the memory access intensity among columns to decrease the average memory access latency and improve the performance.

Implementation Overhead Evaluation

To evaluate the area overhead of 3D-CRP, we assume that each 1-bit register requires 12 transistors, each 1-bit 4-to-1 multiplexer requires 28 transistors and each 1-bit 2-to-

1 multiplexer (mux) has 12 transistors. We need ten 1-bit transistors for LCSRs and RCSRs, one 64-bit 1-to-4 demultiplexer (demux) and one 64-bit 4-to-1 mux for data transfers, one 30-bit 1-to-4 demux and one 30-bit 4-to-1 mux for address transfers (for 4 GB memory, we need 32-bit demux and mux), one 2-bit 4-to-1 mux for destination selection, one 1-to-2 demux to send back the hit signal to remote layers and two AND gates to generate L2 cache requests. Thus, the total number of required transistors 3D-CRP is limited to 5460 (10×1-bit register+ 2×64-bit 1-to-4 demux + 2×30-bit 4-to-1 mux + 1×2-bit 4-to-1 mux + 1×1-bit 1-to-2 demux + 2×AND gate). We assume there are 128 TSVs for two-way data transfer between caches, 60 TSVs for the memory address bits, and four additional TSVs for transferring L2 requests and hit bits between the caches on adjacent layers. To connect to a memory controller, we assume there are 30 TSVs for memory address bits, and 512 TSVs for receiving data from the memory controller. TSV power has been reported to be much lower compared to the overall power consumption of a chip (Zhao et al., 2011); thus, we do not take TSV power into account in our power simulations. We assume that TSVs have 10 μm diameters and a center-to-center pitch of 20 μm. The total area overhead of TSVs is less than 0.1 mm^2, which is negligible compared to the total chip area of 10.9 mm^2. Prior work (Homayoun et al., 2012) shows that the layer to layer delay caused by TSVs is 1.26 ps, which has no impact on the system performance as it is much smaller than the CPU clock period at 1 GHz. If there is on-chip DRAM, the memory controller is also connected to DRAM through TSVs, which brings extra 512 TSVs for data transmission and 32 TSVs to send commands to a memory module.

3.2.2 Cache Resource Management Policy

To efficiently manage the cache resources and improve the energy efficiency using 3D-CRP, we introduce a runtime job allocation and cache resource pooling policy. We first explain the details of the proposed policy for the system shown in Fig. 3·5, where each layer has one core and a 1MB L2 cache. Then we introduce the extended policy for larger 3D-CRP systems.

Overview of Cache Pooling

Based on the fact that different applications need varying cache resources to achieve their optimal energy efficiency, as stated in Section 3.1, we propose a two-stage runtime policy to allocate the cache resources within a single-column 3D-CRP system. The flowchart is shown in Fig. 3·8. The policy contains two stages: (1) *Job allocation*, which decides on the core each job should run on, and (2) *Cache resource pooling*, which distributes the cache resources between a pair of jobs.

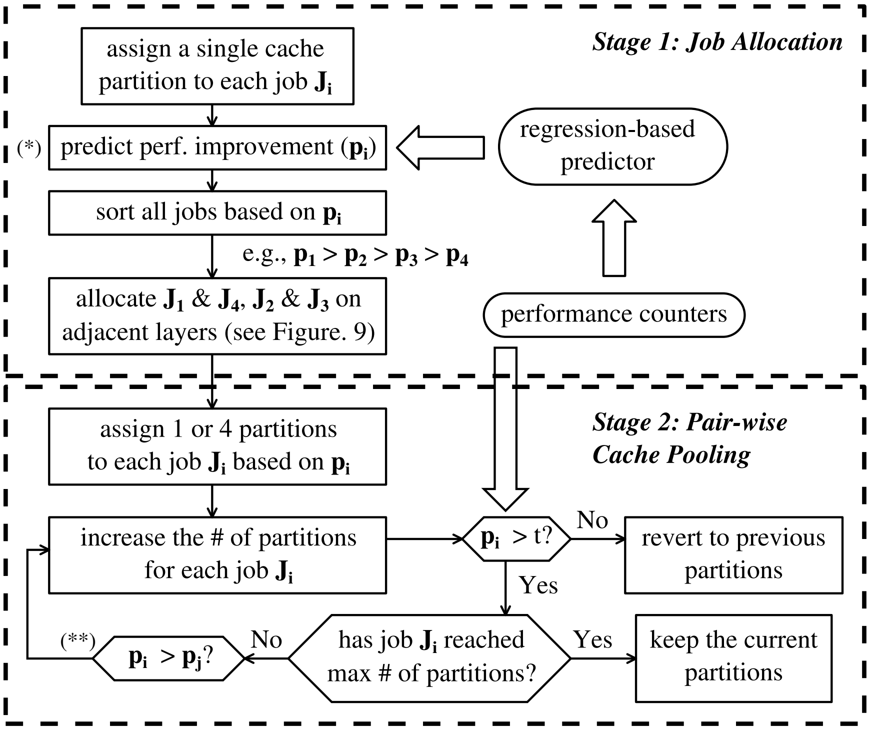
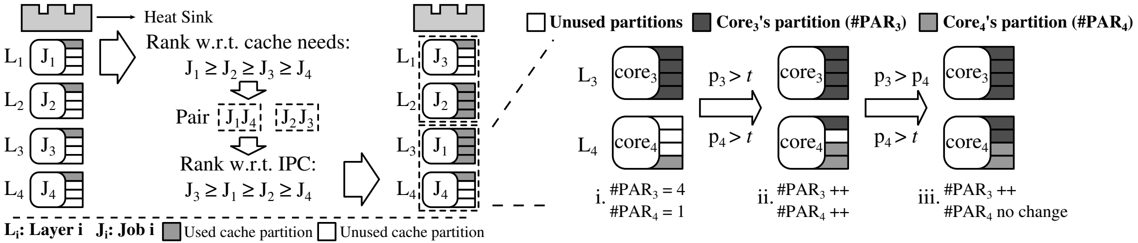

Stage 1: Job Allocation
assign a single cache partition to each job J_i
(*)
predict perf. improvement (p_i)
regression-based predictor
sort all jobs based on p_i
e.g., $p_1 > p_2 > p_3 > p_4$
allocate J_1 & J_4, J_2 & J_3 on adjacent layers (see Figure. 9)
performance counters
Stage 2: Pair-wise Cache Pooling
assign 1 or 4 partitions to each job J_i based on p_i
increase the # of partitions for each job J_i
p_i > t?
No
revert to previous partitions
Yes
has job J_i reached max # of partitions?
Yes
keep the current partitions
No
p_i > p_j?
(**)

Figure 3·8: A flow chart illustrating our runtime job allocation and cache resource pooling policy. (*) p_i represents the predicted IPC improvement for each job when running with 4 cache partitions compared to running with 1 partition. (**) Condition is checked only if J_i and J_j are competing for the same partition.

Stage 1: Job Allocation Across the Stack

In this stage, we allocate the jobs to the 3D system with both energy efficiency and thermal considerations. The allocation is based on an estimation of the jobs' IPC improvement (p_i) when running with four partitions compared to running with one partition. The estimation of p_i is conducted using an off-line linear regression model that takes runtime performance counter data as inputs. At the beginning, we assign n jobs to n cores in the system in a random manner, and start running the jobs for an interval (e.g., 10 ms) using the default reserved cache partition (each core has a reserved L2 cache partition of 256 KB that cannot be pooled). The performance counters that we use in estimation are L2 cache replacements, L2 cache write accesses, L2 cache read misses, L2 cache instruction misses, and number of cycles. The linear regression model is constructed by their linear and cross items. We train the regression model with performance statistics from simulations across 15 of our applications and validate the model using another four applications. The prediction error is less than 5% of the actual performance improvement on average. When implemented in a real system, this predictor can be integrated with the OS. The OS needs to periodically read the hardware performance counters to collect data and feedback to the predictor.

In the next step, we sort the jobs based on their predicted performance improvements and group them in pairs by selecting the highest and lowest ones from the remaining sorted as ($J_1 \geq J_2 \geq J_3 \geq J_4$) according to their p_i. In this case, we group four jobs into two pairs (J_1, J_4 and J_2, J_3). As for temperature consideration, we allocate the job pair with higher average IPC to the available cores closest to heat sink as shown in Fig. 3·9 (a). The reason is that the cores on layers closer to the heat sink can be cooled more effectively in comparison to cores farther from the heat sink (Coskun et al., 2009a).



(a) An example to illustrate the job allocation stage

(b) An example that illustrates the cache resource pooling among a job-pair in our run-time policy. *#PAR* refers to the number of cache partitions

Figure 3·9: The 2-stage intra-column runtime job allocation and cache resource pooling policy.

Stage 2: Cache Resource Pooling Among Job Pairs

In the second stage, we propose a method to manage the cache resources within each job pair. In order to determine whether a job needs more cache partitions, we introduce a performance improvement threshold (t). This threshold represents the minimum IPC improvement a job should get from an extra cache partition to achieve a lower EDP. We use ($Power/IPC^2$) to calculate EDP. The key to derive t is based on the observation: the EDP of cache-hungry jobs decreases when the number of cache partitions of the job increases due to the high performance improvement. On the contrary, for non-cache-hungry jobs, the EDP increases when the acquired cache partitions increase because the performance is only slightly improved while the energy consumption increases. For a lower EDP, the following inequality should be satisfied:

$$\frac{Power}{IPC^2} > \frac{Power + \triangle Power}{(IPC + \triangle IPC)^2} \tag{3.1}$$

IPC and $Power$ refer to performance and power values before we increase the number of cache partitions, while $\triangle IPC$ and $\triangle Power$ are the variations in IPC and power when the job uses an additional partition. From this inequality, we obtain:

$$\frac{\triangle IPC}{IPC} > \sqrt{1 + \frac{\triangle Power}{Power}} - 1 = t \tag{3.2}$$

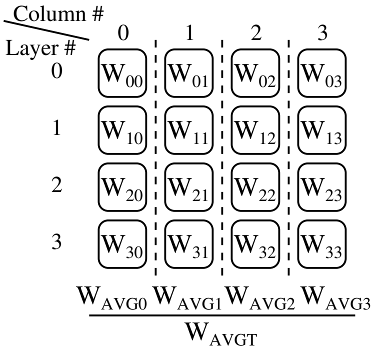
When performance improvement is larger than t, increasing the number of partitions reduces the EDP of the job. We compute t as 3% on average based on our experiments with 19 SPEC benchmarks. We compute the amount of cache partitions to assign to each job by utilizing t and p_i. If p_i of one job is greater than 9.3% ($(1+3\%)^3-1$), we assign four cache partitions to it; otherwise, we keep one partition for the job. The 9% is obtained from the threshold of increasing the partition from one to four. Then, we iteratively increase the number of cache partitions for each job if three conditions are satisfied: (1) $p_i > t$, (2) the job has not reached the maximum number of partitions, and (3) $p_i > p_j$. The maximum number of partitions is seven for jobs that are assigned four partitions, and four for jobs that are assigned one partition. If $p_i < t$, we revert the job to previous partitions. We keep the job with current partitions once it reaches the maximum number of partitions. The last condition is only checked if jobs J_i and J_j are competing for the same cache partition.

We give an example cache assignment where one job in a job-pair is assigned one cache partition and the other job is assigned four cache partitions, as shown in Fig. 3·9 (b). In step i, the performance improvements of both jobs are greater than the threshold, so we increase one cache partition for both $Core_3$ and $Core_4$ in step ii. At last, we assign the cache partition to the job with higher performance improvement ($Core_3$ in this case).

Inter-Column Job Allocation on Larger 3D-CRP Systems

For larger 3D-CRP systems with multiple columns, the cache requirements and performance of cores might be different across the columns. To balance the cache-hungriness among the columns, we perform inter-column job allocation after sorting the jobs as a load balancing policy in such systems.

We first assign weights to each core according to the corresponding cache requirements. We then average the weights for each column (W_{AVGi}) and the whole 3D



Inter-Column Reallocation:

```
while (if $|W_{AVGi}-W_{AVGT}| \geq threshold$ for any i = 0, 1, 2, 3)
  do
  1. Sort $W_{AVGi}$;
  2. Take the task with the largest weight in the column which
     has the largest $W_{AVGi}$ as task 1;
  3. Take the task with the smallest weight in the column which
     has the smallest $W_{AVGi}$ as task 2;
  4. Swap task 1 and task 2;
  5. Compute the new $W_{AVGi}$ for i = 0, 1, 2, 3;
endwhile
```

Figure 3·10: The illustration of inter-column job allocation algorithm.

system (W_{AVGT}) and compare each W_{AVGi} with W_{AVGT} to see the difference. A threshold is set up to check whether the difference between W_{AVGi} and W_{AVGT} is large. If the threshold is exceeded, the highest-weight task in the column with the largest W_{AVGi} and the lowest-weight task in the column with the smallest W_{AVGi} are swapped to balance the cache-hungriness. This process is iterated until the difference between each W_{AVGi} and W_{AVGT} is under the threshold. We perform job migration if needed after the iteration converges. The algorithm is shown in Fig. 3·10. After the inter-column job allocation, the system pairs all jobs and decides the cache resource allocation inside each column as stated in the previous subsections. Furthermore, from the memory access perspective, since the jobs within one column could have higher average memory access than the jobs in the other columns, the memory access latency of this particular column may be potentially higher than the latency of the other columns. Therefore, when doing the inter-column job allocation, we also consider the L2 miss per cycle (MPC). We balance the L2 MPC as well as the cache hungriness to balance the memory accesses among all columns.

Figure 3·11 shows a simple example of job allocation in a 16-core 3D system with cache resource pooling architecture. In this system, we have four columns and each column has four cores; the columns are named C_1, C_2, C_3, and C_4. Columns C_1 and C_4 initially have four and three cache-hungry jobs, respectively, while columns C_2

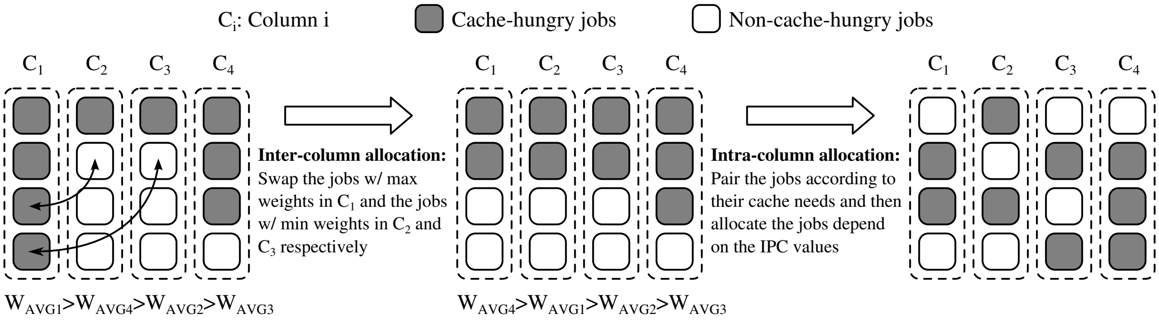

C$_i$: Column i
Cache-hungry jobs
Non-cache-hungry jobs
C$_1$ C$_2$ C$_3$ C$_4$
Inter-column allocation:
Swap the jobs w/ max weights in C$_1$ and the jobs w/ min weights in C$_2$ and C$_3$ respectively
Intra-column allocation:
Pair the jobs according to their cache needs and then allocate the jobs depend on the IPC values
W$_{AVG1}$>W$_{AVG4}$>W$_{AVG2}$>W$_{AVG3}$
W$_{AVG4}$>W$_{AVG1}$>W$_{AVG2}$>W$_{AVG3}$

Figure 3·11: An example of job allocation in a 16-core 3D system.

and C_3 only have one cache-hungry job each. We assume that $W_{AVG1} > W_{AVG4} > W_{AVG3} > W_{AVG2}$. The job with the highest weight in C_1 is swapped with the job with the lowest weight in C_2 and we get $W_{AVG1} > W_{AVG4} > W_{AVG2} > W_{AVG3}$. Similarly, we swap the new highest-weight job in C_1 with the lowest-weight job in C_3 this time and get the difference between each W_{AVGi} and W_{AVGT} under the threshold. After the inter-column job reallocation, the cache-hungriness is balanced across the columns. Next, we perform the proposed intra-column job pairing and allocation to finalize the location of each job. In a larger 3D system, using this inter-column job allocation, the cache needs are balanced and the cache resources can be utilized more efficiently.

Performance Overhead Evaluation

In order to improve the energy efficiency of the 3D system in presence of workload changes, we run our runtime policy periodically every 100 ms. We re-allocate the cache partitions among job pairs and flush the cache partitions whenever there is re-allocation. In the worst case, we decrease the number of cache partitions for a job from four to one or increase the cache partitions from four to seven, which both result in the cache partitions getting flushed three times. Following a new cache configuration, there is no relevant data in the L2 cache. Thus, the applications begin to execute with cold caches. System performance degrades due to the cold start effect in caches. Prior work estimates the cold start effect of a similar SPEC benchmark suite as less

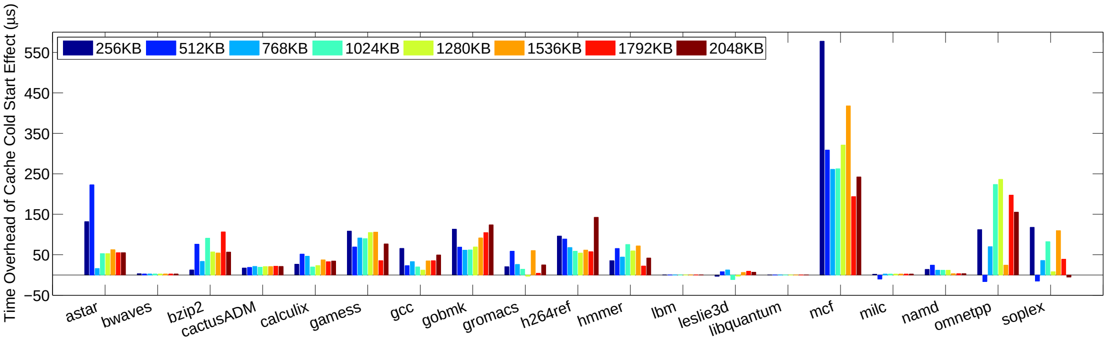


Figure 3·12: Time overhead (μs) of cache cold start effect for all applications under cache sizes from 256 KB to 2048 KB

than 1 ms (Coskun et al., 2009b). We also evaluate the cold start effect overhead by comparing the performance of the benchmark suite with and without cache warmup, as shown in Fig. 3·12. We can see that almost all applications suffer from cache cold start effects, in different amounts. For example, mcf suffers most from cold caches because it is much more cache intensive compared to the other applications. On the contrary, lbm almost has no cold start overhead because it does not use much cache. The highest overhead is around 500 μs from mcf. For most of the applications, the overhead is lower than 150 μs. For multi-core systems, the memory access rate is higher than single-core systems, which increases memory access latency. Thus, we also perform similar experiments for various memory access latencies and the results demonstrate that the cache warmup overhead is still under 1 ms. Other than cache cold start effect, when job migration happens, context switch also introduces performance degradation. However, the context switch overhead is no more than 10 μs (Constantinou et al., 2005; Kamruzzaman et al., 2011), and techniques such as *fast trap* can further reduce time spent on it (Gomaa et al., 2004). Thus, the performance overhead of our policy is negligible for SPEC type of workloads.

Table 3.1: Core Architecture Parameters

Parameter	High-Perf	Low-Power
CPU Clock	2.1 GHz	1.0 GHz
Issue Width	out-of-order 3-way	out-of-order 2-way
Reorder Buffer	84 entries	40 entries
BTB/RAS size	2048/24 entries	512/16 entries
Integer/FP ALU	3/3	2/1
Integer/FP MultDiv	1/1	1/1
Load/Store Queue	32/32 entries	16/12 entries
L1 I/DCache	64 KB, 2-way, 2 ns	16 KB, 2-way, 2 ns
L2 Cache	1 MB, 4-way, 5 ns	1 MB, 4-way, 5 ns
Core Area	15.75 mm^2	3.88 mm^2

3.2.3 Experimental Methodology

Target System

We apply the proposed cache resource pooling technique on low-power and high-performance 3D multi-core systems with four cores and 16 cores, respectively. The core architecture for the low-power system is based on the one used in Intel SCC (Howard et al., 2011). For the high-performance system, we use the core architecture applied in the AMD Magny-Cours processor (Conway et al., 2009). The architectural parameters for both systems are listed in Table 3.1. For the 4-core 3D-CRP system, all four cores are stacked in one column using off-chip DRAM. In the 16-core system, there are four layers and each layer has four cores. Thus, there are four columns in the system and cores could pool cache resources within each column. Each column in the 3D system has a memory controller, which is located on the layer farthest from the heat sink as shown in Fig. 3·7. The stacked DRAM layers are placed at the bottom of the 3D system, as described in Section 3.2.1. Due to the area limit, the low-power 3D system needs two DRAM layers to support 1 GB DRAM while the high-performance system only needs one layer.

Table 3.2: Main Memory Access Latency for the 3D CRP System

LLC-to-MC	0 ns (due to the short latency provided by TSVs)
Memory Controller	Queuing delay, computed by M/D/1 queuing model
Main Memory	On-chip 1 GB DRAM: $t_{RAS} = 36\ ns, t_{RP} = 15\ ns$
Total Delay	Queuing delay + t_{RAS} + t_{RP}
Memory Bus	On-chip memory bus, 2 GHz, 64-byte bus width

Simulation Framework

For our performance simulation infrastructure, we use the system-call emulation mode in Gem5 simulator (Binkert et al., 2011) with X86 instruction set architecture. For single-core simulations shown in Section 3.1, we fast-forward two billion instructions and then execute 100 million instructions in detailed mode for all applications under L2 cache sizes from 0 to 2 MB. For 4-core and 16-core simulations with the proposed CRP technique, we also collect performance results from the same segment of instructions. We run McPAT 0.7 (Li et al., 2009) under 45 nm process to estimate dynamic power consumption of the cores and then calibrate the results using published power values. We use CACTI 5.3 (Thoziyoor et al., 2008) to compute L2 cache power and area, and scale the dynamic L2 cache power based on L2 cache access rate. We use HotSpot 5.02 (Skadron et al., 2003) for thermal simulations.

In this work, we apply the M/D/1 queuing model for each memory controller to model the queuing delay rather than using a unified memory latency for all multi-program workload sets. In the M/D/1 model, arrival rate (λ) and service rate (μ) are required to compute the queuing delay, $t_{queuing}$, as shown below:

$$t_{queuing} = \frac{\lambda}{2\mu(\mu - \lambda)} \tag{3.3}$$

In 3D-CRP system, there is one memory controller in each column, thus for multi-program workloads we sum up the memory access rate of each core in the column as the arrival rate to the corresponding memory controller. We use the DRAM

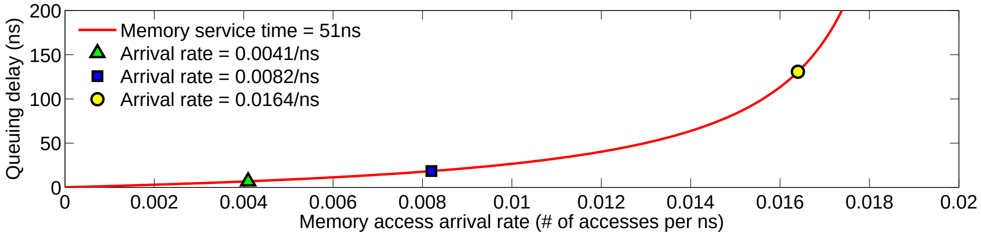


Figure 3·13: The relationship between memory access arrival rate and memory controller queuing delay. The data points from left to right represent the memory access arrival rate of one bzip2, two instances of bzip2 and four instances of bzip2, respectively.

response time (t_{RAS}+t_{RP}) as the memory system service rate. For each multi-program workload, we first assign a sufficiently large value as the memory access latency to ensure that the arrival rate would not exceed the memory system service rate. We then run performance simulations and collect the memory access rate of the workload. Based on this arrival rate (λ_1), we compute the queuing delay (t_1) of the memory controller. The new memory access latency is the sum of LLC-to-MC delay, DRAM module access time and the queuing delay ($t_1 + t_{RAS} + t_{RP}$), as shown in Table 3.2. Next, we feed this new latency back to Gem5 and collect the arrival rate (λ_2) from the second round simulations. If λ_1 and λ_2 converges (e.g., within 10% difference), the new queuing delay (t_2) is similar to t_1 and $t_1 + t_{RAS} + t_{RP}$ is the correct memory access latency in turn. Otherwise, we need to keep iterating until two consecutive arrival rates converge. Based on our experience, the arrival rates always converge to a small range after three iterations. By doing this we assign a memory access latency value according to the various workloads' memory intensiveness, which improves the accuracy of the performance results. In Fig. 3·13 we show the relationship between the memory access arrival rate and queuing delay as computed by Equation 3.3. Here we take bzip2 as an example. When running bzip2 with 4-way 1 MB L2 cache, the memory access rate is 0.0041 $/ns$ and the corresponding queuing latency is 6.75 ns. If there are two instances of bzip2 in the system, the memory access rate doubles and

Table 3.3: Benchmark classification according to memory-intensiveness and cache-hungriness

	Memory-intensive	**Non-memory-intensive**
Cache-hungry	bzip2, omnetpp (1-3), soplex (1-6)	omnetpp (4-7), soplex (7)
Non-cache-hungry	bwaves, gcc, gobmk(1), mcf, libquantum, lbm leslie3d	astar, calculix, cactusADM milc, namd, gobmk (2-7) gromacs, h264ref, hmmer

Table 3.4: 4-core system workload sets

Workload	**Benchmarks**
non-cache-hungry1	bwaves, gromacs, gobmk, milc
non-cache-hungry2	calculix, leslie3d, milc, namd
low-cache-hungry1	gamess, leslie3d, libquantum, omnetpp
low-cache-hungry2	bwaves, hmmer, namd, bzip2
med-cache-hungry1	astar, bzip2, soplex, mcf
med-cache-hungry2	bzip2, cactusADM, hmmer, omnetpp
high-cache-hungry1	gromacs, bzip2, omnetpp, soplex
high-cache-hungry2	h264ref, bzip2, omnetpp, soplex
all-cache-hungry1	soplex, soplex, omnetpp, bzip2
all-cache-hungry2	soplex, bzip2, soplex, bzip2

the queuing delay becomes 18.3 ns. When there are four instances of bzip2 running, the queuing delay increases to 130.7 ns. Figure 3·13 shows this relationship. As the memory access rate increases, the queuing delay increases exponentially. When there are multiple memory controllers in the system, the memory accesses get distributed; thus, the memory access latency is lower.

3.2.4 Evaluation

Multi-program Workload Sets

To test 3D-CRP and cache resource pooling policy, we select 19 applications from the SPEC CPU 2006 benchmark suite as listed in Fig. 3·1. According to the applications' memory-intensiveness and cache-hungriness, we categorize the applications into four classes as shown in Table 3.3. The number following an application refers to the corresponding cache configurations. For example, omnetpp (1-3) means that when running with one to three cache partitions, omnetpp is memory-intensive. For 4-core

Table 3.5: 16-core system workload sets. *nch, lch, mch, hch,ach* represent non-cache-hungry, low-cache-hungry, med-cache-hungry, high-cache-hungry and all-cache-hungry, respectively.

Workload	**Single column workload sets**			
nch + nch	nch1	nch2	nch1	nch2
nch + lch	nch1	nch2	lch1	lch2
nch + mch	nch1	nch2	mch1	mch2
nch + hch	nch1	nch2	hch1	hch2
nch + ach	nch1	nch2	ach1	ach2
lch + ach	lch1	lch2	ach1	ach2
mch + ach	mch1	mch2	ach1	ach2
hch + ach	hch1	hch2	ach1	ach2
ach + ach	ach1	ach2	ach1	ach2

3D systems, we create ten multi-program workload sets with four threads each, by combining cache-hungry and non-cache-hungry applications as shown in Table 3.4. We use *nch* to represent non-cache-hungry workload composition. Similarly, we apply *lch*, *mch*, *hch*, and *ach* to represent the other workload compositions. Among these workloads, *nch* contains only non-cache-hungry applications, *lch*, *mch* and *hch* includes one, two, and three cache-hungry applications respectively, while *ach* includes only cache-hungry applications. For 16-core 3D systems, we group four 4-core workload sets for each 16-core workload set based on cache needs, as shown in Table 3.5. From top to bottom, the number of cache-hungry applications in the workload set increases. When presenting the results, we compare IPC and EDP for each workload set under different 3D systems. Since area is a very important metric for evaluating the 3D systems (die costs are proportional to the 4^{th} power of the area (Rabaey et al., 2003)), we also use energy-delay-area-product (EDAP) as a metric to evaluate the cumulative energy and area efficiency (Li et al., 2009) for the 3D systems.

Performance & Energy Efficiency Evaluation

For both low-power and high-performance 3D systems, we provide three baseline systems where each core has a: (1) fixed 1 MB private L2 cache; (2) fixed 2 MB private L2 cache; (3) 1 MB private cache with selective cache ways (SCW) (Albonesi,

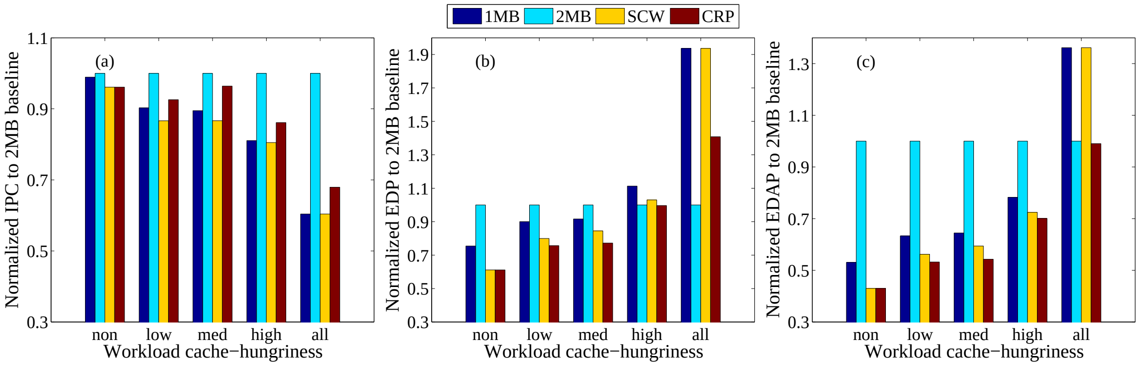


Figure 3·14: Normalized IPC, EDP and EDAP of low-power 3D-CRP system and the 3D baseline systems with 1 MB fixed caches, 2 MB fixed caches and 1 MB caches with selective cache way.

1999). For the SCW baseline system, we also use the proposed policy to decide the best cache partitions for each job, but jobs can only require a maximum of four cache partitions since SCW does not allow pooling cache partitions from the other layers.

4-core 3D System

Figure 3·14 shows the IPC, EDP, EDAP comparison between 3D-CRP system and the other three baselines for the 4-core low-power system. All of the values for each metric are normalized to 2 MB baseline. As expected, the 2 MB baseline always has the best performance among all systems. Among 1 MB baseline, SCW baseline and 3D-CRP, SCW's performance is always slightly lower than 1 MB baseline while 3D-CRP outperforms 1 MB baseline as long as there are cache-hungry jobs in the workloads. The performance improvement of 3D-CRP over 1 MB baseline is up to 11.2% among all workloads. The reason is that 3D-CRP always turns off the unnecessary cache partitions and pools them to cache-hungry jobs, which boosts the system performance. In Fig. 3·14 (b), it is obvious that for all workloads, 3D-CRP provides lower EDP than 1 MB and SCW baselines. On average, 3D-CRP reduces EDP by 18.8% and 8.9% compared to 1 MB and SCW baselines respectively. For SCW, the non-sharing feature limits its improvement in EDP for workload sets with high cache-hungriness (e.g., *hch* and *ach* workloads). For the workloads that contain

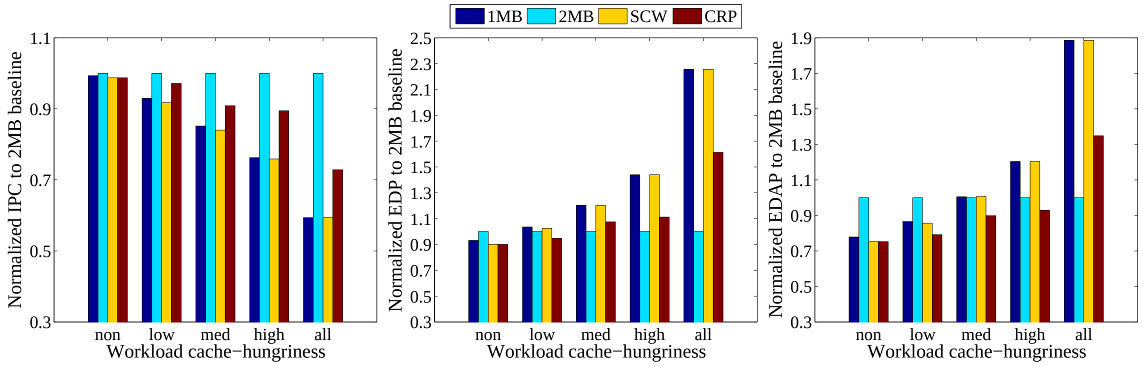


Figure 3·15: Normalized IPC, EDP and EDAP of high-performance 3D-CRP system and the 3D baseline systems with 1 MB fixed caches, 2 MB fixed caches and 1 MB caches with selective cache way.

both cache-hungry jobs and non-cache hungry jobs, 3D-CRP improves the energy efficiency by up to 38.9% compared to 2 MB baseline. Although the 2 MB baseline always provides the best performance, it cannot offer the optimal system energy efficiency. For EDAP, as shown in Fig. 3·14 (c), 3D-CRP outperforms all baselines for all workloads. The improvements brought by CRP over 1 MB and SCW cases are the same as that of EDP because they have the same area, and the average EDAP improvement of CRP over 2 MB baseline is 36.1%.

We also evaluate performance and energy efficiency for the high-performance system using the proposed 3D-CRP design and runtime policy, as shown in Fig. 3·15. Similar to the low-power 3D system, 3D-CRP achieves lower energy efficiency than 1 MB and SCW baselines and the average EDP reduction is 14.8% and 13.9% respectively. When comparing to 2 MB baseline, the EDP results are different because the high-performance core consumes much more power than the low-power core and 1 MB L2 cache. Thus, the percentage of extra power consumption introduced by larger L2 caches is smaller than the percentage of extra performance improvement when it comes to cache-hungry jobs, which can be expressed as:

$$\frac{\triangle IPC}{IPC} > \frac{\triangle Power}{Power} \Rightarrow \frac{Power}{IPC^2} < \frac{Power + \triangle Power}{(IPC + \triangle IPC)^2} \tag{3.4}$$

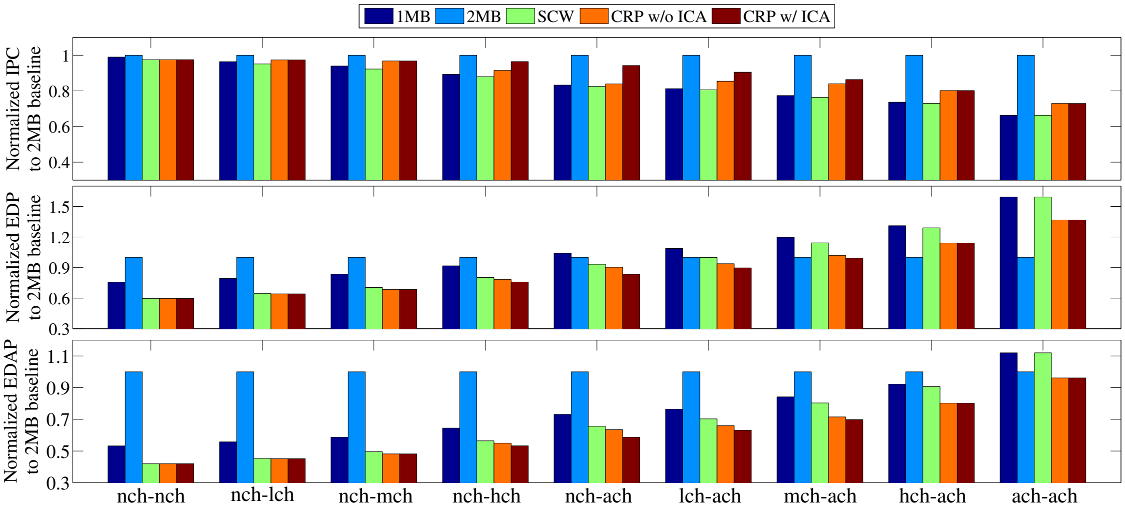


Figure 3·16: Normalized IPC, EDP and EDAP of 16-core low-power 3D-CRP system and the 3D baseline systems with 1 MB fixed caches, 2 MB fixed caches and 1 MB caches with selective cache way. ICA refers to Inter-column job allocation.

Thus, it ends up with lower energy efficiency. Nevertheless, for the workloads mixed with cache-hungry and non-cache-hungry jobs, 3D-CRP performs similarly with 2 MB baseline on EDP, while from EDAP perspective, 3D-CRP still improves over 2 MB baseline by up to 24.7% for *nch* and by 12.7% on average for mixed workloads.

We also integrate 3D systems with microarchitectural resource pooling (MRP) (Homayoun et al., 2012). To evaluate the performance improvement with MRP, we run applications with four times of the default sizes of the performance critical components (reorder buffer, instruction queue, register file, and load/store queue), and compare the IPC results with the results with default settings. For applications running on a single low-power core, our experiments show that MRP improves system performance by 10.4% on average, and combining MRP and CRP provides an extra performance improvement of 8.7% on average in comparison to applying MRP alone.

16-core 3D System

We also evaluate our runtime policy on a 16-core low-power 3D-CRP system with stacked DRAM to investigate our policy's scalability. The 16-core system has four layers with four cores on each layer, and each core has a private L2 cache. The DRAM layers are located at the bottom of the chip. The workloads for 16-core low power system are listed in Table 3.5. Figure 3·16 shows the performance and energy efficiency results of 3D systems with different cache architectures. In this figure, from left to right, the cache-hungriness of the workloads increases. For all nine workloads, 3D-CRP outperforms 1 MB and SCW baselines in IPC, EDP and EDAP. 3D-CRP is always better than the 2 MB baseline on EDAP. In addition, with inter-column job allocation (ICA), there are further IPC and EDP improvement for 3D-CRP. For example, for *nch-ach* workload, *3D CRP + ICA* further improves performance by 12.3% and energy-efficiency by 7.8% compared to 3D CRP only. When considering memory accesses among the columns and adjusting the workload accordingly, the system performance improvement is around 5% for all the workload sets.

Thermal Evaluation

We conduct steady-state temperature simulations to evaluate the impact of the proposed runtime policy on the on-chip temperature in 3D-CRP systems. For a 4-core low-power 3D system, since the cores have quite low power consumption, the temperature benefits are slight. While for a 4-core high-performance 3D system, we observe up to a 6.2 ^oC reduction in peak on-chip temperature compared to the temperature-wise worst possible job allocation for 4-core workloads. Across all workloads there is an average of 3.4 ^oC reduction in peak on-chip temperature. Such observations show that our policy can effectively decrease the system peak temperature and prevent cores from exceeding the temperature threshold. In the 16-core 3D system, our results show that without the temperature-aware job allocation only *mch-ach* and *ach-ach* operate under the system temperature threshold 85 ^oC, while applying temperature-aware job allocation keeps all 16-core workloads operating under 85 ^oC.

3.3 Summary

This chapter has proposed a novel design for 3D cache resource pooling that requires minimal additional circuitry and architectural modification. We have first quantified the impact of cache sizes and memory access latency on the application performance. We have then presented an application-aware job allocation and cache pooling policy to improve the energy efficiency and the thermal behavior of 3D systems. Our policy dynamically allocates the jobs to cores on and distributes the cache resources based on the cache hungriness of the applications. In addition, we have designed a memory controller delay model to adjust the memory access latency for different workloads and leveraged this model for all the 3D multi-core system evaluations. Experimental results show that by utilizing cache resource pooling we are able to improve system EDP and EDAP by 18.8% and 36.1% on average compared to 3D systems with fixed cache sizes. The proposed inter-column job allocation manages to additionally improve performance by up to 12.3% and energy-efficiency by up to 7.8% for larger 3D systems with on-chip DRAM. On the thermal side, our policy reduces the peak on-chip temperature of high-performance systems by up to 6.2 ^oC.

Chapter 4

Memory Resource Management in 3D Multi-core Systems

3D stacking enables new ways of integrating memory resources in computation systems. For example, memory resources could be distributed in each layer within a 3D system composed of homogeneous layers, or a separate DRAM chip could be stacked on top of a logic chip for fast and high-bandwidth memory access. One other option is to have both on-chip memory and off-chip memory in a system, which provides not only high-performance memory access but also large capacity. Along with such advantages enabled by 3D stacking, there also comes challenges in how to efficiently manage these memory resources. In this chapter, we propose memory management techniques to improve the performance and energy efficiency of 3D systems. We first propose a memory access scheduling policy in a 3D modular multi-core architecture (3D-MMC, a 3D system with homogeneous layers), then, we develop a novel method to allocate memory resources in heterogeneous memory systems.

4.1 Memory Access Scheduling in 3D-MMC

This section demonstrates a functional hardware and software design for a 3D-stacked multi-core system. The presented 3D system is a low-power 3D modular multi-core architecture built by vertically stacking identical layers. Each layer consists of cores, private and shared memory units, and communication infrastructures. The system uses shared memory communication and TSVs to transfer data across layers. A se-

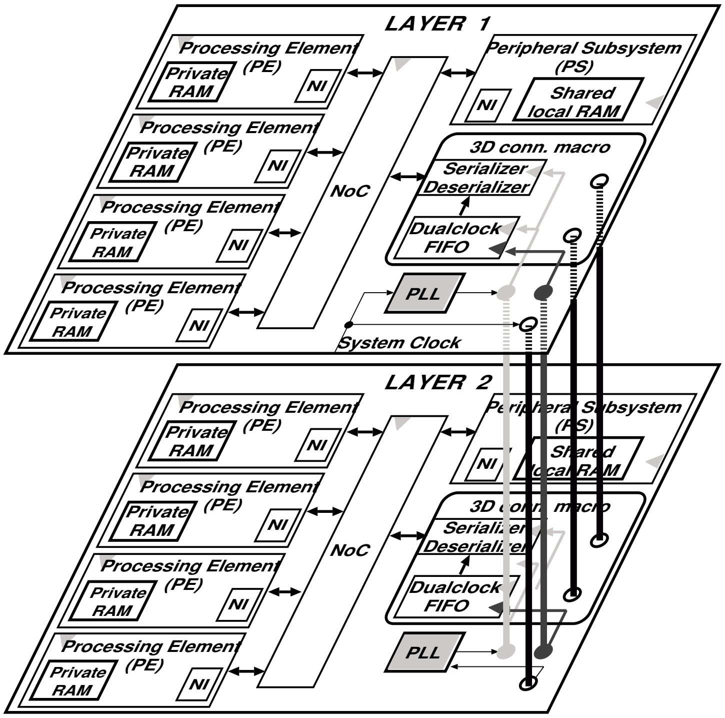


Figure 4·1: Overview of the 3D-MMC built with stacking identical layers. The figure does not show the data TSVs for clarity.

rialization scheme is employed for inter-layer communication to minimize the overall number of TSVs. The proposed architecture has been implemented in HDL and verified on a test chip. We evaluate the performance, power, and temperature characteristics of the architecture using a set of software applications we designed. To efficiently manage the shared memory of the 3D-stacked system, we propose a novel memory resource pooling approach which adapts the memory access based on the available memory resources in the 3D system.

4.1.1 3D Modular Multi-core Architecture Design

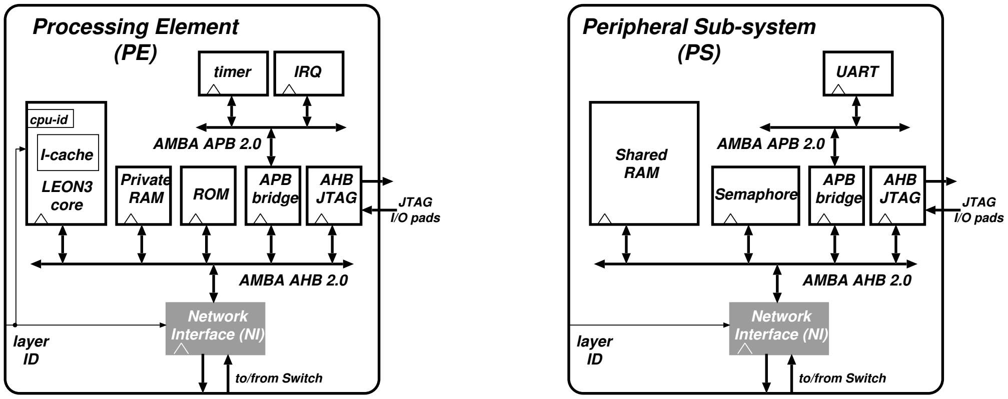
In this subsection, we provide the details of 3D-MMC, describe out performance evaluation setup, and then demonstrate the thermal feasibility of this system. Figure 4·1 provides a diagram of 3D-MMC system composed of two layers interconnected by TSVs. The number of layers in the stack determines the total core count as well as the memory size and the level of memory resource pooling. A single layer can function either as a stand-alone 2D-CMP and as part of a 3D-system when integrated with the 3D communication infrastructure.

Planar (single-layer) architecture

A single layer is composed of four *Processing Elements* (PE) that exchange data through a shared memory, which is placed in the *Peripheral Subsystem* (PS) unit. A system of semaphores arbitrates the access of PEs to the shared memory. The routing between each PE and the shared memory occurs through a NoC. Several 3D-NoC topologies have been proposed recently, and their superior performance over 2D-NoC have been demonstrated (Pavlidis and Friedman, 2007). In 3D-MMC, a specific source-routed NoC is implemented to manage the signals routing to and from six directions (North, South, East, West, Up, Down). In the 3D stack, NoC on different layers are interconnected to enable the management of the signals in both horizontal and vertical directions.

Figures 4·2a and 4·2b illustrate the internal architecture of a PE and a PS, respectively. Each PE is built out of a 32-bit RISC processor, the open-source LEON3 unit from Aeroflex Gaisler, which is connected to the slave modules through an AMBA bus. An AHB JTAG master module is included for debugging purposes. Slave devices for each PE are a privately addressable memory space including a ROM for booting and a private RAM to host the program code. The Network-Interface (NI) block is a master located within both PE and PS. It interfaces the AMBA bus to the NoC, and is responsible of transferring data packets to/from the shared memory, which has an address space visible to each core. Similar to PEs, the PS contains NI and AHB JTAG acting as masters, whereas the remaining units act as slaves.

Each PE in an N-layer system has access to $N + 1$ different memory modules that can be accessed in parallel: a private-RAM contained in its own PE, a shared



(a) (b)

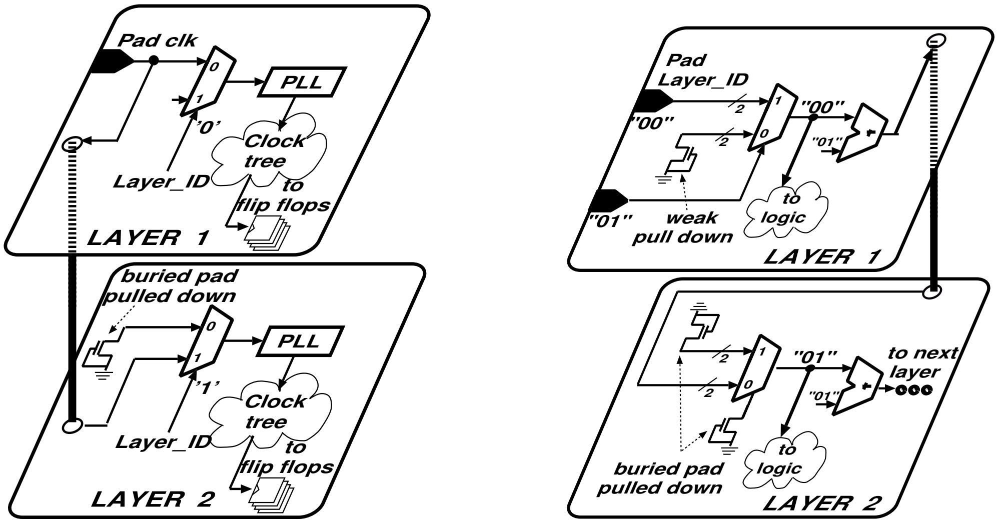
Figure 4·2: (a) PE internal architecture; (b) PS internal architecture.

local-RAM located in the PS of its layer, and $N-1$ shared remote-RAMs situated in the PS of the other stacked layers. The proposed memory hierarchy that uses shared data memory for inter-processor communication simplifies the hardware complexity and avoids memory coherency overhead. The multi-core synchronization is handled at the software level.

3D communication and control infrastructure

Inter-layer communication is achieved through a 3D communication unit, *3D-macro*, which leverages an array of TSVs as a vertical data bus. To limit the TSV area overhead, we use a serializer-deserializer (serdes) module to minimize the total number of TSVs. This serdes module is explained in detail in our prior work (Beanato et al., 2012). Data signals are serialized before the transmission through TSVs, and deserialized at the receiving layer. The bandwidth loss due to serialization can be compensated by increasing the serdes clock frequency. Serialization is more cost-efficient than parallel buses in 3D stacked systems (Sun et al., 2010).

A challenging issue for 3D stacked systems is the reliable distribution of the clock signal (Pavlidis et al., 2008). In 3D-MMC, each stacked layer has an independent clock domain. The clock is injected onto a pad of the top layer, passes through a



(a) **(b)**

Figure 4·3: (a) Clock distribution and propagation between two layers using three redundant TSVs; (b) LayerID generation and propagation between two layers using three redundant TSVs.

PLL module, and is both distributed in the circuit and sent to the next layer. The bottom layer receives the clock from power TSVs, and forwards it to a PLL module to be re-generated for maintaining its integrity. Hence, all layers operate at the same frequency, but are asynchronous from each other due to potential phase shifts among the clocks. In the multi-clock domain approach, signals are transmitted among the layers together with their clock and then they are re-synchronized to the clock domain of the receiving layer using a Dual Clock FIFO, as shown in Fig. 4·3a.

Once identical layers are stacked, they need to operate as a complete system without further modification. For this purpose, a dedicated control signal, namely the *layer identification number* (LayerID) is implemented for enabling auto-configuration of the layers depending on their positions in the 3D-system. As shown in Fig. 4·3b for an example of two layers, the sequence "00", is injected through the pads and selected by a multiplexer as the LayerID of the top layer. The value is forwarded to a half adder that computes the ID for the layer below, "01". The pads of the bottom tier are designed to be pulled-down when no signal is applied to them. As a result, the multiplexer on the second layer selects the LayerID transmitted by the TSVs.

Evaluation platform

We implement the proposed architecture in HDL and verify it on a test chip at an operating frequency of 400 MHz and a vertical bandwidth of 3.2 *Gbps*. We fabricate the 2D-CMPs using a standard UMC 90nm CMOS technology. We test and verify single dies, and then process them for in-house TSV fabrication and stacking.

We conduct the performance study in this work through cycle-accurate post-layout simulations in ModelSim. We design a set of software benchmarks to evaluate performance, and compile these benchmarks using a SPARC compiler. We load the compiled binary files into the ROM, and execute them following the boot script.

Thermal Evaluation

A significant challenge in 3D stacking is the power density increase per footprint, which may cause temperature to increase beyond reliable thresholds. This section demonstrates the thermal feasibility of 3D-MMC.

The power consumption of each component in a layer of the 3D stack is estimated via statistical power analysis using Encounter Power System by Cadence. We assume a switching rate of 50% for each flip flop and each input port, and we use a 100% toggling rate for the clock. The tool automatically estimates the power consumption based on the average toggling rate of each gate. Table 4.1 provides the power consumption of all the components at 400 MHz including leakage power. Core power in the table includes the logic, I-Cache, ROM, and all other sub-blocks of the core except for the local RAM. Table 4.1 highlights the low power consumption of 3D-MMC.

We use HotSpot version 5.02 (Skadron et al., 2003) for thermal simulations. The package and die parameters used in the simulation are provided in Table 4.1. The

Table 4.1: Power Consumption and Thermal Properties of 3D-MMC

Power Consumption Characteristics	
Components	**Power** (mW)
Core	37.98
Local RAM for each core	17.13
Router	10.07
Data TSV arrays	1.6 (smaller array) to 8.11 (larger array)
Shared memory	22.16
PLL	5
Package and Die Thermal Characteristics	
Die area	3.5 mm x 3.5 mm
Die thickness (bottom layer)	280 μm
Die thickness (other layers)	50 μm
Die (Si) resistivity	0.01 mK/W (meter-Kelvin per Watt)
Glue conductivity	0.082 W/mK at 25 ^oC
Glue thickness	2 μm

floorplan of each layer is identical and is shown in Fig. 4·1 and 4·4. To take the impact of TSVs into account during thermal evaluation, we use a modified version of HotSpot that enables modeling heterogeneity within a layer (Meng et al., 2012). We compute joint thermal resistivities for each TSV block based on the ratio of TSV (Cu) area to overall TSV array area (including all the spacing between the TSVs). We simulate the system without a heat sink by using a very small number for the heat sink thickness in HotSpot. Layers are stacked using a glue (interface material) layer of Parylene-C.

Figure 4·4 provides the steady state peak temperatures for a single layer chip and 3D systems including two layers, four layers, and eight layers when all cores are active. In the figure, we also provide a thermal map for the top layer of the 2-layered system. For 2- and 4-layered systems, even though cores overlap on top of each other and all cores are active, we do not observe high temperatures. For the 8-layered stack, peak temperature reaches 74 ^oC, which is still below the typical 85 ^oC thermal thresholds used in most processor chips. As we focus on a 2-layered stack in this work, we do not apply thermal management strategies.

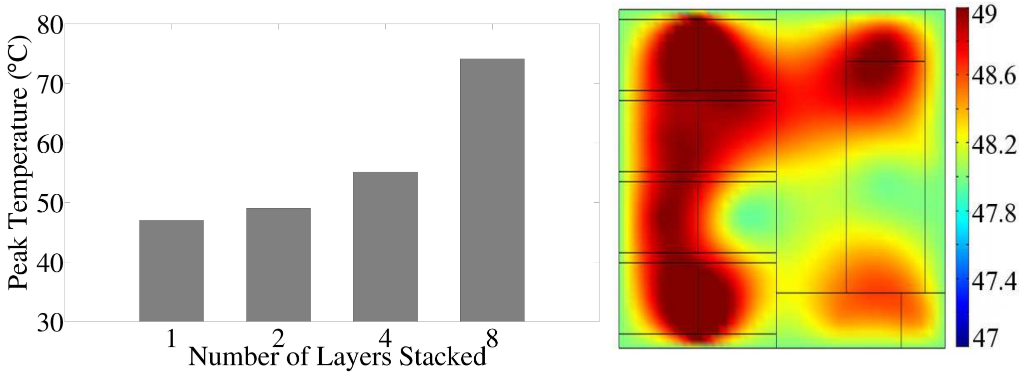


Figure 4·4: The figure demonstrates the peak temperatures at steady state for a single layer as well as 2-, 4-, and 8-layered stack. On the right, we show the thermal map of the top layer for the 2-layered stacks. Thermal variations are similarly low (limited to a few degrees only) for 4- and 8-layered stacks.

4.1.2 Resource Pooling For Efficient Memory Access

In traditional 2D design, as the number of cores increases, the bandwidth of the shared memory becomes a performance bottleneck. In 3D-MMC, the ability of pooling other layers' resources alleviates the memory bottleneck problem. This section first demonstrates the memory bottleneck problem on a 2-layer 3D-MMC, and then proposes a methodology to find the optimal way to apply resource pooling.

3D-MMC's memory subsystem has only one write port and one read port. When the memory access rate exceeds a certain level, access blocking occurs. Aiming to evaluate memory bottlenecks and resource pooling, we create a memory-intensive benchmark, *Memory Stress*, which performs 1000 writes of integer-length values into the shared memory. We perform experiments that write on local (same layer's) shared memory and remote (different layer's) shared memory respectively, with 1-core, 2-core, 3-core, and 4-core cases. In this group of experiments, all active cores are on the same layer. The resulting execution times for both local and remote shared memory cases are shown in Fig. 4·5. This figure shows that as more cores attempt to access shared memory, performance penalty increases. When there are either more than two

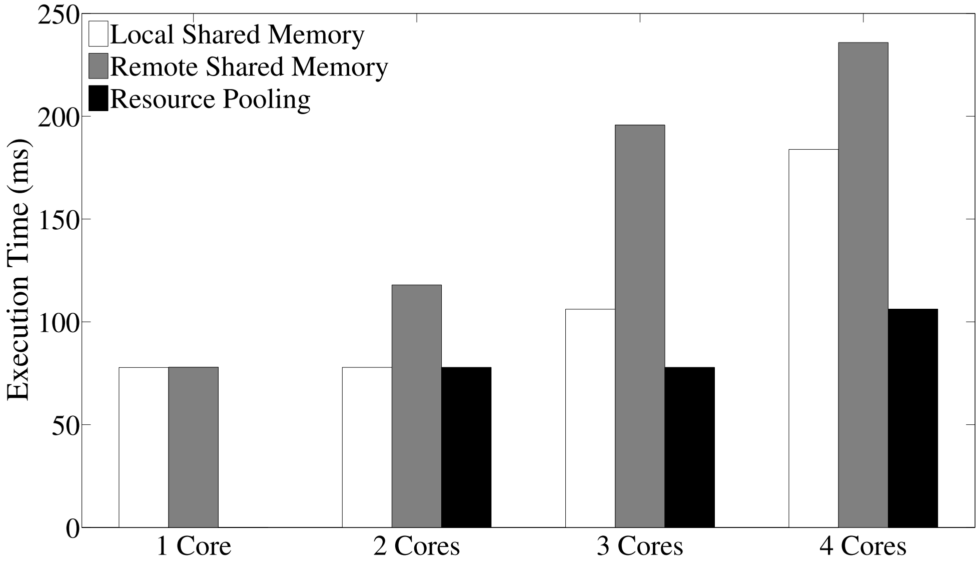


Figure 4·5: Comparison of execution time of *Memory Stress* benchmark when all cores access local memory, all cores access remote memory, and when memory resource pooling is applied.

cores accessing the remote shared memory or more than three cores accessing the local shared memory, the extra cores are blocked for a while. Blocking of cores happens because of the memory bottleneck and the communication limitation between layers.

To mitigate the local shared memory contention, we propose utilizing the remote shared memory for local cores. We call this scenario *Memory Resource Pooling.* For the experiment in Fig. 4·5, for the multi-core cases we assign one core to access the remote shared memory while the others still access the local shared memory. In the 3-core case, one core is assigned to access remote shared memory while the other two write to the local shared memory. For 3-core and 4-core cases, memory resource pooling brings 26.6% and 42.3% runtime reduction, respectively.

The above memory resource pooling strategy schedules memory accesses at the core granularity; thus, we call it Core Level Resource Pooling (CLRP). Task Level Resource Pooling (TLRP) includes adjustable *Workload Allocation* and *Workload*

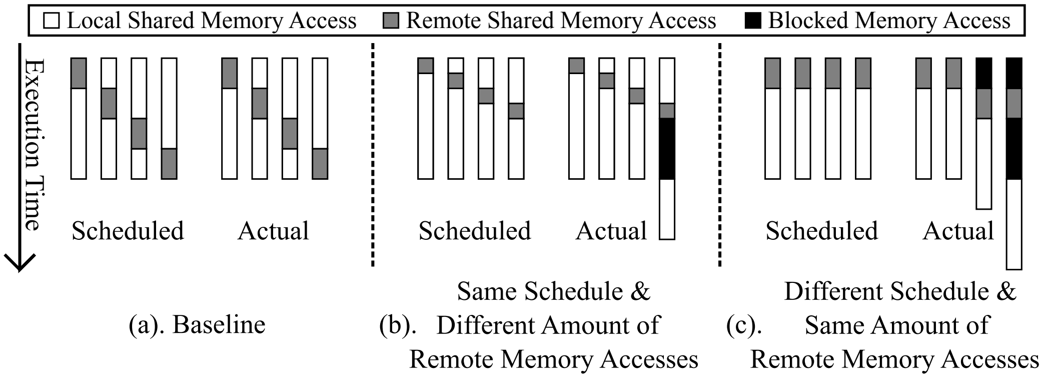
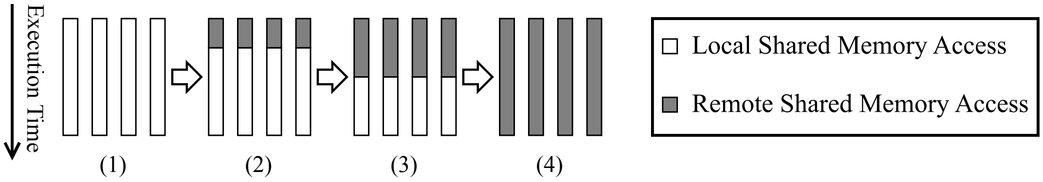

☐ Local Shared Memory Access
■ Remote Shared Memory Access
■ Blocked Memory Access
Execution Time
Scheduled
Actual
Scheduled
Actual
Scheduled
Actual
(a). Baseline
(b). Same Schedule & Different Amount of Remote Memory Accesses
(c). Different Schedule & Same Amount of Remote Memory Accesses

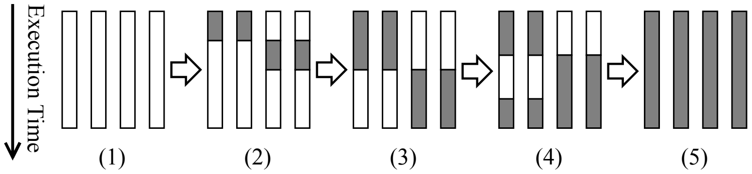
Figure 4·6: Performance under different workload allocation and scheduling combinations. (a) is the baseline, (b) has the same schedule with (a) but has fewer remote memory accesses, while (c) has the same number of remote memory accesses but has a different schedule.

Scheduling within each core. With TLRP, the workload of each core is divided into two parts: local memory accesses and remote memory accesses. For each core in the system, workload allocation determines the ratio of local and remote memory accesses, while workload scheduling defines the execution sequence of memory accesses. We allocate equal amount of workload to all the cores for a fair comparison.

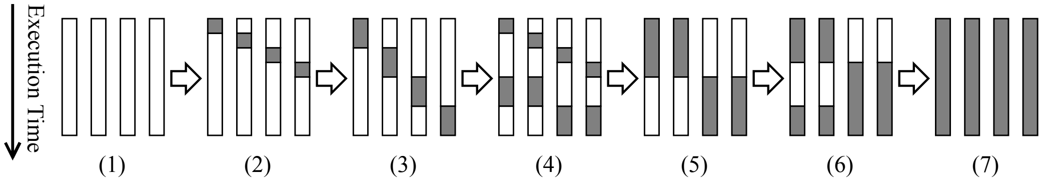
In Fig. 4·6, each group of four bars represents the workload execution of four cores on the same layer. White and gray blocks stand for local and remote memory accesses, respectively, and black ones represent the memory stalls. In the same figure, *Scheduled* refers to the sequence of workload execution planed for each core, while *Actual* shows the real execution. We consider (a) as a baseline case, where simultaneous local memory accesses from four cores are avoided. In this case, all cores behave as scheduled and no core is blocked because of contention. Case (b) shows the situation where the system applies the same workload schedule with (a) but with fewer remote memory accesses. As local shared memory allows for at most three cores to simultaneously access to it, there is a noticeable performance loss once all four cores access the local shared memory. Case (c) demonstrates the system's behavior when the cores have the same amount of remote memory accesses as case (a) but they are



(a). 4-thread-RSM



(b). 2-thread-RSM ⟶ 2+4 thread-RSM



(c). Optimal Schedule:
1-thread-RSM ⟶ 1+2 thread-RSM ⟶ 2+4 thread-RSM

Figure 4·7: Workload schedules for task level resource pooling. (a). Four threads accessing remote shared memory at the same time–*4-thread-RSM*; (b). (1)-(3): Two threads accessing remote shared memory at the same time–*2-thread-RSM*; (c). (1)-(3): One thread accessing remote shared memory–*1-thread-RSM*.

scheduled to access local and remote shared memory at the same time, which causes considerable performance loss compared to case (a). Thus, both workload allocation and scheduling in TLRP affect performance significantly and need to be jointly considered to achieve the best performance.

To optimize performance via memory resource pooling, we should avoid the memory bottleneck as much as possible. Next, we propose an approach for computing the relationship between performance and the number of remote memory accesses. We introduce three workload schedules as shown in Fig. 4·7, where each schedule is applicable to any workload allocation. In Fig. 4·7, remote memory accesses increase gradually from left side to right side. Schedule (a) always makes all four cores access the remote shared memory (*4-thread-RSM*, where RSM stands for remote shared memory). Schedule (b) issues two cores to access remote shared memory at a time (*2-thread-RSM*) from (1) to (3). In (4-7), there are too many remote memory accesses to be scheduled using *2-thread-RSM*, thus we apply mixed *2+4 thread-RSM* until the ratio of remote memory accesses increases to 100%. *1-thread-RSM* has only one thread accessing remote shared memory at a time to minimize simultaneous local memory access, as shown in case (c) from (1) to (3). As the remote memory accesses increase, schedule (c) uses *1-thread-RSM*, *1+2 thread-RSM*, and *2+4 thread-RSM* successively, which minimizes simultaneous local memory accesses.

To compare the performance of these three schedules, we observe the execution time of the whole application, i.e., the longest execution time among all four cores. The execution time on each core is the product of *Instruction Count*, *Cycles per Instruction* and *Cycle Time*. Since most memory-intensive applications contain a large number of memory accesses, in this case we can substitute instructions with memory accesses, which means execution time equals the product of *# of Memory Accesses*, *Cycles per Memory Access* and also *Cycle Time*. To apply TLRP, we need to split memory accesses into local memory accesses and remote memory accesses. In the following equation, T_{exec} stands for the execution time, N_{MemAcc} is the total number of shared memory accesses in the application, W_L and W_R represent the weight (i.e., ratio) of local and remote memory accesses, and C_{Li} and C_{Ri} refer to the number of cycles when i cores are accessing local shared memory and remote shared memory, respectively. C_{Li} and C_{Ri} can be obtained from the shared memory access test. We can calculate the execution time based on the ratio of remote memory accesses using the following equations:

$$T_{exec} = (\sum(W_{Li} \times N_{MemAcc} \times C_{Li}) + \sum(W_{Ri} \times N_{MemAcc} \times C_{Ri})) \times T_{Cycle} \tag{4.1}$$

$$W_L + W_R = \sum W_{Li} + \sum W_{Ri} = 1 \tag{4.2}$$

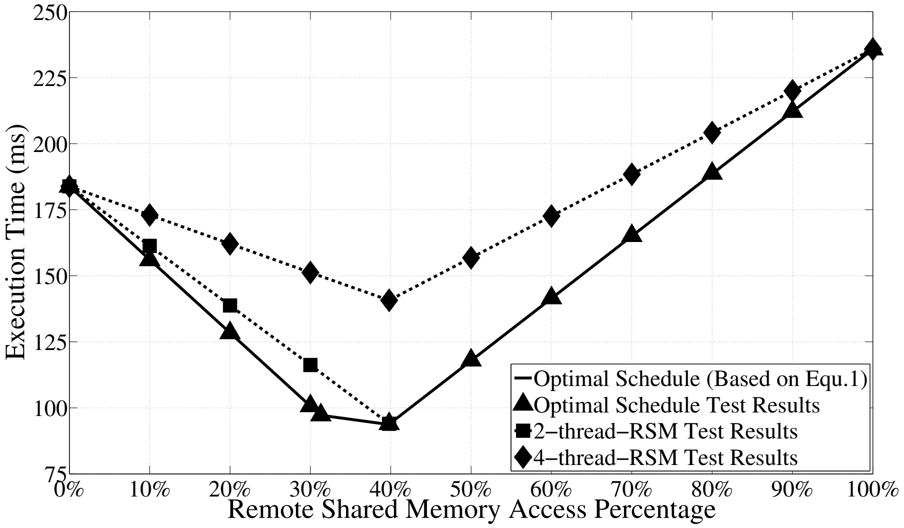


Figure 4·8: Test results of different memory resource pooling schedules and the optimal schedule's curve based on Eqn. (1).

Figure 4·8 shows the test results and fitted curves of the workload schedules in Fig. 4·7. For the optimal schedule, we also draw the theoretical curve based on Eqn. (4.1). The experimental results fit very well with the theoretical curve. Although all of the three schedules can take advantage of resource pooling, the optimal one improves the performance by up to 48.9%, which coincides with the curve for 2-thread-RSM. The optimal schedule shown in the figure demonstrates the potential benefits of memory resource pooling and TLRP workload scheduling.

In Eqn. (4.1), N_{MemAcc} is related to the application, T_{cycle} depends on the architecture, and C_{Li} and C_{Ri} are both application and architecture related. Thus, for most of the shared-memory systems and applications, the proposed approach is applicable for quantifying the potential performance improvement of memory resource pooling.

4.1.3 Performance evaluation

This section demonstrates the performance benefits of 3D-MMC. To evaluate the performance, we design four benchmarks: **1D DCT**, a single dimension 8×8 matrix

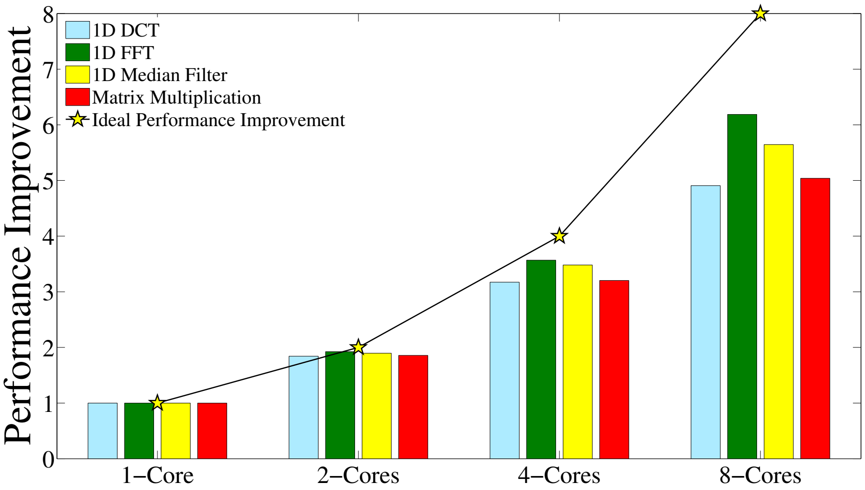


Figure 4·9: Performance improvement compared to single core.

discrete cosine transformation; **1D FFT**, 8 × 8 matrix 12 butterfly Fast Fourier Transformation; **1D Median Filter** with a window size of three and an input array with 64 integers; **Matrix Multiplication**, 8 × 8 multiplication implemented using divide and conquer algorithm.

The test results are shown in Fig. 4·9. Ideal performance improvement refers to the improvement we can get from the multi-core system if the benchmarks can be fully parallelized. For 1-core to 4-core cases, the cores are all on one layer and thus the system can be viewed as a 2D layer only. As for 8-core case, we have two 2D layers with four cores on each, which is the 3D-MMC architecture described above. It can be observed from this figure that the performance improves significantly (61% on average) from 4-core (2D) to 8-core (3D). The difference of improvement among benchmarks is because the benchmarks vary in their scalability. For example, both DCT and FFT have the same input matrix and large amounts of computation, but FFT is more computation-bound compared to DCT. Reading the input can be viewed as the serial part of a benchmark and the computation is the parallel part since it can be done locally in each PE. Thus, the scalability of DCT is lower compared to FFT,

Table 4.2: Execution time of different shared memory access scenarios when all eight cores are active.

Benchmark	All cores access a single shared memory (ns)	All cores access their local shared memory (ns)
1D DCT	16716	16495
1D FFT	26260	26063
Median Filter	8674	7420
Matrix Multiplication	52838	51935

and this difference in turn results in different performance improvements. Assuming a negligible area overhead is imposed by stacking, performance per area is 61% better than a 2D chip on average.

For the 8-core case there are two ways of accessing shared memory for the cores: accessing a single shared memory in the system or each core accessing their local shared memory only. Table 4.2 shows the execution times of these two scenarios. For DCT and FFT, execution times differ by 1.3% and 0.8% only. Matrix Multiplication has 1.7% difference between these two situations because of its slightly higher memory access rate. There is a much larger difference (16.9%) for Median Filter because it is the most memory-intensive benchmark.

Finally, we apply memory resource pooling to the Median Filter benchmark. As this benchmark also needs a lot of private memory accesses, four cores running this benchmark do not stress the shared memory sufficiently to reach the memory bottleneck, limiting the performance improvement to 5%. The available benefit from memory resource pooling is proportional to the memory access rate. The memory access rate becomes higher with a larger number of cores on 2D layer and/or by running more memory-intensive applications. When applying resource pooling to more than 2 layers in a 3D system, the benefits are expected to increase as the cores can utilize a larger number of shared memory blocks across different layers.

4.2 Object-Level Memory Management in Heterogeneous Memory Systems

In the previous section, we have investigated performance improvement of a homogeneous 3D system by pooling on-chip scratchpad memory resources. Main memory often plays an even more important role in system performance and energy efficiency. An outstanding feature of 3D stacking is the ability to integrate different technologies into a single chip by stacking layers with various technologies, such as logic + DRAM. Stacked DRAM can provide shorter access latency and higher bandwidth utilizing TSVs (Loh, 2008). However, stacked DRAM's capacity is limited by chip area and temperature thresholds, which motivates including both on-chip stacked DRAM and off-chip DRAM, forming a heterogeneous memory system. Including multiple types of memory modules in the system, in traditional 2D systems as well as in 3D-stacked systems, allows for higher performance and energy efficiency.

In this section, we study applications' sensitivities to memory characteristics (bandwidth, access latency and power consumption) and explore the performance and energy efficiency benefits of heterogeneous memory systems[1]. In our investigation, we observe that not only applications differ in their memory requirements, but there are memory objects within each application that also exhibit different sensitivities to memory characteristics. We propose a technique to classify memory objects within applications and allocate each memory object to their best-fitting memory module to improve the system performance and energy-efficiency. We present the details on our motivation, technique, and experimental results in the following subsections.

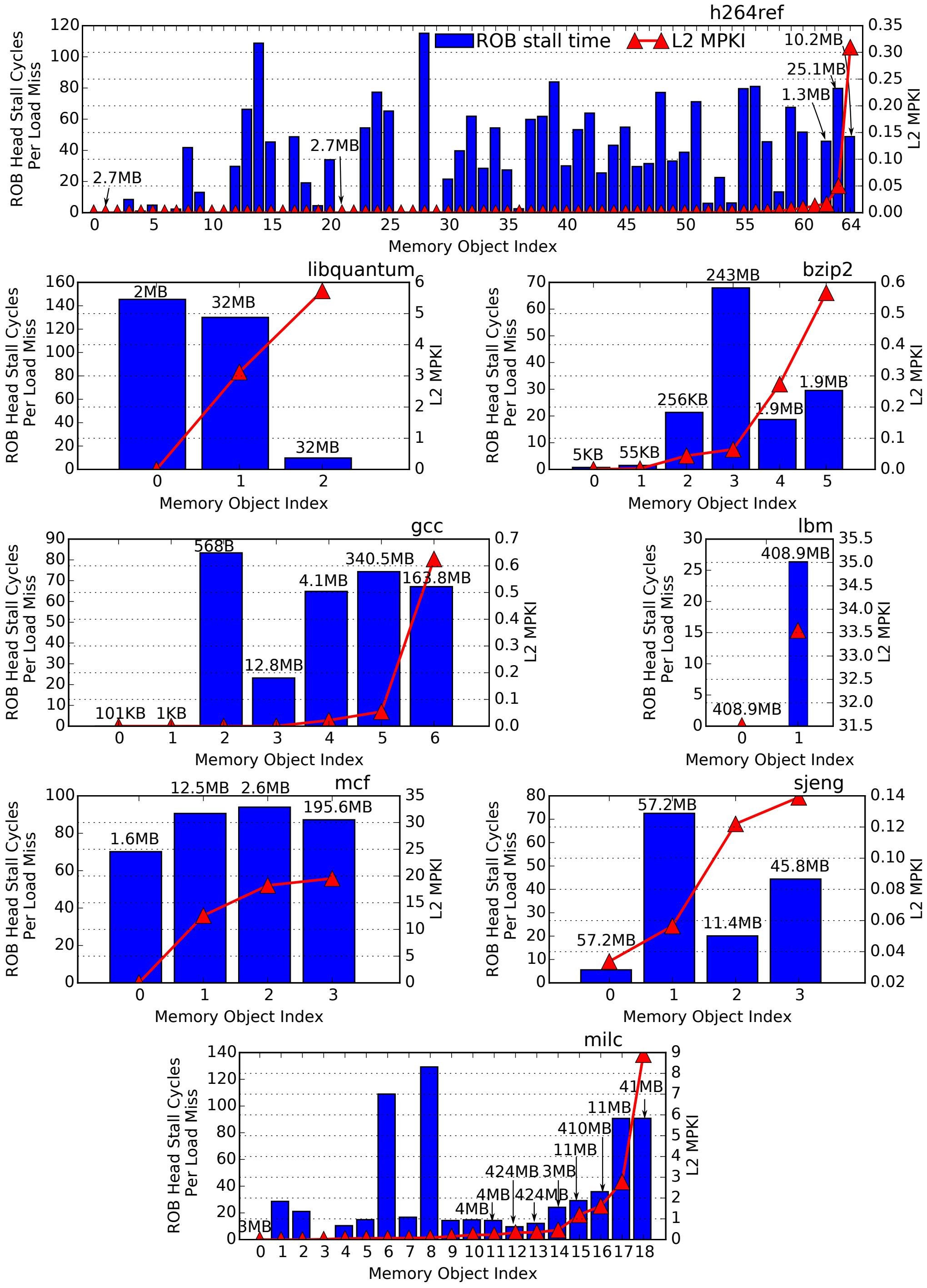


Figure 4·10: Memory access characteristics of memory objects for selected SPEC CPU2006 applications. The x-axis shows different memory objects in the profiled applications, and each object's footprint is denoted on the bar (in MB). For *milc* and *h264ref*, we only mark the objects larger than 1 MB.

4.2.1 Motivation

Memory vendors provide memory modules with various performance and power characteristics, targeting a wide range of system requirements. For example, *Reduced Latency DRAM (RLDRAM)* is a type of memory optimized for low access latency, which makes it ideal for switch and router applications. RLDRAM's bandwidth is smaller and power consumption is significantly higher compared to *DDR3* memory modules. On the other hand, *Low Power DRAM (LPDRAM)* reduces power consumption substantially, but has higher access latency and lower bandwidth; thus, it is attractive for low-power platforms. However, there is no single memory module that can provide the lowest latency, highest bandwidth, and lowest power consumption.

Homogeneous memory systems (i.e., the typical way systems are designed today) employ a single type of memory module. However, applications differ widely in their memory access behavior, making homogeneous memory systems sub-optimal in terms of energy efficiency. A computation-bound workload can achieve similar performance using any type of memory module but the overall energy consumption can be reduced using LPDRAM. On the other hand, a memory-intensive workload achieves substantially better performance using memory modules with low access latency or high bandwidth. A heterogeneous memory system, which contains different memory modules, is able to cater to a wide range of applications and provide higher energy efficiency (Phadke and Narayanasamy, 2011; Chatterjee et al., 2012). Yet, achieving higher energy efficiency is contingent upon placing an application's data in the right memory module.

In addition to application-level differences of memory use, most applications contain a number of memory objects, which are often accessed at different frequencies,

---

[1] Our investigation in this subsection does not specifically focus on 3D-stacked memory, but instead, we design methods for efficient management of heterogeneous memory system broadly, with applications to both 2D and 3D systems.

have different memory footprints, and different LLC miss rates. We claim that it would therefore be beneficial to classify memory objects and conduct *object-level data placement.* Figure 4·10 shows the heterogeneity in memory access characteristics for memory objects for a set of SPEC applications. We plot the LLC misses per kilo-instruction (L2 MPKI, since we have two levels of caches in the tested system) and the Reorder Buffer (ROB) stall time[2] incurred by various memory objects within each application. These metrics together indicate whether a given memory object is latency-sensitive, bandwidth-sensitive, or not sensitive to main memory properties. A high L2 MPKI and a high ROB stall time are indicative of a *latency-sensitive* object that lacks MLP. On the other hand, a high L2 MPKI but low ROB stall time (low stall time indicates good MLP) for an object indicates a *bandwidth-sensitive* memory object. If a memory object is neither latency-sensitive nor bandwidth-sensitive, it can be allocated with pages from a low-power memory module. In Fig. 4·10, we see a wide range across both LLC miss rates and ROB stall times among the memory objects of the applications. In order to tap into this heterogeneity within an application, we design an object-level memory allocation technique, which allocates memory objects within an application based on their memory access behavior, to the best-fitting memory module.

4.2.2 MOCA: Memory Object Classification and Allocation

To leverage the advantages of heterogeneous memory systems for system performance and energy efficiency improvement, we propose an object-level memory allocation technique, MOCA, which profiles the memory objects within an application, classifies them, and allocates pages to these memory objects based on their classification. The flow of MOCA is shown in Fig. 4·11. First, we profile memory objects in each

---

[2] ROB stall time is computed as the average number of cycles spent waiting at the head of ROB for each LLC miss. This metric has been shown as an effective measure of MLP in prior work (Mutlu et al., 2006).

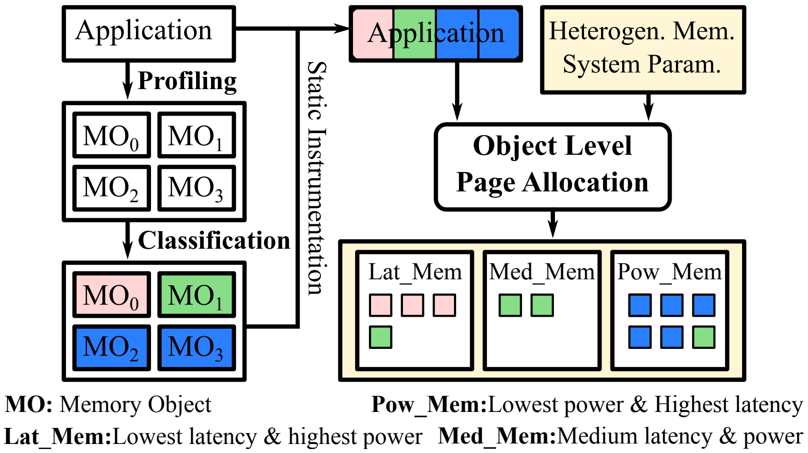
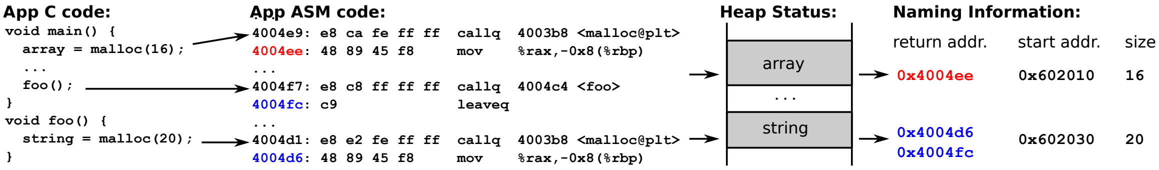

Application
Profiling
MO0
MO1
MO2
MO3
Classification
MO0
MO1
MO2
MO3
Static Instrumentation
Application
Heterogen. Mem. System Param.
Object Level Page Allocation
Lat_Mem
Med_Mem
Pow_Mem
MO: Memory Object
Pow_Mem:Lowest power & Highest latency
Lat_Mem:Lowest latency & highest power
Med_Mem:Medium latency & power

Figure 4·11: The workflow of MOCA. The profiling stage uniquely names objects and profiles their memory access behavior. Classification stage uses this information to classify objects. At runtime, each memory object is allocated pages from its best-fitting memory module based on its classification.



App C code:

```
void main() {
  array = malloc(16);
  ...
  foo();
}
void foo() {
  string = malloc(20);
}
```

App ASM code:

```
4004e9: e8 ca fe ff ff   callq  4003b8 <malloc@plt>
4004ee: 48 89 45 f8      mov    %rax,-0x8(%rbp)
...
4004f7: e8 c8 ff ff ff   callq  4004c4 <foo>
4004fc: c9               leaveq
...
4004d1: e8 e2 fe ff ff   callq  4003b8 <malloc@plt>
4004d6: 48 89 45 f8      mov    %rax,-0x8(%rbp)
```

Naming Information:

return addr.	start addr.	size
0x4004ee	0x602010	16
0x4004d6 0x4004fc	0x602030	20

Figure 4·12: An example of memory object naming convention.

application based on their memory access behavior. Using these profiling results, we classify memory objects based on their sensitivity to memory access latency and memory bandwidth. These two steps are conducted offline. At runtime, MOCA allocates each memory object based on its classification to the appropriate memory module.

Memory Object Profiling

The profiling stage uniquely names each memory object and collects performance statistics for them. We use the return addresses of (1) the memory allocation function (e.g., *malloc()*) and (2) its caller functions in the stack to assign a name to a memory object, as these addresses are unique for each object. We also record the starting virtual memory address and the size of each memory object to identify load operations belonging to a given memory object. Figure 4·12 shows an example of our memory

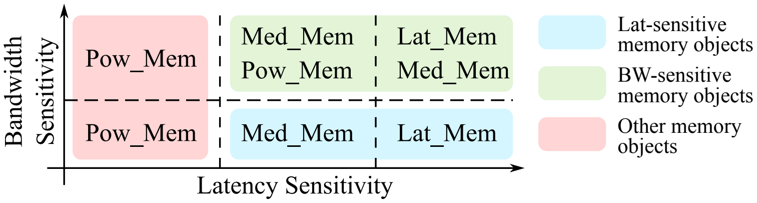


Figure 4·13: Classification of memory objects.

object naming process. We collect two performance statistics for each memory object: (a) average number of stall cycles at the ROB head per LLC read miss (because read misses cause significant delays in application executions) and (b) LLC MPKI, as discussed in Section 4.2.1.

Memory Object Classification

The classification stage uses the output of profiling stage (memory objects, their LLC MPKI, ROB head stall time and footprints) to classify objects as being latency-sensitive or bandwidth-sensitive or neither. Memory objects with high LLC MPKI are generally memory-bound. Among these, those exhibiting low ROB head stall time have high MLP (memory latencies are largely hidden as indicated low ROB stalls), i.e., such objects will benefit from a memory module which can process multiple requests in parallel or a high-bandwidth memory module. We classify such objects as bandwidth-sensitive. The rest of the memory-bound objects with low MLP (high ROB head stall time) are sensitive to access latency of memory modules (i.e., larger latencies translate to larger ROB stall time) and we classify them as latency-sensitive. Objects with low LLC MPKI are neither sensitive to latency or bandwidth. Such objects can be placed in low-power memory modules without affecting performance, saving memory power consumption.

We empirically classify the memory objects within each application based on their latency and bandwidth sensitivity. Figure 4·13 depicts this classification, where Lat_Mem is a *RLDRAM* module, Pow_Mem is a *LPDRAM* module and Med_Mem is a traditional *DDR3* module. We further classify latency-sensitive and bandwidth-sensitive memory objects into two subcategories each based on their latency sensitivity. For example, for bandwidth-sensitive memory objects with higher latency sensitivity, we spread the pages of such memory objects across Lat_Mem and Med_Mem to prevent the degradation caused by Pow_Mem. As for the rest of bandwidth-sensitive memory objects, we use Med_Mem and Pow_Mem only.

Page Allocation for Memory Objects

When running applications in a system with heterogeneous memory, we use the memory object classification (Section 4.2.2) to conduct object-level physical memory page allocation during application execution. We also implement application-level page allocation (Phadke and Narayanasamy, 2011) as a baseline of comparison.

Application-level page allocation

To implement application-level page allocation, we collect cumulative memory access patterns for each application and classify available applications into three categories: latency-sensitive, bandwidth-sensitive, or neither (i.e., can use low-power memory). When allocating physical pages to applications, we select the pages from its best-fitting memory module first. Eventually, if there are not enough pages left in the best-fit module, we first select pages from the bandwidth-optimized memory module and then the low-power module.

Object-level page allocation

In MOCA, to enable object-level page allocation in a heterogeneous memory system, we (1) modify the memory allocation function in *glibc* to separate the heap memory space based on the memory object types and add an extra input argument in this function to specify the memory object type, and (2) divide the physical memory space based on the available memory modules.

Then, when a memory object is instantiated through a modified memory allocation function call (including the extra input of object type), this object is allocated with the virtual memory pages depending on its type. When doing page translation, based on a memory object's virtual page number, the OS can identify its type and then allocate a physical page from the memory module corresponding to its type, as shown in Fig. 4·14. If there is enough space in the requested memory module, the memory object gets the physical pages it needs from its favored memory modules. Otherwise, we first use the pages from Med_Mem and then the ones from Pow_Mem to map a sufficient number of physical pages to the memory object.

4.2.3 Implementation

In a real-life system, MOCA first conducts memory object profiling and classification for a given application using a set of typical (training) inputs (i.e., as in any profiling-based approach). The memory object classification obtained via profiling can then be integrated into the application binaries through application instrumentation. Then, when running an instrumented application, the OS allocates memory based on the object classification information provided by the application and the types of memory modules available in the system.

We implement an experimental framework that mimics this real-life scenario in

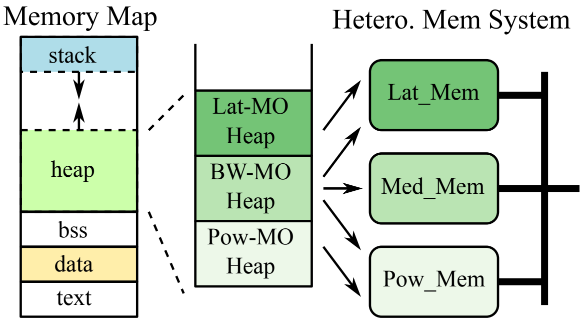


Figure 4·14: Virtual and physical memory space separation for MOCA support in real systems.

an instruction-level performance simulator (Gem5 (Binkert et al., 2011)). The framework includes a memory object profiler, statistics collection, and page allocation. We track memory objects allocated using the memory allocation library of C language (*malloc()*). This framework can be implemented in other simulators and other programming languages as well. Details of our implementation are provided below.

Memory Object Profiler (Offline)

To profile memory objects within an application, we need to give each of them a unique name and enable the performance simulator to keep track of them during application runs. As introduced in Section 4.2.2, we use return addresses of memory allocation function and its caller functions to uniquely name memory objects. To do this, we modify the memory allocation function to get its call stack information and pass the memory object information (including the call stack information) to the simulator. For profiling, we modify the simulator to register memory objects based on the received memory object information, and collect the performance statistics for these memory objects.

For modifying the memory allocation function, we use a built-in function in C language, *__builtin_return_address*(), to get the return addresses of each memory allocation function and its caller functions. The return addresses, with the start address and size of a memory object, form the memory object information. We add pseudo instructions as part of the simulated ISA in the simulator to create a path transmitting the memory object information from the simulated system to the simulator. When the simulator receives memory object information, it registers the incoming memory object and compares it with the existing ones based on the call stack information. If there is no match in the registered memory objects, the simulator instantiates a new counter for this memory object. In case of a match, the simulator relates this memory object with the counter that has the same call stack information.

Table 4.3: Timing and architectural parameters for various memory modules used in this work.

	$DDR3$ (MICRON, 2011)	$RLDDR3$ (MICRON, 2016)	$LPDDR2$ (MICRON, 2013)
Burst length	8	4	2
# of banks	8	16	4
Row buffer size	128 B	16 B	128 B
# of rows	32 K	8 K	8 K
Device width	8	32	16
t_{CK}	1.07	1.25	1.875
t_{RAS}	35	6	42
t_{RCD}	13.75	2	18
t_{RC}	48.75	8	60
t_{RFC}	160	110	130
Standby Power	256 mW/GB	33 mW/GB	5.6 W/GB

Statistics collection (Offline)

For the registered memory objects, we collect ROB stall cycles and LLC MPKI to identify their memory access behavior. For each simulated clock tick, when the ROB stalls (or when there is a LLC miss), we poll all the registered memory objects to see which memory object the requested address belongs to, and then the simulator increments the corresponding counter for that memory object.

Page allocation (Runtime)

When a memory allocation function is called, based on the type of memory object, the function allocates virtual pages in the corresponding heap space for this memory object. Since the OS is aware of the correspondence between virtual memory (heap) and physical memory, it maps corresponding physical pages to the virtual pages according to the heap space they reside in, as shown in Fig. 4·14. We implement the page allocation mechanism inside the simulator by maintaining our own page table.

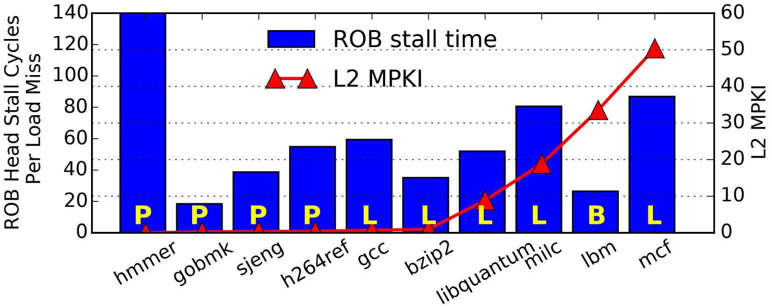


Figure 4·15: Memory access statistics (average ROB head stall cycles per L2 miss and L2 MPKI) of selected SPEC applications. The letter in each bar represents the category of that application. L: latency-sensitive; B: bandwidth-sensitive; P: neither (placed in low-power memory).

4.2.4 Experimental Results

Simulation Methodology

We model a AMD Magny-Cours processor using Gem5 (Binkert et al., 2011) for performance simulations. We use McPAT (Li et al., 2009) to get the core and cache power values, and MICRON data sheets (MICRON, 2011; MICRON, 2013; MICRON, 2016) to calculate memory power consumption. To demonstrate the benefits of the proposed technique we run C-based applications from SPEC CPU2006 benchmark suite with reference input set by fast-forwarding two billion instructions and executing 100 million instructions. For memory object profiling, we generate the simpoints (Hamerly et al., 2005) of applications with training input sets and run applications using these simpoints to collect memory object statistics.

For performance and energy efficiency evaluation, we have three 2 *GB* homogeneous memory systems, each of which is composed of DDR3 memory, LPDDR2 memory or RLDDR3 memory, respectively. To test the application level page allocation and proposed MOCA design, we model a heterogeneous memory system composed of a 768 *MB* DDR3 module, a 256 *MB* RLDDR3 module and a 1 *GB* LPDDR2 module. This heterogeneous memory system consists of four memory chan-

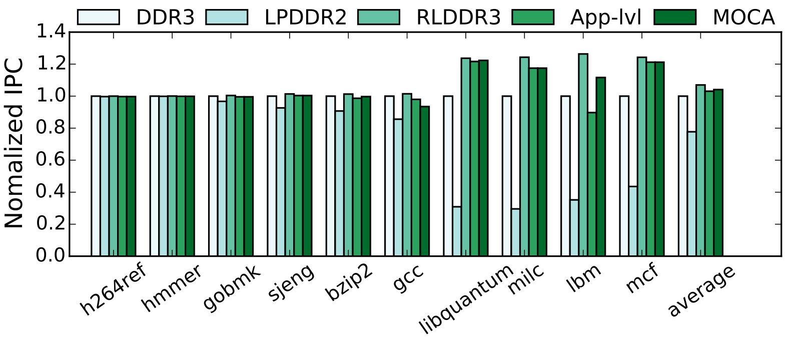


Figure 4·16: Performance of homogeneous memory systems and heterogeneous memory systems with different page allocation algorithms.

nels and each channel is connected to a type of memory module (DDR3, RLDDR3 or LPDDR2). Each memory channel has a continuous memory address range (e.g., 0x0000000-0xFFFFFFF for 1 MB memory channel). We use a dedicated memory controller for each channel as the device timing parameters differ among different memory modules. The architectural, timing and power parameters of the applied memory modules are shown in Table 4.3 (MICRON, 2011; MICRON, 2013; MICRON, 2016). We use the proposed profiler to profile the target applications and classify these applications based on L2 MPKI and ROB stall time. The application-level profiling and classification results are shown in Fig. 4·15. In the following discussion, we denote application-level memory allocation as *App-lvl* and our proposed object-level memory allocation as *MOCA*.

Homogeneous vs. heterogeneous memory system

Figure 4·16 and 4·17 show the normalized performance and normalized energy delay square product (ED^{2}P) of the baseline memory systems and proposed heterogeneous memory systems, respectively. Besides these, we also show the memory ED^{2}P in Fig. 4·18. As shown in the figures, LPDDR2 has the worst performance, but due to its low power cost, it still has the better memory energy efficiency than other systems for low-power applications (e.g., h264ref and gobmk). On the other hand, RLDDR3

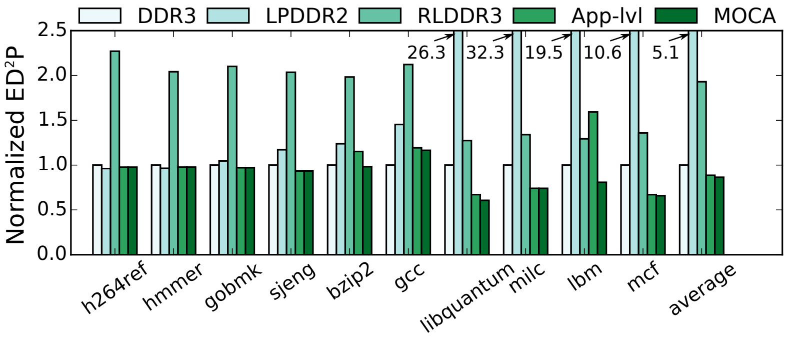


Figure 4·17: System ED^{2}P of homogeneous memory systems and heterogeneous memory systems with different page allocation algorithms.

always performs the best compared to the other systems, however, its high power consumption significantly degrades its energy efficiency. DDR3 has the best memory energy efficiency at a cost of 8.4% performance degradation compared to RLDDR3. As for heterogeneous memory systems (both App-lvl and MOCA) outperform DDR3. They achieve better system ED^{2}P (11.6% for App-lvl and 13.6% for MOCA) and have similar memory ED^{2}P with DDR3 (9.2% for App-lvl and 3.1% for MOCA). Thus, the heterogeneous memory systems can achieve better energy efficiency over homogeneous memory systems with improved performance.

Application-level allocation vs. MOCA

For applications without object diversity (e.g., all objects belong to the same category or an application has single memory object), MOCA has the same performance/energy efficiency as App-lvl. However, when there are various memory objects within an application, MOCA can achieve better performance and energy efficiency. For example, the memory object with highest L2 MPKI in bzip2 has low ROB stall time (as shown in Fig.4·10), thus this memory object is bandwidth-sensitive. The other memory objects within bzip2, however, are neither latency-sensitive nor bandwidth-sensitive, so they should be placed in LPDDR2 modules. As for lbm, App-lvl allocates pages from all memory modules to its memory objects in a round robin

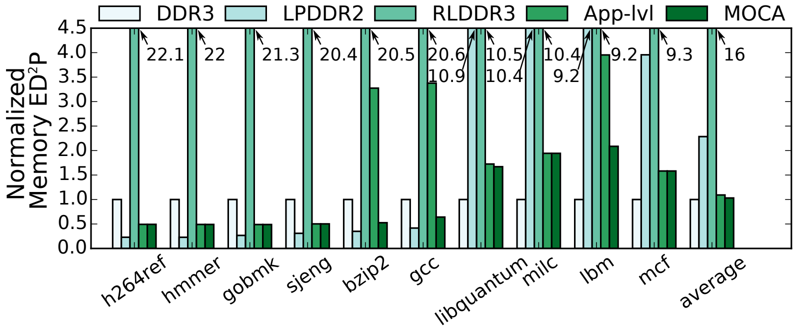


Figure 4·18: Memory ED^{2}P of homogeneous memory systems and heterogeneous memory systems with different page allocation algorithms.

fashion, however, the low-power memory module hurts the general performance due to its low bandwidth and high access latency. By allocating pages only from DDR3 and RLDRAM module, we can improve the performance by 24.4% and reduce system ED^{2}P by 49.3%. Overall, for memory intensive applications, MOCA can improve the performance by 11% and reduce system ED^{2}P by 17.3% on average compared to traditional homogeneous memory system composed of DDR3 modules. For applications with memory object diversity, MOCA outperforms App-lvl by 5.4% in performance and reduce system ED^{2}P by 19%.

4.3 Summary

In this chapter, we have demonstrated the potential of 3D stacking to build a low power multi-core system based on homogeneous stacking through 3D-MMC. We have described an optimal way of exploiting the additional resources offered by the 3D-stacked system through *memory resource pooling*. In addition, this work has shown an analytical approach to evaluate the benefits of resource pooling. On the other hand, we have proposed a memory object level management for heterogeneous memory systems, which shows a greater potential of 3D stacking on improving the system performance and energy efficiency. We have demonstrated that in comparison to a homogeneous memory system with DDR3 modules, for memory-intensive applications, MOCA improves performance by 11% and reduces system ED^{2}P by 17.3% on average. For applications with memory object diversity, MOCA outperforms application-level page allocation by 5.4% in performance and by 19% in system ED^{2}P.

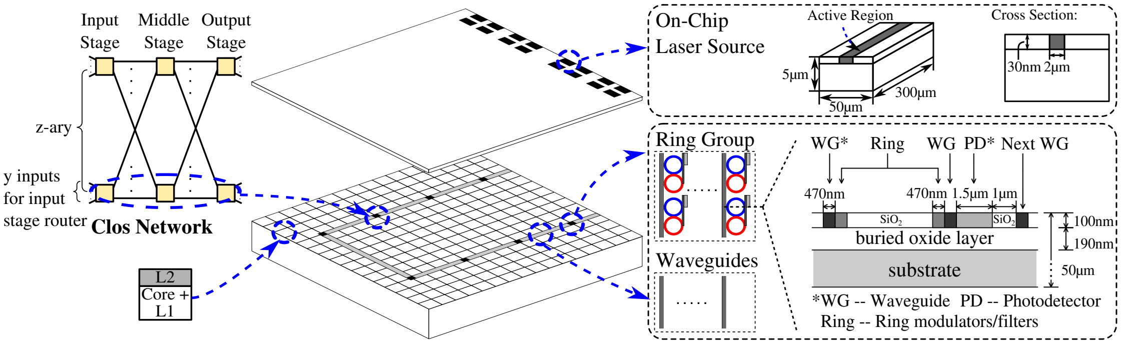
Chapter 5

Thermal Management of 3D Stacked Many-core Systems with PNoC

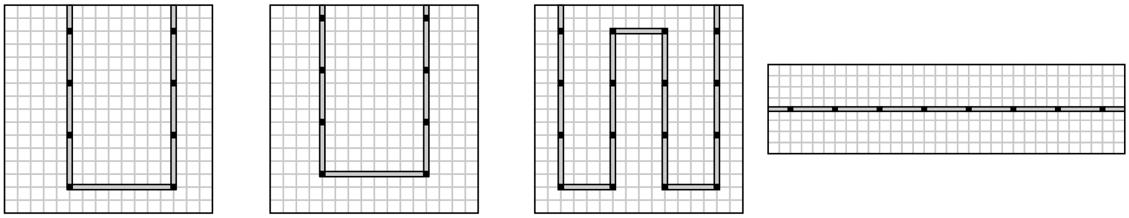
Silicon-photonic links are projected to replace the electrical links for global on-chip communication in future many-core systems, however, the thermal sensitivity of photonic devices limits its wide adoption. Thus, it is in critical need for thermal management for silicon-photonic network-on-chip (PNoC). In this chapter, we present novel runtime and design-time thermal management techniques to guarantee the optical link integrity and improve the power efficiency of many-core systems with PNoC. Our proposed runtime management policy adjusts the temperature of optical devices through workload allocation to match their optical frequencies as much as possible and the remaining difference in optical frequencies is tuned to the same level during runtime using our proposed tuning policy. Our proposed design-time methods reduce the localized thermal tuning power and laser source power consumption through on-chip silicon-photonic devices placement and floorplanning. We start this chapter with a introduction of many-core systems with PNoC, followed by our experimental methodology. Then, we present the proposed runtime and design-time thermal/power management techniques in details.

5.1 Many-core Systems with PNoC

Many-core systems, designed for a high degree of parallel processing, is composed of a large number of simple and independent cores and an on-chip communication network.



(a) 256-core system with U-shape layout of PNoC.



(b) 256-core system with horizontally shifted ring groups.

(c) 256-core system with vertically shifted ring groups.

(d) 256-core system with W-shape physical layout.

(e) 256-core system with 1:4 chip aspect ratio and chain-shape physical layout.

Figure 5·1: Target many-core system with a PNoC (a), many-core systems with 8-ary 3-stage Clos topology and shifted physical layouts (b)-(c), and many-core systems with different logical topology and physical layout combinations (d)-(e). (d) is designed with 16-ary 3-stage Clos topology and W-shape physical layout; (e) is designed with 8-ary 3-stage Clos topology and chain-shape physical layout.

To design a many-core system with PNoC, we must consider the requirements and constraints for both electronic components and silicon-photonic components. In this section, we introduce the architecture of our target many-core system and a general PNoC design flow. Table 5.1 shows the notations used in this chapter.

5.1.1 Many-core System Architecture

In our work, we use a 256-core system designed using a typical 22 nm SOI CMOS process, operating at 1 GHz with 0.9 V supply voltage. For each core, we use an architecture similar to the IA-32 core from Intel SCC (Howard et al., 2011). Every core consists of a 16 KB I/D L1 cache and a 256 KB private L2 cache. We scale the core architecture to 22 nm, resulting in a single core area of 0.93 mm^2 (including the

Table 5.1: Notations used in this work.

Variable	**Definition**	**Unit**
i	Ring group index	
j	Core index	
k	Thread index	
l	Laser source index	
s	Simulation step index	
g	Material index	
R	Thermal resistivity	$m \cdot K/W$
R_{joint}	Thermal resistivity of the NoC block	$m \cdot K/W$
$\Delta f_R/\Delta f_{LS}$	Thermal sensitivity of ring resonators / laser sources in frequency domain	GHz/K
$\Delta \lambda_R/\Delta \lambda_{LS}$	Thermal sensitivity of ring resonators / laser sources in wavelength domain	pm/K
η_R/η_{LS}	Thermal tuning efficiency of ring resonators / laser sources	W/K
n_g	Refractive index of the ring resonator material	
r	Ring resonator radius	μm
V	Volume	m^3
M	Total number of ring groups	
H	Total number of rings in a ring group	
N	Total number of cores	
S	Total number of threads	
Q	Total number of laser sources	
n_λ	Number of wavelengths per waveguide	
x	Number of middle stage routers	
y	Number of I/O ports on first or last stage routers	
z	Number of first or last stage routers	
w	Weight factor	K/W
w_{ij}	Weight factor for ring group i and core j for temperature impact	K/W
Δw	Weight factor difference	K/W
t	Time	ms
T	Temperature	^oC
T_{RG_i}	Temperature of ring group i	^oC
T_{LS_l}	Temperature of laser source l	^oC
P_j	Power of core j	W
P_{FT}	Optical frequency tuning power	W
P_{leak}	Leakage power	W
λ	Wavelength	nm
F	Frequency	GHz
F_{RG_i}	Frequency of ring group i	GHz
F_{LS_l}	Frequency of laser source l	GHz

L1 cache), and an L2 cache area of 0.35 mm^2. Our total chip area[1] is 326.5 mm^2. The average power consumption for each core is 1.17 W. The system is organized into 64 equal tiles. In each tile, four cores are connected via an electrical router. There are 16 memory controllers that are uniformly distributed along two edges of the chip. We use an 8-ary 3-stage Clos network topology to connect the L2 caches and memory controllers. Our Clos can be described by the triplet (x=8, y=10, z=8), where x is the number of middle stage routers, y is the number of I/O ports on the first or last stage routers, and z is the number of first or last stage routers. Therefore, the 8-ary 3-stage Clos PNoC has 128 channels in total.

We map the 8-ary 3-stage Clos topology to a U-shaped physical layout of silicon-photonic waveguides as shown in Fig. 5·1(a), where each ring group is assigned to the nearest eight tiles and two memory controllers. We apply the silicon-photonic link technology described in prior work (Orcutt et al., 2012; Moss et al., 2013; Georgas et al., 2014), where optical devices are monolithically integrated with CMOS devices. In this system, single crystal Si is utilized for waveguides and ring resonators, and Ge on Si is utilized for photodetectors. Ring resonators are designed in Si by ion implantation and are tuned with metal heaters. We combine the ring modulators and filters from one electrical router of each of the three network stages into a *ring group (RG)*. The optical waves from laser sources arrive at a ring group and are modulated. The modulated optical waves traverse the network and are filtered by the ring filters in the destination ring group, where a photodetector converts the optical signal into an electrical current that is fed to the link receiver circuit. We assume on-chip laser sources, which simplify packaging, reduce cost and improve laser source control. Several approaches have been proposed for realizing on-chip laser sources (Song et al., 2015; Wang et al., 2011; Lourdudoss, 2012). Specifically,

---

[1] There are commercial products with similar die size and power consumption, e.g., SPARC T4 processor (Oracle, 2011).

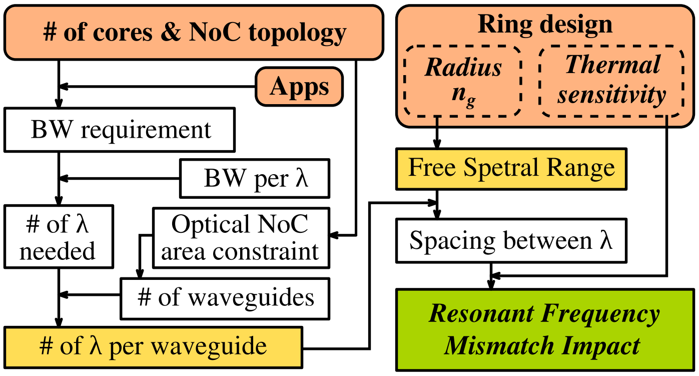


Figure 5·2: PNoC design flow chart.

we assume heterogeneous integration to incorporate laser sources above the logic and silicon-photonic devices. Such a monolithic approach can be cost effective because it does not require separate fabrication of laser sources and would avoid chip attachment steps that require precise alignment.

Alternative core and cache architectures may require different logical topology and physical layout combinations, so we also present system designs with other layouts and ring group locations in Fig. 5·1(b) to 5·1(e). Figures 5·1(b) and 5·1(c) show two layouts that use the same logical topology but have horizontal and vertical shifts, respectively, in ring group locations. Figure 5·1(d) presents a system with 16-ary 3-stage logical topology and a W-shaped physical layout. Figure 5·1(e) shows a rectangular chip with 8-ary 3-stage logical topology and chain-shaped physical layout.

5.1.2 PNoC Design Flow

Designing a PNoC for a many-core system has many factors to consider, e.g., bandwidth requirement and area constraints of the target system, data rate of the optical waves as well as the design of ring resonators. To investigate the design space of a PNoC, we adopt a cross-layer approach where we jointly consider the photonic device design and NoC architecture design. Figure 5·2 shows the design flow adopted for jointly choosing the ring dimensions, the number of wavelengths per waveguide, and the number of waveguides for a given thermal gradient and area constraint. We consider area overhead as a constraint in the design flow because monolithic integration increases die area, resulting in increased manufacturing cost.

The bandwidth requirement of a PNoC depends on targeted applications in a many-core system. In our work, we simulate selected SPLASH-2 (Woo et al., 1995), PARSEC (Bienia et al., 2008) and UHPC (Campbell et al., 2012) applications on our many-core system and determine the peak NoC bandwidth (BW) requirement to be 512 GB/s, which corresponds to 64 $bits/cycle$ for each photonic channel in our 8-ary 3-stage Clos network. A monolithically integrated silicon-photonic link with 2.5 $Gbps/\lambda$ bandwidth has been demonstrated in prior work (Moss et al., 2013). In this work, we assume a bandwidth of 4 $Gbps/\lambda$. This is reasonable considering the performance of current silicon-photonic devices that operate beyond 25 $Gbps$ (Baehr-Jones et al., 2012). The link bandwidth and the required bandwidth of the applications define the total number of wavelengths needed in the PNoC. We constrain PNoC area to be at most 10% of the total die area. This constraint puts an upper limit on the number of waveguides in the system and thus a lower limit on the number of wavelengths that need to be mapped to a waveguide. We ignore the non-linearity limit on the power that can be injected into a waveguide (Joshi et al., 2009). As for the ring resonator design, we choose 10 μm as the ring radius.

Within each ring group in a PNoC, there are ring resonators with varying frequencies belonging to different silicon-photonic links. Each silicon-photonic link is multiplexed with other links on a waveguide and has one ring modulator (on the transmitter side) in one ring group and a ring filter (on the receiver side) with the same resonant frequency in another ring group. For the sake of brevity, we refer to "resonant frequencies of ring resonators within a ring group" as "the resonant frequency of a ring group". We use "resonant frequency difference between two ring

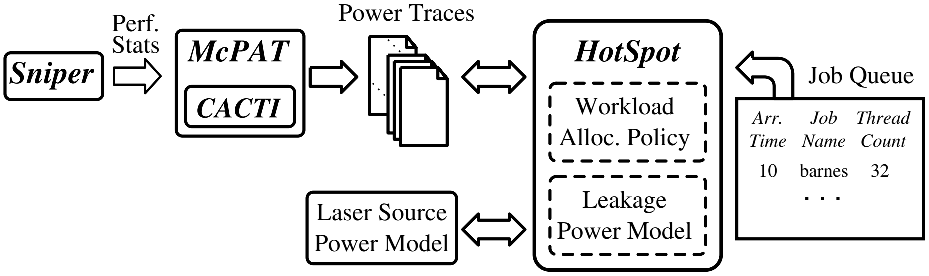


Figure 5·3: Our performance (Carlson et al., 2011), power (Thoziyoor et al., 2008; Li et al., 2009) and thermal (Skadron et al., 2003) simulation setup for modeling many-core systems with a PNoC.

groups" to represent "resonant frequency difference between a ring modulator in one ring group, and the corresponding ring filter in the other ring group".

For systems without process variations, a corresponding resonant frequency difference between two ring groups can be computed using the temperature gradient (ΔT) between them: $\Delta F = \Delta T \times \Delta f_R$. Due to manufacturing process variations, there are variations in the dimensions of the waveguides across a chip (Chen et al., 2013). Since the resonant frequency is very sensitive to these dimensions, there is an initial gradient in frequency across ring groups. Thus, temperature alone cannot accurately indicate the frequency difference among ring groups.

5.2 Experimental Methodology

To investigate thermal variations' impact on optical frequencies of ring resonators and laser sources at runtime when running realistic workloads on a many-core system with PNoC, we set up a simulation infrastructure composed of performance, power and thermal simulators, as shown in Fig. 5·3. We use Sniper (Carlson et al., 2011) to simulate performance. Sniper comes interfaced with McPAT (Li et al., 2009) to estimate the power consumption of the simulated system. The power traces generated by McPAT are given as inputs to the HotSpot 3D Extension (Skadron et al., 2003; Meng et al., 2012) for transient thermal simulations.

5.2.1 Performance and Power Simulation

For performance simulations, we simulate the region of interest of a representative set of multi-threaded applications from the SPLASH-2 (Woo et al., 1995) (barnes, lu_cont and water_nsq), PARSEC (Bienia et al., 2008) (blackscholes and canneal), and UHPC (Campbell et al., 2012) (md and shock) benchmark suites. To investigate the impact of core thermal variations on the photonic devices under varying system utilizations, we run each application on a target many-core system (explained in Section 5.1.1) with 32, 64, 96 and 128 threads.

We use the performance statistics from Sniper as input to McPAT to calculate power for cores and caches. After generating all power traces, we use published power dissipation data from Intel Single-Chip Cloud Computer (SCC) (Howard et al., 2011), scaled to 22 nm, to calibrate our dynamic power data. HotSpot takes these power traces as inputs, and outputs corresponding temperature traces. We assume that idle cores are put into sleep states and consume 0 W. We also assume that 35% of the average core power (1.17 W) at 70 ^oC comes from leakage (Meng et al., 2012). We calculate the average core power consumption in one core for each application and categorize the applications as shown in Table 5.2. We compose and evaluate different workload combinations based on this categorization in Section 5.5.2.

To simulate thermal behavior of the cores more accurately, we implement a linear leakage power model inside HotSpot. This model is suitable due to the relatively limited range in the operating temperature on our target system (Su et al., 2003). We use published data for Intel 22 nm commercial processors (Wong, 2012) to extract this linear leakage power model as shown in Equation (5.1). In this equation, $T(t_{s-1})$ is the temperature in ^oC at time t_{s-1} and $P_{leak}(t_s)$ is the leakage power in W at time t_s, where s is the thermal simulation step index and t_s is the time at which the leakage power is recalculated. c_1 and c_2 are constant coefficients with values 1.4e-3 and 0.31,

Table 5.2: Classification of applications.

High Power (HP) Apps	md (2.15 W), shock (1.7 W)
Medium Power (MP) Apps	blackscholes (1.46 W), barnes (1.3 W)
Low Power (LP) Apps	canneal (0.9 W), water_nsq (0.7 W), lu_cont (0.75 W)

respectively. During thermal simulations, we update the leakage power for every core based on its temperature at t_{s-1}.

$$P_{leak}(t_s) = c_1 \times T(t_{s-1}) + c_2 \tag{5.1}$$

A novel part of our simulation infrastructure is modeling the laser source power consumption at runtime as a function of temperature, and including this model in our HotSpot transient thermal simulations. Previous work (Li et al., 2015a) implemented a temperature-dependent laser source power model for steady state thermal simulations. We put together a similar framework that works with both steady state simulations and transient simulations. In our framework, we generate a lookup table for laser power by employing the theory described in prior work (Coldren et al., 2012). The laser source power that contributes to heat dissipation is the difference between the required input electrical power and the required output optical power. The required input electrical power depends on the required output optical power and the laser source efficiency. The required output optical power is determined by the optical loss during optical wave transmission in the PNoC, and thus is fixed for a given PNoC design. The laser source efficiency is based on the required output optical power and laser source temperature. Thus, the lookup table takes required output optical power and laser source temperature as inputs, and computes the required input electrical power based on the corresponding laser source efficiency. During transient thermal simulations, we update the laser source power at the beginning of each simulation step for each laser source based on its temperature.

5.2.2 Thermal Simulation

We perform transient thermal simulations to evaluate our proposed dynamic workload allocation policies. To do this, we enable HotSpot to read the upcoming jobs from a job queue, in which each job entry has an arrival time, an application name, and a required thread count. We also integrate a workload allocation module in HotSpot. When a job arrives, this module allocates the threads to cores. HotSpot assigns a power value for each core at each simulation step based on the specific thread it runs (assigned from a power trace database generated via Sniper-McPAT). Thread migration can be applied to this framework as needed.

In our HotSpot setup, we use the default configuration with 35 ^oC ambient temperature, and the properties of the materials shown in Table 5.3. The floorplans of the target systems are shown in Fig. 5·1. For our system, we assume monolithic integration of waveguides, ring resonators and photodetectors on the logic layer (Orcutt et al., 2012), while laser sources are on a separate layer. On the laser source layer, the laser sources are placed along the upper chip edge, arranged in two groups surrounding the waveguides in a matrix fashion, as shown in Fig. 5·1(a).

The number of laser sources depends on the design choice of laser source type, sharing degree and required network bandwidth. Sharing a laser source among multiple waveguides has been shown to improve laser source efficiency and reduce total on-chip power (Chen et al., 2014). We choose 32 waveguides for our PNoC design to allow for laser source sharing between waveguides while remaining within the 10% area overhead maximum given for the photonic devices in our system.

We aggregate waveguides, ring resonators and photodetectors into larger simulation blocks in our floorplan in HotSpot. We calculate the joint thermal resistivity for each PNoC block based on the percentage of each material's volume and thermal resistivity of each material using $R_{joint} = V_{total}/\Sigma(V_g/R_g)$, where V_{total} represents the

Table 5.3: Properties of the materials in our target system.

	Thickness (mm)	Side (mm)
Heat Sink	6.9	80
Spreader	1	40
Interface Material	0.02	
Laser Source Layer	0.005	
Core Layer	0.05	
	Thermal Conductivity $(W \cdot m^{-1} \cdot K^{-1})$	Specific Heat $(J \cdot g^{-1} \cdot K^{-1})$
Spreader	400	
Interface Material	4	
Si	100	0.71
InP	68	0.31
	Photonic Device Dimensions	Material
Laser Source Size	300 μm × 50 μm	InP
Ring Radius	10 μm	Si

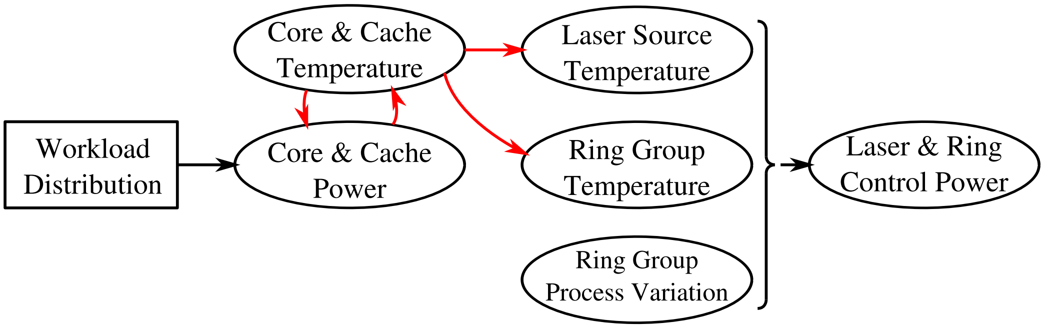


Figure 5·4: Thermal dependence between workload distribution and optical device frequency control power.

total volume of a PNoC block, R_g refers to the thermal resistivity of material g and V_g indicates the volume of material g in this PNoC block. R_{joint} of the ring blocks is 1.006e-2 $m \cdot K/W$, while R_{joint} for the waveguide blocks is 1.004e-2 $m \cdot K/W$ (both are almost identical to the thermal resistivity of Si).

Transient thermal simulations are initialized with a steady state simulation. As the temperature-dependent leakage model changes the power traces, we run each transient thermal simulation for another round to ensure convergence of temperature.

5.3 Runtime Thermal Management Through Workload Allocation

In a PNoC, WID process and thermal variations cause optical frequency difference among ring groups and laser sources, which needs to be compensated through localized thermal tuning technique. Process variations depend on the quality of the manufacturing process while the temperature variations are highly dependent on the workload distribution in the many-core system. The thermal dependence between workload distribution and optical device frequency control power is shown in Fig. 5·4. Our target is to change the chip thermal map through workload allocation to reduce the resonant frequency difference among all the ring groups. On top of this, we propose an adaptive frequency tuning method to match the remaining optical frequency difference between laser sources and ring groups.

We describe our proposed ring-location-aware workload allocation policy, *RingAware* (Zhang et al., 2014), and its improved version, *FreqAlign* (Abellan et al., 2016), in Section 5.3.1. In Section 5.3.2, we introduce a baseline frequency tuning policy and present a novel adaptive frequency tuning policy for many-core systems with on-chip laser sources. We discuss the performance overhead of *FreqAlign* in Section 5.3.2.

5.3.1 Workload Allocation Policies

RingAware

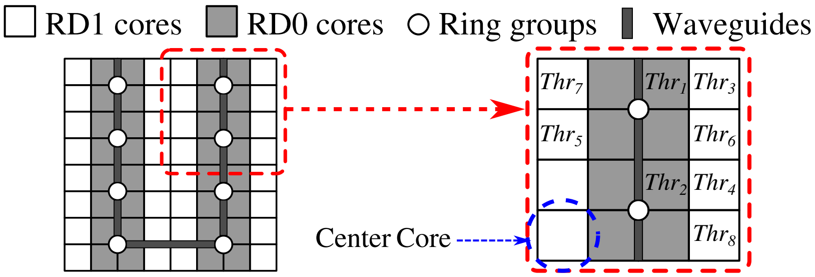
The *RingAware* workload allocation policy balances the ring group temperatures by maintaining similar power profiles around each ring group. For a given layout, this policy categorizes cores based on the distance of a core from its closest ring group. We use *RD#* notation for each region, where # represents the cores' relative distance to the ring group, as shown in Fig. 5·5. Since *RD*0 cores have the highest impact on a ring group's temperature, *RingAware* maintains similar power profile across the

$RD0$ regions for all ring groups to minimize their temperature gradients.

We use single-threaded cores and each workload is composed of S threads. For an N-core system with M ring groups, if there are S threads to allocate, we first compare S with the total number of *non-RD0* cores. If S is larger, the *RD0* cores need to be

```
Identify RD0 cores → RD0 core list;
Partition the system into 4 quadrants;
Sort all threads based on their power consumption;
if $S > N - \#RD0Cores$ then
    foreach ring group in the system do
        assign $\lceil \frac{S-(N-\#RD0Cores)}{M} \rceil$ threads;
    end
    Each quadrant ← $\frac{S - \lceil \frac{S-(N-\#RD0Cores)}{M} \rceil * M}{4}$ threads;
else
    Each quadrant ← $\frac{S}{4}$ threads;
    foreach quadrant in the system do
        foreach thread left in queue do
            $allocatedCore \leftarrow 0$;
            $nextThread \leftarrow 0$;
            foreach alternative core j on boundary do
                if core j is idle & core j ∉ RD0 core list then
                    $allocatedCore \leftarrow j$;
                    $nextThread \leftarrow 1$;
                    break;
                end
            end
            if $nextThread == 0$ then
                foreach alternative core j in inner area do
                    if core j is idle & core j ∉ RD0 core list then
                        $allocatedCore \leftarrow j$;
                        $nextThread \leftarrow 1$;
                        break;
                    end
                end
            end
        end
    end
end
```

Algorithm 1: Pseudocode for *RingAware* (Zhang et al., 2014) policy.



(a) (b)

Figure 5·5: (a) Classification of *RD*0 cores and (b) an example of *RingAware* allocation in a 64-core system.

utilized to run all the threads and we assign $\lceil \frac{S-(N-\#RD0Cores)}{M} \rceil$ threads to each *RD0* region. The *RD0* regions of all ring groups need to have the same active core count to minimize the ring group temperature gradient. Then, we partition the system into four quadrants and then assign the rest of the threads evenly in each quadrant. The residual threads, if any, are allocated to the quadrants in a round-robin fashion. For each quadrant, *RingAware* activates *non-RD0* cores alternately from the outer boundaries to the inner part of the chip (i.e., to reduce chip temperature) until all threads are allocated, starting from the corner core, as shown in Algorithm 1. If there are power variations among threads, we rank the threads according to their power consumptions at the beginning and start the allocation process with the high-power threads in the order (Thr_1 to Thr_8) shown in Fig. 5·5(b).

RingAware allocation effectively reduces the ring group temperature gradient, which results in a low resonant frequency gradient when the system does not have process variations. For systems with process variations, only balancing the temperatures of ring groups is not sufficient to reduce the resonant frequency difference among ring groups. Also, when the ring groups are not symmetrically placed on the chip, *RingAware* starts to require larger thermal tuning power. Hence, we now propose an improved policy that jointly accounts for thermal variations and process variations. This policy works even for asymmetric placement of ring groups.

FreqAlign

The goal of *FreqAlign* workload allocation policy is to reduce the resonant frequency difference among ring groups. To do this, we estimate the ring group resonant frequency for every potential workload allocation decision by estimating the ring group temperatures. In a system that has M ring groups and N cores, for each ring group, we use an $M \times N$ weight matrix of w_{ij} that contains the steady state temperature impact per unit of power of core j on ring group i. This weight matrix is used to estimate the temperature of ring groups. For example, if core j has a weight factor of 0.5 for ring group i, it means at steady state, ring group i's temperature increases by 0.5 K when core j consumes 1 W. This weight matrix can be obtained using HotSpot for a given physical layout.

When calculating the resonant frequency shifts of ring resonators in ring group i due to thermal variations, we use Eqn. (5.2) and (5.3), where ${F_{RG_i}}^{post}$ and ${T_{RG_i}}^{post}$ are the resonant frequency and temperature, respectively, of ring group i after an updated workload allocation. ${F_{RG_i}}^{pre}$ and ${T_{RG_i}}^{pre}$ are the resonant frequency and temperature of ring group i before this updated workload allocation. Δf_R is 9.7 GHz/K, P_j is the power of core j, and w_{ij} is the weight factor of core j to ring group i.

$${F_{RG_i}}^{post} = {F_{RG_i}}^{pre} - \Delta f_R \times ({T_{RG_i}}^{post} - {T_{RG_i}}^{pre}) \tag{5.2}$$

$${T_{RG_i}}^{post} = \sum_{j=1}^{N} w_{ij} \times P_j \tag{5.3}$$

Algorithm 2 shows the pseudocode of *FreqAlign*. Here, we define a job as an application with a number of threads to be allocated (our target system has single-threaded cores, so we can only assign one thread per core), and we put the threads into a queue and allocate them to the available cores in the many-core system. The objective function of the optimization is to minimize the sum of the absolute differences in resonant

```
Sort all threads based on their power consumption;
foreach thread in queue do
    $\Delta w_{min} \leftarrow -1$;
    $allocatedCore \leftarrow 0$;
    foreach available core j in many-core system do
        foreach ring group i in many-core system do
            $w_{est} \leftarrow w_{curr} +$ core j impact on RG i;
        end
        $\Delta w_{est} \leftarrow \max(w_{est}) - \min(w_{est})$;
        if $\Delta w_{est} < \Delta w_{min}$ or $\Delta w_{min} == -1$ then   $\Delta w_{min} \leftarrow \Delta w_{est}$;
        $allocatedCore \leftarrow j$;
        else continue;
    end
    allocatedCore in coreArray ← active;
    $w_{curr} \leftarrow w_{curr} +$ core(allocatedCore) impact on RGs;
end
```

Algorithm 2: Pseudocode for *FreqAlign* Policy.

frequencies of all the ring groups ($\sum_{i=1}^{M-1} \sum_{i'=i+1}^{M} |F_{RG_i} - F_{RG_{i'}}|$).

Every ring group has a designed resonant frequency value F_i^0. Due to process variations, this value varies depending on the ring group location. The variations in the resonant frequency could be diagnosed after the chip is manufactured. We track the resonant frequencies using an array (w_{curr} in Algorithm 2) for the ring groups during the system operation. This array contains the initial values of w which depend on the resonant frequency shift of each ring group caused by its process variations. For example, an initial array of $[5, 0, 0, 0, 0, 0, 0, -5]$, means that RG_1 has a resonant frequency $5\ K \times 9.7\ GHz/K = 48.5\ GHz$ lower than the designed frequency while RG_8 has a resonant frequency $48.5\ GHz$ higher than the designed frequency. Every time a core is activated, we update this array based on the core's impact on the ring groups. Our target in workload allocation is to equalize the values in this array.

During the system operation, when an application with S threads arrives, we rank the threads based on their power consumption (which can be estimated through previous runs or performance counters history) and assign them to the cores while balancing the resonant frequency of the ring groups. After all S threads are allocated to the corresponding cores, the system starts to run. As Algorithm 2 shows, when assigning the threads, we go through all the available cores in the system. For each available core, we calculate the expected resonant frequency difference among all ring groups if a thread is assigned to that core. For each thread, we select the core that leads to the smallest resonant frequency difference among all ring groups (Δw_{min}). After assigning a thread, we update the estimated resonant frequency values for all ring groups. We iterate this process until all threads are assigned. If there are jobs currently running on the many-core system and a new job arrives, we rank them together with new threads based on their power consumptions and reallocate all workloads. The potential workload migration when redoing the allocation induces context switch overhead and cache cold start effect to the system. *FreqAlign* can be integrated with the operating system scheduler and run on any available core in the system. We discuss the performance overhead of *FreqAlign* in Section 5.3.2.

5.3.2 Frequency Tuning Methods

Baseline Frequency Tuning

Workload allocation can help decrease the resonant frequency difference among the ring groups. We use localized tuning to compensate for the remaining resonant frequency difference as well as the optical frequency difference between ring groups and laser sources. Resonant frequencies of ring resonators can be controlled through thermal tuning devices such as micro-heaters. As for on-chip laser sources, their optical frequencies can be controlled in a number of ways, depending on the laser source type. For example, multi-section distributed Bragg reflector laser sources comprise of wavelength tuning control elements such as mirrors and a phase section. The wavelengths of distributed feedback lasers, which we use in this work, are controlled by injecting current. More advanced laser sources on silicon-photonic platforms may comprise of extra ring filters within the laser cavity for tuning purposes.

Our baseline frequency tuning method is *Target Frequency Tuning (TFT).* In this tuning method, at any given time during system operation, all ring groups and laser sources are first tuned to their optical frequencies at the temperature threshold of the target many-core system (90 ^oC in our case), and then are individually tuned further to compensate for process variations to match their optical frequencies. We also assume that all the ring resonators within a ring group share the same temperature. Since the material used for the laser sources and ring resonators have different thermo-optic coefficients, their respective tuning efficiencies (8 mW/nm (Larson et al., 2015) for the laser sources and 2.6 mW/nm (Orcutt et al., 2012) for the ring resonators) also differ. The temperature sensitivity values for laser sources and ring resonators are 12.5 GHz/K (Kimoto et al., 2003) and 9.7 GHz/K (Dong et al., 2010a), respectively. For a fixed target optical frequency, the amount of frequency tuning power (P_{FT}) required is shown in Equation (5.4), where F_{LS_l} is the frequency of laser source l, $F_{target_{i/l}}$ is the desired target optical frequency of a photonic device i/l at the target temperature, Δf_{LS} is the thermal sensitivity of the laser source, η_{LS} is the tuning efficiency of the laser sources, F_{RG_i} is the frequency of ring group i, Δf_R is the thermal sensitivity of the ring resonators, η_R is the tuning efficiency of a single ring resonator, Q is the total number of laser sources, M is the total number of ring groups, and H is the amount of ring resonators in a ring group.

$$\begin{aligned} P_{FT} = \sum_{l=1}^{Q} \frac{|F_{LS_l} - F_{target_l}|}{\Delta f_{LS}} \times \eta_{LS} + \\ \sum_{i=1}^{M} \frac{F_{RG_i} - F_{target_i}}{\Delta f_R} \times \eta_R \times H \end{aligned} \tag{5.4}$$

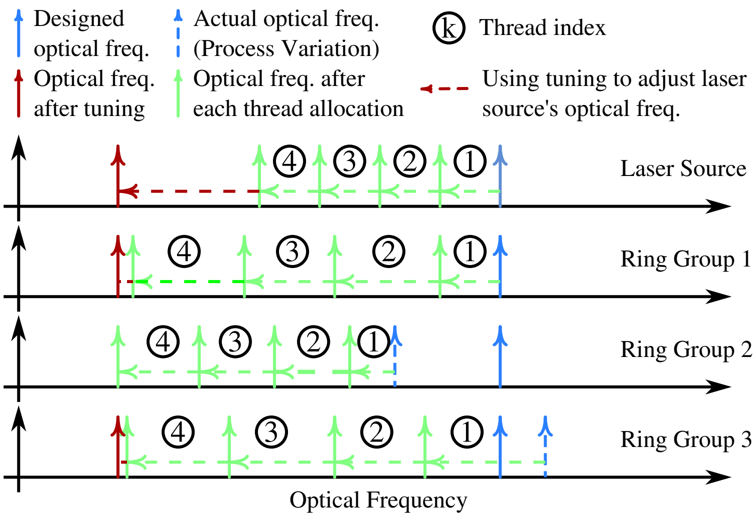

Designed optical freq.
Actual optical freq. (Process Variation)
ⓚ Thread index
Optical freq. after tuning
Optical freq. after each thread allocation
Using tuning to adjust laser source's optical freq.
④ ③ ② ①
Laser Source
Ring Group 1
Ring Group 2
Ring Group 3
Optical Frequency

Figure 5·6: Illustration of *FreqAlign* workload allocation policy and adaptive frequency tuning *AFT* policy. Every thread allocated by *FreqAlign* increases the temperatures of ring groups and causes a downward shift in their frequencies. When all threads are allocated, thermal tuning is used to bring all ring groups to the lowest common resonant frequency. Above, ring groups 1 and 3, as well as the laser source, are tuned to match the resonant frequency of ring group 2.

Adaptive Frequency Tuning

TFT tunes all ring groups and laser sources in PNoC to a target optical frequency. Using this method, the total tuning power depends directly on the sum of differences between the optical frequency of optical devices and the target frequency. When the system is underutilized, the tuning power becomes significant due to the low average temperature. Since in our work we use on-chip laser sources, which provide a much shorter control loop compared to off-chip laser sources, we propose a new tuning method to match the optical frequency of laser sources and ring resonators, called *Adaptive Frequency Tuning (AFT)*. In this tuning method, we set the lowest frequency among the ring groups as the target frequency and tune all the other devices to this target frequency. Because ring groups' resonant frequencies change with their temperatures, the target frequency is chosen adaptively based on the current lowest resonant frequency among all ring groups. Therefore, the tuning power depends on the combination of relative differences between the lowest resonant frequency among the ring groups and the optical frequencies of other optical devices. As a result, *FreqAlign* requires lower power consumption for optical frequency tuning (Fig. 5·6).

Performance Overhead Analysis

The performance overhead of *FreqAlign* policy is composed of two parts: (1) the execution time of *FreqAlign* and (2) the potential thread migration overhead in a many-core system. To evaluate the execution time of *FreqAlign*, we carried out an off-line experimental analysis considering the worst-case scenario of allocating one thread to each core in the target 256-core system. For our analysis, we implement *FreqAlign* in C programming language, compile it using *gcc* with -O3 flag, and run it on Sniper. The simulation results show that the allocation of 256 threads to 256 cores takes a total of 192 μs.

Whenever a new job enters the system, thread allocation in *FreqAlign* also involves a reallocation process, in which we migrate the existing threads if necessary. Thread migration may hurt application performance due to the context switch and the cache cold start effect. We use Sniper along with its hardware thread migration scheme (Van Craeynest et al., 2013) to investigate the impact of thread migration overhead on our target system's performance. Once the pipeline of the core a thread is originally running on has been drained, its architectural state is transferred to the destination core. The destination core then starts running the thread and endures the cache cold start effect. The overhead from migrating a thread from one core to another core includes three major components: (1) a fixed penalty of 1000 cycles for storing and restoring the core's architectural state (Van Craeynest et al., 2013); (2) the time to drain the source core's pipeline prior to migration; and (3) the cache cold start effect. As quantified in several previous studies (Van Craeynest et al., 2013;

Van Craeynest et al., 2012), cache cold start effect is the dominant component in migration overhead and can be two orders of magnitude larger than the other two components combined. In our thread migration scheme, there is no flushing of the source core's caches. Every cache miss in the destination core sends a memory request to the source core's L2 cache instead of memory. This lowers the number of memory accesses. Any source core's L2 cache block that is in shared/modified state triggers a writeback/invalidation using the normal cache coherency protocol.

As thread migration overhead varies with both workload and the number of threads to migrate, we carry out a comprehensive evaluation that considers all applications under varying number of threads used in this work (further details in Section 5.5.2). We configure Sniper with multiple thread migration intervals (time slice after which a thread migrating process occurs): 500 μs, 1 ms and 10 ms. For each interval case, we configure the migration percentage, i.e., the number of threads that actually migrate: 0.0 (baseline case without thread migration), 0.05 (5% of the threads migrate), 0.1, 0.25, 0.5 and 1.0 (all threads migrate). We compare the migration cases with the baseline cases to calculate the average cycle count per migration for each of the combinations. From the results, we observe a maximum of 147.3 μs (14.3 μs on average) increment in running time due to thread migration. The running times of our applications with native input size vary from hundreds of milliseconds to seconds (Carlson et al., 2012). Real-life running times for similar scientific applications vary from minutes to hours. As *FreqAlign* executes only when a new job arrives at the system, we conclude that it entails negligible performance overhead.

5.3.3 Experimental Results

To demonstrate the benefits and scalability of *FreqAlign*, we conduct experiments using systems with different logical topology / physical layout combinations and process variations and compare *FreqAlign* with two other policies: assignment of threads starting from the lowest indexed core (*Clustered*, shown in Algorithm 3) and *RingAware* (shown in Algorithm 1). In our target system, the cores are indexed from left to right and from bottom to top. There are 32 waveguides in the PNoC, and the data rate for each wavelength is 4 *Gbps*. Our design of experiments contains the following cases:

1. Six workload combination cases using two different jobs (see Table 5.4) at the same time: HPHP, HPMP, HPLP, MPMP, MPLP, LPLP; Job 1 arrives at 1 *ms* and Job 2 arrives at 2 *ms* after the start of each simulation. Jobs have different running times (see Table 5.5), and the shorter job repeats itself until the longer job finishes execution.
2. Six utilization cases: 25% (Job 1: 32 cores + Job 2: 32 cores), 50% (32+96, 64+64, 96+32 cores), 75% (96+96 cores), 100% (128+128 cores).
3. One case without process variations and four cases with process variations in different directions.
4. Five logical topology and physical layout combinations (Fig. 5·1(a) to 5·1(e)).

```
Sort all threads based on their power consumption;
foreach thread in queue do
    allocatedCore ← 0;
    for j = 1 to N do
        if core j is available then
            allocatedCore ← j;
            break;
        end
    end
end
```

Algorithm 3: Pseudocode for *Clustered* policy.

Table 5.4: Workload combinations. HP: High-Power; MP: Medium-Power; LP: Low-Power.

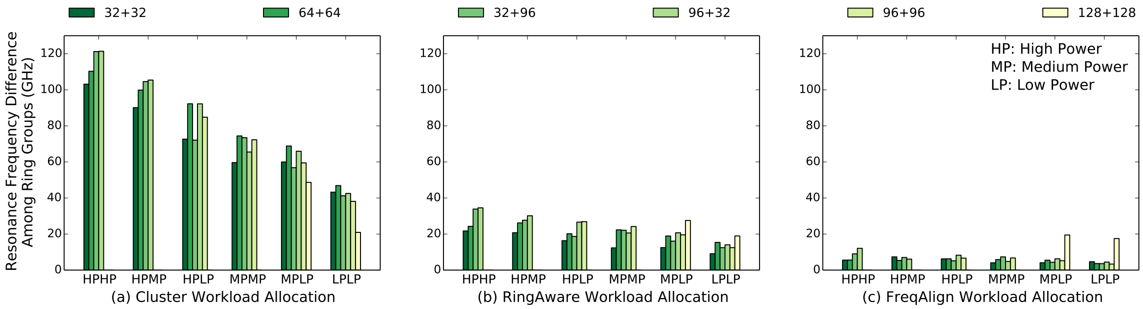
Workload	Job 1	Job 2
HPHP	md	shock
HPMP	md	blackscholes
HPLP	shock	lu_cont
MPMP	barnes	blackscholes
MPLP	barnes	water_nsq
LPLP	lu_cont	canneal

Table 5.5: Running times of jobs (Unit: *ms*).

Thread count	md	shock	blackscholes	lu_cont	barnes	water_nsq	canneal
32	320	397	30	140	237	17	72
64	295	309	24	128	218	16	68
96	221	205	16	104	158	16	62
128	167	188	12	89	139	16	59

Optical Frequency Tuning Evaluation

We compare the optical frequency difference and required tuning power for the three policies. Figure 5·7 shows resonant frequency differences among the ring groups for the three policies using a U-shape 8-ary 3-stage Clos PNoC (as shown in Fig. 5·1(a)). We can see that *Clustered* results in highest resonant frequency gradient among all three policies. *FreqAlign* reduces resonant frequency difference by 60.6% on average compared to *RingAware*. This is because *RingAware* only focuses on RD0 cores, but



(a) Cluster Workload Allocation (b) RingAware Workload Allocation (c) FreqAlign Workload Allocation

Figure 5·7: Average resonant frequency differences when using *Clustered*, *RingAware* and *FreqAlign* workload allocation policy for U-shape layout with 8-ary 3-stage Clos topology shown in Fig. 5·1(a). Each bar represents a workload and utilization combination case.

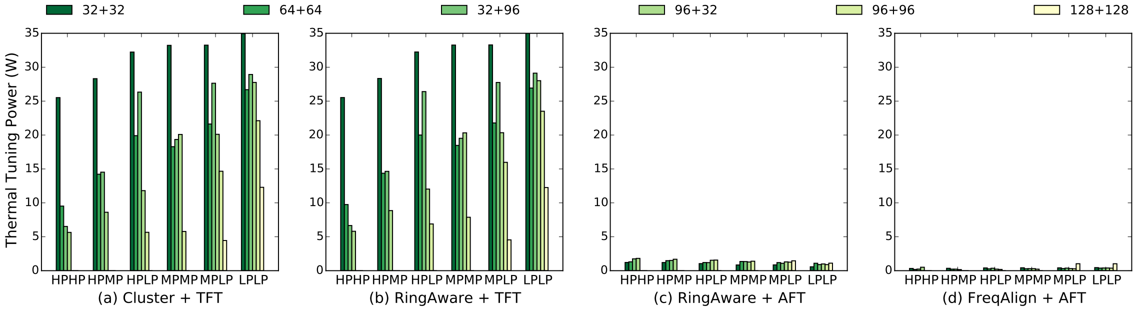


Figure 5·8: Average optical tuning power when using localized tuning for *Clustered*, *RingAware* and *FreqAlign* workload allocation policy for 8-ary 3-stage Clos topology with U-shape layout in Fig. 5·1(a). *TFT*: Target Frequency Tuning; *AFT*: Adaptive Frequency Tuning.

the aggregation of non-RD0 cores still has a huge impact on ring group temperature. *FreqAlign* estimates the impact of allocating a thread to a core on all ring group temperatures, which reduces resonant frequency difference among all ring groups.

In Fig. 5·8, we present the thermal tuning power when applying different workload allocation policies and tuning mechanisms. Here, we do not show the cases (e.g., HPHP with 128+128 threads) in which the maximum on-chip temperature is higher than the temperature threshold, 90 ^oC. This rule also applies to all the other figures in this section. Since *TFT* requires every ring group and laser source to be tuned to the resonant frequency at 90 ^oC, the required tuning power only depends on the absolute operating temperatures of the photonic devices. Under such scenarios, temperature balancing techniques without proper tuning strategies do not show advantage on reducing thermal tuning power. Thus *Clustered* and *RingAware* have similar required tuning power. *Adaptive Frequency Tuning (AFT)*, on the other hand, tunes the laser source frequency to align with the lowest of the current resonant frequencies of ring groups, which is balanced through the proposed workload allocation policy. *FreqAlign+AFT* saves 19.28 W thermal tuning power on average and up to 34.57 W compared to *RingAware+TFT*. This result demonstrates the need for proper control of the on-chip laser source and ring resonator optical frequency tuning mechanism.

Apart from edge-placed laser sources, we also test the proposed policy and baseline policies for the same many-core chip, but this time with locally-placed laser sources, where on-chip laser sources are placed around the ring groups along the U-shape waveguide. We observe similar percentages of thermal tuning power reduction for *FreqAlign*. For off-chip laser sources, due to the lack of runtime control, AFT is not applicable in this scenario. Thus, systems with off-chip laser sources have similar ring resonator thermal tuning power as the cases with TFT (Fig. 5·8(a) and (b)).

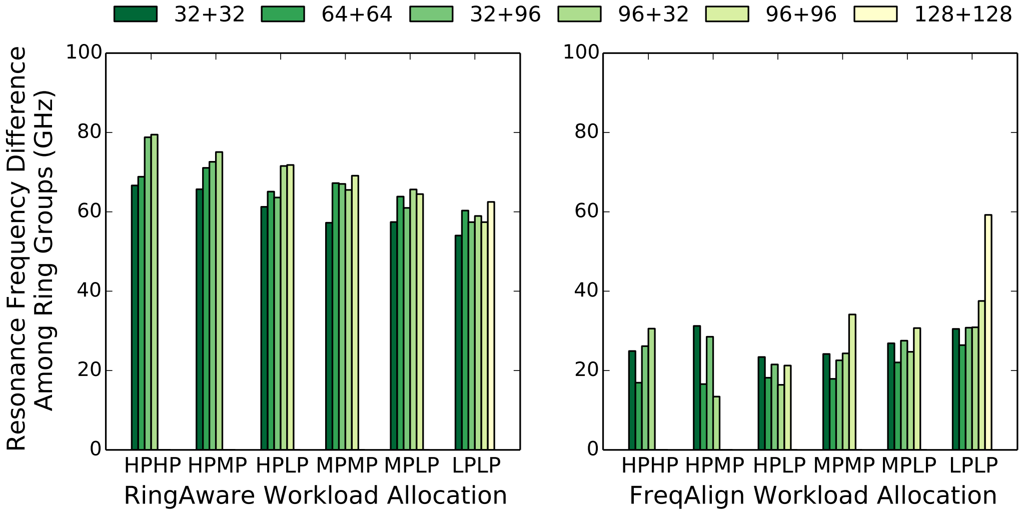
Case Study on Process Variation

Ring resonators are sensitive to process variations, and the resonant frequency can vary approximately linearly with distance on the scale of waveguide length (Zortman et al., 2010). To study the impact of process variations, we consider a wavelength variation of 400 pm/cm due to process variations by considering a linear process variation of 0.76 nm (Mohamed et al., 2010) over our approximately 1.9 cm long chip.[2] In this case study, we consider four process variation directions as shown in Fig. 5·10: (a) horizontal, (b) vertical, (c) diagonal which results in largest process variations among ring groups and (d) diagonal which results in largest process variations across the chip. We evaluate both *RingAware* and *FreqAlign* policies, with the assumed process variation directions. The results in Fig. 5·9 show that *FreqAlign* reduces the resonant frequency difference by 52.7% on average compared to *RingAware*. This is because *RingAware* is designed to balance the operating temperatures among the ring groups, and does not account for process variations. On the other hand, *FreqAlign* considers both temperature and process variations, so it significantly reduces the average resonant frequency difference.[3]

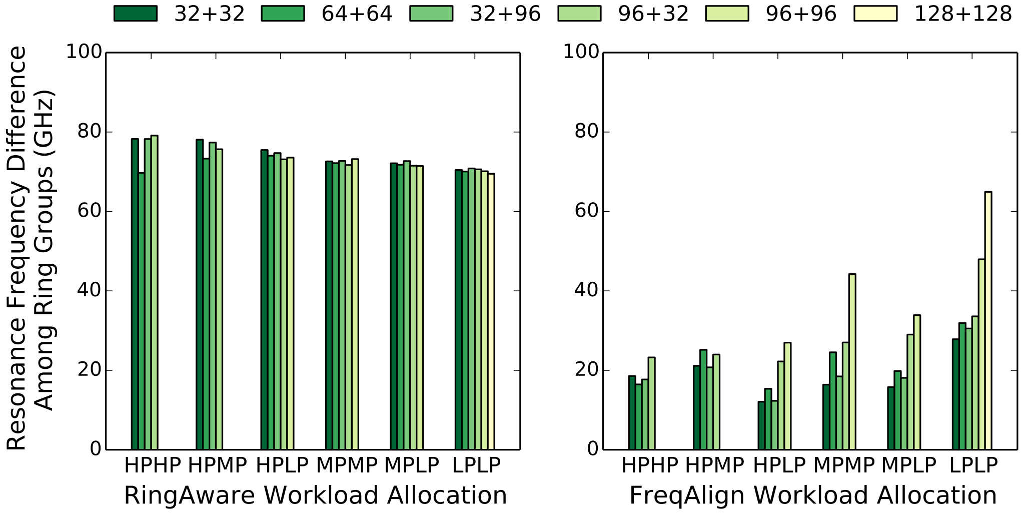
---

[2] While we are using linear process variation for our case study, *FreqAlign* comprehends unique process variation parameters for each ring group in a system, so any pattern of process variation between ring groups could be considered.

[3] We also investigate systematic WID process variations for both ring resonators and cores. We assume 50% leakage power variations across the chip from top to bottom due to WID process

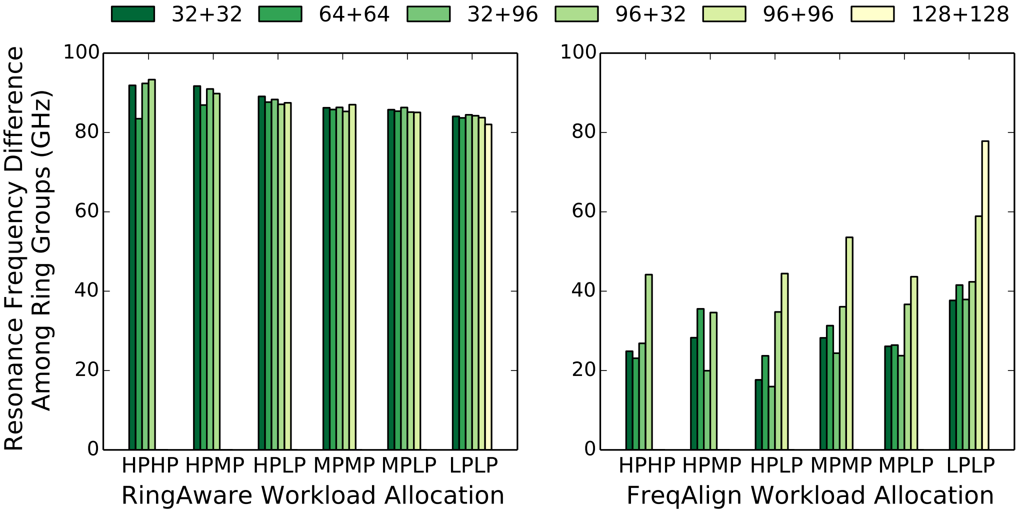


(a) Horizontal process variation gradient. Average (maximum) power reduction is 1.83 *W* (2.77 *W*), 62.9% (82.5%).

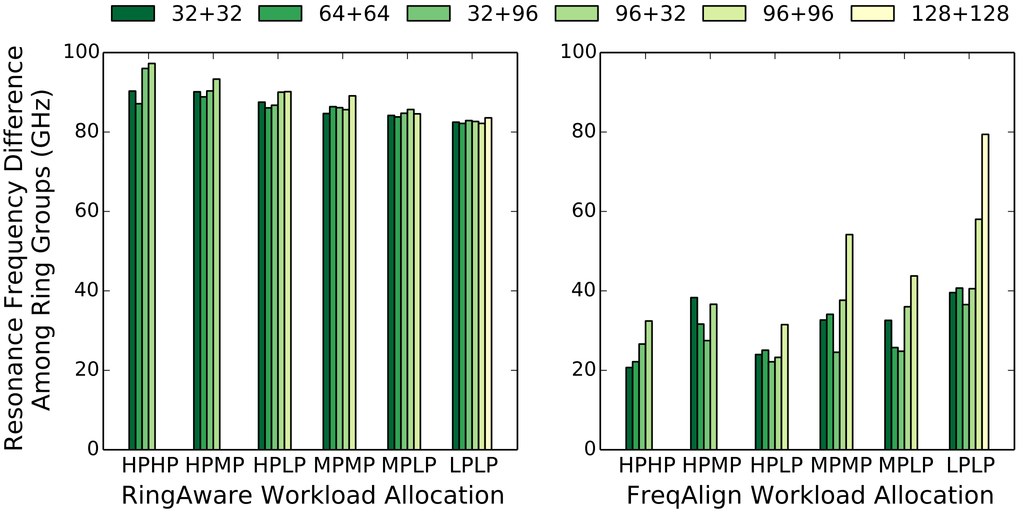


(b) Vertical process variation gradient. Average (maximum) power reduction is 1.81 *W* (2.91 *W*), 64.7% (84.3%).

Figure 5·9: Average resonant frequency difference when comparing *RingAware* and *FreqAlign* workload allocation policy for U-shape layout with 8-ary 3-stage Clos topology and a wavelength variation of 400 *pm*/*cm* in multiple directions due to process variations. The process variation cases and tuning power reduction of *FreqAlign*+*AFT* compared to *RingAware*+*AFT* are shown in the captions of subfigures.



(c) Maximum process variation among ring groups. Average (maximum) power reduction is 2.04 W (3.26 W), 60.1% (80.8%).



(d) Maximum on-chip process variation. Average (maximum) power reduction is 2.12 W (3.13 W), 61.4% (76.2%).

Figure 5·9: Average resonant frequency difference when comparing *RingAware* and *FreqAlign* workload allocation policy for U-shape layout with 8-ary 3-stage Clos topology and a wavelength variation of 400 pm/cm in multiple directions due to process variations. The process variation cases and tuning power reduction of *FreqAlign*+AFT compared to *RingAware*+AFT are shown in the captions of subfigures.

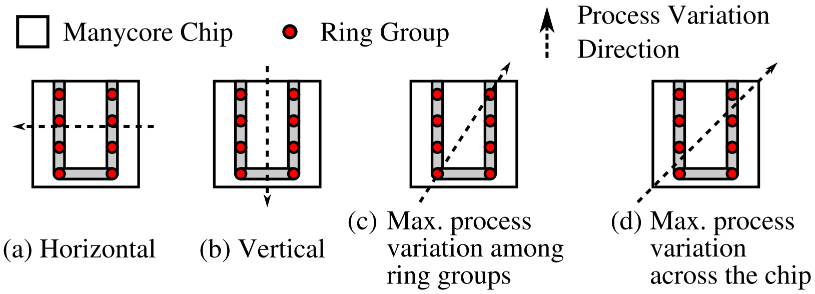

Manycore Chip
Ring Group
Process Variation Direction
(a) Horizontal
(b) Vertical
(c) Max. process variation among ring groups
(d) Max. process variation across the chip

Figure 5·10: Process variation directions considered for case study.

Case Study on Layout Sensitivity

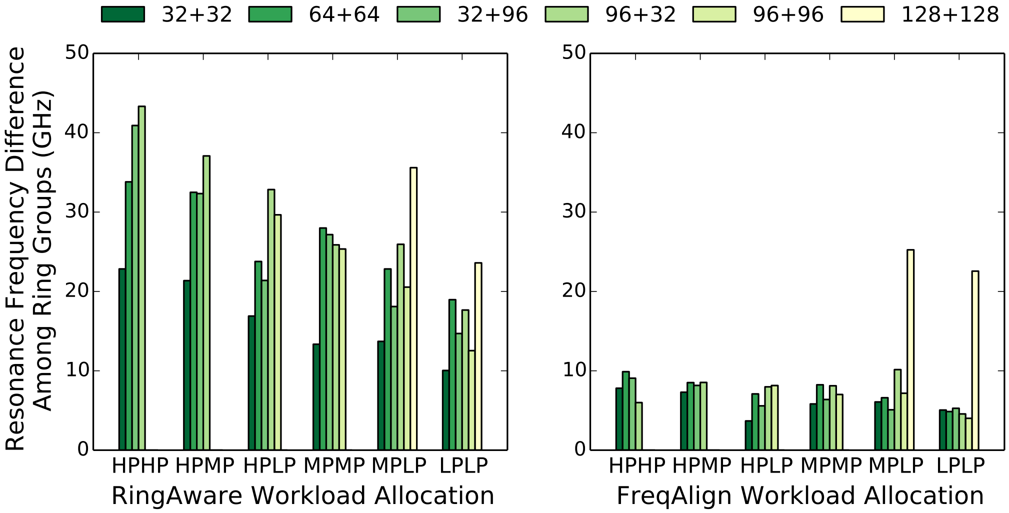
In addition to process variations, we also evaluate *FreqAlign* for various PNoC layouts. We conduct this case study in two aspects: (1) using the same PNoC logical topology and physical layout combination but slightly shifted ring group locations; (2) using different PNoC logical topology and physical layout combinations.

For Case (1), we test the two layouts with shifted ring group placements in Fig. 5·1(b) and Fig. 5·1(c). When using these two layouts, *FreqAlign* has 60% and 50.7% reduction in resonant frequency difference, respectively, compared to *RingAware* (see Fig. 5·11(a) and Fig. 5·11(b)) since it considers the estimated resonant frequencies of ring groups and their placement, when allocating the jobs. This case study demonstrates that *FreqAlign* is adaptive to minor changes in system layout.

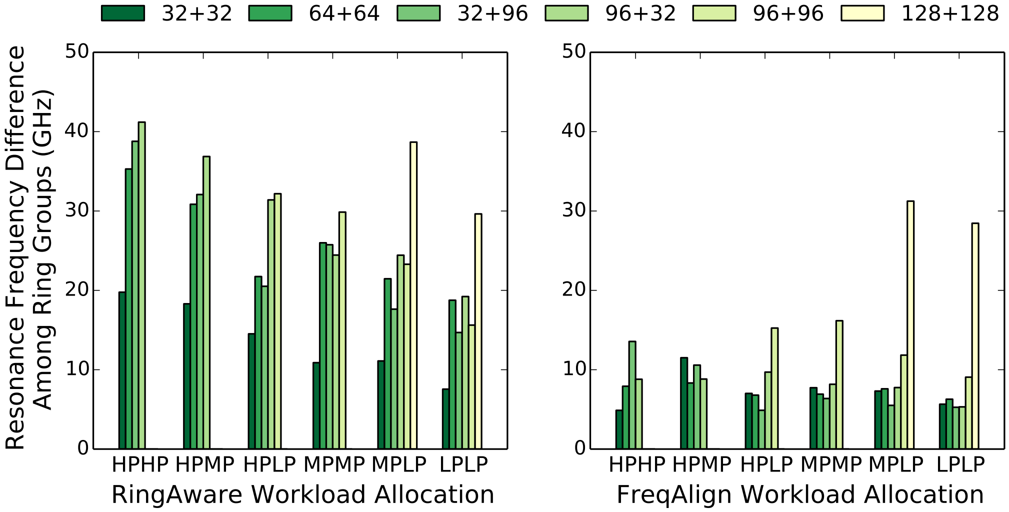
For Case (2), we use two other logical topology and physical layout combinations shown in Fig. 5·1(d) (16-ary 3-stage Clos topology and a W-shape layout) and Fig. 5·1(e) (8-ary 3-stage Clos topology and chain-shape layout). The comparisons of average resonant frequency difference among ring groups for these two cases between *RingAware* and *FreqAlign* are shown in Fig. 5·11(c) and Fig. 5·11(d), respectively. Compared to *RingAware*, *FreqAlign* reduces the average resonant frequency difference by 42.8% and 59.2% for W-shape 16-ary 3-stage PNoC and chain-shape 8-ary

---

variations in a decreasing gradient (Dighe et al., 2011). Our results show that *FreqAlign* reduces the resonant frequency difference by 57.3% compared to *RingAware*. Without WID process variations for cores, the reduction is 55.6%.

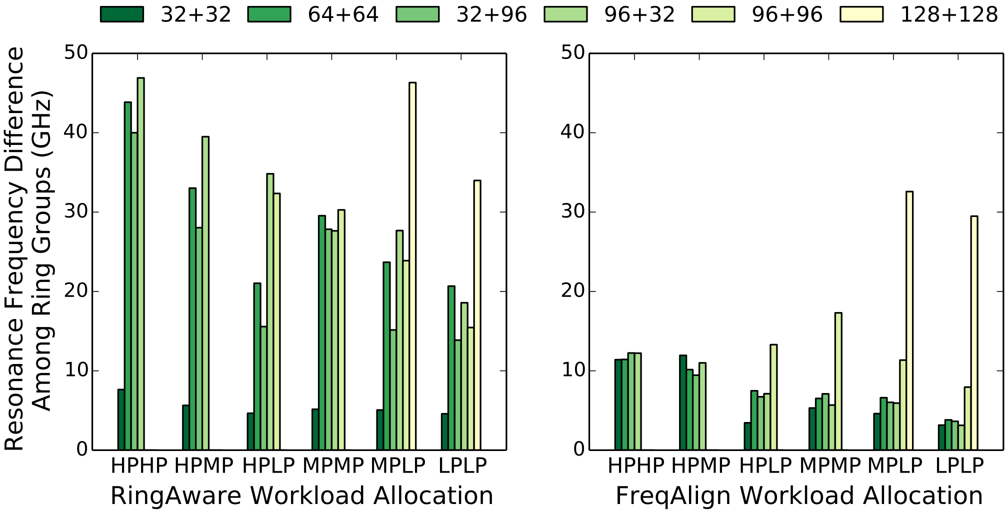


(a) Horizontally shifted U-shape layout with 8-ary 3-stage Clos topology (Fig. 5·1(b)). Average (maximum) power reduction is 1.03 W (2.05 W), 72.7% (96.3%).

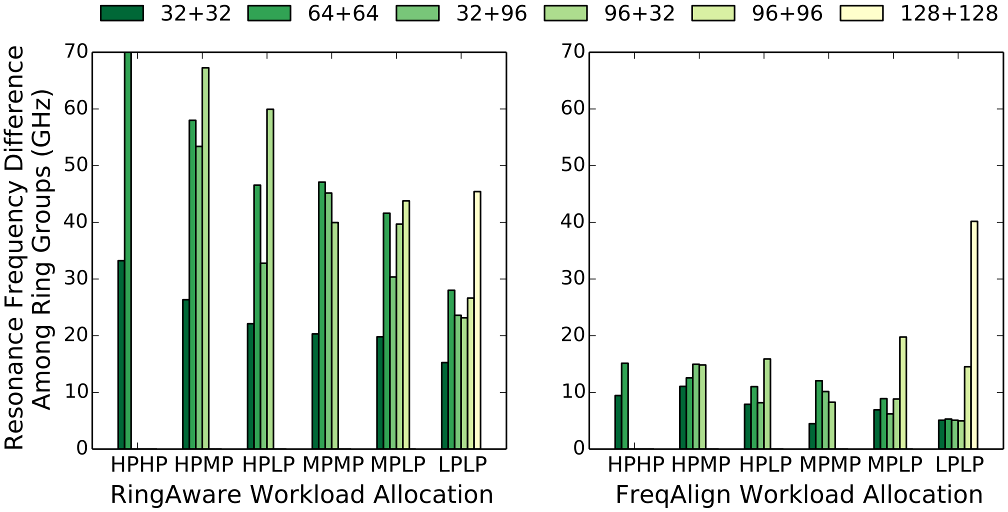


(b) Vertically shifted U-shape layout with 8-ary 3-stage Clos topology (Fig. 5·1(c)). Average (maximum) power reduction is 0.80 W (1.47 W), 65.8% (90.3%).

Figure 5·11: Comparison of average resonant frequency difference among ring groups for different PNoC logical topology and physical layout combinations between *RingAware* and *FreqAlign*. The PNoC cases and tuning power reduction of *FreqAlign+AFT* compared to *RingAware+AFT* are shown in captions of subfigures.



(c) W-shape layout with 16-ary 3-stage Clos topology (Fig. 5·1(d)). Average (maximum) power reduction is 1.63 W (3.47 W), 30.1% (82.7%).



(d) Chain-shape layout with 8-ary 3-stage Clos topology (Fig. 5·1(e)). Average (maximum) power reduction is 0.92 W (1.8 W), 70.3% (77.9%).

Figure 5·11: Comparison of average resonant frequency difference among ring groups for different PNoC logical topology and physical layout combinations between *RingAware* and *FreqAlign*. The PNoC cases and tuning power reduction of *FreqAlign+AFT* compared to *RingAware+AFT* are shown in captions of subfigures.

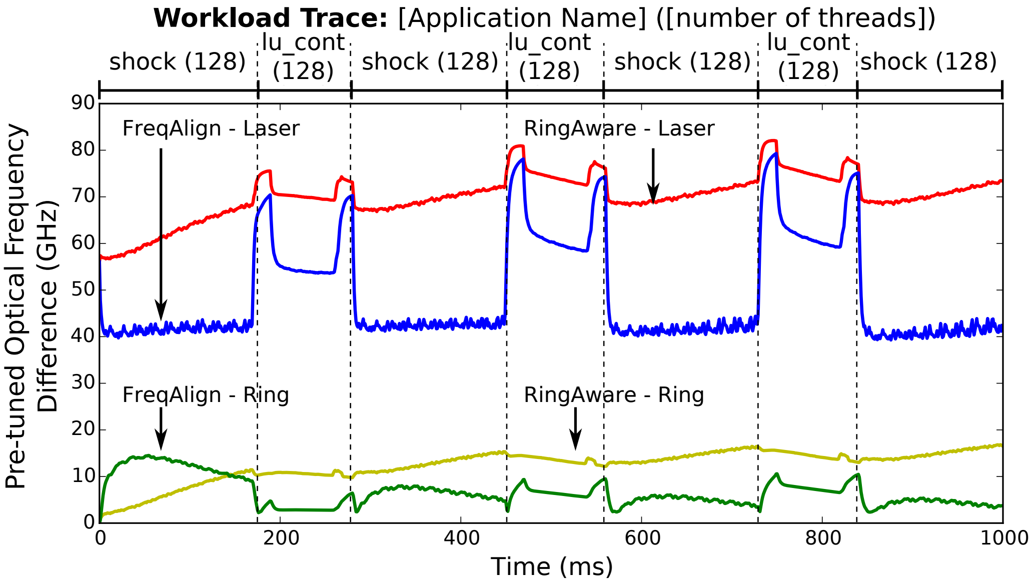


Figure 5·12: Runtime optical frequency difference trace for transient case with large temporal on-chip temperature gradients prior to applying tuning methods.

3-stage PNoC, respectively. Thus, *FreqAlign* is more adaptive to different PNoC logical topology and physical layout combinations.

Transient Resonance Frequency Investigation

To compare the transient behavior of *RingAware* and *FreqAlign*, we conduct a test case where the target system has a large temporal temperature gradient during operation. We use HPLP (shock + lu_cont) as the jobs and let them run alternately. We use the U-shape layout and 8-ary 3-stage Clos topology for the many-core system and set the initial temperatures of all units at 65 ^oC, so *RingAware* and *FreqAlign* have the same starting point. We also limit the running time to be 1000 ms. Figure 5·12 shows the maximum optical frequency difference between the target frequency and the optical frequencies of ring groups and on-chip laser sources for this test case prior to applying tuning methods. We observe that at first *RingAware* results in lower frequency difference than *FreqAlign*. This is because *RingAware* has a more even

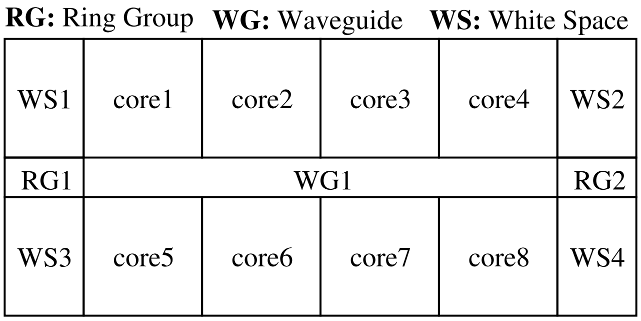


Figure 5·13: Floorplan of a 2× 4 system.

workload distribution than *FreqAlign*. However, such a distribution causes the inner ring groups to be hotter than the outer ring groups. As the time progresses and the system enters its steady state, the resonant frequency difference of *RingAware* increases. On the other hand, *FreqAlign* achieves much lower resonant frequency difference. The jittering during shock (128) and variation during lu_cont (128) in the traces are caused by applications going through different phases during execution. *FreqAlign* reduces the amount of optical frequency that the on-chip laser sources need to be tuned compared to *RingAware*. Since scientific applications usually have long running times (minutes or hours), which is sufficient for the system to enter steady state, using *FreqAlign* can achieve lower resonant frequency difference than *RingAware* in general.

5.3.4 Comparison with Optimal Workload Allocation

An important issue, which is difficult to address in the context of heuristics for NP-hard optimizations, is the degree of suboptimality of heuristic solutions. To offer some insight into the quality of *FreqAlign* workload allocation solutions relative to optimal solutions, we have studied the various heuristics on a smaller chip architecture (a 2×4-core system with two ring groups placed on the two shorter opposite edges, as shown in Fig. 5·13). This is because finding the optimal solution is practically infeasible for larger systems ($N!$ workload allocation solutions for a fully utilized N-core system).

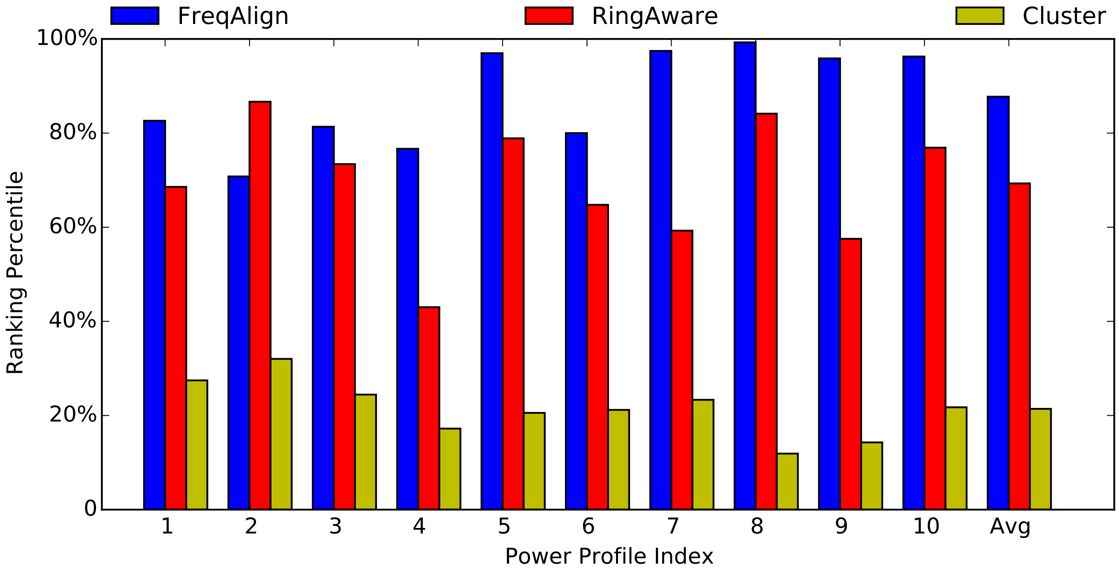


Figure 5·14: Ranking percentile of *FreqAlign*, *RingAware* and *Clustered* that indicates the ranking of these workload allocation algorithm among all possible workload allocation solutions for ten randomly generated power profiles (varying from 0.4 W to 2.8 W) for cores.

For the 2×4 system, we run all 10080 workload allocation solutions ($10080 = 8!/4$ after accounting for horizontal and vertical symmetries) for ten power profiles generated by picking power numbers randomly between 0.4 W and 2.8 W for each core. On average, *FreqAlign* results in lower resonant frequency difference between two ring groups than 87.7% of all workload allocation solutions, while *RingAware* outperforms 69.3% of all workload allocation solutions, as shown in Fig. 5·14. This suggests that *FreqAlign* returns solutions that are substantially closer to optimal than those of *RingAware*. We leave the closing of this suboptimality gap for future research.

5.4 Design-time Power Management Through Laser Source Placement

In addition to runtime thermal management, design-time choices also impact the on-chip thermal conditions and PNoC power. For example, laser source power, one of the major components of PNoC power, significantly depends on the routing of waveguides and the placement of router groups in a many-core system. Thus, we also design techniques in floorplanning and placing on-chip silicon-photonic devices to reduce PNoC power.

This section describes our on-chip laser source placement and sharing strategies. We first present the laser source model we use to evaluate the laser source operating regimes. We then discuss the tradeoffs among laser source optical output power, wall-plug efficiency (WPE) and temperature, followed by the proposed methodology to determine the sharing and placement of on-chip laser sources for minimizing laser source power.

5.4.1 Laser source power model

A laser source's power consumption depends on its input current and operating temperature. A laser source takes electrical input power and converts a portion of it to optical output power. The ratio of optical output power (P_o) to electrical input power (P_{IN}) is named as laser source WPE (η_{WPE}):

$$\eta_{WPE} = \frac{P_o}{P_{IN}}, \tag{5.5}$$

$$P_o = \eta_i \eta_d \frac{hc}{\lambda q}(I - I_{th}), \tag{5.6}$$

where η_i and η_d are the laser source internal efficiency and differential quantum efficiency, respectively; h, c and q are Planck's constant, the speed of light, and the elementary electric charge, respectively; λ is the laser source operating wavelength; I and I_{th} are the drive and threshold currents, respectively (Coldren et al., 2012).

The electrical input power of a laser source is the product of the input current and the total voltage across the laser source's terminals and can be calculated as:

$$P_{IN} = I^2 R_s + I V_d, \tag{5.7}$$

where R_s is the laser source series resistance and V_d represents the diode voltage.

For semiconductor laser sources, P_o has a strong dependence on temperature. Fortunately, simple empirical formulae match well with the measured characteristics of many different semiconductor diode laser sources (Coldren et al., 2012).

$$I_{th} = I_{th}^0 e^{T/T_0}, \tag{5.8a}$$

$$\eta_d = \eta_d^0 e^{-T/T_\eta}, \tag{5.8b}$$

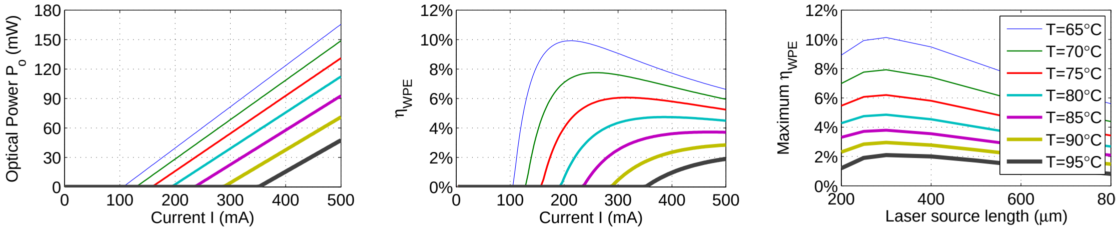
where T_0 and T_η are the characteristic temperatures of the threshold current and the differential quantum efficiency, respectively, and I_{th}^0 and η_d^0 are the threshold current and the differential quantum efficiency projected to a reference temperature. Additionally, the diode voltage V_d is determined through the Shockley diode equation:

$$V_d = \frac{k_B T}{q} ln\left(\frac{I}{I_s}\right), \tag{5.9}$$

where k_B is the Boltzmann constant and I_s is the reverse bias saturation current. By substituting Eqs. (5.8) and (5.9) into Eqs. (5.6) and (5.7), simple relationships are expressed for the dependence of WPE on operating temperature. This model is well established and the parameters are extracted from measurement results (Hu et al., 1994b; Hu et al., 1994a). We consider a strained InP-based multi-quantum well laser source structure.

5.4.2 Laser source power, efficiency and temperature tradeoffs

Since laser source power and WPE strongly depend on the input current and the operating temperature, for energy-efficient computing, it is essential to operate a laser



(a) **(b)** **(c)**

Figure 5·15: (a) P-I characteristics of a laser source, (b) WPE vs. input current, and (c) WPE vs. laser source lengths at various temperatures.

source at its highest possible efficiency. The following discussion provides insights on finding a laser source's optimal operation point (i.e., a set of input current and operating temperature).

The P-I characteristic (the optical output power of a laser source versus the input current) of the target laser source under various temperatures is shown in Fig. 5·15a. It shows that the threshold current I_{th} increases along with temperature while the laser optical output power decreases with increment in temperature for a given input current. For a fixed temperature, as the input current increases, the WPE initially increases, reaches a peak value and then decreases, as shown in Fig. 5·15b. The peak efficiency decreases at higher temperature as expected from the laser source power model (Eqs. 5.8). Thus, it is preferred to keep the laser source operating at a low temperature and ensure that the input current is at the value where the WPE is maximized. In addition to input current and operating temperature, a laser source's dimension also impacts its WPE. Figure 5·15c shows the relationship between WPE and laser source length under various temperatures. We fix the laser source width at 50 μm while vary the laser source length from 200 μm to 700 μm. We can observe from Fig. 5·15c that a 300 μm long laser source has the highest efficiency at all temperatures. This is because for short cavity length, high-order effects result in a reduction of the carrier density above threshold, which in turn decreases η_i. For long

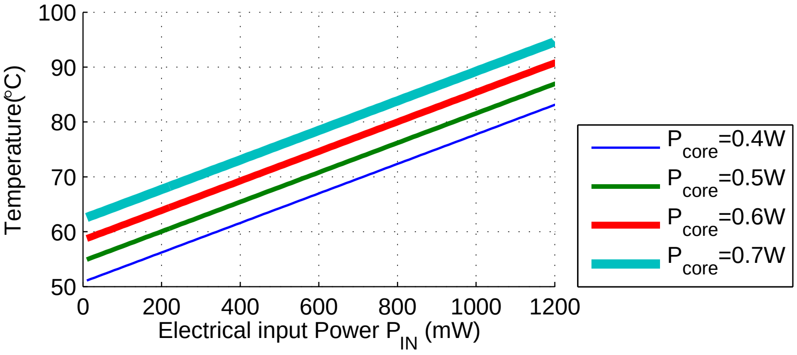
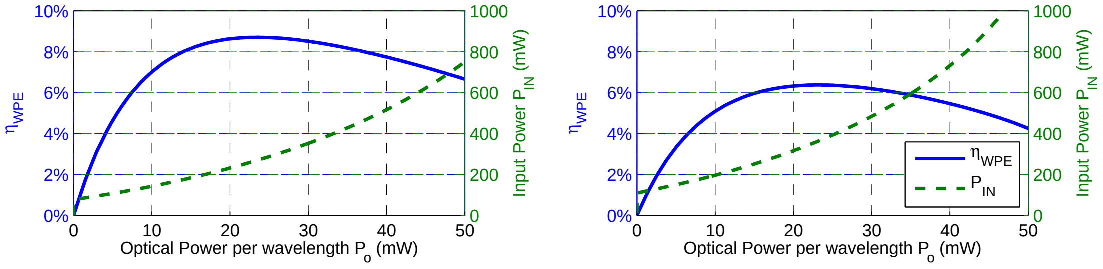


Figure 5·16: Laser source temperature vs. electrical input power for a 300 μm × 50 μm laser source.



(a) **(b)**

Figure 5·17: Wall-plug efficiency vs. optical output power by the laser source for different granularities of sharing while a background logical layer operates at 0.4 W per core (a) and 0.7 W per core (b).

cavity length, η_d dominates and the efficiency decreases. Based on this model, we select a 300 μm × 50 μm laser source for the following analysis.

To determine the impact of electrical input power on laser source temperature, we ran thermal simulations for a 256-core system with each core consuming 0.4 W, 0.5 W, 0.6 W and 0.7 W of power. The laser sources are placed on a separate layer on top of the logic layer that contains cores and caches. As shown in Fig. 5·16, as electrical input power increases, the laser source temperature increases, and the WPE decreases in return. The increase in core power also increases the temperature of the laser source, which further lowers the WPE.

Based on the power-temperature-efficiency tradeoffs of the laser source, Fig. 5·17 shows the laser source efficiency and electrical input power as a function of optical output power of laser sources for two scenarios – one where each core has 0.4 W power and the second where each core has 0.7 W. The required optical output power from a laser depends on the optical loss in the photonic link being driven by that laser source. For 0.4 W case, Fig. 5·17a shows that the optimal operation point is a laser output power of 23 mW per wavelength, where the laser source has its peak efficiency of 8.2% and an electrical input power of 268 mW. When every core's power is 0.7 W, the optimal laser output power is still approximately 23 mW, but the laser efficiency decreases to 6.2% due to the higher operating temperature, which results in an electrical input power of 355 mW. This analysis demonstrates that the electrical input power of a laser source increases along with core power consumption. Hence, it is essential to consider system power profile for laser source power reduction.

As shown in Fig. 5·17, a laser source outputs a specific optical output power at its maximum efficiency. Depending on the optical power required per λ and the applied laser technology, it may be desirable to share laser source output power across two or more waveguides. In the next section, we investigate the impact of laser source sharing on the total laser source power consumption. We assume the sharing of one laser source among multiple waveguides is done by splitting the emitted lightwave from the laser source (the corresponding optical loss is 0.2 dB/split).

5.4.3 On-chip laser source sharing and placement strategy

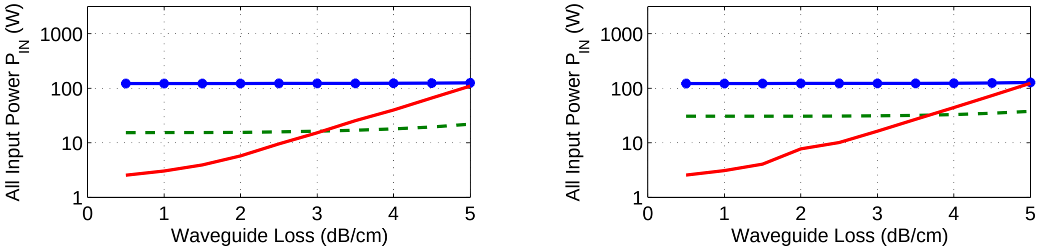
To determine the sharing and placement of laser sources in a many-core system, we propose a cross-layer approach where we jointly consider the NoC bandwidth constraints driven by the target applications, thermal constraints driven by the power of cores and laser sources, and physical layout constraints driven by the losses in silicon-photonic devices. Following the flow in Fig. 5·2, we can determine the number of silicon-photonic links, i.e., number of wavelengths required by the target system. The chosen logical topology for an NoC can be mapped to several different layouts (Batten et al., 2013). We identify various potential physical layouts of PNoC with three candidate schemes for the on-chip laser source placement and sharing – 1) all laser sources are placed locally next to the router with each laser source emitting one wavelength for one waveguide (no sharing); 2) all laser sources are placed locally next to the router with each laser source emitting one wavelength that is shared across multiple waveguides; and 3) all laser sources are placed along the chip edge with each laser source emitting one wavelength shared by two or more waveguides. For a fixed set of physical layout and bandwidth per channel, based on the on-chip thermal conditions and the laser source power model, we use the placement and sharing scheme with the lowest laser source power among the three candidates.

5.4.4 Case Studies

In this section, we present two case studies to show how the sharing and placement of laser sources change with logical topologies and physical layouts. We assume the target system bisection bandwidth is 512 GB/s, and each core is consuming 0.7 W power. We use the described laser source power model to calculate laser source WPE and power. Since the waveguide propagation optical loss is the major component of total optical loss, we vary the per unit length (cm) waveguide loss in the analysis to provide an insight for waveguide selection when designing PNoCs.

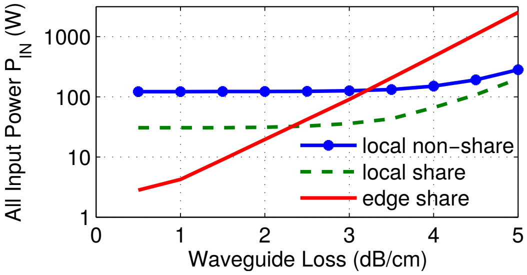
Laser source placement and sharing across various logical topologies

First, we look into the impact of logical topologies on the choice of on-chip laser source placement and sharing scheme. Figure 5·18a shows the total electrical input power of laser sources for an 8-ary 3-stage Clos topology for different placements of laser sources. There are 64 photonic channels connecting the 1^{st} and 2^{nd} stage of routers and another 64 connecting the 2^{nd} and 3^{rd} stage of routers. The detailed



(a) 8-ary 3-stage Clos with U-shaped layout

(b) 16-ary 3-stage Clos with U-shaped layout



(c) 16-ary 3-stage Clos with W-shaped layout

Figure 5·18: Total laser power vs. waveguide loss for various sharing scenarios and placements of on-chip laser sources. (a) and (b) compare various topologies with U-shaped layout. (b) and (c) compare various layouts for 16-ary 3-stage Clos topology. We assume each core in the logical layer consumes a power of 0.7 W.

specifications of the 8-ary 3-stage Clos are given in Table 5.6. The photonic channels are mapped to a U-shaped layout shown in Fig. 5·1. For a typical waveguide loss of 2 dB/cm, 0.15 mW of optical output power per wavelength is required. If local laser sources are not shared, the efficiency of each local laser source for 0.15 mW optical output power is 0.12%. This results in a total electrical input power for laser sources of 243 W (119 mW per laser source).

Given that the routers in the 1^{st}, 2^{nd} and 3^{rd} stage of the Clos network are placed next to each other, we can share the local laser sources among the 16 photonic channels, whereby the optical output power of a laser source is split and routed into the waveguides associated with these photonic channels. Such sharing mechanism increases the total optical output power of each local laser source, and thus, improves its WPE. For this particular example, each one of the 16 photonic channels is mapped to a waveguide with 16 λ per photonic channel, i.e. 16 λ per waveguide. If each laser source is shared across these 16 waveguides, the total optical output power is 2.4 mW for each λ, which corresponds to a laser source efficiency of 1.3% and a total electrical input power of 23.63 W (185 mW per laser source).

For the laser source considered in this study, the maximum efficiency is achieved at an optical output power of 23 mW. To use a laser source that outputs 23 mW optical power, we propose to place laser sources along the chip edge and share them among all waveguides. In this case, the optical output power required for each waveguide is 0.18 mW for each λ. This value is higher than local share scheme because average waveguide propagation loss increases with lengths of waveguides. We share 16 laser sources (1 for each λ) across 128 waveguides so that they can operate at 6.38% efficiency. This results in a total electrical input power of the lasers of 5.74 W (359 mW per laser source).

As the waveguide loss per cm increases, the total required optical output power increases. In the case of local non-shared laser sources, this increment in optical output power improves WPE. As a result, the total electrical input power does not increase significantly. In the case of local shared laser sources, such increase in optical output power drives the WPE towards the peak value. Hence, similar to local non-shared laser sources, there is only a marginal increment in the total electrical input power. The layout with laser sources located along the edge has longer waveguides, and so the optical loss increases significantly when waveguide propagation loss increases, which in turn lowers the WPE. Overall, if the waveguide loss is low (< 3 dB/cm), using edge-placed shared laser sources is the best option. If the waveguide loss is high ($>$ 3 dB/cm), placing shared laser sources locally is beneficial.

For the same 256-core target system, we could use a 16-ary 3-stage Clos network

Table 5.6: Architectural-level parameters for three tested PNoCs. U-shaped and W-shaped layouts are shown in Fig. 5·1.

Logical topology	**Physical layout**	**Dimension**	**Concentration**	**λ/Channel**	**Number of channels**
Clos	U-shaped	8-ary 3-stage	4	16	128
Clos	U-shaped	16-ary 3-stage	1	4	512
Clos	W-shaped	16-ary 3-stage	1	4	512

for less contention among cores at the input of each router. This 16-ary 3-stage Clos topology has 48 routers (16 routers in each stage) with each router in the 1^{st} and 3^{rd} stage connected to 16 cores (one core per router input). This network topology requires a total of 512 channels. To match the bisection bandwidth of this topology with the 8-ary 3-stage Clos, each channel needs four λ, and the system has a total of 128 waveguides with four channels (four λ for each channel) sharing one waveguide (16 λ in each waveguide). In general, the trends for the electrical input power of the laser for the 16-ary 3-stage Clos are similar to the trends for the 8-ary 3-stage Clos. One exception is that the electrical input power for the case using shared local laser sources is higher for the 16-ary 3-stage topology due to the decrease in the degree of sharing of laser sources.

Overall, the best sharing and placement of on-chip laser sources depend on the network topology. For example, at 3.5 dB/cm waveguide loss, for 8-ary 3-stage Clos topology mapped to U-shape physical layout, using shared local laser sources minimizes the electrical input power, while for 16-ary 3-stage Clos using shared laser sources located along the edge is the better choice. On the other hand for a 2 dB/cm waveguide loss, shared local laser sources are preferable for all three logical topologies.

Laser source placement and sharing across various physical layouts

Depending on alternate power, performance and area design constraints, the placement and routing tools may generate physical layouts that are different from the U-shaped layout that we considered in the earlier subsection. The choice of laser source sharing and placement changes with a change in the physical layout. For example, mapping the 16-ary 3-stage Clos topology to a W-shaped layout (see Fig. 5·1) increases optical waveguide losses, and hence the electrical input power required for a laser source. Figure 5·18c shows the total electrical input power for the lasers varying with waveguide loss per *cm* for the 16-ary 3-stage Clos with W-shaped layout, respectively. Similar to the U-shaped layout, for low waveguide loss shared laser sources placed along the edge are preferable, while for high waveguide loss shared local laser sources are preferable. The crossover point (i.e., where the choice of laser source placement and sharing changes from shared laser sources placed along the edge to shared laser source placed locally) is different for the two layouts. For the 16-ary 3-stage Clos topology, the crossover points are 3.6 dB/cm and 2.2 dB/cm for the U-shaped and W-shaped layouts, respectively.

5.5 Design-time Thermal Management Through PNoC Floorplanning

Placing laser sources on-chip and sharing them among waveguides is not the only way to reduce laser source power consumption. Since the total optical loss primarily comes from waveguide propagation loss, crossing loss, and bending loss, the routing of waveguides among computation clusters also plays an important role. In this section, we propose a cross-layer optimization of PNoC and core cluster floorplan for many-core systems. Our technique takes a many-core system's parameters (e.g., number of cores, core parameters, aspect ratio) as input and output a floorplan with the lowest PNoC power consumption. The core of our technique is a mixed-integer linear programming (MILP) formulation that minimizes PNoC power, including (1) laser source power due to propagation, bending, and crossing losses; (2) electrical and electrical-optical-electrical conversion power; and (3) thermal tuning power.

5.5.1 MILP-Based PNoC Floorplan Optimization

Our floorplan optimization comprehends the many-core chip and the PNoC as follows (see Fig. 5·19). In the chip, *cores* are grouped together to form *tiles*. All communication within a tile is local (i.e., does not go through the PNoC) and electrical. In the studies reported below, we assume a fixed bandwidth of 512 GB/s for the PNoC (Chen et al., 2014). We also assume an *8-ary 3-stage Clos* logical topology of the PNoC. The PNoC consists of *router groups*, each assigned to a set of tiles that constitute a *cluster*. All communication among tiles within a given cluster and among routers in the same router group goes through electrical links. Two router groups across clusters communicate with each other using an optical link. The connection from one router group to another is called a *net*, which we must route legally within the routing graph. Implicitly, the studies reported below consider monolithic integration (Orcutt et al., 2012; Georgas et al., 2014) (as opposed to TSV-based stacked-die integration) of the photonic components with serpentine routing of all waveguides together (due to the cost of the trenches on the die). We assume on-chip laser sources are placed next to the router groups on a separate layer (Chen et al., 2014) where the link begins and ends.

Notation Used in the MILP

Table 5.1 gives parameters and notations that we use in formalizing our MILP. The PNoC is defined by the locations of each router group in the set C, the orientations of their corresponding clusters, and the specific waveguides used to connect the router groups according to the topology implied by the set of nets N. As shown in Fig. 5·19, each router group is associated with a rectangular cluster of tiles around it. The cluster can be oriented vertically or horizontally, with the router group itself at the cluster's geometric center. A is the set of all available edges in the routing graph, where a_{vrq} (resp. a_{hrq}) denotes a vertical edge from vertex (r, q) to $(r + 1, q)$ (resp. a horizontal edge from vertex (r, q) to $(r, q + 1)$). N is the predefined set of nets connecting the router groups according to the logical topology of the PNoC. Each net n has a given source cluster s_n and sink cluster t_n, where $s_n, t_n \in C$.

Formal MILP Statement

We minimize a objective function (Equation (5.10)) that is a weighted combination of the PNoC area and power, where α and β are user-specified scaling factors.

$$\textbf{Minimize:} \quad \alpha \cdot P_{PNoC} + \beta \cdot AREA_{PNoC} \tag{5.10}$$

Subject to:

$$\sum_{r\in R, q\in Q, f\in\{0,1\}} \gamma^c_{frq} = 1, \qquad \forall c \in C, \quad \gamma^c_{frq} \in \{0,1\} \tag{5.11}$$

$$o_{crq} = \sum_{r'\in R, q'\in Q, f\in 0,1} o_{fr'q'}(r,q)\gamma^c_{fr'q'}, \quad \forall c \in C \tag{5.12}$$

$$\sum_{c\in C} o_{crq} \leq 1, \qquad \forall q \in Q, r \in R \tag{5.13}$$

$$2v^n_{rq} - e^n_{hrq-1} - e^n_{vr-1q} - e^n_{hrq} - e^n_{vrq} - \sum_{f\in 0,1} \gamma^{s_n}_{frq} - \sum_{f\in 0,1} \gamma^{t_n}_{frq} = 0, \quad \forall n \in N, r \in R, q \in Q \tag{5.14}$$

$$r_c = \sum_{r\in R, q\in Q, f\in 0,1} r \cdot \gamma^c_{frq}, \quad q_c = \sum_{r\in R, q\in Q, f\in 0,1} q \cdot \gamma^c_{frq}, \quad \forall c \in C \tag{5.15}$$

$$f_c = \sum_{r\in R, q\in Q, f\in 0,1} f \cdot \gamma^c_{frq}, \quad \forall c \in C \tag{5.16}$$

Structural Constraints

A number of constraints enforce proper structure of the core cluster placement and the PNoC routing. Using the 0-1 indicator variable γ^c_{frq}, Equation (5.11) ensures that exactly one vertex (r, q) and one orientation (horizontal or vertical) are chosen for each router group c. Equation (5.15) captures the vertex (r_c, q_c) in the routing

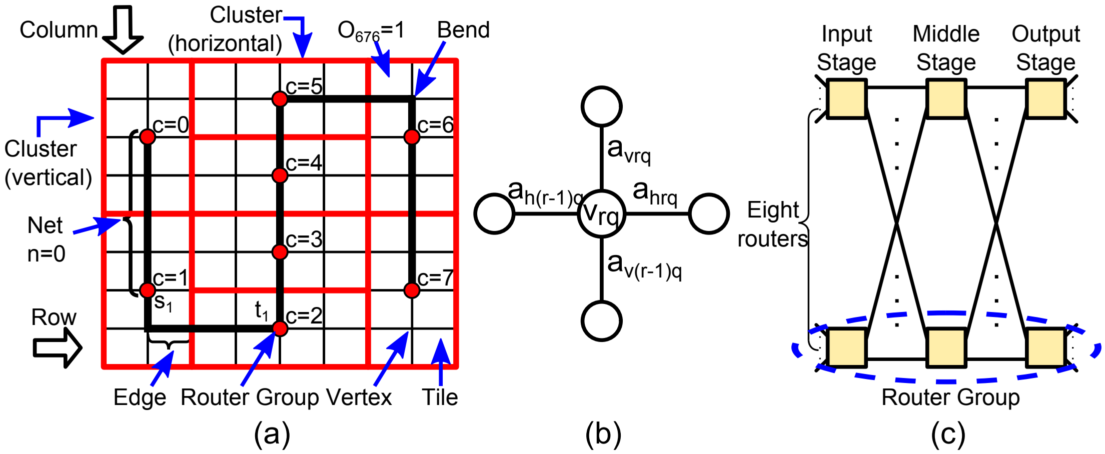


Figure 5·19: (a) Example of chip floorplan to illustrate our terminology. (b) A vertex and its surrounding edges in the routing graph. (c) 3-stage Clos topology with 8 router groups per stage.

graph where the router group c is placed, and Equation (5.16) captures the orientation f_c of the cluster of router group c.

Equation (5.12) captures which tiles on a chip are occupied by which cluster. A given o_{crq} indicates whether tile (r, q) is occupied by the cluster of router group c. $o_{fr'q'}(r, q)$ is a pre-calculated two-dimensional array that indicates whether tile (r, q) would be occupied by a cluster of a router group placed at (r', q') with orientation f. The array has an entry of one at each location that is occupied, and zero everywhere else. Equation (5.13) enforces the constraint that no tile on the chip can belong to more than one cluster. This ensures legal placement of clusters. If a tile is not in the footprint of any placed cluster (implying whitespace in the floorplan, e.g., for components other than the cores that communicate through the PNoC), for that tile we will have $\sum_{c \in C} o_{crq} = 0$. Equation (5.14) (Jafari et al., 2009) imposes flow conservation, i.e., a well-formed path of routing graph edges for each net n from its source s_n to its sink t_n. The 0-1 indicator variable v^n_{rq} captures the use of vertex (r, q) in the routing of net n; $e^n_{h/vrq}$ is a 0-1 indicator of whether edge $a_{h/vrq}$ is used in the routing of net n.

Equations for Area and Power

The area component of our objective function is determined by the following constraints. Equation (5.17) uses γ^{c}_{frq} to identify a binary indicator for the row (x_{cr}) and column (y_{cq}) the router group c is in. There is only one γ^{c}_{frq} that can be non-zero, and there is only one value in an array of all rows and an array of columns that is non-zero for each router group. Equations (5.18) and (5.19) indicate which rows and columns have router groups assigned to them. Router group locations cause extra area to be taken up in the chip, so by counting the number of rows and columns that are occupied we can obtain a figure of merit for how much area is required for photonic components.

$$x_{cr} = \sum_{q \in Q, f \in 0,1} \gamma^{c}_{frq}, \quad y_{cq} = \sum_{r \in R, f \in 0,1} \gamma^{c}_{frq} \;\; \forall c \in C \tag{5.17}$$

$$used_r = \begin{cases} 1 & \text{if } \sum_{c \in C} x_{cr} \geq 1, \forall r \in R \\ 0 & \quad \text{otherwise} \end{cases} \tag{5.18}$$

$$used_q = \begin{cases} 1 & \text{if } \sum_{c \in C} y_{cq} \geq 1, \forall q \in Q \\ 0 & \quad \text{otherwise} \end{cases} \tag{5.19}$$

$$\Delta H = H_C \cdot \sum_{r \in R} used_r \tag{5.20}$$

$$\Delta W = W_C \cdot \sum_{q \in Q} used_q \tag{5.21}$$

$$AREA_{PNoC} = (H + \Delta H) \cdot (W + \Delta W) - H \cdot W \tag{5.22}$$

The power component of the objective is determined by the following constraints. We convert P_{loss} (dbM) to P_{laser} (mW) using a piecewise-linear approximation and the laser source WPE to obtain the electrical input power required to operate the laser. P_{tuning} is the thermal tuning power needed to keep the ring groups at a similar temperature. $P_{electrical}$ is the power required for EOE conversion. $P_{modulator}$, $P_{detector}$, P_{SERDES} and $P_{cluster}$ values are obtained from code extracted from DSENT (Sun et al., 2012).

$$P_{PNoC} = P_{laser} + P_{tuning} + P_{electrical} \tag{5.23}$$

$$P_{loss} = P_{prop} \sum_{n\in N} \sum_{a_{h/vrq}\in A} d^n_{h/vrq} \dot{e}^n_{h/vrq} + P_{cross} \cdot n_{cross} + P_{bend} \cdot n_{bend} + P_{constant} \tag{5.24}$$

$$P_{electrical} = P_{modulator} + P_{detector} + P_{SERDES} + P_{cluster} \tag{5.25}$$

Thermal tuning power is proportional to the difference between the thermal impact of a given router group (θ_c) and the maximum thermal impact (θ_{max}) over all router groups. Equation (5.26) calculates the thermal impact of each router group using the power profile of the system, with each tile's power level contributing a thermal weight $w_{r'q'}(r,q)$ to the router group at (r,q). Given that θ_c is a product of two binary variables, we must linearize it using the following technique.

$$\theta_c = \sum_{r\in R, q\in Q, f\in 0,1, r'\in R, q'\in Q} \gamma^c_{frq} \cdot w_{r'q'}(r,q) \cdot p_{r'q'}, \quad \forall c \in C \tag{5.26}$$

$$p_{r'q'} = \sum_{c\in C} o_{cr'q'} \cdot p_c, \quad \forall r' \in R, q' \in Q, p_c\ is\ fixed \tag{5.27}$$

$$P_{tuning} = P^0_{tuning} \sum_{c\in C} (\theta_{max} - \theta_c) \tag{5.28}$$

Accounting for Optical Bends and Crossings

We include the bends and crossings in routing solutions. The 0-1 indicator SV^n_{rq} (respectively, SH^n_{rq}) captures the existence of a straight vertical (respectively, horizontal) route through vertex (r,q) for net n. We derive SV^n_{rq} and SH^n_{rq} from $e^n_{h/vrq}$.

$$SV^n_{rq} \le e^n_{vr-1q};\ SV^n_{rq} \le e^n_{vrq};\ SV^n_{rq} \ge e^n_{vr-1q} + e^n_{vrq} - 1,\ \forall n \in N, r \in R, q \in Q \tag{5.29}$$

$$SH^n_{rq} \leq e^n_{hrq-1};\ SH^n_{rq} \leq e^n_{hrq};\ SH^n_{rq} \geq e^n_{hrq-1} + e^n_{hrq} - 1,\ \forall n \in N, r \in R, q \in Q \tag{5.30}$$

To account properly for all bends in the routing solution, we define a 0-1 indicator B_{rq} to capture the existence of a bend at vertex (r, q), and $\hat{B}_{rq}$ as a binary indicator for a vertex used with no bends. SH_{rq}, SV_{rq}, and v_{rq} respectively indicate straight vertical routes, straight horizontal routes, and vertex used at each (r, q) coordinate, for the superposition of all routed nets $n \in N$. Finally, we add the number of bends across all (r, q) to obtain the total number of bends in the routing solution.

$$\begin{aligned} &\hat{B}_{rq} \leq SH_{rq} + SV_{rq};\ \hat{B}_{rq} \geq SH_{rq};\ \hat{B}_{rq} \geq SV_{rq}; \\ &\qquad B_{rq} + \hat{B}_{rq} - v_{rq} + \sum_{c \in C, f \in 0,1} \gamma^c_{frq} = 0,\ \forall r \in R, q \in Q \end{aligned} \tag{5.31}$$

$$n_{bend} = \sum_{q \in Q, r \in R} B_{rq} \tag{5.32}$$

We also include all straight-straight crossings in our power loss equation, using the same variables. The 0-1 indicator variable CR_{rq} captures the existence of a crossing at vertex (r, q), enabling us to obtain the total number of crossings across all (r, q).

$$\begin{aligned} CR_{rq} \geq SH_{rq} + SV_{rq} - 1;\ CR_{rq} \leq SH_{rq};\ CR_{rq} \leq SV_{rq} \\ \forall r \in R, q \in Q \end{aligned} \tag{5.33}$$

$$n_{cross} = \sum_{q \in Q, r \in R} CR_{rq} \tag{5.34}$$

MILP Instance Complexity and Scalability

Using the notation and from the formulation given above, the complexity of an instance of our MILP is as follows.

- The number of variables: $8NRQ + 3CRQ + 4RQ + C + CR^2Q^2$.

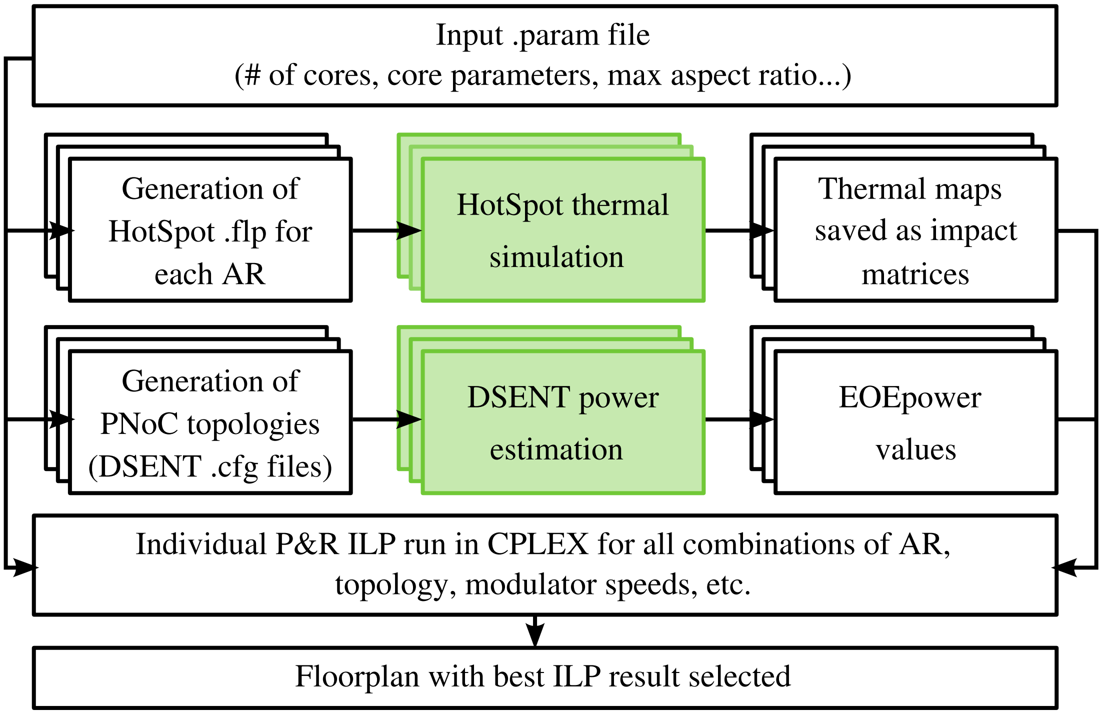

Input .param file
(# of cores, core parameters, max aspect ratio...)
Generation of HotSpot .flp for each AR
HotSpot thermal simulation
Thermal maps saved as impact matrices
Generation of PNoC topologies (DSENT .cfg files)
DSENT power estimation
EOEpower values
Individual P&R ILP run in CPLEX for all combinations of AR, topology, modulator speeds, etc.
Floorplan with best ILP result selected

Figure 5·20: The floorplan optimization flow.

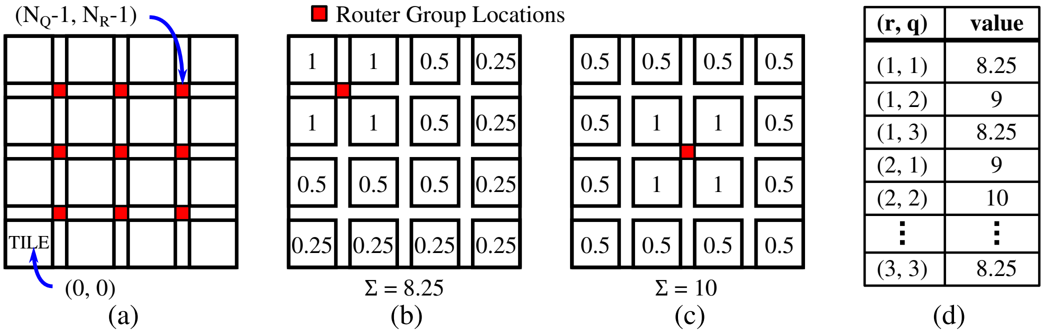
- The number of constraints: $3CR^2Q^2 + NRQ + 14RQ + 5C + 1$.

For a typical instance that we study in the experiments reported below, $C = 8$, $R = 8$, $Q = 8$ and $N = 7$, implying 38152 variables and 99689 constraints. Both the number of variables and the number constraints have terms that scale (i) linearly with the number of router groups, and (ii) quadratically with the number of tiles (RQ). If we assume that the number of cores per tile is fixed, then these parameters respectively translate to (i) the number of cores, and (ii) the size of the chip. For instances of this complexity, runtimes of ILOG CPLEX v12.5.1 (CPLEX, 2012) range from 10 seconds to several minutes on a 2.8 GHz Xeon server.

Optimization Flow

The key details of our optimization flow and setup are as follows.

(1) Our floorplan optimizer takes as input a *.param file* with the following contents:



(r, q)	value
(1, 1)	8.25
(1, 2)	9
(1, 3)	8.25
(2, 1)	9
(2, 2)	10
⋮	⋮
(3, 3)	8.25

Figure 5·21: Core impact matrix generation: (a) illustrative floorplan with 16 tiles (64 cores) and nine potential router group positions; (b) sample core impact calculation for router group (1,3); (c) sample core impact calculation for router group (2,2); (d) a 1x9 core impact array generated for the floorplan.

(i) CoreParams (N_{cores}, W_{core}, H_{core}, Core Power); (ii) AspectRatio (AR_{min}, AR_{max}); and (iii) OpticalParams (loss mechanisms, waveguide dimensions and spacing, ring dimensions and spacing, and photodetector sensitivity).

(2) Since it is not practical to run HotSpot inside a high-dimensional optimization, and MILP approach is fundamentally incompatible with running a thermal simulation "in the loop", we work around this issue by pre-characterizing a *core impact matrix* that captures the steady-state temperature impact of each running core on each possible router group location. The core impact matrix contains the thermal impact in K/W due to a 1 W core at each core location. We assume a linear superposition of core impacts due to all cores to calculate a final temperature at each vertex. We compare the temperature profile based on superposition with the data from HotSpot (with all the cores active simultaneously) and confirm less than 3% error.

(3) We extract code from DSENT (Sun et al., 2012) distribution to calculate the EOE power (modulator, detector, SERDES) and the electrical power for the NoC within the clusters. For the link bandwidths considered in our analysis (Table 5.7), we leverage DSENT's capability to perform datapath power optimization by balancing insertion loss and extinction ratio with modulator/receiver and laser power.

Table 5.7: Experimental configurations studied.

#cores	Clos size	(chip AR, cluster AR)	optical datarate (Gbps)	#waveguides
64	8-ary (1 core/tile)	(1:1,1:2), (1:4,1:2)	8	8,16,32,64,128
			4	16,32,64,128
			2	32,64,128
128	8-ary (2 core/tile)	(1:2,1:1), (1:2,1:4)	8	8,16,32,64,128
			4	16,32,64,128
			2	32,64,128
256	8-ary (4 core/tile)	(1:1,1:2), (1:1,1:8)	8	8,16,32,64,128
			4	16,32,64,128
			2	32,64,128
	16-ary (1 core/tile)	(1:1,1:1), (1:1,1:4), (1:4,1:1), (1:4,1:4)	8	32,64,128
			4	64,128
			2	128

5.5.2 Experimental Results and Discussion

Simulation Infrastructure

To test our optimization model, we use the same core architecture and technology assumption for many-core systems as the one in Section 5.1.1. We use HotSpot's default configuration, but scale the heat spreader and heat sink lengths to be 2X and 4X the longest chip side length, respectively, for each floorplan. We also modify the configuration file in DSENT to match our experiments as follows: 22 *nm* technology; 1 GHz operating frequency; 2, 4, 8 *Gbps* link data rates for all the test cases; 3-stage 8-ary Clos or 3-stage 16-ary Clos topology according to different test cases; 64, 128, or 256 cores according to different test cases.

Design of Experiments

From above, our optimizer finds the optimal packing of clusters and routing of waveguides, based on given design inputs. To validate our optimization approach over a large design space, we use a set of configurations shown in Table 5.7. We consider WPE values of 5% and 15%, for current and future on-chip laser sources, respectively.

Workloads are intrinsically different from each other, which leads to potential

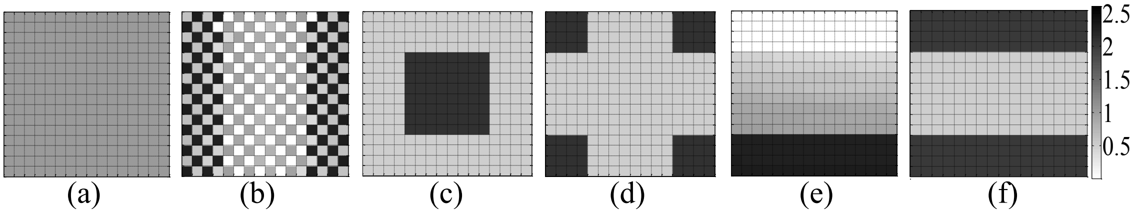
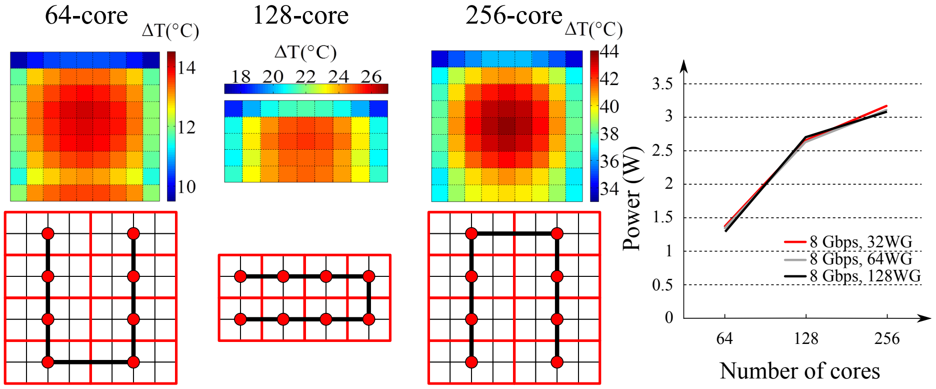


Figure 5·22: Six power profiles studied. Darker tiles indicate higher power cores.

Table 5.8: Losses in PNoCs.(Joshi et al., 2009)

Loss Mechanism	**Loss Contribution**
Splitter Through Loss	0.2 dB per split
Waveguide Propagation Loss	2 dB per cm
Waveguide Crossing Loss	0.05 dB per crossing
Ring Drop Loss	1.5 dB per wavelength per ring
Ring Insertion Loss	0.1 dB per wavelength per ring
Ring Through Loss	0.01 dB per wavelength per ring
Photodetector Loss	0.1 dB per photodetector
Merge Loss	5 dB per merge

variations in their power profiles. Especially in a many-core system, it is common to have multi-program workloads and thus imbalanced power profiles (Lu et al., 2015; Coskun et al., 2008). Also, the emergence of heterogeneous systems exacerbates the imbalance within the power profiles. Thus, optimizing for known imbalances in power profiles may work as a viable goal for many real-life systems. Our experiments consider the power profiles in Fig. 5·22(a)-(f). We include these power profiles in our design of experiments to demonstrate that the optimal floorplan is sensitive to the power profile, and that designers can potentially determine the floorplan based on a power profile of a use case or combination of use cases (average, weighted-average, or worst-case). We assume the optical loss coefficients listed in Table 5.8.



(a) (b)

Figure 5·23: Accumulated thermal weight profile and optimal floorplan vs. N_{cores}.

Results and Discussion

In all cases that we consider, the logical topology is a chain from router group $c = 0$ to router group $c = |C|$, with $|N| = |C| - 1$ nets. Figure 5·23 shows how the accumulated thermal weight profile and the optimal floorplan vary with change in N_{cores} for a given NoC topology, optical data rate and number of waveguides. We see that although waveguide lengths increase with N_{cores}, the required thermal tuning power tends to flatten out in larger chips due to more symmetry.

Figure 5·24 shows how the thermal weight profile and floorplan vary with the aspect ratio (AR) of the chip. In general, a skewed chip AR leads to a larger periphery, creating more asymmetry in the thermal weight profile as shown in Fig. 5·24(a), but at the same time allowing for a shorter waveguide length.

Figure 5·25 shows the thermal weight profiles and floorplans for the different power profiles described in Fig. 5·22. We note that the PNoC power varies by nearly 1.7X across the different power profiles. We also note that the optimal floorplans vary when we change WPE from 5% to 15%. A poorer laser source efficiency tends to favor the U-shaped floorplan. In comparison to a baseline vertical U-shaped floorplan, the

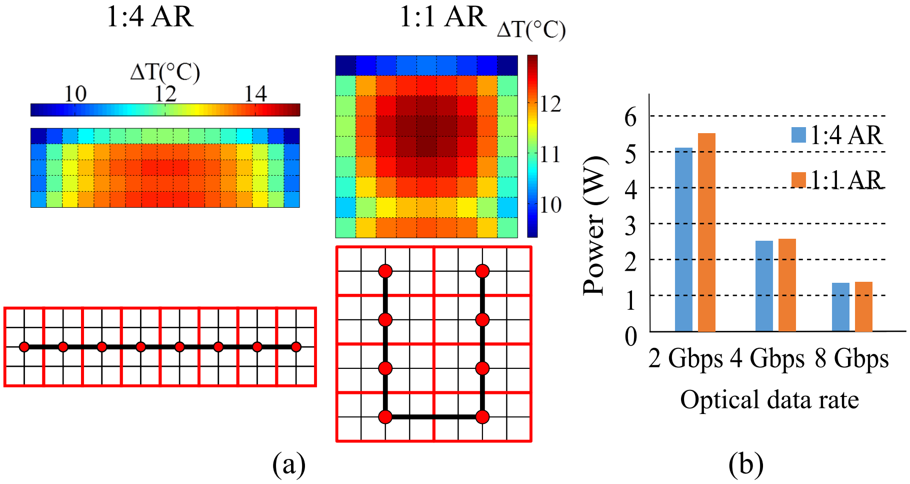


Figure 5·24: Accumulated thermal weight profile and optimal floorplan vs. AR.

floorplan in Fig. 5·25(e) saves up to 15% power under the heterogeneous power profile in Fig. 5·22(e).

From our experiments, we arrive at the following general conclusions.

- Both thermal tuning power and laser power are important sources of power in the PNoC. Sensitivity to thermal weight profiles is especially important for cases with better WPE.
- Larger chips present an economy of scale for the PNoC power due to the more symmetric thermal weight profiles of larger chips.
- Skewed chip aspect ratios provide larger periphery and create asymmetry in the thermal weight profiles.
- The maximum achievable optical data rate is always preferred.
- It is important to consider different power profiles during the design time, since heterogeneous power profiles expose inherent weaknesses to certain router group

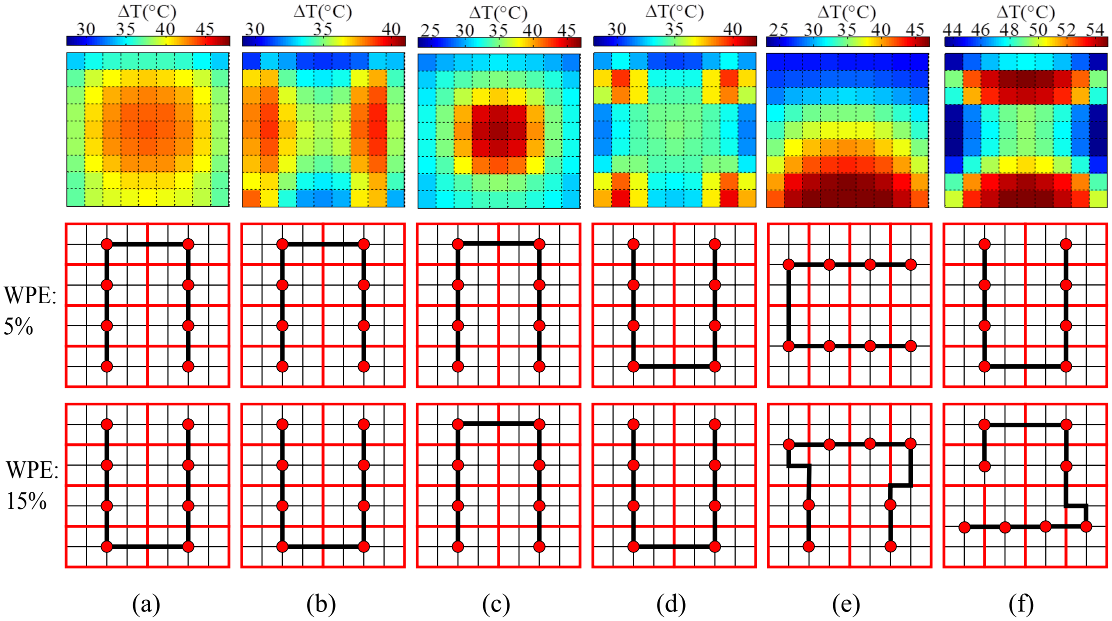


Figure 5·25: Accumulated thermal weight profile on the first row, and optimal floorplan with WPE of 5% and 15% on the second and third row respectively for power profiles (a) - (f) in Figure 5·22.

locations. Being thermally aware of runtime management issues during floorplan optimization provides a key cross-layer advantage to such an optimization. Weighting the power profiles based on duty cycle and benchmarking metrics could provide a way to choose an optimal floorplan that is aware of the heterogeneous runtime power profiles.

- Allowing for power weights associated with clusters provides an additional knob to investigate the best mix and locations for high- and low-performance clusters, and the impact of dark silicon considerations on the optimal floorplan.

5.6 Summary

PNoC is a promising replacement for ENoC in many-core systems. Adoption of PNoC relies on developing techniques that efficiently manage the thermal conditions (indirectly manage the optical frequencies) of the optical devices. In this chapter, we present our work on both runtime and design-time thermal management of many-core systems with PNoC.

Our runtime thermal management technique contains a workload allocation policy coupled with an adaptive tuning technique to align the optical frequencies of on-chip laser sources and ring resonators. Our proposed technique can jointly compensate for the difference in the optical frequency due to thermal and process variations, which in turn reduces the power consumed in localized thermal tuning. This is the first time resonant frequency matching of on-chip laser sources and ring resonators has been investigated, and their transient impact considered with dynamic workload allocation. We demonstrate that the proposed technique reduces the localized tuning power from 20 W on average to below 1 W and by up to 34.57 W, and it achieves similar benefits across systems with different PNoC logical topology and physical layout combinations.

Our design-time thermal/power management includes strategies for on-chip laser source placement and sharing and optimization for PNoC floorplanning. Our analysis shows that the choice of on-chip laser source placement and sharing changes with the choice of logical topology, physical layout and waveguide loss. Additionally, the optimal PNoC P & R solution is sensitive to thermal weight and power profiles, optical data rate, number of cores, and chip aspect ratios. Compared to thermally-agnostic solutions, our technique saves up to 15% PNoC power.

Chapter 6

Conclusions and Future Directions

6.1 Summary of Major Contributions

3D stacking is a promising integration technology that provides more diverse and flexible system designs for achieving dramatically improved performance and energy efficiency, especially in multi-/many-core era. It enables integrating a larger number of on-chip resources as well as stacking dies manufactured using different technologies (e.g., logic with DRAM or PNoC) into a single chip. However, the potential of 3D stacking in energy-efficient computing has not been fully exploited due to under-utilized on-chip resources and high chip temperatures and thermal gradients. The resource under-utilization leads to unnecessary performance drops, and the varying thermal conditions endanger the functionality of specific on-chip components (e.g., silicon-photonic devices) and increase the system power consumption.

By investigating application-dependent resource needs for on-chip resources and thermal conditions, this thesis has claimed that intelligent resource/thermal management is essential for unveiling the true efficiency potential of 3D stacking. The proposed techniques in this thesis address the urgent needs for low-power, application-dependent, and cross-layer resource and thermal management for 3D stacked systems.

In this thesis, we have identified workloads' sensitivity to cache and main memory architectures. Motivated by varying cache requirements of workloads, we have first proposed a cache resource pooling architecture (3D-CRP) for 3D homogeneous systems, where a core is able to pool cache resources from its adjacent layers. In

3D-CRP, we have designed a policy to allocate workloads within the system and then assign cache resources between adjacent layers dynamically based on the estimated performance and power consumption of each core. Using the proposed policy, 3D-CRP can improve system EDP by 18.8% on average compared to 3D systems with fixed cache size.

Second, we have applied the concept of resource pooling to on-chip scratchpad memory in 3D-MMC. We have developed a task-level resource pooling policy to avoid local resource contention and exploiting the additional resources on other layers offered by 3D stacking. For a memory intensive workload, we can achieve up to 48.9% performance improvement. In addition, motivated by the diversity on memory access patterns among the memory objects within each workload, we have designed a memory-object-level memory management approach – MOCA. MOCA profiles and classifies memory objects within each workload based on their sensitivity to memory bandwidth and access latency, and then allocates these memory objects to their best-fitting memory modules during runtime. Compared to application-level memory management, MOCA outperforms by 5.4% in performance and 19% in system ED^{2}P.

Since the NoC bandwidth requirement increases along with the number of on-chip cores, PNoC is a promising replacement for ENoC in many-core systems. However, PNoC comes with high power overhead due to process/thermal variations and high laser source power consumption. This thesis contributes to runtime workload allocation policies and design-time techniques to counter the thermal challenges in many-core systems with PNoC. We have provided a detailed analysis on silicon-photonic devices' sensitivity to process and thermal variations and performance-power-thermal tradeoff of on-chip laser sources. Motivated by the thermal requirements of silicon-photonic devices, we have proposed a workload allocation policy, RingAware, to balance the temperatures among these devices. We have enhanced RingAware with the awareness of both process and thermal variations and proposed FreqAlign, a workload allocation policy aiming at balancing the optical frequencies of on-chip silicon-photonic devices. We have designed Adaptive Frequency Tuning (AFT) technique to operate in conjunction with FreqAlign so as to lower the localized thermal tuning power for many-core systems with PNoC. Combining FreqAlign and AFT, we are able to reduce the thermal tuning power from 20 W on average to below 1 W.

Design-time solutions such as P & R of silicon-photonic links affect PNoC power substantially. At the design level, we have explored silicon-photonic devices floorplanning techniques to reduce the PNoC power consumption. We have proposed placement and sharing schemes of on-chip laser sources to reduce total laser source power consumption. We have also designed a MILP-based PNoC floorplan optimizer to generate optimized P & R solutions for silicon-photonic devices under given power profiles. The proposed optimizer saves up to 15% PNoC power compared to thermally-agnostic floorplanning solutions.

In summary, the proposed techniques in this thesis significantly improve the energy efficiency of 3D multi-/many-core systems using application-dependent, cross-layer resource and thermal management. Based on our results, we believe that cumulatively, proposed methods in this thesis for cache, memory, NoC, and overall system management can improve system energy efficiency by several times, compared to the state-of-the-art. The investigations conducted in this thesis provide insights and points to open challenges for future research as well.

6.2 Future Research Directions

6.2.1 Heterogeneous Memory Architecture in 3D Systems

We have demonstrated the benefits of heterogeneous memory systems on performance and energy efficiency using MOCA, however, there are many interesting open research challenges. First, integrating heterogeneous memory architecture into 3D stacking architecture involves several design-time decisions. For example, with the limited area, the question of what type of memory to stack on top of a logic die arises. There are several choices: (1) stacking memory modules with low access latency or high bandwidth, and (2) stacking memory modules with low power consumption. Option (1) can provide even better memory access time due to the shortened path between memory and processor, but the high power consumption of such memory modules may increase the on-chip temperature in return. Option (2) helps with the on-chip thermal conditions but does not provide benefits for memory-intensive workloads. There needs to be a thorough and detailed investigation regarding the impact of different options on the performance and energy efficiency to make the best choice.

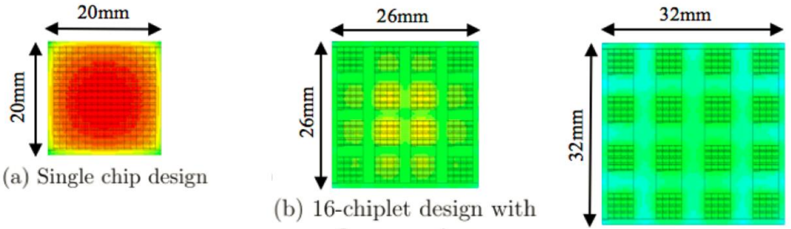
Second, the runtime data placement in 3D systems with stacked DRAM also plays an important role in performance and energy efficiency. For example, for a 3D multi-core system with stacked memory, if the data placement is centralized to a single DRAM bank, the access rate of this DRAM bank increases, which in turn raises its power consumption. As a result, the temperatures of this DRAM bank and the core (or other logic component) beneath it also increase, which might hit the system temperature threshold, cause throttling, and decrease the system performance (Meng and Coskun, 2012). If a 3D system has both on-chip memory and off-chip memory, there is more flexibility in data placement and on-chip thermal management, which could potentially provide better performance with lower on-chip temperature.

Another important challenge is to integrate cache heterogeneity together with main memory heterogeneity. We have showed that cache heterogeneity and memory heterogeneity can both bring benefits to performance and energy efficiency, however, there could be even more improvement if they are combined and jointly configured when running a workload.

6.2.2 Thermal/Power Management for PNoC with On-chip Laser Sources and 2.5D Integration

So far, our investigation and work on many-core systems with PNoC are based on either monolithic integration (in a single layer) or 3D stacking. For a many-core system, the high power density and the resultant thermal violation may prevent the system from being fully utilized. 2.5D integration is a promising technology that enables integrating a heterogeneous set of chiplets onto a silicon interposer, which alleviate the thermal issues. 2.5D integration provides extra wiring resources in the interposer, which can be leveraged to provide higher bandwidth connectivity among the chiplets and improve performance. This integration technology also adds extra spacing among the chiplets, which can provide more thermal headroom and alleviate the on-chip thermal challenges. This allows for more aggressive on-chip resource utilization and claims the potential performance from dark silicon. However, there is an urgent need for techniques to intelligently organize the chiplets and place them on the interposer for temperature advantages at the lowest cost.

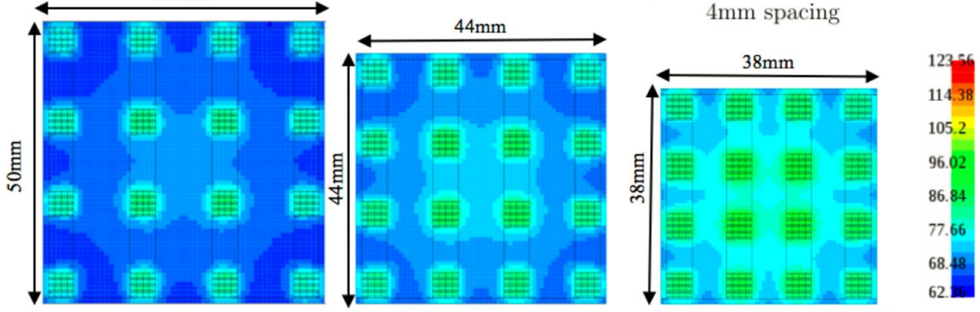
We have investigated the impact of chiplets organization on the on-chip thermal conditions. We partition the 256-core system shown in Fig. 5·1 into 16 chiplets, organize these chiplets in a 4×4 matrix on an interposer, and adjust the spacing among the chiplets to see the corresponding impact on chip thermal map. Figure 6·1 shows the thermal map of the tested system for an example benchmark (cholesky) running with 256 threads at 1 GHz. In single chip case (without using 2.5D integration), the peak temperature is 124 ^oC, which is not feasible in real systems due to the violation of the typical temperature threshold (85 ^oC). After applying 2.5D integration, as we increase the spacing between adjacent chiplets from 2 mm to 10 mm, the peak on-chip temperature decreases from 105 ^oC to 85 ^oC, where the system can achieve maximum possible performance.



(a) Single chip design

(b) 16-chiplet design with 2mm spacing

(c) 16-chiplet design with 4mm spacing



(f) 16-chiplet design with 10mm spacing

(e) 16-chiplet design with 8mm spacing

(d) 16-chiplet design with 6mm spacing

Figure 6·1: Thermal maps of single chip case and 16-chiplet case with different spacings.

One other benefit of 2.5D integration is the ability of integrating heterogeneous chiplets on a single interposer. For example, laser sources can be integrated into a single chiplet and connected to the other logic chiplets through the interposer. There could also be specialized chiplets for caches or memories. In such cases, the floorplanning of chiplets becomes extremely important due to its impact on chip power profiles. A good P&R solution could help reduce the PNoC power consumption greatly and provide more thermal headrooms for potential performance improvement.

References

Abellan, J. L., Coskun, A. K., Gu, A., Jin, W., Joshi, A., Kahng, A. B., Klamkin, J., Morales, C., Recchio, J., Srinivas, V., and Zhang, T. (2016). Adaptive tuning of photonic devices in a photonic NoC through dynamic workload allocation. *To appear in IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems.*

Agarwal, N., Nellans, D., Stephenson, M., O'Connor, M., and Keckler, S. W. (2015). Page placement strategies for GPUs within heterogeneous memory systems. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 607–618.

Albonesi, D. H. (1999). Selective cache ways: On-demand cache resource allocation. In *Proceedings of International Symposium on Microarchitecture*, pages 248–259.

Baehr-Jones, T., Ding, R., Ayazi, A., Pinguet, T., Streshinsky, M., Harris, N., Li, J., He, L., Gould, M., Zhang, Y., Lim, A., Liow, T.-Y., Teo, S. H.-G., Lo, G.-Q., and Hochberg, M. (2012). A 25 Gb/s silicon photonics platform. *arXiv preprint arXiv:1203.0767.*

Batten, C., Joshi, A., Stojanovć, V., and Asanović, K. (2013). Designing chip-level nanophotonic interconnection networks. In *Integrated Optical Interconnect Architectures for Embedded Systems*, pages 81–135. Springer New York.

Beanato, G., Giovannini, P., Cevrero, A., Athanasopoulos, P., Zervas, M., Temiz, Y., and Leblebici, Y. (2012). Design and testing strategies for modular 3-D-multiprocessor systems using die-level through silicon via technology. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2(2):295 –306.

Bienia, C., Kumar, S., Singh, J. P., and Li, K. (2008). The PARSEC benchmark suite: Characterization and Architectural Implications. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, pages 72–81.

Binkert, N., Beckmann, B., Black, G., Reinhardt, S. K., Saidi, A., Basu, A., Hestness, J., Hower, D. R., Krishna, T., Sardashti, S., Sen, R., Sewell, K., Shoaib, M., Vaish, N., Hill, M. D., and Wood, D. A. (2011). The gem5 simulator. *ACM SIGARCH Computer Architecture News*, 39(2):1–7.

Black, B., Annavaram, M., Brekelbaum, N., DeVale, J., Jiang, L., Loh, G. H., McCauley, D., Morrow, P., Nelson, D. W., Pantuso, D., Reed, P., Rupley, J., Shankar, S., Shen, J., and Webb, C. (2006). Die stacking (3D) microarchitecture. In *Proceedings of International Symposium on Microarchitecture*, pages 469–479.

Bogdan, P., Marculescu, R., Jain, S., and Gavila, R. (2012). An optimal control approach to power management for multi-voltage and frequency islands multiprocessor platforms under highly variable workloads. In *Proceedings of IEEE/ACM International Symposium on Networks on Chip*, pages 35–42.

Campbell, D., Bader, D., Brandt, S., Cook, D., Gokhale, M., Hornung, R., Keasler, J., LeBlanc, P., Marin, G., Mulvaney, B., Richards, M., Vetter, J., and I.Walker (2012). Ubiquitous High Performance Computing: Challenge problems specification. Technical Report HR0011-10-C-0145, Georgia Institute of Technology.

Carlson, T., Heirman, W., and Eeckhout, L. (2011). Sniper: Exploring the level of abstraction for scalable and accurate parallel multi-core simulations. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–12.

Carlson, T., Heirman, W., Eeckhout, L., and Hur, I. (2012). Benchmark native execution. `http://snipersim.org/documents/gainestown-native.html`.

Chatterjee, N., Shevgoor, M., Balasubramonian, R., Davis, A., Fang, Z., Illikkal, R., and Iyer, R. (2012). Leveraging heterogeneity in DRAM main memories to accelerate critical word access. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, pages 13–24.

Chen, C., Zhang, T., Contu, P., Klamkin, J., Coskun, A. K., and Joshi, A. (2014). Sharing and placement of on-chip laser sources in silicon-photonic nocs. In *Proceedings of International Symposium on Networks-on-Chip*, pages 88–95.

Chen, X., Mohamed, M., Li, Z., Shang, L., and Mickelson, A. R. (2013). Process variation in silicon photonic devices. *Applied optics*, 52(31):7638–7647.

Chiou, D., Devadas, S., Rudolph, L., and Ang, B. S. (2000). Dynamic cache partitioning via columnization. In *TechReport, Massachusetts Institute of Technology.*

Cianchetti, M. J., Kerekes, J. C., and Albonesi, D. H. (2009). Phastlane: a rapid transit optical routing network. *ACM SIGARCH Computer Architecture News*, 37(3):441–450.

Coldren, L. A., Corzine, S. W., and Mashanovitch, M. L. (2012). *Diode lasers and photonic integrated circuits*, volume 218. John Wiley & Sons.

Constantinou, T., Sazeides, Y., Michaud, P., Fetis, D., and Seznec, A. (2005). Performance implications of single thread migration on a chip multi-core. *ACM SIGARCH Computer Architecture News*, 33(4):80–91.

Conway, P., Kalyanasundharam, N., Donley, G., Lepak, K., and Hughes, B. (2009). Blade computing with the AMD opteron processor. `http://www.hotchips.org/wp-content/uploads/hc_archives/hc21/2_mon/HC21.24.100.ServerSystemsI-Epub/HC21.24.110.Conway-AMD-Magny-Cours.pdf`.

Coskun, A., Gu, A., Jin, W., Joshi, A., Kahng, A., Klamkin, J., Ma, Y., Recchio, J., Srinivas, V., and Zhang, T. (2016). Cross-layer floorplan optimization for silicon photonic NoCs in many-core systems. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 1309–1314.

Coskun, A. K., Ayala, J. L., Atienza, D., Rosing, T. S., and Leblebici, Y. (2009a). Dynamic thermal management in 3D multicore architectures. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 1410–1415.

Coskun, A. K., Rosing, T. v., Whisnant, K. A., and Gross, K. C. (2008). Static and dynamic temperature-aware scheduling for multiprocessor SoCs. *IEEE Transactions on Very Large Scale Integration Systems*, 16(9):1127–1140.

Coskun, A. K., Strong, R., Tullsen, D. M., and Rosing, T. S. (2009b). Evaluating the impact of job scheduling and power management on processor lifetime for chip multiprocessors. In *SIGMETRICS/Performance – Joint Conference on Measurement and Modeling of Computer Systems*, pages 169–180.

CPLEX (2012). IBM ILOG CPLEX. www.ilog.com/products/cplex/.

Das, R., Ausavarungnirun, R., Mutlu, O., Kumar, A., and Azimi, M. (2012). Application-to-core mapping policies to reduce memory interference in multi-core systems. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, pages 455–456.

Demir, Y. and Hardavellas, N. (2015). Parka: Thermally insulated nanophotonic interconnects. In *Proceedings of the International Symposium on Networks-on-Chip*, pages 1:1–1:8.

DeRose, C., Watts, M., Trotter, D., Luck, D., Nielson, G., and Young, R. (2010). Silicon microring modulator with integrated heater and temperature sensor for thermal control. In *Proceedings of Conference on Lasers and Electro-Optics and Quantum Electronics and Laser Science*, pages 1–2.

Dighe, S., Vangal, S. R., Aseron, P., Kumar, S., Jacob, T., Bowman, K. A., Howard, J., Tschanz, J., Erraguntla, V., Borkar, N., De, V. K., and Borkar, S. (2011). Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor. *IEEE Journal of Solid-State Circuits*, 46(1):184–193.

Ding, D., Yu, B., and Pan, D. (2012). Glow: A global router for low-power thermal-reliable interconnect synthesis using photonic wavelength multiplexing. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 621–626.

Djordjevic, S. S., Shang, K., Guan, B., Cheung, S. T. S., Liao, L., Basak, J., Liu, H.-F., and Yoo, S. J. B. (2013). CMOS-compatible, athermal silicon ring modulators clad with titanium dioxide. *Optics Express*, 21(12):13958–13968.

Dong, P., Shafiiha, R., Liao, S., Liang, H., Feng, N.-N., Feng, D., Li, G., Zheng, X., Krishnamoorthy, A. V., and Asghari, M. (2010a). Wavelength-tunable silicon microring modulator. *Optics Express*, 18(11):10941–10946.

Dong, X., Xie, Y., Muralimanohar, N., and Jouppi, N. P. (2010b). Simple but effective heterogeneous main memory with on-chip memory controller support. In *Proceedings of ACM/IEEE International Conference for High Performance Computing, Networking, Storage and Analysis*, pages 1–11.

Dulloor, S. R., Roy, A., Zhao, Z., Sundaram, N., Satish, N., Sankaran, R., Jackson, J., and Schwan, K. (2016). Data tiering in heterogeneous memory systems. In *Proceedings of European Conference on Computer Systems*, pages 15:1–15:16.

Fick, D., Dreslinski, R. G., Giridhar, B., Kim, G., Seo, S., Fojtik, M., Satpathy, S., Lee, Y., Kim, D., Liu, N., Wieckowski, M., Chen, G., Sylvester, D., Blaauw, D., and Mudge, T. (2012). Centip3De: A 3930DMIPS/W configurable near-threshold 3D stacked system with 64 ARM cortex-M3 cores. In *Proceedings of IEEE International Solid-State Circuits Conference*, pages 190–192.

Garrou, P., Bower, C., and Ramm, P. (2008). *Handbook of 3D Integration: Technology and Applications of 3D Integrated Circuits*. Number 2. John Wiley & Sons.

Georgas, M., Moss, B., Sun, C., Shainline, J., Orcutt, J., Wade, M., Chen, Y.-H., Nammari, K., Leu, J., Srinivasan, A., Ram, R., Popovic, M., and Stojanovic, V. (2014). A monolithically-integrated optical transmitter and receiver in a zero-change 45nm SOI process. In *Proceedings of Symposium on VLSI Circuits Digest of Technical Papers*, pages 1–2.

Gomaa, M., Powell, M. D., and Vijaykumar, T. N. (2004). Heat-and-run: Leveraging SMT and CMP to manage power density through the operating system. In

Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems, pages 260–270.

Guha, B., Kyotoku, B. B. C., and Lipson, M. (2010). CMOS-compatible athermal silicon microring resonators. *Optics Express*, 18(4):3487–3493.

Hameed, F., Faruque, M. A., and Henkel, J. (2011). Dynamic thermal management in 3D multi-core architecture through run-time adaptation. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 1–6.

Hamerly, G., Perelman, E., Lau, J., and Calder, B. (2005). Simpoint 3.0: Faster and more flexible program phase analysis. *Journal of Instruction Level Parallelism*, 7(4):1–28.

Heck, M. and Bowers, J. (2014). Energy efficient and energy proportional optical interconnects for multi-core processors: Driving the need for on-chip sources. *IEEE Journal of Selected Topics in Quantum Electronics*, 20(4):1–12.

Homayoun, H., Kontorinis, V., Shayan, A., Lin, T.-W., and Tullsen, D. M. (2012). Dynamically heterogeneous cores through 3D resource pooling. In *Proceedings of International Symposium on High Performance Computer Architecture*, pages 1–12.

Howard, J., Dighe, S., Vangal, S., Ruhl, G., Borkar, N., Jain, S., Erraguntla, V., Konow, M., Riepen, M., Gries, M., Droege, G., Lund-Larsen, T., Steibl, S., Borkar, S., De, V., and Van Der Wijngaart, R. (2011). A 48-core IA-32 processor in 45 nm CMOS using on-die message-passing and DVFS for performance and power scaling. *IEEE Journal of Solid-State Circuits*, 46(1):173–183.

Hu, S., Corzine, S., Law, K.-K., Young, D., Gossard, A., Coldren, L., and Merz, J. (1994a). Lateral carrier diffusion and surface recombination in InGaAs/AlGaAs quantum-well ridge waveguide lasers. *Journal of Applied Physics*, 76(8):4479–4487.

Hu, S., Young, D., Corzine, S., Gossard, A., and Coldren, L. (1994b). High-efficiency and low-threshold InGaAs/AlGaAs quantum-well lasers. *Journal of Applied Physics*, 76(6):3932–3934.

Ipek, E., Kirman, M., Kirman, N., and Martinez, J. F. (2007). Core fusion: Accommodating software diversity in chip multiprocessors. In *Proceedings of International Symposium on Computer Architecture*, pages 186–197.

Jafari, R., Ghasemzadeh, H., Dabiri, F., Nahapetian, A., and Sarrafzadeh, M. (2009). An efficient placement and routing technique for fault-tolerant distributed embedded computing. *ACM Transactions on Embedded Computing Systems*, 8(4):28.

Joshi, A., Batten, C., Kwon, Y., Beamer, S., Shamim, I., Asanovic, K., and Stojanovic, V. (2009). Silicon-photonic Clos networks for global on-chip communication. In *Proceedings of International Symposium on Networks-on-Chip*, pages 124–133.

Jung, J., Kang, K., and Kyung, C.-M. (2011). Design and management of 3D-stacked NUCA cache for chip multiprocessors. In *Proceedings of ACM/IEEE Great Lakes Symposium on VLSI*, pages 91–96.

Kamruzzaman, M., Swanson, S., and Tullsen, D. M. (2011). Inter-core prefetching for multicore processors using migrating helper threads. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 393–404.

Kim, D. H., Athikulwongse, K., Healy, M. B., Hossain, M. M., Jung, M., Khorosh, I., Kumar, G., Lee, Y.-J., Lewis, D. L., Lin, T.-W., Liu, C., Panth, S., Pathak, M., Ouellette, B., Ren, M., Shen, G., Song, T., Woo, D. H., Zhao, X., Kim, J., Choi, H., Loh, G. H., Lee, H.-H. S., and Lim, S. K. (2012). 3D-MAPS: 3D massively parallel processor with stacked memory. In *Proceedings of IEEE International Solid-State Circuits Conference*, pages 188–190.

Kimoto, T., Shinagawa, T., Mukaihara, T., Nasu, H., Tamura, S., Numura, T., and Kasukawa, A. (2003). Highly reliable 40-mW 25-GHz× 20-ch thermally tunable DFB laser module, integrated with wavelength monitor. *Furutaka Review*, 24:1–5.

Kirman, N., Kirman, M., Dokania, R., Martinez, J., Apsel, A., Watkins, M., and Albonesi, D. (2006). Leveraging optical technology in future bus-based chip multiprocessors. In *Proceedings of IEEE/ACM International Symposium on Microarchitecture*, pages 492–503.

Kumar, R., Zyuban, V., and Tullsen, D. M. (2005). Interconnections in multi-core architectures: Understanding mechanisms, overheads and scaling. In *Proceedings of International Symposium on Computer Architecture*, pages 408–419.

Larson, M., Bhardwaj, A., Xiong, W., Feng, Y., dong Huang, X., Petrov, K., Moewe, M., Ji, H., Semakov, A., Lv, C., Kutty, S., Patwardhan, A., Liu, N., Li, Z., Bao, Y., Shen, Z., Bajwa, S., Zhou, F., and Koh, P.-C. (2015). Narrow linewidth sampled-grating distributed Bragg reflector laser with enhanced side-mode suppression. In *Proceedings of Optical Fiber Communication Conference*, page M2D.1.

Lee, M., Gupta, V., and Schwan, K. (2013). Software-controlled transparent management of heterogeneous memory resources in virtualized systems. In *Proceedings of ACM SIGPLAN Workshop on Memory Systems Performance and Correctness*, pages 5:1–5:6.

Li, H., Fourmigue, A., Beux, S. L., Letartre, X., O'Connor, I., and Nicolescu, G. (2015a). Thermal aware design method for VCSEL-based on-chip optical interconnect. In *Proceedings of Design, Automation Test in Europe Conference Exhibition*, pages 1120–1125.

Li, S., Ahn, J.-H., Strong, R., Brockman, J., Tullsen, D., and Jouppi, N. (2009). McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures. In *Proceedings of International Symposium on Microarchitecture*, pages 469–480.

Li, Z., Qouneh, A., Joshi, M., Zhang, W., Fu, X., and Li, T. (2015b). Aurora: A cross-layer solution for thermally resilient photonic network-on-chip. *IEEE Transactions on Very Large Scale Integration Systems*, 23(1):170–183.

Loh, G. H. (2008). 3D-stacked memory architectures for multi-core processors. In *Proceedings of International Symposium on Computer Architecture*, pages 453–464.

Loh, G. H. (2009). Extending the effectiveness of 3D-stacked DRAM caches with an adaptive multi-queue policy. In *Proceedings of International Symposium on Microarchitecture*, pages 201–212.

Lourdudoss, S. (2012). Heteroepitaxy and selective area heteroepitaxy for silicon photonics. *Current Opinion in Solid State and Materials Science*, 16(2):91–99.

Lu, S. J., Tessier, R., and Burleson, W. (2015). Reinforcement learning for thermal-aware many-core task allocation. In *Proceedings of ACM Great Lakes Symposium on VLSI*, pages 379–384.

Luk, C.-K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V. J., and Hazelwood, K. (2005). Pin: Building customized program analysis tools with dynamic instrumentation. In *Proceedings of ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 190–200.

Manipatruni, S., Dokania, R. K., Schmidt, B., Sherwood-Droz, N., Poitras, C. B., Apsel, A. B., and Lipson, M. (2008). Wide temperature range operation of micrometer-scale silicon electro-optic modulators. *Optical Letter*, 33(19):2185–2187.

Meng, J. and Coskun, A. K. (2012). Analysis and runtime management of 3D systems with stacked DRAM for boosting energy efficiency. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 611–616.

Meng, J., Kawakami, K., and Coskun, A. (2012). Optimizing energy efficiency of 3-D multicore systems with stacked DRAM under power and thermal constraints. In *Proceedings of Design Automation Conference*, pages 648–655.

Meng, J., Zhang, T., and Coskun, A. K. (2013). Dynamic cache pooling for improving energy efficiency in 3D stacked multicore processors. In *Proceedings of IFIP/IEEE International Conference on Very Large Scale Integration*, pages 210–215.

Meswani, M. R., Blagodurov, S., Roberts, D., Slice, J., Ignatowski, M., and Loh, G. H. (2015). Heterogeneous memory architectures: A HW/SW approach for mixing die-stacked and off-package memories. In *Proceedings of International Symposium on High Performance Computer Architecture*, pages 126–136.

MICRON (2011). DDR3 SDRAM power calculator. https://www.micron.com/products/dram/ddr3-sdram.

MICRON (2013). LPDDR2 SDRAM power calculator. http://www.micron.com/products/dram/lpdram.

MICRON (2016). RLDRAM3 power calculator. http://www.micron.com/products/-dram/rldram-memory.

Mohamed, M., Li, Z., Chen, X., Shang, L., Mickelson, A., Vachharajani, M., and Sun, Y. (2010). Power-efficient variation-aware photonic on-chip network management. In *Proceedings of International Symposium on Low-Power Electronics and Design*, pages 31–36.

Moss, B., Sun, C., Georgas, M., Shainline, J., Orcutt, J., Leu, J., Wade, M., Chen, Y.-H., Nammari, K., Wang, X., Li, H., Ram, R., Popovic, M., and Stojanovic, V. (2013). A 1.23pJ/b 2.5Gb/s monolithically integrated optical carrier-injection ring modulator and all-digital driver circuit in commercial 45nm SOI. In *Proceedings of IEEE International Solid-State Circuits Conference*, pages 126–127.

Mutlu, O., Kim, H., and Patt, Y. N. (2006). Efficient runahead execution: Power-efficient memory latency tolerance. *IEEE Micro*, 26(1):10–20.

Nitta, C., Farrens, M., and Akella, V. (2011). Addressing system-level trimming issues in on-chip nanophotonic networks. In *Proceedings of International Symposium on High Performance Computer Architecture*, pages 122–131.

Noia, B. and Chakrabarty, K. (2014). *Design-for-Test and Test Optimization Techniques for TSV-based 3D Stacked ICs, 1st edition.* Springer International Publishing.

Oracle (2011). SPARC T4 processor data sheet. `http://www.oracle.com/us/products/servers-storage/servers/sparc-enterprise/t-series/sparc-t4-processor-ds-497205.pdf`.

Orcutt, J. S., Moss, B., Sun, C., Leu, J., Georgas, M., Shainline, J., Zgraggen, E., Li, H., Sun, J., Weaver, M., Urošević, S., Popović, M., Ram, R. J., and Stojanović, V. (2012). Open foundry platform for high-performance electronic-photonic integration. *Optical Express*, 20(11):12222–12232.

Owens, J. D., Dally, W. J., Ho, R., Jayasimha, D. J., Keckler, S. W., and Peh, L.-S. (2007). Research challenges for on-chip interconnection networks. *IEEE Micro*, 27(5):96–108.

Pan, Y., Kumar, P., Kim, J., Memik, G., Zhang, Y., and Choudhary, A. (2009). Firefly: Illuminating future Network-on-Chip with nanophotonics. In *Proceedings of International Symposium on Computer Architecture*, pages 429–440.

Pavlidis, V. and Friedman, E. (2007). 3-D topologies for Networks-on-Chip. *IEEE Transactions on Very Large Scale Integration Systems*, 15(10):1081–1090.

Pavlidis, V., Savidis, I., and Friedman, E. (2008). Clock distribution networks for 3-D integrated circuits. In *Proceedings of Custom Integrated Circuits Conference*, pages 651 –654.

Pavlovic, M., Puzovic, N., and Ramirez, A. (2013). Data placement in HPC architectures with heterogeneous off-chip memory. In *Proceedings of IEEE International Conference on Computer Design*, pages 193–200.

Peón-quirós, M., Bartzas, A., Mamagkakis, S., Catthoor, F., Mendías, J. M., and Soudris, D. (2015). Placement of linked dynamic data structures over heterogeneous memories in embedded systems. *ACM Transactions on Embedded Computer System*, 14(2):37:1–37:30.

Phadke, S. and Narayanasamy, S. (2011). MLP aware heterogeneous memory system. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 1–6.

Ponomarev, D., Kucuk, G., and Ghose, K. (2006). Dynamic resizing of superscalar datapath components for energy efficiency. *IEEE Transactions on Computers*, 55(2):199–213.

Qureshi, M. K. and Patt, Y. N. (2006). Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *Proceedings of International Symposium on Microarchitecture*, pages 423–432.

Rabaey, J. M., Chandrakasan, A., and Nikolic., B. (2003). *Digital Integrated Circuits: A Design Perspective, 2nd edition.*

Ramini, L., Bertozzi, D., and Carloni, L. (2012). Engineering a bandwidth-scalable optical layer for a 3D multi-core processor with awareness of layout constraints. In *Proceedings of International Symposium on Networks-on-Chip*, pages 185–192.

Rupp, K. (2015). 40 years of microprocessor trend data. `https://www.karlrupp.net/2015/06/40-years-of-microprocessor-trend-data/`.

Shacham, A., Bergman, K., and Carloni, L. P. (2007). On the design of a photonic Network-on-Chip. In *Proceedings of International Symposium on Networks-on-Chip*, pages 53–64.

Shen, D., Liu, X., and Lin, F. X. (2016). Characterizing emerging heterogeneous memory. In *Proceedings of ACM SIGPLAN International Symposium on Memory Management*, pages 13–23.

Skadron, K., Stan, M. R., Huang, W., Velusamy, S., Sankaranarayanan, K., and Tarjan, D. (2003). Temperature-aware microarchitecture. In *Proceedings of International Symposium on Computer Architecture*, pages 2–13.

Snavely, A. and Tullsen, D. M. (2000). Symbiotic jobscheduling for a simultaneous multithreaded processor. In *Proceedings of International Conference on Architectural Support for Programming Languages and Operating Systems*, pages 234–244.

Song, B., Contu, P., Stagarescu, C., Pinna, S., Abolghasem, P., Ristic, S., Bickel, N., Bowker, J., Behfar, A., and Klamkin, J. (2015). 3D integrated hybrid silicon laser. In *Proceedings of European Conference on Optical Communication*, pages 1–3.

Su, H., Liu, F., Devgan, A., Acar, E., and Nassif, S. (2003). Full chip leakage-estimation considering power supply and temperature variations. In *Proceedings of International Symposium on Low Power Electronics and Design*, pages 78–83.

Sun, C., Chen, C.-H. O., Kurian, G., Wei, L., Miller, J., Agarwal, A., Peh, L.-S., and Stojanovic, V. (2012). DSENT - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling. In *Proceedings of International Symposium on Network on Chip*, pages 201–210.

Sun, F., Cevrero, A., Athanasopoulos, P., and Leblebici, Y. (2010). Design and feasibility of multi-Gb/s quasi-serial vertical interconnects based on TSVs for 3D ICs. In *Proceedings of IEEE/IFIP International Conference on VLSI and System-on-Chip*, pages 149–154.

Sun, G., Dong, X., Xie, Y., Li, J., and Chen, Y. (2009). A novel architecture of the 3D stacked MRAM L2 cache for CMPs. In *Proceedings of International Symposium on High Performance Computer Architecture*, pages 239–249.

Thoziyoor, S., Muralimanohar, N., Ahn, J. H., and Jouppi, N. P. (2008). CACTI 5.1. Technical Report HPL-2008-20, HP Labs.

Tran, L., Kurdahi, F. J., Eltawil, A. M., and Homayoun, H. (2013). Heterogeneous memory management for 3D-DRAM and external DRAM with QoS. In *Proceedings of Asia and South Pacific Design Automation Conference*, pages 663–668.

Van Craeynest, K., Akram, S., Heirman, W., Jaleel, A., and Eeckhout, L. (2013). Fairness-aware scheduling on single-ISA heterogeneous multi-cores. In *Proceedings of International Conference on Parallel Architectures and Compilation Techniques*, pages 177–188.

Van Craeynest, K., Jaleel, A., Eeckhout, L., Narvaez, P., and Emer, J. (2012). Scheduling heterogeneous multi-cores through performance impact estimation (PIE). In *Proceedings of International Symposium on Computer Architecture*, pages 213–224.

Vantrease, D., Schreiber, R., Monchiero, M., McLaren, M., Jouppi, N., Fiorentino, M., Davis, A., Binkert, N., Beausoleil, R., and Ahn, J. (2008). Corona: System implications of emerging nanophotonic technology. In *Proceedings of International Symposium on Computer Architecture*, pages 153–164.

Varadarajan, K., Nandy, S., Sharda, V., Bharadwaj, A., Iyer, R., Makineni, S., and Newell, D. (2006). Molecular caches: A caching structure for dynamic creation of application-specific heterogeneous cache regions. In *Proceedings of International Symposium on Microarchitecture*, pages 433–442.

Wang, T., Liu, H., Lee, A., Pozzi, F., and Seeds, A. (2011). 1.3-μm InAs/GaAs Quantum-dot lasers monolithically grown on Si substrates. *Optics Express*, 19(12): 11381–11386.

Wong, H. (2012). A comparison of intel's 32nm and 22nm core i5 CPUs: Power, voltage, temperature, and frequency. http://blog.stuffedcow.net/2012/10/intel32nm-22nm-core-i5-comparison/.

Woo, S., Ohara, M., Torrie, E., Singh, J., and Gupta, A. (1995). The SPLASH-2 programs: characterization and methodological considerations. In *Proceedings of International Symposium on Computer Architecture*, pages 24–36.

Zhang, T., Abellán, J. L., Joshi, A., and Coskun, A. K. (2014). Thermal management of manycore systems with silicon-photonic networks. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 307:1–307:6.

Zhang, T., Aga, S., Narayanasamy, S., and Coskun, A. K. (2017). MOCA: memory object classification and allocation in heterogeneous memory systems. In *review.*

Zhang, T., Cevrero, A., Beanato, G., Athanasopoulos, P., Coskun, A. K., and Leblebici, Y. (2013). 3D-MMC: A modular 3D multi-core architecture with efficient resource pooling. In *Proceedings of Design, Automation and Test in Europe Conference and Exhibition*, pages 1241–1246.

Zhang, T., Meng, J., and Coskun, A. K. (2015). Dynamic cache pooling in 3D multicore processors. *ACM Journal on Emerging Technologies in Computing*, 12(2):14:1–14:21.

Zhao, X., Minz, J., and Lim, S.-K. (2011). Low-power and reliable clock network design for through-silicon via (TSV) based 3D ICs. *IEEE Transactions on Components, Packaging and Manufacturing Technology*, 1(2):247–259.

Zhou, X., Xu, Y., Du, Y., Zhang, Y., and Yang, J. (2008). Thermal management for 3D processors via task scheduling. In *Proceedings of International Conference on Parallel Processing*, pages 115–122.

Zhu, C., Gu, Z., Shang, L., Dick, R. P., and Joseph, R. (2008). Three-dimensional chip-multiprocessor run-time thermal management. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(8).

Zortman, W. A., Trotter, D. C., and Watts, M. R. (2010). Silicon photonics manufacturing. *Optics Express*, 18(23):23598–23607.

CURRICULUM VITAE

Tiansheng Zhang

Education

Ph.D., Boston University, 01/2017
Electrical and Computer Engineering Department
Advisor: Professor Ayse K. Coskun
Dissertation Title: "Resource and Thermal Management in 3D Stacked Multi-/Many-core Systems"

B.S., Harbin Institute of Technology, 07/2010
Department of Microelectronics

Professional Experience

Oracle Corporation, Santa Clara, CA, U.S., 06/2016 to 09/2016
Software Engineering Intern, *Supervisor:* Darrin Johnson
Compression algorithm improvement for SPARC M7 processors.

MediaTek Inc., Woburn, MA, U.S., 06/2015 to 09/2015
Software Engineering Intern, *Supervisor:* Dr. Yuan Lin & Dr. Henry Cox
Cross-architecture performance prediction of scalar code on embedded systems.

Research Experience

Performance & Energy-Aware Computing Lab, Boston University, Boston, MA, U.S., 09/2011 to 01/2017
Research Assistant, *Advisor:* Prof. Ayse K. Coskun
Power and performance simulation and optimization of 3D-stacked multi-/many-core systems; Thermally-aware run-time management of manycore systems with silicon-photonic networks-on-chip.

Microelectronic Systems Lab, EPFL, Switzerland, 05/2012 to 09/2012
Research Intern, *Supervisor:* Prof. Yusuf Leblebici
Simulation, debugging and testing of a 3D-stacked prototype chip.

Advanced SoC Research Center, HIT, China, 05/2008 to 07/2011
Undergraduate Research Assistant, *Advisor:* Prof. Jinxiang Wang
Design of a solution with high performance and low area-cost for fault-tolerant routing in 2D-Mesh NoCs.

Teaching Experience

EC440: Introduction to Operating Systems, Spring 2012

EC413: Computer Organization, Fall 2011

Book Chapters

1. **Tiansheng Zhang**, Jonathan Klamkin, Ajay Joshi, and Ayse K. Coskun. "Thermal Management of Silicon Photonic NoCs in Many-core Systems". To appear in *Optical Interconnect for Computing Systems*, 2017.
2. **Tiansheng Zhang**, Fulya Kaplan, and Ayse K. Coskun. "Thermal Modeling and Management in 3D Stacked Systems". In *Physical Design for 3D Integrated Circuits.* Editors: Aida Todri-Sanial, and Chuan Seng Tan. CRC Press, (ISBN: 978-1-498-71036-7), pp. 229-244, 2015.

Refereed Journal Publications

1. Jose L. Abellan, Ayse K. Coskun, Anjun Gu, Warren Jin, Ajay Joshi, Andrew B. Kahng, Jonathan Klamkin, Cristian Morales, John Recchio, Vaishnav Srinivas, and **Tiansheng Zhang**. "Adaptive Tuning of Photonic Devices in a Photonic NoC Through Dynamic Workload Allocation". To appear in *IEEE Transactions on Computer Aided Design of Integrated Circuits and Systems (TCAD).*
2. **Tiansheng Zhang**, Jie Meng, and Ayse K. Coskun. "Dynamic Cache Pooling in 3D Multicore Processors". In *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, Volumn 12 Issue 2, August 2015.

Refereed Conference Publications

1. Ayse K. Coskun, Ajay Joshi, Andrew B. Kahng, Yenai Ma, Saiful Mojumder, and **Tiansheng Zhang**. "TACO: Thermally-Aware Chiplet Organization for 2.5D Integrated Manycore Systems". *In review.*

2. **Tiansheng Zhang**, Shaizeen Aga, Satish Narayanasamy, and Ayse K. Coskun. "MOCA: Memory Object Classification and Allocation in Heterogeneous Memory Systems". *In review.*

3. Ayse K. Coskun, Anjun Gu, Warren Jin, Ajay Joshi, Andrew B. Kahng, Jonathan Klamkin, Yenai Ma, John Recchio, Vaishnav Srinivas, and **Tiansheng Zhang**. "Cross-layer Floorplan Optimization for Silicon Photonic NoCs In Many-core Systems". In *Proc. Design, Automation and Test in Europe (DATE)*, pp. 1309-1314, March 2016.

4. Raphael Landaverde, **Tiansheng Zhang**, Ayse K. Coskun, and Martin Herbordt. "An Investigation of Unified Memory Access Performance in CUDA". In *Proc. IEEE High Performance Extreme Computing Conference (HPEC)*, pp. 1-6, September 2014.

5. Chao Chen, **Tiansheng Zhang**, Pietro Contu, Jonathan Klamkin, Ayse K. Coskun, and Ajay Joshi. "Sharing and Placement of On-chip Laser Sources in Silicon-Photonic NoCs". In *Proc. International Symposium on Networks-on-Chip (NOCS)*, pp. 88-95, September 2014.

6. **Tiansheng Zhang**, Jose Abellan, Ajay Joshi, and Ayse K. Coskun. "Thermal Management of Manycore Systems with Silicon-Photonic Networks". In *Proc. Design, Automation and Test in Europe (DATE)*, pp. 1-6, March 2014.

7. Jie Meng, **Tiansheng Zhang**, and Ayse K. Coskun. "Dynamic Cache Pooling for Improving Energy Efficiency in 3D Stacked Multicore Processors". In *Proc. the IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, pp. 210-215, October 2013.

8. **Tiansheng Zhang**, Alessandro Cevrero, Giulia Beanato, Panagiotis Athanasopoulos, Ayse K. Coskun, and Yusuf Leblebici. "3D-MMC: A Modular 3D Multi-Core Architecture with Efficient Resource Pooling". In *Proc. Design, Automation and Test in Europe (DATE)*, pp. 1241-1246, March 2013.

9. Jinxiang Wang, Fangfa Fu, **Tiansheng Zhang**, Yuping Chen. "A Small-Granularity Solution in 2D-Mesh Network-on-Chip". In *Proc. IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, pp. 382-384, November 2010.

Refereed Workshop Publications

1. **Tiansheng Zhang**, and Ayse K. Coskun. "Resource Management Design in 3D-Stacked Multicore Systems for Improving Energy Efficiency". In *Proceedings of Boston Area Architecture Workshop (BARC)*, Januarary 2015.