Designing Data-Aware Network-on-Chip for Performance

Abhijit Das

Supervisor: Dr. John Jose

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

Doctor of Philosophy

Department of Computer Science and Engineering
INDIAN INSTITUTE OF TECHNOLOGY GUWAHATI

October 2021

TH-2538_156101018

TH-2538_156101018

To the four pillars of my life . . .

My parents Subhadra and Sadhan Chandra Das, and my siblings Ruma and Satyajit.

TH-2538_156101018

TH-2538_156101018

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other institute/university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgements.

October 2021
Guwahati, Assam, India

Abhijit Das

TH-2538_156101018

TH-2538_156101018

Certificate

This is to certify that the dissertation titled **Designing Data-Aware Network-on-Chip for Performance**, submitted by **Abhijit Das**, to the Indian Institute of Technology Guwahati in partial fulfilment of the requirements for the degree of Doctor of Philosophy, is a record of bona fide research work done under my supervision. To the best of my knowledge, the contents of this dissertation have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other institute/university. Therefore, this dissertation is worthy of consideration for the award of the degree of **Doctor of Philosophy**.

Dr. John Jose
Dissertation Supervisor
Assistant Professor
Department of Computer Science and Engineering
Indian Institute of Technology Guwahati

October 2021
Guwahati, Assam, India

TH-2538_156101018

TH-2538_156101018

Acknowledgements

What a pleasure it is to thank the incredible people who have made this dissertation possible!

My dissertation supervisor Dr. John Jose and I joined IIT Guwahati around the same time. He was kind enough to accept me as his first PhD student even though I came from a different research background. He believed in me, and since then, he has been my friend, family and supervisor, precisely in the given order. With his first research grant, we set up Multi-Core ARchitecture and Systems (MARS) Research Lab at IIT Guwahati. When it comes to research, his technical expertise, critical insights, and systematic problem-solving approach will stay with me throughout my professional career. Besides research, Dr. Jose helped me improve my communication and presentation skills. I am a fan of his immense energy, optimism and positive attitude towards everything in life. He always made himself available whenever I needed his guidance. During our travels for conferences in UAE and Italy, I learnt how cool and fun a company he is. Thank you for everything Dr. Jose!

During the summer of 2017, when Dr. Maurizio Palesi, University of Catania, Italy, visited IIT Guwahati, I could never imagine getting an opportunity to work with a researcher of his calibre. Today, he is my collaborator and co-author for most of the dissertation-related projects and publications. He always encourages and challenges me to push an idea further. His in-depth knowledge of NoC related concepts has been invaluable for me. What a warm welcome and hospitality Dr. Jose and I received when Dr. Palesi and his family hosted us in Catania, Italy, in the year 2018, during the NOCS 2018 and VLSI-SoC 2018 conferences. I want to sincerely thank Dr. Palesi for his support and look forward to future collaborations.

In another summer of 2019, I had the privilege of knowing Prof. Prabhat Mishra, University of Florida, USA, when he visited IIT Guwahati. He immediately treated me like an old friend he is meeting after years. This heartwarming gesture from a distinguished personality like him was overwhelming for me. Those days, I was preparing for the finals of Qualcomm Innovation Fellowship (QIF) 2019. Amidst his tightly packed schedule, Prof. Mishra took the time to listen to my presentation and offered invaluable feedback. That day we began our collaboration, and today, he is my co-author in multiple publications. I have greatly benefited from his technical expertise in multiple domains and his vast experience about life. I want to extend my heartfelt gratitude to Prof. Mishra for being so kind to me.

TH-2538_156101018

My Doctoral Committee members, Dr. T. Venkatesh (Chair), Dr. Arnab Sarkar, Dr. Moumita Patra and Dr. Gaurav Trivedi, supported me throughout my PhD journey. They believed in my research ideas and engaged in lengthy discussions to make them better. Dr. Venkatesh was very kind to help me with the preparations for QIF 2019. Dr. Sarkar was always available for discussion, and his attention to detail improved my ideas. I am always grateful to all the members for their time and support. I also want to thank the co-authors and anonymous reviewers of all my publications for sharing their knowledge with me. Finally, I am also thankful to all the funding sources for their support with my research and travel.

One of the most important people in my life is Dr. Abahan Sarkar. In the first year of my Diploma in Computer Science and Technology, I joined his tuition classes back in 2007. He is one of the smartest people I know, and he saw something in me. Soon, he became my elder brother and took the job of bringing out the best in me. His belief in me and continuous support made me reach where I am today. I can not possibly thank him enough for everything he has done for me. I just want him to know that I am forever grateful. May Allah bless him.

MARS Research Lab was my home away from home for the last few years. Dipika Deb, Manjari Saha, Sivakumar S, Manju R, Amit Puri, Tathagatha Barik and Abhishek Kumar were like my family. Each one of them helped me to grow, both professionally and personally. I express my heartfelt thanks to all of them and wish them the best for the future. Special mention to Manju for taking the best care of me (especially the food) when I fractured my leg, and Siva for being there whenever I needed any help. I am also thankful to all the MTPs, BTPs, Interns and visitors of the lab I got the opportunity to work with and learn from.

During my time at IIT Guwahati, I have made a lot of friends whose company kept me happy and motivated. Starting with *Bro-Gang*, my closest buddies consisting of Brijesh Singh, Subrata Nandi and Arunangshu Pal. Brijesh treated me like his younger brother, Subrata instilled in me the value of family, and Arunangshu taught me professionalism. I have had some of the best times of my life with them. Another gang, *2GB*, consisting of Anasua Mitra, Aparajita Dutta and Palash Das, are my first set of friends at IIT Guwahati. I had a great time with Sriniwas Pandey, Partha Sarathi Pati, Madhurima Buragohain, Sheel Sindhu Manohar, Pranav Kumar Singh, Sisir Kumar Jena, Shrestha Tripathy, Cherinet Kejela Addise, Ujjwal Biswas, Hema Kumar Yarnagula, Durgesh Kumar, Dr. Shounak Chakraborty and Dr. Shirshendu Das. This list is not exhaustive, and I am sure that I am missing out on a lot of names. However, I thank all of them for the beautiful memories we made together.

Among other important people, I want to thank my best friend since school Shishir Das Chowdhury and my best friend from college Albert Mundu. They are my constant support. I am also thankful to Rajat Sarkar, Sourav Sarkar, Debika Debnath, Sapna Mukherjee, Priyanka

TH-2538_156101018

Das, Amalendu Deb Barma, Bikash Roy, Sumit Saurabh, Abhishek Kumar, Pfoseo Mao, Ramandeep Singh, Ranjan Singh, Manoj Pant, Sulabh Katiyar and Dipayan Dev.

Finally, and most importantly, I am deeply grateful to my family. Dadu and Dida have always been a source of guidance throughout my life, and this dissertation is a result of their blessings. Maa is my biggest support system and has made uncountable efforts and sacrifices for me. Babu has always believed that I am made for greater things and backed me for every decision I made in life. My elder sister Ruma and her husband Sunil are the best motivators I can ask for. My younger brother Maman (Satyajit) understands me like no one else. Without these people in my life, I do not have an existence. So, I want to say a big thank you!

TH-2538_156101018

TH-2538_156101018

Abstract

We are now in an era where data drives everything, and the demand for information processing is increasing exponentially. This increasing demand is driving a parallel increase in the number of processing cores in Tiled Chip Multi-Processors (TCMPs). It is indeed visible in the industry, with Intel Xeon Phi, AMD EPYC and Ampere Altra processors featuring up to 128 cores in their TCMPs. As the number of cores continues to increase in modern TCMPs, on-chip communication plays a pivotal role in determining the performance. Multi-hop packet-based Network-on-Chip (NoC) is a widely adopted communication infrastructure in modern TCMPs as it provides a high transfer bandwidth and is scalable. Data-driven applications expect minimum memory access latency, where NoC plays a significant role. In fact, it is reported that NoC is responsible for 60% to 75% of the miss latency experienced by the applications. Nevertheless, most of the existing memory access optimisations are NoC oblivious. These optimisations focus on the memory hierarchy and treat the underlying NoC like a black box. Since NoC plays a vital role in memory access latency, ignoring the nature of its infrastructure may severely impact the performance of applications in TCMPs.

This dissertation advocates for considering NoC and memory hierarchy together when designing techniques and proposing optimisations for TCMPs. The dissertation shows that designing data-aware NoC with the help of memory hierarchy improves resource utilisation, reduces memory access latency and improves system performance in TCMPs. Specifically, the architectures proposed in this dissertation are able to improve overall system performance by up to 19% with negligible storage, area and power overhead. While TCMPs continue to scale with the help of NoC, the proposed architectures can help with future design decisions.

TH-2538_156101018

TH-2538_156101018

Table of Contents

List of Figures xxi

List of Tables xxv

Nomenclature xxvii

1 Introduction 1

1.1 Data-Aware Network-on-Chip . . . 2

1.2 Explored Problems with Solutions . . . 3

1.2.1 Critical Packet Prioritisation . . . 4

1.2.2 Critical Word Prioritisation . . . 4

1.2.3 Opportunistic Caching . . . 5

1.2.3.1 Exploiting Underutilised Router Buffers . . . 5

1.2.3.2 Exploiting Unused Trace Buffers . . . 5

1.3 Summary . . . 6

2 Background 9

2.1 Tiled Chip Multi-Processors . . . 9

2.2 Message, Packet, and Flit . . . 9

2.3 Network-on-Chip Overview . . . 10

2.3.1 Topology . . . 11

2.3.2 Routing and Arbitration . . . 12

2.3.3 Flow Control . . . 13

2.3.4 Router Microarchitecture . . . 16

2.4 Memory Hierarchy Overview . . . 16

2.4.1 Organisation . . . 17

2.4.2 Coherence . . . 18

2.4.3 Traffic through NoC . . . 20

2.5 Evaluation Methodology . . . 21

TH-2538_156101018

2.5.1 Simulation Infrastructure 21
2.5.2 Performance Metrics 22
2.6 Chapter Summary 23

3 Critical Packet Prioritisation **25**
3.1 Introduction 25
3.2 Motivation 27
3.2.1 Exploiting Slack and its Diversity 27
3.3 SAR Architecture 30
3.3.1 Slack Estimation 30
3.3.2 Minimal Path Estimation 30
3.3.3 Modified Router Microarchitecture 32
3.3.3.1 Packet Pre-processor (PP) 32
3.3.3.2 Look-Ahead Re-router (LR) 35
3.3.4 Comparison and Design Challenges 35
3.4 Performance Evaluation 37
3.4.1 Simulation Framework and Workloads 37
3.4.2 Result Analysis and Discussion 39
3.4.2.1 Reply Difference Time 39
3.4.2.2 Usage Wait Time 40
3.4.2.3 Network Stall Time 41
3.5 Overhead Analysis 42
3.5.1 Timing Overhead 42
3.5.2 Storage, Area and Power Overhead 42
3.6 Related Work 43
3.7 Chapter Summary 43

4 Critical Word Prioritisation **45**
4.1 Introduction 45
4.2 Background 47
4.2.1 Cache Organisation in NoC based TCMPs 47
4.2.2 Cache Optimisations for Miss Penalty Reduction 48
4.2.2.1 Early Restart (ER) 49
4.2.2.2 Critical Word First (CWF) 49
4.3 Data Criticality in Applications 50
4.4 Motivation 52
4.5 NoC-Aware Early Restart Optimisation 53

4.5.1 Critical Flit Identification . . . 54
4.5.2 Critical Flit Prioritisation . . . 55
4.6 Performance Evaluation . . . 57
4.6.1 Simulation Framework and Workloads . . . 57
4.6.2 Result Analysis and Discussion . . . 58
4.6.2.1 L1 Cache Miss Penalty . . . 58
4.6.2.2 System Speedup . . . 59
4.6.3 Comparison with CWF Optimisation . . . 60
4.7 Sensitivity and Overhead Analysis . . . 61
4.7.1 Impact on Multi-Programmed Workloads . . . 61
4.7.2 Impact of Channel Width . . . 62
4.7.3 Storage, Area and Power Overhead . . . 63
4.8 Related Work . . . 63
4.9 Chapter Summary . . . 64

5 Opportunistic Caching by Exploiting Underutilised Router Buffers 65
5.1 Introduction . . . 65
5.2 Background . . . 67
5.2.1 LLC Miss Penalty . . . 67
5.2.2 VC Availability . . . 69
5.3 Motivation . . . 70
5.4 Proposed Architecture . . . 71
5.4.1 Local Store: Keeping Evicted LLC Blocks in Router Buffers . . . 72
5.4.2 Local Reply: Responding to Block Requests from Routers . . . 79
5.4.3 Block Forward and Drop: Releasing Stored Blocks . . . 80
5.4.4 Maintaining Cache Coherence . . . 80
5.5 Performance Evaluation . . . 81
5.5.1 Simulation Framework and Workloads . . . 82
5.5.2 Result Analysis and Discussion . . . 83
5.5.2.1 LLC Miss Penalty . . . 83
5.5.2.2 Network Stall Time . . . 84
5.5.2.3 System Speedup . . . 84
5.5.3 Case Study: Proposed Architecture for L1 Cache . . . 85
5.6 Sensitivity and Overhead Analysis . . . 87
5.6.1 Impact of Number of VCs . . . 87
5.6.2 Without Corner Router VCs . . . 88
5.6.3 Storage, Area and Power Overhead . . . 89

5.7 Related Work 89
5.8 Chapter Summary 90

6 Opportunistic Caching by Exploiting Unused Trace Buffers 91
6.1 Introduction 91
6.2 Background 93
6.2.1 L1 Cache Miss Penalty 93
6.2.2 VC Availability 94
6.2.3 Embedded Trace Buffer 95
6.3 Motivation 97
6.4 Opportunistic Caching in NoC 98
6.4.1 Block Store in Router Buffers 100
6.4.2 Block Store in Trace Buffers 101
6.4.3 Block Reply from Routers 104
6.4.4 Block Forward and Drop from Routers 105
6.4.4.1 Time-Triggered Block Forward (TT-BF) 106
6.4.4.2 Message-Triggered Block Forward (MT-BF) 106
6.4.5 Cache Coherence 106
6.5 Performance Evaluation 108
6.5.1 Simulation Framework and Workloads 109
6.5.2 Result Analysis and Discussion 110
6.5.2.1 L1 Cache Miss Penalty 110
6.5.2.2 Network Stall Time 111
6.5.2.3 System Speedup 112
6.5.3 Qualitative Comparison with An Existing Work 113
6.6 Sensitivity and Overhead Analysis 116
6.6.1 Impact of Number of VCs 116
6.6.2 Impact of Trace Buffer Size 117
6.6.3 Impact of Time Threshold 118
6.6.4 Storage, Area and Power Overhead 118
6.7 Related Work 119
6.8 Chapter Summary 120

7 Conclusions and Future Work 123
7.1 Dissertation Summary 123
7.2 Future Research Directions 126

References 127

List of Publications 137

Vita 139

TH-2538_156101018

TH-2538_156101018

List of Figures

1.1 Summary of contributions in the dissertation 7

2.1 Conceptual view of an NoC based TCMP, where, PE: Processing Element, R: Router, MC: Memory Controller, and NIC: Network Interface Controller. 10
2.2 Structure of a message, packet and flit 10
2.3 Popular NoC topologies 11
2.4 Deadlock-free routing turn models 13
2.5 Example of a store-and-forward flow control 14
2.6 Example of a virtual cut-through flow control 15
2.7 Data transfer between different levels of memory. A data request packet consists of a single head flit (H) carrying the header, whereas a data reply packet consists of a head flit followed by multiple body flits and ended by a tail flit (H, B0, B1, B2, T) carrying header and data. The critical word (W3) and the flit carrying it (B1) are shown in *red*. 17
2.8 Venn diagram of classic five-state MOESI model 19

3.1 Conceptual view of an NoC based TCMP with 2-levels of on-chip cache memories. A data request packet (shown in *blue*) consists of a single head flit (H) carrying the header, whereas a data reply packet (shown in *green*) consists of a head flit followed by multiple body flits and ended by a tail flit (H, B0, B1, B2, T) carrying header and data. 26
3.2 Conceptual illustration of the presence of slack in NoC packets 28
3.3 No-slack packets travelling through NoC routers 29
3.4 Modified message/packet header 29
3.5 Quadrant and Region based on the position of destination 31
3.6 Possibility of alternate minimal path 32
3.7 Proposed SAR router microarchitecture 35
3.8 Illustrative example of slack-aware re-routing 36

TH-2538_156101018

3.9 Reply difference time . . . 39
3.10 Usage wait time . . . 40
3.11 Network stall time . . . 41

4.1 Data transfer between different levels of memory. A data request packet consists of a single head flit (H) carrying the header, whereas a data reply packet consists of a head flit followed by multiple body flits and ended by a tail flit (H, B0, B1, B2, T) carrying header and data. The critical word (W3) and the flit carrying it (B1) are shown in *red*. . . . 46
4.2 Position of critical word in the requested blocks . . . 49
4.3 Reply difference time . . . 53
4.4 Modified L1 cache controller . . . 54
4.5 Modified router microarchitecture . . . 56
4.6 L1 cache miss penalty . . . 58
4.7 System Speedup . . . 60
4.8 Comparison with CWF optimisation . . . 61
4.9 Impact on multi-programmed workloads . . . 61
4.10 Impact of channel width . . . 62

5.1 Conceptual view of an NoC based TCMP, where data transfer request is travelling from L1 cache to LLC (shown in *blue*) and then from LLC to MC (shown in *green*). . . . 66
5.2 LLC miss penalty . . . 68
5.3 VC availability in local input port . . . 69
5.4 Re-reference time of evicted LLC blocks . . . 70
5.5 Conceptual illustration of the proposed router microarchitecture . . . 71
5.6 Modified message/packet header . . . 72
5.7 LLC miss penalty . . . 83
5.8 Network stall time . . . 84
5.9 System speedup . . . 85
5.10 L1 cache miss penalty with the proposed architecture for L1 cache . . . 86
5.11 Network stall time with the proposed architecture for L1 cache . . . 87
5.12 System speedup with the proposed architecture for L1 cache . . . 87
5.13 Impact of number of VCs . . . 88
5.14 Without corner router VCs . . . 88

TH-2538_156101018

6.1 Conceptual view of an NoC based TCMP, where data transfer request is travelling from L1 cache to LLC (shown in *blue*) and then from LLC to MC (shown in *green*). . . . 93
6.2 VC availability in local input port 95
6.3 Re-reference time of evicted L1 cache blocks 96
6.4 Conceptual view of the proposed router microarchitecture. All the additional units and links are shown in *red*. Evicted L1 cache blocks enter the NoC as packets and get divided into multiple smaller units called flits (H, B0, B1, T). Based on whether a block is clean or dirty, the corresponding flits get stored in either the trace buffer or the local VCs. 99
6.5 Modified message/packet header 100
6.6 Eviction of a dirty L1 cache block 100
6.7 Eviction of a clean L1 cache block 102
6.8 L1 cache miss on a requested block 102
6.9 Messages to maintain cache coherence 107
6.10 L1 cache miss penalty 110
6.11 Network stall time 111
6.12 System speedup 113
6.13 Comparison with *VCache* architecture 114
6.14 Impact of number of VCs 115
6.15 Impact of trace buffer size 116
6.16 Smallest re-reference time 117
6.17 Impact of time threshold 117

7.1 Summary of contributions in the dissertation 124

TH-2538_156101018

List of Tables

2.1 System configuration. *Some of the parameters vary for specific architecture. 21

3.1 SAR priority vector description . . . 33
3.2 System configuration . . . 37
3.3 Benchmark characteristics . . . 38
3.4 Workload mixes . . . 39

4.1 Position of critical word in the corresponding flits. Note that head flit (H) does not carry any words of the data block and hence its corresponding column is left empty. . . . 51
4.2 System configuration . . . 57

5.1 MOESI Directory Protocol - Stable states of LLC . . . 72
5.2 LLC/DIR Controller - Block replacement. REPLACEMENT: block replacement from LLC, ACK-PUT: acknowledgement from MC for PUT request, PUTE/PUTO/PUTM: write-back request for E/O/M state, DATA: evicted block, INV: invalidation, REQ: L1 requester, DIR: directory, MEM: off-chip memory connected through MC. . . . 73
5.3 LLC/DIR Controller - Block miss. GETS: read request from L1, GETX: write request from L1, DATA: shared block from off-chip memory, EX-DATA: exclusive block from off-chip memory, CAN-PUT: cancel PUT request for coherence . . . 77
5.3 Continued . . . 78
5.4 System configuration . . . 82
5.5 Workload mixes . . . 83
5.6 Workload mixes . . . 86

6.1 System configuration . . . 108
6.2 Workload mixes . . . 109

6.3 Overhead compared to the baseline 119

TH-2538_156101018

Nomenclature

Acronyms / Abbreviations

AI	Artificial Intelligence
CWF	Critical Word First
DfD	Design-for-Debug
DL	Deep Learning
DOR	Dimension-Ordered Routing
DRAM	Dynamic Random Access Memory
DSA	Domain-Specific Architecture
ER	Early Restart
ETB	Embedded Trace Buffer
FIFO	First In First Out
HDL	Hardware Description Language
HoL	Head-of-Line
ILP	Instruction-Level Parallelism
IN	Interconnection Network
IPC	Instructions Per Cycle
IP	Intellectual Property
ITRS	International Technology Roadmap for Semiconductors

TH-2538_156101018

LLC Last Level Cache

MC Memory Controller

ML Machine Learning

MLP Memory-Level Parallelism

MPKI Misses Per Kilo Instruction

MP Miss Penalty

MSHR Miss Status Handling Register

NIC Network Interface Controller

NoC Network-on-Chip

NST Network Stall Time

NUCA Non-Uniform Cache Architecture

NVM Non-Volatile Memory

OCIN On-Chip Interconnection Network

OCN On-Chip Network

ODN On-Die Network

OoO Out-of-Order

PE Processing Element

QoS Quality of Service

RC Route Compute

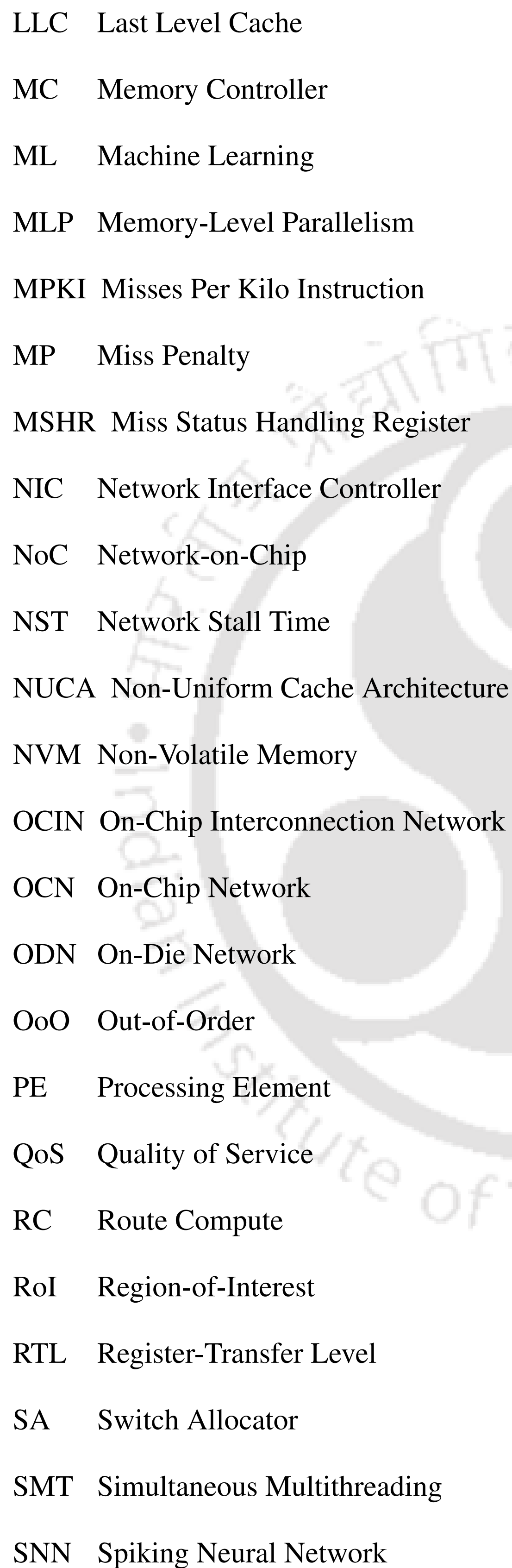
RoI Region-of-Interest

RTL Register-Transfer Level

SA Switch Allocator

SMT Simultaneous Multithreading

SNN Spiking Neural Network



TH-2538_156101018

SRAM Static Random Access Memory

S System Speedup

TCMP Tiled Chip Multi-Processors

TSMC Taiwan Semiconductor Manufacturing Company

VA VC Allocator

VC Virtual Channel

VN Virtual Network

WSC Warehouse-Scale Computer

TH-2538_156101018

TH-2538_156101018

Chapter 1

Introduction

Multiple factors including, the continuance of Moore's Law [1], end of Dennard's Scaling [2] and limits of Instruction-Level Parallelism (ILP) [3] has led to the paradigm shift from uni-core to multi-core systems. We are now in an era where data drives everything, and the demand for information processing is increasing exponentially. As a result, almost all of the processing devices in and around us are realised by some form of a multi-core system.

All modern multi-core systems are also popularly referred to as Tiled Chip Multi-Processors (TCMPs) because of the way they are designed. 2015 International Technology Roadmap for Semiconductors (ITRS) report predicts that the increasing demand for information processing will drive a 30-fold increase in the number of processing cores by 2030 [4]. It is indeed visible in the industry with Intel Xeon Phi [5], AMD EPYC [6] and Ampere Altra [7] processors featuring up to 128 cores in their TCMPs. As the number of cores continues to increase in TCMPs, on-chip communication plays a pivotal role in determining the performance. Global wires, shared buses, and monolithic crossbars have failed to provide the necessary communication infrastructure in TCMPs. As a result, a shared Network-on-Chip (NoC) based communication infrastructure is employed in most of the modern TCMPs [8].

NoC is a multi-hop packet-based communication infrastructure that connects different cores with each other through routers. NoC provides a high transfer bandwidth, and the infrastructure is scalable. NoC is preferred over others as it is reliable with well-controlled and highly predictable electrical properties. For data-driven applications, NoC plays a significant role in memory access latency. In fact, it is reported that NoC is responsible for 60% to 75% of the miss latency experienced by applications [9]. Nevertheless, most of the popular applications, as well as existing memory access optimisations, are NoC oblivious.

Applications running on different cores access on-chip cache memories and off-chip main memory through the underlying NoC. However, most of the available memory access optimisations are unconcerned about the on-chip communication infrastructure. These

TH-2538_156101018

optimisations focus on the memory hierarchy and treat the underlying NoC like a black box. The main reason behind this practice is the assumption that the on-chip communication latency is fixed (defined) and hardly changes over time. Such an assumption may hold true for traditional on-chip communication infrastructures (global wires, shared buses and monolithic crossbars), but not for NoC. Unlike those infrastructures, NoC uses packet-based communication, where the nature of a transfer is discrete (not continuous). Also, the transfer is multi-hop and experiences unknown router delay at each hop due to routing and arbitration decisions. Hence, the communication latency in NoC is undefined and varies according to the available traffic. As NoC plays a vital role in memory access latency, ignoring the nature of its infrastructure may severely impact the performance of the applications in TCMPs.

We are in the era of data-driven applications, and these applications demand minimum memory access latency. While TCMPs continue to scale with the help of NoC, designing memory access optimisations that are aware of the NoC infrastructure (resources) is necessary. Existing literature advocates that considering NoC and memory hierarchy together is the way forward in TCMP design [9]. Existing literature also considers efficient utilisation of NoC resources as a fundamental challenge in TCMP design, as currently, it is only about 20% [10]. This dissertation accepts both; the suggestion as well as the challenge. There are innovative solutions available in the literature about using NoC for caching and coherence [11][12][13][14][15]. Data and application-aware solutions are also available for efficient utilisation of NoC resources [16][17][18][19][20]. However, most of these solutions focus on independently optimising the NoC without bothering about its interaction with the memory hierarchy, thus leading to sub-optimal performance. This dissertation attempts to establish a dynamic cooperation between NoC and the memory hierarchy for improved performance. By gathering information about the data travelling from one level of memory to another, we exploit underutilised NoC resources to design techniques and propose optimisations that reduce memory access latency and improve overall system performance.

1.1 Data-Aware Network-on-Chip

Most of the existing techniques and optimisations do not consider NoC and memory hierarchy together. Instead, they optimise for individual shared resources, like NoC is optimised for packet latency, Last Level Cache (LLC) is optimised for hit rate, and Memory Controller (MC) is optimised for bank access latency, etc. However, these optimised metrics are not directly related to the performance experienced by the applications. There are multiple valid reasons as to why optimising for individual shared resources may not be adequate for overall system performance. For example, service latency of packets might be hidden due to

TH-2538_156101018

Memory-Level Parallelism (MLP) [21], and hence packet latency might not be indicative of the network-related stall time at the core (processor)[22]. Similarly, LLC is distributed across multiple cores as banks, and a requested data can be anywhere, from the nearest to the farthest bank. Hence, just the hit rate alone might not be indicative of the LLC access time at the processor [23]. Furthermore, optimising individual shared resources to provide minimum service guarantees often leads to over-provision of the resources. Studies on Warehouse-Scale Computers (WSCs) have reported average resource utilisation to be between 10% and 50% only [24][25]. This is why efficient utilisation of shared resources is a fundamental challenge.

To design techniques and propose optimisations that can directly impact the overall system performance, shared resources must be considered together. Moreover, all the shared resources must understand the characteristics and importance of the applications to improve their utilisation. Hence, rather than ignoring the NoC, all the techniques and optimisations (including memory access optimisations) should consider the NoC and memory hierarchy together. Also, NoC should be made aware of the characteristics and importance of the applications that run on the TCMP. However, labelling a particular application as more important than others at all times can be misleading. This is because, for the same application, data requested at different times will have different importance to the processor. Hence, this dissertation goes one level deeper and focuses on designing data-aware NoC for improving overall system performance. While the characteristics and importance of the data are relatively well understood at memory hierarchy, i.e. on-chip cache memories and off-chip main memory, less is known at NoC. The complex and chaotic interaction between the cores in a TCMP makes it difficult for the NoC to understand about the travelling data (packet) on its own. Additional challenges include varying packet injection rate, queuing delay, MLP, spatial location of the LLC banks and MCs, etc. Hence, NoC must interact with the memory hierarchy to understand about the data travelling from one level of memory to another.

This dissertation advocates for considering NoC and memory hierarchy together when designing techniques and proposing optimisations for TCMPs. The dissertation shows that designing data-aware NoC with the help of memory hierarchy improves resource utilisation, reduces memory access latency and improves overall system performance in TCMPs.

1.2 Explored Problems with Solutions

A brief discussion about the explored problems and their solutions proposed as part of this dissertation is presented here. Subsequent chapters discuss each one of them in detail.

TH-2538_156101018

1.2.1 Critical Packet Prioritisation

Different data (packets) travelling through the NoC can have a differential impact on the overall system performance. For example, some packets may be more critical and can not be delayed as they stall execution in the processor, whereas some other packets may tolerate delay as their latency is hidden by the outstanding latency of their predecessors. In this work, we consider a metric from the literature called *slack* that represents the relative importance of packets [26]. Specifically, the slack of a packet is the number of system clock cycles that packet can be delayed in the NoC without impacting the execution in the processor. NoC obtains the slack of the incoming packets by interacting with the on-chip cache controllers. Lower slack packets are more critical than higher slack packets, whereas no-slack packets are the most critical. An existing work prioritises lower slack packets over higher slack packets but does not deal with the no-slack packets [17]. However, we observe that due to low packet injection rate in modern applications, no-slack packets dominate the NoC. Since no-slack packets are the most critical ones, their delay in NoC will severely hamper the overall system performance. We propose a novel technique with few optimisations to prioritise lower slack packets over their higher counterparts and find an alternate minimal path for no-slack packets [27]. Our proposed prioritisation makes sure that no-slack packets are not delayed in the NoC and reach their destination at the earliest to resume execution.

1.2.2 Critical Word Prioritisation

At any given point in execution, a processor usually requests for a single word called *critical word* from the memory hierarchy [28]. When the processor encounters an L1 cache miss on the critical word, a data transfer from the next level of memory is triggered. The smallest unit of data transfer between different levels of memory is always in the unit of blocks containing multiple words. So, even though the processor requests a single word, an entire data block (containing the critical word) is brought from the next level of memory in the form of a packet through the underlying NoC. Nevertheless, transfer in NoC is discrete and transfer bandwidth is limited to the channel width called *flit*, such that flit $<<$ packet. A data block (packet) is divided into multiple flits and usually sent in sequence to the requester. The critical word can be in any of the incoming flits which experience unknown router delay along the way. Two of the most popular memory access optimisations to reduce miss latency (penalty) of the critical word are *Early Restart (ER)* and *Critical Word First (CWF)* [28]. However, both of these optimisations are NoC oblivious and hence, when employed in NoC based TCMPs neglect the concept of flit based discrete transfer and router delay. As a result, the effectiveness of ER and CWF and all other NoC oblivious memory access optimisations is less in modern

TH-2538_156101018

TCMPs. In this work, we show that CWF optimisation might not be necessary and propose a novel NoC-aware ER optimisation where the flits carrying the critical words are prioritised to reach their destination as soon as possible to resume processor execution at the earliest [29].

1.2.3 Opportunistic Caching

Due to the increasing core counts, limited on-chip area and associated cost, most of the modern TCMPs employ 2-levels of on-chip cache memories. Even though some TCMPs have 3-levels, due to the limited size, data-driven applications with large memory footprints encounter frequent cache misses. Such applications suffer from recurring miss penalties when they re-reference recently evicted cache blocks. On the other hand, due to the very low packet injection rate of only around 5% in modern applications [30][31][32], a lot of NoC resources remain underutilised, or even worse, unused. We propose techniques and optimisations to exploit these resources to accommodate evicted cache blocks in NoC routers. Future re-references to the evicted cache blocks can be serviced from the routers without any delay, which significantly reduces miss penalty and improves overall system performance.

1.2.3.1 Exploiting Underutilised Router Buffers

To meet the worst-case performance requirements and scalable transfer bandwidth, NoC routers are provisioned with input port buffers. Packets on their way from source to destination are temporarily stored in these buffers while taking part in routing and arbitration decisions. However, due to low packet injection rate, router buffers remain underutilised except during NoC congestion. We propose to store evicted cache blocks in such buffers without hampering the usual NoC transfer [33][34]. An evicted cache block can either be *clean* or *dirty*, where the former is discarded while the latter is sent to the next level of memory for write-back. We propose to store the evicted, dirty cache blocks in the router buffers on their way for the write-back. When a recently evicted cache block is re-referenced, we facilitate a local reply with the matching stored block from the routers. Local reply completely avoids the on-chip travel, which significantly reduces miss penalty and improves system performance. We also bring some evicted, clean cache blocks to the routers (which are otherwise discarded) and store them in the buffers to increase our chances of local replies.

1.2.3.2 Exploiting Unused Trace Buffers

When we bring evicted, clean cache blocks to be stored in the routers to increase our chances of local replies, they compete against the evicted, dirty cache blocks for a place in the buffers. The number of clean blocks evicted from the cache memory is much higher than the dirty

blocks. Router buffers are underutilised, but they are also limited by size; hence making the clean blocks compete with the dirty blocks defeats the purpose of accommodating more evicted cache blocks for local replies. Due to the design complexity of NoC based TCMPs, post-silicon debug is usually practised to validate a proposed design before going into the production. An important phase of the debug involves validating the on-chip interaction between different cores, and to aid the process, Design-for-Debug (DfD) hardware are embedded across various modules and cores in a TCMP [35]. *Trace buffers* are DfD hardware embedded in NoC routers to record their state for post-silicon debug and validation. However, when a TCMP design goes into production, most of the DfD hardware become non-functional. Since the usage of DfD hardware (including the trace buffers) is sporadic and rare after the production, most of them are left unused. We re-purpose the unused trace buffers embedded in NoC routers to store evicted, clean cache blocks [36][37]. With the dirty blocks stored in router buffers and the clean blocks stored in trace buffers, we accommodate more evicted cache blocks, which significantly increases our chances of local replies.

1.3 Summary

With the ever-increasing number of data-driven applications, minimum memory access latency is desirable in modern NoC based TCMPs. Even though NoC plays a significant role in memory access latency, it is treated like a black box and ignored by existing techniques and optimisations. In this dissertation, we propose four different techniques and optimisations to address the interaction gap. We show that considering NoC and memory hierarchy together while designing techniques and proposing optimisations for TCMPs improves resource utilisation, reduces memory access latency and improves overall system performance. Figure 1.1 summarises our contributions towards designing data-aware NoC for performance.

In the remainder of this dissertation, Chapter 2 presents relevant background on NoC and memory hierarchy. It also details the simulation framework and performance metrics we consider for evaluation. Chapter 3 describes our work on critical packet based prioritisation in NoC. Chapter 4 goes one level deeper and presents critical word based prioritisation in NoC. Chapter 5 details the first part of our work on opportunistic caching, where we exploit NoC router buffers to store evicted cache blocks. Whereas Chapter 6 completes the discussion on opportunistic caching, where we exploit both router buffers and trace buffers. Finally, Chapter 7 concludes the dissertation and discusses possible future research directions.

TH-2538_156101018

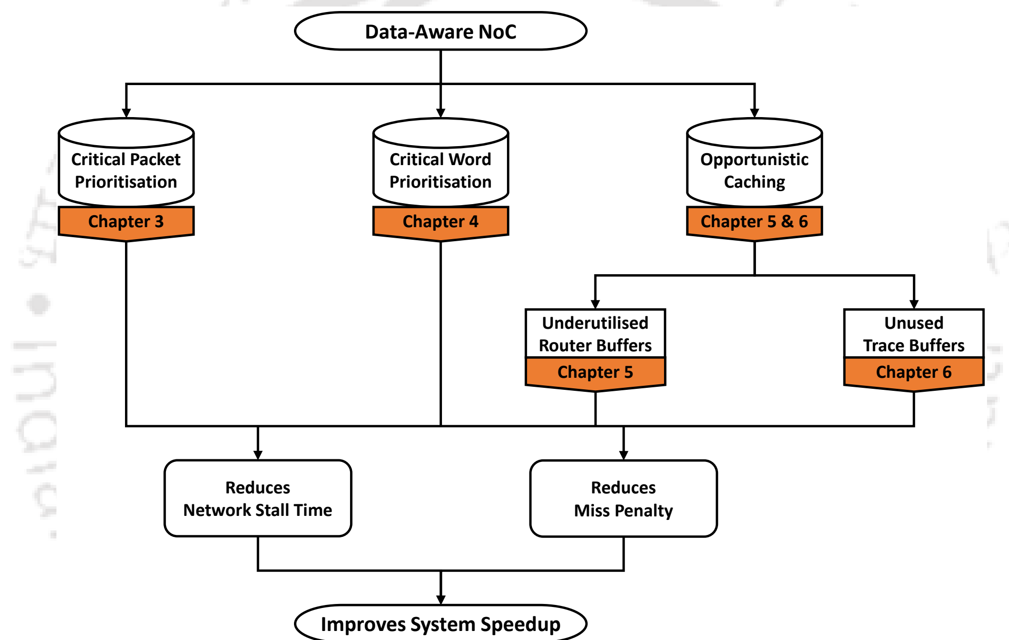


Figure 1.1: Summary of contributions in the dissertation

TH-2538_156101018

TH-2538_156101018

Chapter 2

Background

This chapter presents the key features of NoC communication infrastructure to understand the dissertation. Necessary background about the memory hierarchy is also discussed to familiarise the readers about coherence and the traffic through NoC. Finally, evaluation methodology, including simulation infrastructure and performance metrics, is described.

2.1 Tiled Chip Multi-Processors

Figure 2.1 shows the conceptual view of an NoC based TCMP, including the Processing Element (PE) and router microarchitecture. Since the cores are arranged like regular tiles, the term TCMP became synonymous with multi-core systems. The shown TCMP has 2-levels of on-chip cache memories, where the L2 cache serves as the LLC. L2 cache is divided into multiple banks and distributed across all the cores. Each PE houses an Out-of-Order (OoO) superscalar processor, an L1 Instruction (I) and Data (D) cache, and an L2 cache bank. PEs use Network Interface Controller (NIC) to enter the underlying NoC for communication.

2.2 Message, Packet, and Flit

Cores in a TCMP communicate with each other by exchanging *messages*. The typical structure of a message has two parts: header and payload, as shown in Figure 2.2. The header carries information about message exchange, including source and destination cores, and the payload carries the requested data block. When cores communicate through the underlying NoC, a message becomes one or more *packet(s)*. In modern NoC based TCMPs, one message can be accommodated within a single packet. Similar to the structure of a message, a packet has a head and a body. However, due to limited NoC channel width, packets are further

TH-2538_156101018

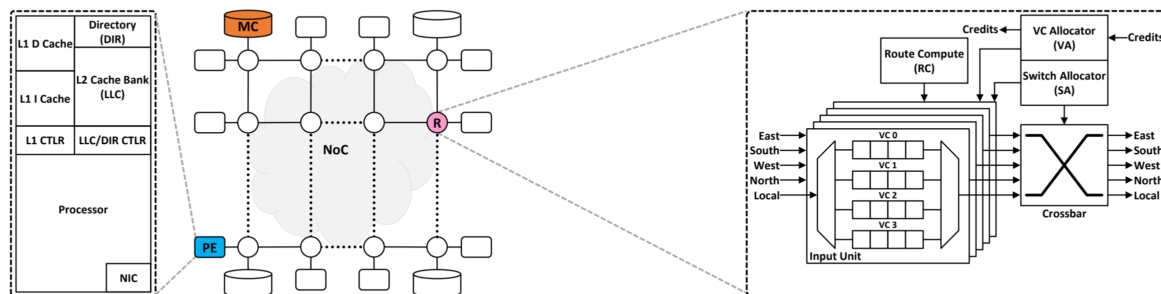


Figure 2.1: Conceptual view of an NoC based TCMP, where, PE: Processing Element, R: Router, MC: Memory Controller, and NIC: Network Interface Controller.

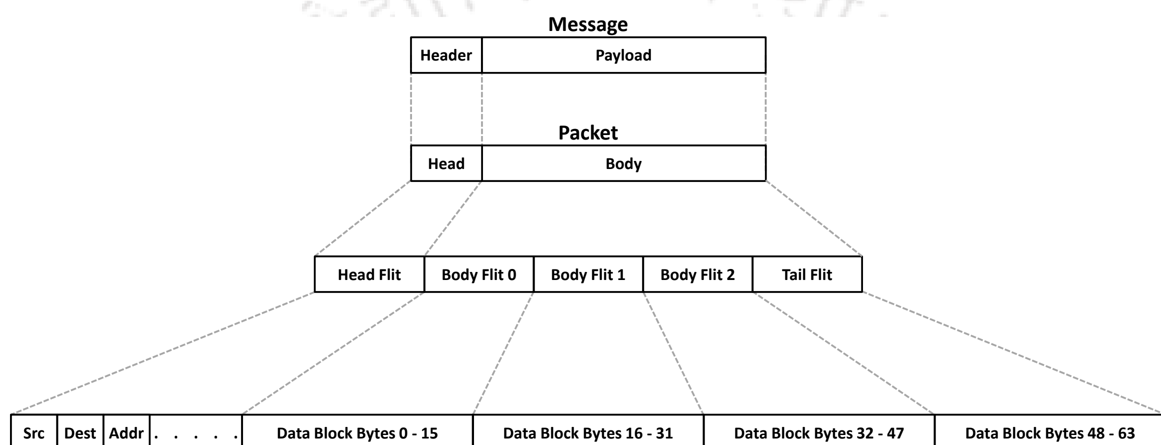


Figure 2.2: Structure of a message, packet and flit

divided into smaller flow control units called *flits*. A head flit marks the beginning of a packet and carries the header. Multiple body flits, ended by a tail flit carries the requested data block. An NoC packet P_i can be represented as:

$$P_i = \{F^i_{Head} \mid F^i_{Body_0} \mid \dots \mid F^i_{Body_{n-1}} \mid F^i_{Tail}\} \tag{2.1}$$

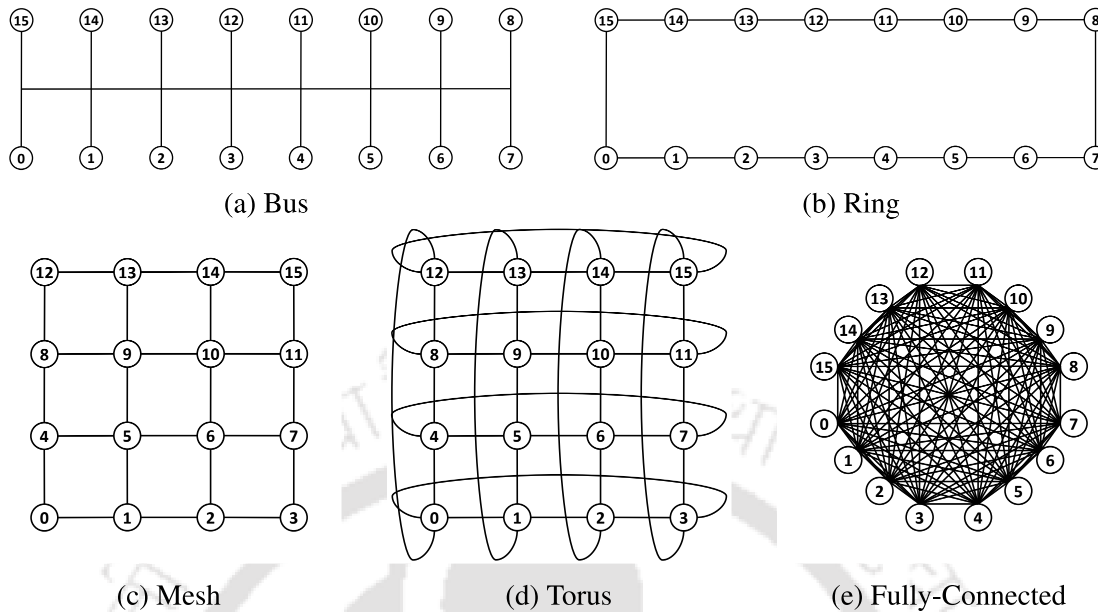
Since a flit F^i_j contains multiple words, it can represented as:

$$F^i_j = \{W^i_k \mid W^i_{k+1} \mid \dots \mid W^i_{l-2} \mid W^i_{l-1}\} \tag{2.2}$$

2.3 Network-on-Chip Overview

Owing to its growing popularity in the research community, multiple alternate names of NoC have emerged, like On-Chip Network (OCN), On-Die Network (ODN), Interconnection Network (IN), On-Chip Interconnection Network (OCIN), etc., and are used interchangeably.

TH-2538_156101018



(a) Bus (b) Ring

(c) Mesh (d) Torus (e) Fully-Connected

Figure 2.3: Popular NoC topologies

As core counts continue to increase, NoC is the preferred communication infrastructure in modern TCMPs. The key features that define an NoC communication infrastructure are: topology, routing and arbitration, flow control, and router microarchitecture. We describe each of these features to an extent necessary to understand the dissertation. A more comprehensive description can be found in any of the classic books on NoC [38][39][40].

2.3.1 Topology

Topology describes the way cores (routers) are connected via channels (links) to form the on-chip communication network. *Hop* refers to the physical connection between the neighbouring routers. Topology defines the minimum number of hops between a source and destination, which has a significant role in the performance of NoC. Some of the popular NoC topologies are shown in Figure 2.3. Before scalability became an issue, *Bus* topology has been extensively used due to its ease of design and low power consumption. *Ring* topology is introduced as a scalable alternative to *Bus*; however, the number of hops increases linearly with the number of routers. The solution to this problem came in the form of 2D topologies for NoC, like *Mesh* and *Torus*, where the number of hops increases as a square-root of the number of routers. An impractical topology due to the layout issues is called *Fully-Connected*, which always offers a 1-hop traversal between any pair of source and the destination routers.

Topology also determines the path diversity; the number of alternate shortest paths available for the NoC packets to travel from source to their destination. To understand with an

TH-2538_156101018

example, from source ① to destination ⑦, paths available in different topologies are:

Ring:
① → ② → ③ → ④ → ⑤ → ⑥ → ⑦

Mesh:
① → ② → ③ → ⑦
① → ② → ⑥ → ⑦
① → ⑤ → ⑥ → ⑦

Torus:
① → ② → ③ → ⑦
① → ② → ⑥ → ⑦
① → ⑤ → ⑥ → ⑦
① → ⓪ → ③ → ⑦
① → ⓪ → ④ → ⑦
① → ⑤ → ④ → ⑦

2.3.2 Routing and Arbitration

After determining the topology, a routing algorithm dictates the path NoC packets should take to reach from source to their destination. Based on the routing algorithm, the dictated paths can be either minimal or non-minimal. A routing algorithm is called minimal if it always dictates paths with minimum number of hops between a source and destination. At every hop on a minimal path, a packet moves closer to its destination. On the other hand, a non-minimal routing algorithm allows misrouting but makes sure that the packets find their way to the destination. Non-minimal routing algorithms are able to avoid congested regions.

The most important goals of any routing algorithm are deadlock avoidance and traffic distribution. Based on additional goals, routing algorithms can be identified as deterministic, oblivious and adaptive. A deterministic routing algorithm always selects the same path (same set of hops) for a given pair of source and destination. One of the most popular deterministic routing algorithms is called Dimension-Ordered Routing (DOR). To understand with an example, in 2D-Mesh topology (Figure 2.3c), an XY-DOR algorithm dictates that a packet must travel along the X dimension (East or West direction) first, followed by Y dimension (North or South direction) to reach from source to its destination. Oblivious and adaptive routing algorithms allow packets to take different paths even for the same source and

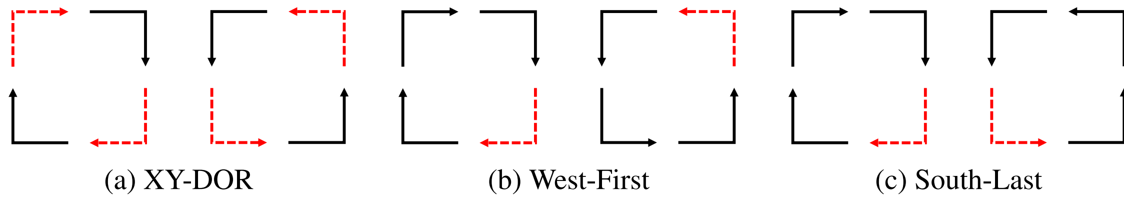


Figure 2.4: Deadlock-free routing turn models

destination pair at different instances. An oblivious routing algorithm prefers randomness or uses some metric (other than NoC congestion) to select a path. Whereas an adaptive routing algorithm targets ideal traffic distribution and considers NoC congestion while selecting a path. To avoid a deadlock, routing algorithms usually adopt one of the popular turn models, which restricts certain turns that eliminate the possibility of cyclic resource dependency [41]. Figure 2.4 shows turn models for three popular deadlock-free routing algorithms (XY-DOR, West-First and South-Last), where the restricted turns are highlighted in dotted-red.

Since the resources in NoC are shared, multiple cores can compete for a given resource at the same time. For example, multiple packets can request the service of a particular router at the same time. To tackle with such situations, a fair arbitration policy is needed to be employed that uses some criteria to select a winner from the competitors. There are two arbitration stages in an NoC router, which will be discussed with greater details in Section 2.3.4. Those two stages of arbitration play a significant role in the delay experienced by competing packets since an unfair arbitration policy can heavily penalise some packets. Usually, routers use a simple and local arbitration policy to decide which packet should be scheduled next. *Round-Robin* is one of the most popular arbitration policies that give fair chances to all the competing packets in an order. Another popular arbitration policy called *Aging* tries to avoid starvation by selecting the oldest packet from the competitors.

2.3.3 Flow Control

Flow control manages NoC transfer and decides when a packet (flit) can traverse the next link on its path, and when it has to wait at some router. An efficient flow control mechanism minimises latency at low NoC traffic and maximises throughput at high NoC traffic. These goals can be achieved by ensuring that NoC resources are not idle while the flits are waiting to use them. While topology and routing algorithm decides the theoretical latency and throughput, flow control determines how close to that expectation an NoC can operate.

There are two broad classes of flow control mechanisms: circuit switching and packet switching. In circuit-switched NoC, a dedicated connection is set up between the source and destination by reserving all the links of the path. Hence, data can be transferred with

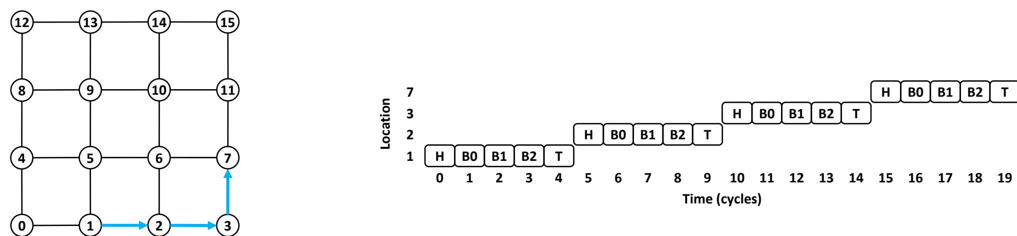


Figure 2.5: Example of a store-and-forward flow control

lowest possible latency without any contention or delay at the routers. However, there is a delay associated with setting up and tearing down the dedicated connection. Moreover, if significant amount of data is not transferred through the dedicated connection, reserved links remain underutilised, reducing the throughput. In packet-switched NoC, the source directly injects packets into the NoC for the destination without setting up any connection. These packets reserve links hop-by-hop all the way till the destination and waits at intermediate routers during contention. Thus packet-switched NoC avoids dedicated connection overhead and improves link utilisation. Modern NoC based TCMPs prefer packet-switching based flow control mechanism as it improves resource utilisation and maximises throughput [39].

Packet-switching based flow control mechanisms can be broadly classified into the following categories, based on how NoC resources (router buffers and links) are allocated:

- **Store-and-Forward:** Each router waits until an entire packet has been received before forwarding any part of the packet (i.e. flit) to the next router [39]. As a result, long delays are incurred at each hop, which makes store-and-forward unsuitable for modern NoC based TCMPs that target minimum latency. Moreover, store-and-forward requires large enough router buffers to be able to hold an entire packet which is another undesirable in modern NoC based TCMPs. A 5-flit packet travelling from source ① to destination ⑦ using store-and-forward is shown in Figure 2.5. The underlying NoC uses 2D-Mesh topology (Figure 2.3c) and employs XY-DOR algorithm (Figure 2.4a).

- **Virtual Cut-Through:** To reduce the delay experienced by packets at each hop, virtual cut-through allows forwarding flits to the next router before the entire packet is received at the current router [42]. Packet latency in virtual cut-through is significantly reduced over store-and-forward, as shown in Figure 2.6. 20 system clock cycles are required to send a packet in store-and-forward (Figure 2.5), whereas only 8 system clock cycles are required in virtual cut-through (without delay). However, NoC resources are still allocated at packet granularity. For example, a packet is forwarded to the next router only if the buffer in that router can hold the entire packet. As shown in Figure 2.6, the entire packet is delayed (shown in red) while travelling from router ② to router ③

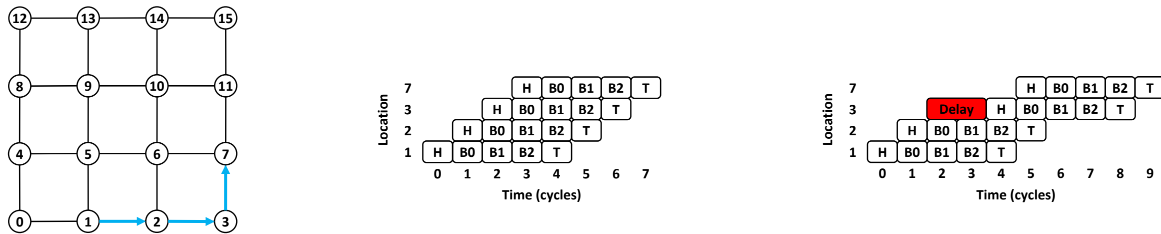


Figure 2.6: Example of a virtual cut-through flow control

even though the buffer in router ③ can hold some flits (and not all) of the packet. A flit is forwarded to router ③ only after ensuring that the buffer can hold all the 5 flits.

- **Wormhole:** Similar to virtual cut-through, wormhole also allows forwarding flits to the next router before the entire packet is received at the current router [43]. However, unlike store-and-forward and virtual cut-through, wormhole allocate NoC resources at flit granularity. Thus a flit can be forwarded to the next router if the buffer can hold it irrespective of whether all 5 flits of the packet can be accommodated. So, wormhole can be used with relatively smaller router buffers, which is why it is the predominant flow control mechanism in modern NoC based TCMPs. Nevertheless, wormhole makes inefficient use of NoC links which reduces throughput. Flits of the same packet travel in sequence and all the links in the path are reserved until the last flit (tail) reaches destination. Since wormhole allocates buffer at flit granularity, flits of the same packet can potentially be in multiple routers. So, when a packet is blocked, all the physical links held by that packet remains idle as other packets can not use them.

- **Virtual Channel:** Initially proposed as a solution for deadlock avoidance [44], Virtual Channels (VCs) are also used to mitigate Head-of-Line (HoL) blocking in flow control to improve throughput. All the previously discussed flow control mechanisms use a single buffer (queue) at each input port of the router (will be discussed in Section 2.3.4). As a result, when a packet at the head of the queue gets blocked, all the subsequent packets are stalled even when they wanted to take a different path, thus resulting in HoL blocking. VC is a separate queue, and multiple VCs can be associated with each input port of the router. These multiple VCs share the same physical link between two adjacent routers. When a packet holding a particular VC is blocked, other packets can still use the physical link through other VCs, thus avoiding HoL blocking. Technically, VCs can be applied to all the previously discussed flow control mechanisms.

TH-2538_156101018

2.3.4 Router Microarchitecture

Most of the features of NoC presented so far are implemented by the microarchitecture of a router. Figure 2.1 shows the microarchitecture of a state-of-the-art 5-ported NoC router. Each input port has buffers that are organised into separate VCs. These buffers are First In First Out (FIFO) queues usually implemented using Static Random Access Memories (SRAMs). Each input port connects to a crossbar switch which is able to provide non-blocking connectivity from any input port to any output port. There are 3 primary units inside an NoC router:

- **Route Compute (RC):** It directs the head flit of a packet to the appropriate output port and also dictates a valid VC within the selected input port in the next router. RC unit gathers destination-related information from the head flit and uses a routing algorithm (deterministic or adaptive) to compute the output port towards the next router.
- **VC Allocator (VA):** It arbitrates among all the head flits requesting the same VCs in the input ports of next routers and decides on winners. All the subsequent flits (body and tail) of the same packet will follow the winning head flit through the same VC. Round-Robin and Aging are two of the most popular arbitration policies in NoC.
- **Switch Allocator (SA):** It arbitrates among all the head flits requesting the same output ports through the crossbar switch and decides on winners. The winning head flits are allowed to traverse the crossbar switch and placed on the output ports towards their respective destination. SA unit employs either of the Round-Robin and Aging.

Usually, each unit of a router requires one system clock cycle to be completed. Early NoC router prototypes were modeled similar to traditional 5-stage off-chip network routers [39][45]. The number of stages is further reduced by pipelining and paralleling them. In fact, techniques like look-ahead routing [46] and speculative VC allocation [47] has made it possible to design as low as 2-stage, or even 1-stage NoC routers [48][49] for modern TCMPs.

2.4 Memory Hierarchy Overview

Most of the modern TCMPs employ a shared memory hierarchy, where the NoC interconnects different memory subsystems (L1 cache, LLC, directory, MC, etc.). Hence, the traffic passing through the underlying NoC are either data or control packets travelling between different memory subsystems. The key features that define a shared memory hierarchy in TCMPs are: organisation, coherence, and traffic through NoC. We describe each of these features to an extent necessary to understand the dissertation. A more comprehensive description of

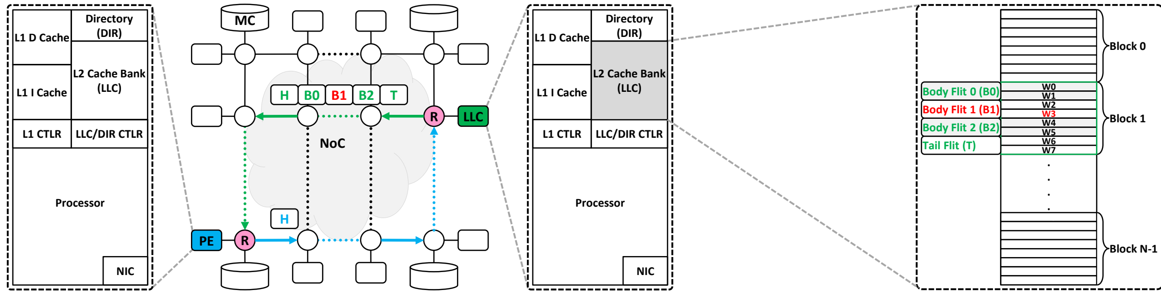


Figure 2.7: Data transfer between different levels of memory. A data request packet consists of a single head flit (H) carrying the header, whereas a data reply packet consists of a head flit followed by multiple body flits and ended by a tail flit (H, B0, B1, B2, T) carrying header and data. The critical word (W3) and the flit carrying it (B1) are shown in *red*.

memory hierarchy in general, as well as each of these features in particular, can be found in some of the classic books available in the existing literature [50][51][52][53].

2.4.1 Organisation

As shown in Figure 2.1, modern NoC based TCMPs generally have a private, write-back L1 cache memory in each tile (core) and a shared, write-back L2 cache memory distributed as banks across all the cores. L2 cache memory serves as the LLC and communicates with the MCs for off-chip data transfer. At any given point in execution, a processor usually requests for a single word called *critical word* from the memory hierarchy. Due to limited on-chip area and associated cost, the size of cache memories is very small in TCMPs. Hence, once in a while, the processor encounters an L1 cache miss on the requested word (critical word). The processor initiates a data transfer request from the next level of memory (LLC), i.e. a data request packet is sent to the appropriate LLC bank, as shown in Figure 2.7. The smallest unit of data transfer between different levels of memory is in *blocks*, which contains multiple words. Block-level transfer is preferred as it exploits spatial locality, avoids large tag arrays and improves Dynamic Random Access Memory (DRAM) row buffer utilisation [19]. So, even though the processor requests a single word, an entire block (containing the critical word) is replied from the LLC bank. A block B_i where $0 \leq i < N$ can be represented as:

$$B_i = \{W_0^i \mid W_1^i \mid \ldots \mid W_{c-1}^i \mid W_c^i \mid W_{c+1}^i \mid \ldots \mid W_{n-1}^i\} \tag{2.3}$$

where W_j^i is one of the many words ($0 \leq j < n$) that constitutes a block B_i. Usually, each W_j^i is a group of few bytes. W_c^i is the actual word requested by the processor (critical word).

Due to limited channel width, the smallest unit of transfer through the underlying NoC is in flits (flit $<<$ block). Hence, fetching an entire block (data reply packet) requires multiple

flits to be transferred from LLC to L1 cache, as shown in Figure 2.7. The cost of an L1 cache miss, also known as *miss penalty* is the time required to replace an existing block in L1 cache with the requested, incoming block. L1 cache miss penalty (MP_{L1}) can be given as:

$$MP_{L1} = t^{L1-LLC}_{Request} + T^{LLC}_{Access} + t^{LLC-L1}_{Reply} \quad (2.4)$$

where

$$T^{LLC}_{Access} = \begin{cases} T^{LLC}_{Hit} & \text{if LLC Hit} \\ T^{LLC}_{Miss} + MP_{LLC} & \text{if LLC Miss} \end{cases} \quad (2.5)$$

Here, T^{i}_{j} is the time taken by module i to complete a task j whereas, t^{i-j}_{k} is the time taken by message k to travel from module i to module j through the underlying NoC. For example, T^{LLC}_{Access} is the time taken by an LLC bank to access a block whereas, $t^{L1-LLC}_{Request}$ is the time taken by a cache miss request to travel from L1 cache to the corresponding LLC bank.

2.4.2 Coherence

In a shared memory hierarchy, multiple cores (processors) may try to read and write to a single address space (block) at the same time. Hence, the memory hierarchy employs a coherence protocol to make sure that data remains coherent across different memory levels. A coherence protocol allows either a single write or multiple reads to a particular block at any given point in time. While processors may read and write in words or other sizes, coherence is usually maintained at block granularity. There are two broad classes of coherence protocols:

- **Snooping Protocols:** In these protocols, any request is broadcast to all the coherence controllers to get a response. Requests travel on an ordered broadcast network which ensures that every coherence controller observes the same series of requests in the same order, i.e., there is a total order of coherence requests. This total order guarantees that all coherence controllers can correctly update the state of a requested block.

- **Directory Protocols:** In these protocols, a directory is maintained which keeps track of the location as well as state of the blocks. A request is unicast to the directory, which either responds or forwards the request to one or more controllers who can respond. Coherence requests are ordered in the directory. Thus directory protocols avoid both the ordered broadcast network as well as having each controller process every request.

Snooping protocols offer low-latency coherence transactions and a conceptually simpler design than directory protocols. However, due to the nature of broadcasting, snooping

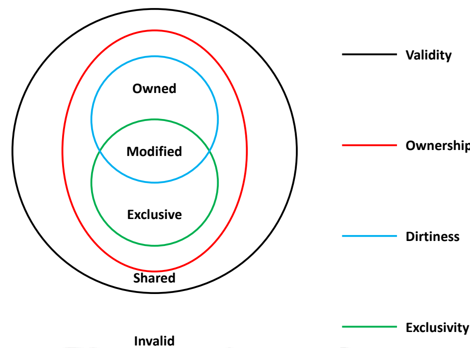


Figure 2.8: Venn diagram of classic five-state MOESI model

protocols are not scalable. Since they use unicasting, directory protocols scale well, but latency of some coherence transactions may be more. Additionally, the choice of coherence protocol also affects the on-chip communication infrastructure (NoC) in a modern TCMP.

Most of the popular coherence protocols use a subset of the classic five-state MOESI model introduced in late 80's [54]. These MOESI states refer to the states of blocks in a cache memory, and the fundamental 3 states are: M, S, and I, which are described as follows:

- **M**(odified): The block is valid, exclusive, owned, and potentially dirty. The L1 cache memory has the only valid copy of the block, and it must respond to the requests for the block. The copy of the block at LLC/main memory is potentially stale.
- **S**(hared): The block is valid but not exclusive, not owned, and not dirty. The L1 cache memory has a read-only copy of the block. Other cache memories may have valid, read-only copies of the block.
- **I**(nvalid): The block is invalid. The L1 cache memory either does not has the block, or it has a potentially stale copy of the block that may not be read or written to.

In addition to these fundamental states, MOESI model also specifies 2 other states: O and E, which are used to optimise certain situations. O and E are described as follows:

- **O**(wned): The block is valid, owned, potentially dirty, but not exclusive. The L1 cache memory has a read-only copy of the block, and it must respond to the requests for the block. Other cache memories may have valid, read-only copies of the block, but they are not owners. The copy of the block at LLC/main memory is potentially stale.
- **E**(xclusive): The block is valid, exclusive, owned, and clean. The L1 cache memory has a read-only copy of the block, and it must respond to the requests for the block. No other cache memories have a valid copy of the block. The copy of the block at LLC/main memory is up-to-date.

TH-2538_156101018

A Venn diagram of the classic MOESI model showing the characteristics of each state is illustrated in Figure 2.8. All the states besides I are valid, while M, O and E also have ownership. Both M and E states possess exclusivity, where no other cache memories have a valid copy of the block. Both M and O states indicate that the block is potentially dirty.

2.4.3 Traffic through NoC

All the traffic passing through the NoC are of different coherence transactions, either data or control messages (packets) travelling between different memory subsystems. Based on the employed coherence protocol, the traffic can be classified into different communication patterns. For example, in an N-core NoC based TCMP, the communication patterns of coherence protocols can be classified as 1-to-1, 1-to-M, and M-to-1, where M refers to multiple sources or destinations ($1 << M <= N$). 1-to-1 communication happens in unicast requests/responses exchanged between different cores. 1-to-M communication happens during broadcast and multicast requests. Whereas M-to-1 communication happens with acknowledgements and tokens. NoC is expected to efficiently handle all kinds of patterns.

Messages exchanged as part of a coherence transaction fall within different message classes. For example, based on the employed coherence protocol, message classes can be *request*, *forward*, *response*, *unblock*, etc. However, protocol-level deadlock is possible if a request for a block from an LLC bank is unable to enter the network as the LLC bank is expecting a response for a previous request, while the response is unable to reach the LLC bank since all queues in the network are full of waiting requests. To avoid such a scenario, coherence protocols require different message classes to use separate sets of queues within the network. This arrangement is made by using **Virtual Networks (VNs)** within the physical network. VNs are identical to VCs (refer Section 2.3.3) in terms of implementation, except that the number of VNs are fixed by the coherence protocol based on message classes. Additionally, each VN can have one or more VCs within each input port of the NoC routers.

Usually, control messages (request/forward/unblock) fit within a single flit, while data messages (response) require multiple flits. For example, if an NoC based TCMP uses 64B blocks and 128-bit flits, a data message will fit in 5 flits. This is why request (control) packets only have a head flit, whereas reply (data) packets have multiple flits (head, body and tail) carrying the data. Also, VCs within the request, forward and unblock VNs (control VCs) are of 1-flit depth, whereas VCs within the response VN (data VCs) often have more depth.

2.5 Evaluation Methodology

This section details the simulation infrastructure, applications and major performance metrics the dissertation has considered to evaluate and analyse the proposed architectures.

Table 2.1: System configuration. *Some of the parameters vary for specific architecture.

Processor	64 OoO x86 cores, 1.3GHz
L1 Cache	32KB, 8-way, private, split (instruction and data)
L2 Cache (LLC)	512KB×64 cores, 16-way, shared
Memory Bank	4; one located at each corner
Coherence	MESI/MOESI distributed directory
NoC	8×8 2D-Mesh topology, 128-bit channel width 3 Virtual Networks (VNs) 3 Virtual Channels (VCs) per VN 1-flit depth control VC, 4-flit depth data VC
Routing	2-stage routers (1.54ns), XY-DOR algorithm VC based wormhole packet-switching
Packets	1-flit for control packet, 5-flit for data packet
Word/Flit/Block	64-bit/128-bit/64B; 2-words/flit, 8-words/block
Benchmarks	SPEC CPU2006 (multi-programmed) PARSEC 3.0 and SPLASH-2x (multi-threaded)

2.5.1 Simulation Infrastructure

All the architectures proposed in this dissertation are modelled on event-driven gem5 simulator [55], and the system configuration is given in Table 2.1. The system configuration is similar to Intel Xeon Phi Processor 7235 [56] with shared and distributed L2 cache (LLC). Due to certain limitations in gem5, the exact configuration of Intel Xeon Phi Processor 7235 could not be modelled. Mostly, *GARNET* [57] (except in Chapter 3) and *Ruby* modules present inside the gem5 simulator are modified to establish a dynamic cooperation between NoC and memory hierarchy, respectively. Chapter 3 models NoC by modifying cycle-accurate trace-driven BookSim simulator [58]. The traces are generated from the memory hierarchy modelled by modifying the Ruby module of the gem5 simulator.

All the additional hardware-related area and power overheads are obtained using ORION 2.0 [59] at 65nm processor technology, DSENT [60] and McPAT [61] at 22nm processor technology, all at 1GHz operating frequency. Some of the additional hardware are also implemented in structural Register-Transfer Level (RTL) Verilog Hardware Description Lan-

TH-2538_156101018

guage (HDL) and synthesised in Synopsis Design Compiler using a Taiwan Semiconductor Manufacturing Company (TSMC) 65nm standard cell library, at 1GHz operating frequency.

To evaluate and analyse the performance, multi-programmed, as well as multi-threaded applications, are considered. For multi-programmed workloads, we consider SPEC CPU2006 benchmarks [62] to mimic a modern NoC based TCMP running multiple applications in parallel. We create different workload mixes (as many as 45) based on various characteristics of the benchmarks. By separately profiling each benchmark, a smaller representative window of instructions is chosen to have a tractable simulation time. For multi-threaded workloads, we consider PARSEC 3.0 [63] and SPLASH-2x [64] benchmarks to mimic a modern NoC based TCMP running multiple threads of a single application. A mix of (as many as 21) computation-intensive, communication-intensive and memory-intensive benchmarks is identified. We consider *sim-medium* input set of the identified benchmarks, and simulate their Region-of-Interest (RoI). In general, a multi-programmed workload is run for at least 320 million instructions (5 million per core, similar to [17]) after fast-forwarding and warm-up, whereas a multi-threaded workload is run for the entire RoI. For a relative comparison, most of the performance results are normalised with respect to a standard baseline architecture.

2.5.2 Performance Metrics

Though multiple performance metrics are considered to evaluate and analyse the performance of each of the proposed architectures, the most important of those metrics are the following:

- **Network Stall Time** (NST)**:** It is defined as the number of cycles a processor stalls waiting for a network packet. We prefer NST over *Packet Latency/Network Latency* as the former is a more appropriate metric to evaluate network-related slowdown in NoC based TCMPs [22]. Reduced NST implies better performance **[Lower the better]**.
- **L1 Cache Miss Penalty** (MP_{L1})**:** It is defined as the number of cycles required to replace an existing block in L1 cache with the requested, incoming block (refer Equation (2.4)). MP_{L1} reflects whether a proposed architecture is able to reduce memory access latency. Reduced MP_{L1} implies better performance **[Lower the better]**.
- **System Speedup** (S)**:** For multi-programmed workloads, S is defined as:

$$S = \frac{IPC_{Proposed}}{IPC_{Baseline}} \quad (2.6)$$

TH-2538_156101018

where $IPC_{Baseline}$ and $IPC_{Proposed}$ are the total Instructions Per Cycle (IPC) of the baseline and proposed architectures, respectively. Whereas, for multi-threaded workloads:

$$S = \frac{ExecTime_{Baseline}}{ExecTime_{Proposed}} \tag{2.7}$$

where $ExecTime_{Baseline}$ and $ExecTime_{Proposed}$ are the total execution time of the baseline and proposed architectures, respectively. We prefer execution time as multi-threaded workloads have synchronisation primitives like locks and barriers, which brings variation in IPC. Increased S implies better performance **[Higher the better]**.

2.6 Chapter Summary

This chapter provided the necessary background to the readers so that they can understand and appreciate the contributions of the dissertation. It started with the description of NoC communication infrastructure, followed by a familiarity with the memory hierarchy in modern TCMPs. The chapter concluded by talking about the evaluation methodology this dissertation has considered to evaluate and analyse the performance of the proposed architectures.

The next chapter describes our work on critical packet based prioritisation in NoC.

Indian Institute of Technology Guwahati

TH-2538_156101018

TH-2538_156101018

Chapter 3

Critical Packet Prioritisation

This chapter describes the first work of the dissertation proposed for critical packet prioritisation in NoC based TCMPs. A criticality defining metric for packets is calculated at run-time and used during routing and arbitration decisions at the NoC routers. When compared with a state-of-the-art architecture, the proposed work significantly reduces network stall time.

3.1 Introduction

The most fundamental challenges in the design of general-purpose TCMPs include devising efficient resource sharing and scheduling policies. Behaviour and interference of applications for basic shared resources like NoC [16][17][65], LLC [66][67][68] and memory bandwidth [69][70][71] are explored in different capacities. NoC trivially becomes the most critical shared resource as it is the communication backbone for the entire system. Even other shared resources, including LLC and memory bandwidth, are dependent on NoC directly or indirectly. However, NoC has various hidden and indirect but significant performance defining factors, which should be explored further. Some of the important factors are queueing delay, MLP, irregular traffic patterns, unpredictable application interferences, etc. These network-level factors can have a significant impact on the application-level performance.

Conceptual view of an NoC based TCMP with data request and reply packets is shown in Figure 3.1. Tile, core and PE, more or less mean the same thing and are used interchangeably throughout the text. Each core may run an independent application, and multiple applications may run concurrently across different cores. When an application encounters an L1 cache miss, a request packet gets triggered, which consequently triggers a reply packet. In an NoC based TCMP, packets of different applications mainly interact (rather compete) with one another in the routers. The arbitration policy employed in these routers decides which packet (of which application) is to be prioritised over others when they request the same output port.

TH-2538_156101018

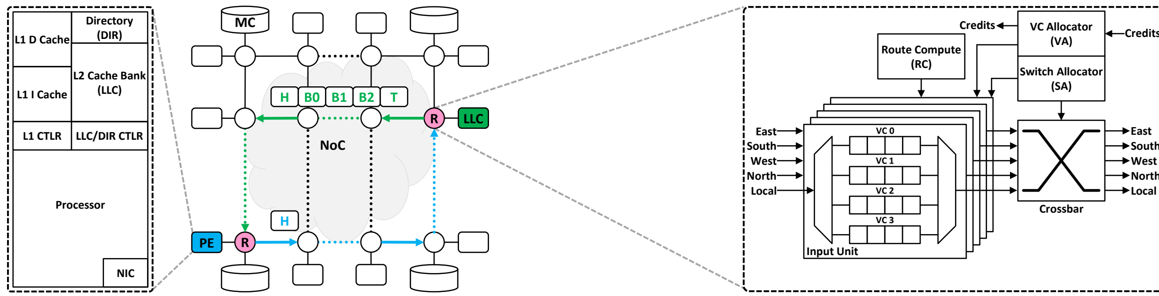


Figure 3.1: Conceptual view of an NoC based TCMP with 2-levels of on-chip cache memories. A data request packet (shown in *blue*) consists of a single head flit (H) carrying the header, whereas a data reply packet (shown in *green*) consists of a head flit followed by multiple body flits and ended by a tail flit (H, B0, B1, B2, T) carrying header and data.

Traditional *Round-Robin* and *Aging* arbitration policies are application oblivious; i.e., they treat packets of all the applications equally. However, applications can be heterogeneous in nature with different Quality of Service (QoS) requirements and hence, each of these packets will have a different impact on the application-level performance. Moreover, different packets of the same application may even have a differential impact on the performance. One of the main reasons for this differential impact is the presence of MLP in the outstanding requests. Servicing multiple memory requests in parallel reduces the application stall time, and criticality of each of these requests to the application depends on the degree of MLP.

Consider the Following Example: Assume that an application issues two network requests (cache misses), one after another, first to a distant tile in the network, and second to a closer tile. The application can continue execution only after the reply to these requests are received. The first request packet travels far and hence take more time to return, whereas the second request packet travels less and come back before the first packet. Even after the second reply packet arrives, the application continues to stall because the first reply packet is expected. Clearly, the second packet is less critical and can be delayed for multiple cycles without adding any stall to the application's execution. This is because the latency of second packet is hidden under the first packet, which takes more time. Thus, the delay tolerance of each packet can be different with respect to its impact on the application's performance.

We study the diversity and interference of packets to design data-aware NoC for general-purpose TCMPs. We differentiate packets based on a metric called *slack* [26], which is a measure of the packet's criticality. Slack of a packet is defined as the number of cycles the packet can be delayed in the network without affecting application execution [17]. Therefore, packets with available slack are non-critical compared to the packets with no available slack (*no-slack*). Increasing the latency of no-slack packets stalls application execution. We propose an NoC based TCMP that prioritises critical packets by a Slack-Aware Re-routing

TH-2538_156101018

(SAR) technique. Proposed SAR enable routers to prioritise lower slack packets over higher slack packets, like in Aergia [17]. But when two no-slack packets have a port conflict, we re-route one of them through an alternate minimal path towards destination. Results show that our prioritisation policy effectively improves application-level performance compared to the existing policies. Our main contributions of this work can be summarised as follows:

- **Run-Time Slack Estimation:** We enable a dynamic interaction between the L1 cache controllers and NoC to obtain the slack of incoming packets. Based on the MLP of the predecessor misses, slack is estimated for an L1 cache miss request packet at run-time, and the value is added in the packet header to be used for priority at the NoC routers.
- **Slack-Aware Re-routing:** We modify NoC routers to prioritise lower slack packets during routing and arbitration decisions and re-route no-slack packets without any delay, when the desired output port is unavailable. We adopt a look-ahead routing technique to facilitate re-routing of no-slack packets through alternate minimal paths.
- **Comparison with State-of-the-Art:** We evaluate the prioritisation policy in the proposed SAR architecture for both multi-programmed and multi-threaded applications. We compare the proposed SAR architecture with traditional Round-Robin and state-of-the-art Aergia [17] policy based architectures for application-level performance.

3.2 Motivation

Most of the modern NoC based TCMPs employ different MLP based techniques like OoO execution, runahead execution etc., to reduce the penalty of load misses for the executing applications. These techniques issue parallel memory requests with an intention to overlap future load misses with the current load misses. If the behaviour of an application shows MLP in the NoC, the latencies of outstanding packets overlap and introduce slack cycles.

3.2.1 Exploiting Slack and its Diversity

If the NoC routers are aware of the available slack, they can take routing and arbitration decisions by prioritising lower slack packets over their higher counterparts. We identify few cases where exploiting slack information of packets can reduce stall time of applications.

- **Case 1: Interference between Different Slack Packets:**

  Consider a 64-core NoC based TCMP as shown in Figure 3.2. Two applications, one in Core-A (tile 57) and the other in Core-B (tile 46) are running simultaneously. Core-A

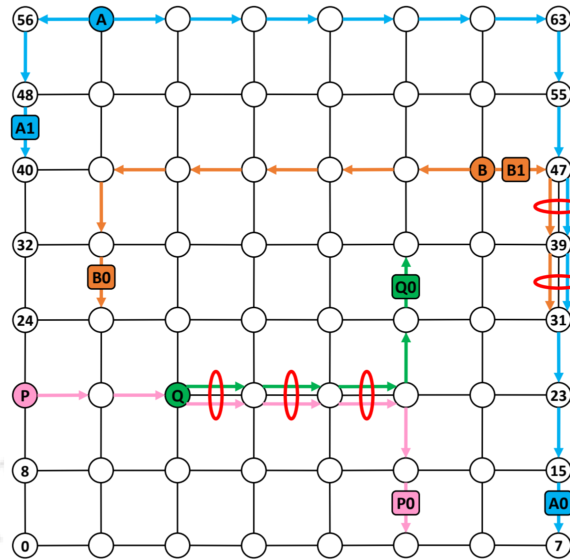


Figure 3.2: Conceptual illustration of the presence of slack in NoC packets

encounters two load misses and generates two packets (A0 and A1). The first packet A0 is sent to tile 7 and is not preceded by any outstanding packet; hence it has a latency of 13 hops and a slack of 0 hops. In the next cycle, the second packet A1 is sent to tile 40 with a latency of 3 hops. Since packet A1 is preceded (and thus overlapped) by the 13-hops packet A0, it has a slack of minimum 10 hops (13 - 3 hops). Similarly, for Core-B, the first packet B0 (with no predecessor) has a latency of 7 hops and a slack of 0 hops, while the second packet B1 has a latency of 3 hops and a slack of 4 hops.

Packets A0 and B1 interfere at 2 points (tiles 47 and 39) as shown with *red* circles. A traditional application oblivious, slack-unaware VC and switch arbitration policy that prioritises B1 over A0 degrades the application-level performance as A0 is more critical than B1. In contrast, if the NoC routers are slack-aware, they will prioritise packet A0 over B1 and reduce the stall time of Core-A without actually increasing the stall time of Core-B. Workload characterisation in Aergia policy shows that there exists sufficient diversity in slack of NoC packets across various benchmarks [17]. This observation has led to the exploration of slack-aware routing techniques in NoC [72][73].

- **Case 2: Interference between No-Slack Packets:**

  Consider another pair of applications in Core-P (tile 16) and Core-Q (tile 18), which are also running simultaneously with Core-A and Core-B on the same 64-core NoC based TCMP as shown in Figure 3.2. Core-P generates a packet P0 with a latency of 7 hops and a slack of 0 hops. While Core-Q generates a packet Q0 that has a latency of 5 hops and a slack of 0 hops. Both P0 and Q0 are no-slack packets and are most critical.

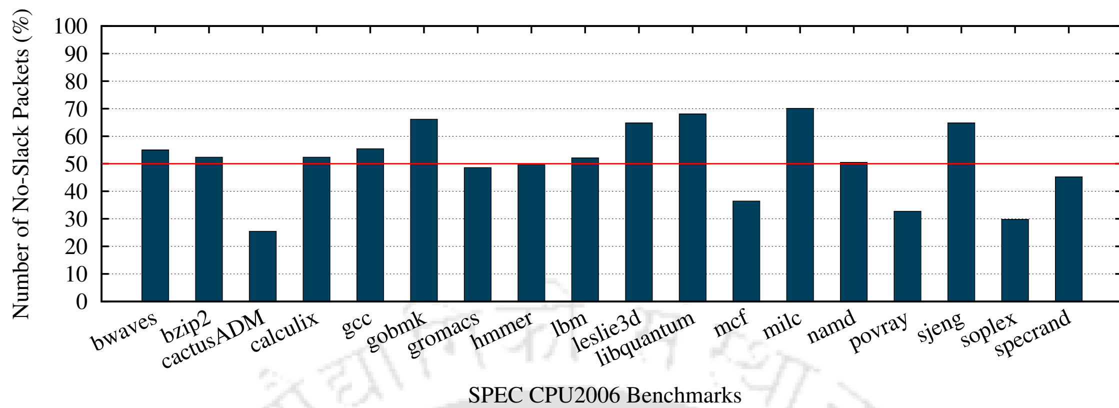


Figure 3.3: No-slack packets travelling through NoC routers

Src	Dest	Addr	. . .	Slack	Quadrant	Region	InterDest	Redirect	Local
				1-bit	1-bit	1-bit	3-bits	1-bit	1-bit

Figure 3.4: Modified message/packet header

Packets P0 and Q0 also interfere at 3 points (tiles 18, 19 and 20) as shown with *red* circles. Since both P0 and Q0 are equally most critical, delaying either of them degrades application-level performance of the other. In this case, as per Aergia [17] only one of the packets will get productive output port and the other will be delayed at least for 1 cycle. There are cases where no-slack packets are delayed for up to 8 cycles due to this port conflict. In contrast, if the NoC routers are slack-aware and can forward one of the no-slack packets through an alternate minimal path, both P0 and Q0 can progress in parallel. This reduces the stall time of both Core-P and Core-Q.

Figure 3.3 presents the percentage of no-slack packets in a representative set of SPEC CPU2006 benchmarks. This set is a mix of heterogeneous applications with different network-related characteristics. X-axis lists all the benchmarks, whereas Y-axis shows the percentage of no-slack packets in them. For example, in a 64-core NoC based TCMP running a multi-programmed benchmark *bwaves*, 54.96% no-slack packets are observed. A trend is clearly visible: *most of the presented benchmarks have more than 50% no-slack packets*. As a result, an NoC based TCMP with a simple slack-aware routing policy [17] can not guarantee optimal performance. In another critical observation, up to 34% cases of port conflict is found between two no-slack packets in the routers. We address this issue with a novel packet re-routing technique and attempt a policy where no-slack packets are not delayed.

TH-2538_156101018

Proposed Solution:

Prioritise lower slack packets over higher slack packets during routing and arbitration decisions, whereas re-route no-slack packets through an alternate minimal path.

3.3 SAR Architecture

In this section, we explain the working of the proposed SAR architecture. We start with how slack is estimated at run-time with the help L1 Cache Controllers (L1 CTLRs). Then, we describe the estimation of alternate minimal path at the NoC routers. Finally, we present the modified router microarchitecture along with its comparison and design challenges.

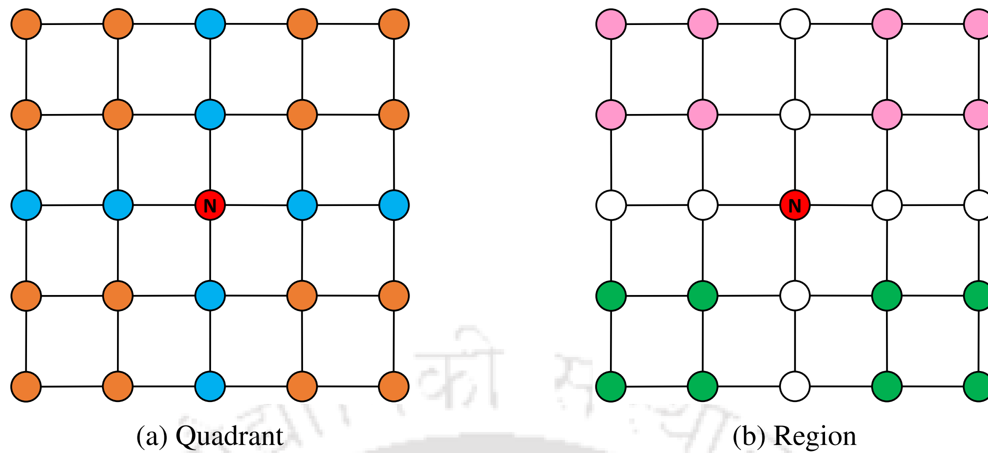
3.3.1 Slack Estimation

We estimate slack with respect to outstanding network transactions (L1 cache miss requests). We define slack of a packet as the difference between the maximum expected latency of its predecessor (i.e. any outstanding packet that was injected before this packet) and its own expected latency with proper adjustments on the injection time. This latency is based on the minimum distance to be traversed by a packet in the network (number of hops). Literature has other indirect metrics like L2 cache access status (hit or miss), number of miss predecessors (predecessors of a packet that are L2 cache miss) etc., which also correlates with slack and criticality of packets. However, such estimations become computation (hit/miss predictor) and storage expensive (list of miss predecessor). We intuitively assume all L2 cache access are hits and avoid the off-chip slack as it is irregular and cannot be quantised accurately.

We modify the structure of L1 Miss Status Handling Registers (MSHRs) to include predecessor related information. Before a cache miss request packet is injected into the network, slack is computed using this information from MSHRs. The slack is then quantised as a 1-bit flag (*Slack*) and stored in the message/packet header, as shown in Figure 3.4. All the no-slack packets are quantised as 1, and all the higher slack packets are quantised as 0. This *Slack* flag is used to enable priority-based routing and arbitration decisions at routers.

3.3.2 Minimal Path Estimation

A slack-based priority policy is used to make sure that a no-slack packet is not delayed in the intermediate routers. But when two no-slack packets compete in a router for the same output port, one has to be delayed. Rather than delaying a no-slack packet, we explore the



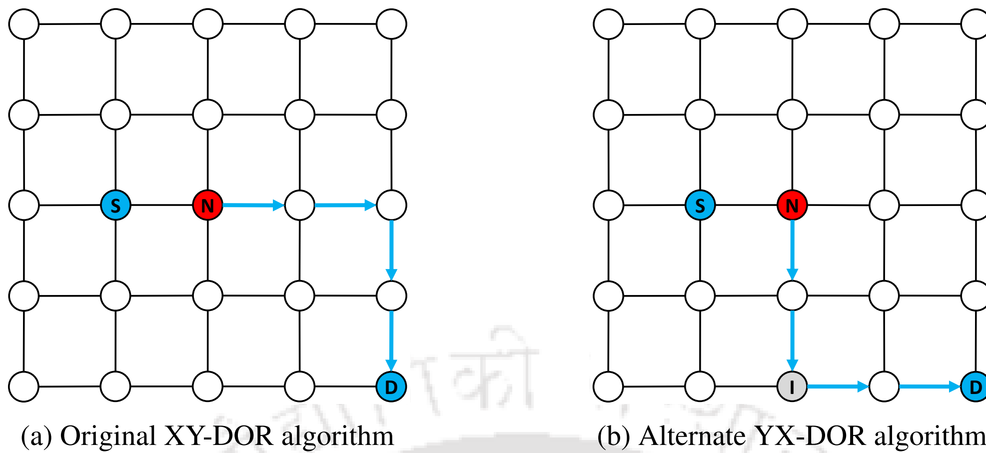
(a) Quadrant (b) Region

Figure 3.5: Quadrant and Region based on the position of destination

possibility of assigning another productive output port. Re-routing is a technique where a packet is forwarded to an intermediate router within the minimal quadrant of the current and destination routers. This makes sure that both the conflicting no-slack packets get a productive port. SAR routers use some additional flags to estimate alternate minimal path.

When a packet with destination router D leaves a source router S to N (N is neighbour of S), two 1-bit flags; *Quadrant* and *Region* is computed and quantised in its header. *Quadrant* and *Region* flags indicate the relative position of D with respect to N. If N and D are on an axis of the N, i.e. on the same row or same column, then the *Quadrant* flag is set to 0 else 1, as shown in Figure 3.5a. For example, if the destination router D is one of the *blue* cores, *Quadrant* flag is set to 0, else 1. If *Quadrant=1* then *Region=0* indicates D is in lower region of N and *Region=1* indicates D is in upper region of N. As shown in Figure 3.5b, if the destination router D is one of the *green* cores, *Region* flag is set to 0, else 1. If *Quadrant=0* then *Region* flag is irrelevant. This *Quadrant* and *Region* flag update happens on each router before the packet moves to its crossbar stage. *Region* helps to identify an alternate minimal path towards destination if the desired output port is not available at N. A packet with its *Quadrant* flag set to 1 can be re-routed at N through an alternate minimal path towards D.

Another flag called *InterDest* stores the address to be used for re-routing using an alternate minimal path. It is the last router in Y direction from N if YX-DOR algorithm is used to reach the destination of the packet. Figure 3.6b shows *InterDest* (router I) with an example and verifies its position on the minimal path towards destination. For our evaluation of an 8×8 2D-Mesh topology, a 3-bit *InterDest* along with a 1-bit flag *Redirect* is used. Only the column number (3-bits) is stored, as *InterDest* (router I) is on the same row as that of the actual destination (router D). If the *Redirect* flag is 0, *InterDest* flag is invalid (ignored).



(a) Original XY-DOR algorithm (b) Alternate YX-DOR algorithm

Figure 3.6: Possibility of alternate minimal path

Another 1-bit flag *Local* is used for deadlock prevention (will be discussed when deadlock is addressed in Section 3.3.4). A total of 8 additional bits are added as different flags (as given in Figure 3.4) to the packet header and communicated through the head flit (H) (refer Figure 3.1) to the NoC routers. Table 3.1 describes each of these flags with sufficient details.

3.3.3 Modified Router Microarchitecture

A conceptual block diagram of the proposed router microarchitecture that implements SAR is presented in Figure 3.7. Like any generic router employed in 2D-Mesh topology based NoCs, a SAR router also has 5 input and 5 output ports/channels; one from each direction (east, south, west, north) and one from the local core. However, north and south input ports have additional demultiplexers (shown in *green* and *blue*, respectively) to redirect packets to local input port. Multiplexers (shown in *red*) in local input port send the re-routed packets to the appropriate VCs. SAR routers use XY-DOR algorithm and VC based wormhole packet-switching, where only the head flit (H) participates in routing and arbitration decisions. The RC, VA, and SA units are same as that of generic routers. Two additional units **Packet Pre-processor (PP)** and **Look-Ahead Re-router (LR)** facilitates the technique of SAR.

3.3.3.1 Packet Pre-processor (PP)

This unit is an addition to the generic routers and works in parallel across all the input ports before the RC unit begins its operation. The additional flags in the modified packet header are used by the PP unit to initiate re-routing operations for every incoming head flits. This unit works in conjunction with the 4-bit *Output Port Select (OPS)* structure (refer Figure 3.7) to identify port conflicts of no-slack packets. PP unit identifies all no-slack packets in the

TH-2538_156101018

Table 3.1: SAR priority vector description

Flag	Value	Inference
Slack	0	All other packets
	1	No-slack packets
Quadrant	0	Destination is on the axis (X/Y) of neighbour (N)
	1	Destination is on one of the quadrants of neighbour (N)
Region	0	Destination is on one of the lower quadrants
	1	Destination is on one of the upper quadrants
InterDest	000 to 111	Last router in Y direction if YX-DOR algorithm is used
Redirect	0	Packet is not re-routed
	1	Packet is re-routed
Local	0	Ignore
	1	Packet is sent to local input port of *InterDest*

input port VCs and directs them towards productive output ports by enabling re-routing when required and if possible. The detailed working of the PP unit is presented in Algorithm 1.

Algorithm 1: Working of Packet Pre-processor (PP) unit

Input: $Input\ VCs$, $Packet\ Header$, OPS
Output: $Minimal\ Output\ Port$
Parameters: $n : Number\ of\ VCs$
Variables: $P_i : Packet\ in\ VC_i$

```
1  for ∀VC_i | VC_i ≠ NULL do
2      P_i[Local] ← RESET
3      if P_i[Slack] == SET ∧ P_i[Quadrant] == SET then
4          if OPS[E] == RESET ∨ OPS[W] == RESET then
5              OPS[E] ← SET ∨ OPS[W] ← SET
6          else if P_i[Region] == SET ∧ OPS[N] == RESET then
7              OPS[N] ← SET
8              Swap Column Bits of Dest with InterDest
9              P_i[Redirect] ← SET
10         else if P_i[Region] == RESET ∧ OPS[S] == RESET then
11             OPS[S] ← SET
12             Swap Column Bits of Dest with InterDest
13             P_i[Redirect] ← SET
```

Algorithm 2: Working of Look-Ahead Re-router (LR) unit

Input: $Input\ VCs, Packet\ Header$
Output: $Quadrant, Region\ and\ InterDest$
Parameters: $n : Number\ of\ VCs, Neighbour(Next)$
Variables: $P_i : Packet\ in\ VC_i, RowDiff, ColDiff, TempDest$

```
1  for ∀VC_i | VC_i ≠ NULL ∧ P_i[Slack] == SET do
2      if P_i[Dest] ≠ Next then
3          RowDiff ← Row of Dest − Row of Next
4          ColDiff ← Column of Dest − Column of Next
5          if RowDiff == 0 ∨ ColDiff == 0 then
6              Quadrant ← RESET
7          else
8              Quadrant ← SET
9              if RowDiff < 0 then
10                 Region ← RESET
11             else
12                 Region ← SET
13             TempDest ← Next + RowDiff ∗ Network Radix
14             InterDest ← Column Bits of TempDest
15     else if P_i[Dest] == Next ∧ Redirect == SET then
16         Replace Column Bits of Dest with InterDest
17         Redirect ← RESET
18         Local ← SET
```

TH-2538_156101018

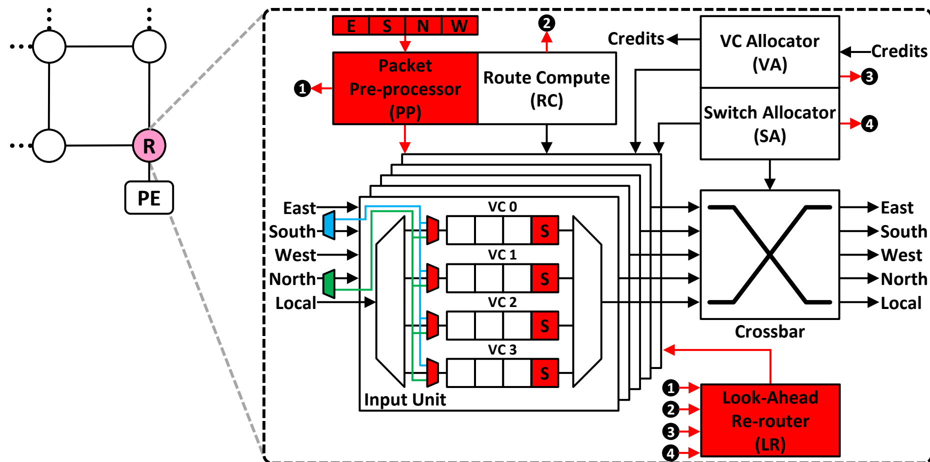


Figure 3.7: Proposed SAR router microarchitecture

3.3.3.2 Look-Ahead Re-router (LR)

This is another additional unit in the proposed SAR routers. LR unit uses the next router information from the RC unit of the current router and calculates alternate minimal path related flags in advance only to be used by the PP unit of the next router. Algorithm 2 describes the detailed working of the LR unit in the proposed SAR routers. LR unit works in parallel with the VA and SA units since the flags it calculates are used only by the next router. Hence, LR unit is not in the critical path of the router pipeline and incurs no additional delay.

In the proposed SAR routers, each VC has an extra priority field (S), which stores the 1-bit value of the *Slack* flag from the head flit (H) when it reserves the channel (refer Figure 3.7). This field is used by the subsequent body flits for priority-based arbitration. An illustrative example of packet re-routing from router Q is given in Figure 3.8. Packet P0 is re-routed in an alternate minimal path through an intermediate destination I without any penalty.

3.3.4 Comparison and Design Challenges

We compare the effectiveness of our technique with Aergia [17] that estimates slack in packets and prioritises lower slack packets over higher slack packet during VC and switch arbitration. Aergia also uses batching to prevent higher slack packets from starvation. All the same slack packets within a batch are treated as equal in Aergia and prioritised at random. However, we have seen in Figure 3.3 that no-slack packets dominate almost across all the presented benchmarks. Hence, Aergia suffers from performance degradation when one no-slack packet is prioritised over the other. In contrast, our proposed SAR architecture works similar to Aergia to prioritise lower slack packets but also re-routes no-slack packets through alternate minimal paths when required. We do not use batching to prevent starvation as our

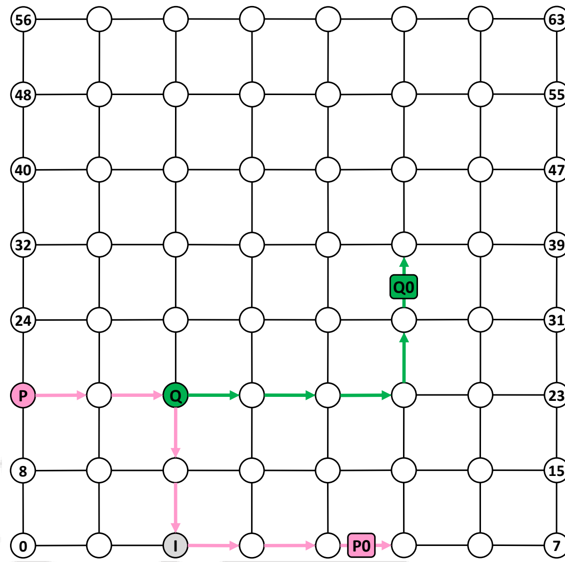


Figure 3.8: Illustrative example of slack-aware re-routing

1-bit slack based priority does not add any significant unfairness to the SAR architecture. Our proposal can be used as a compliment with any other packet prioritisation technique.

Starvation: When we evaluate the proposed SAR architecture, we observe that our 1-bit slack based priority does not add any significant unfairness to the system when compared to traditional Round-Robin policy. Thus, we do not use any additional metric for starvation prevention. Our policy can always be extended with techniques like batching [16][17].

Livelock: In the proposed SAR routers, a packet always travels on a minimal path towards destination whether or not it is re-routed. Lower slack packets are prioritised over higher slack packets and no-slack packets are re-routed through alternate minimal path; but forward progress is always ensured. Hence, the proposed SAR architecture is livelock free.

Deadlock: Proposed SAR routers use XY-DOR algorithm where packets are first routed in X direction followed by Y direction. However, when a packet is re-routed through an alternate minimal path, it takes an early Y-direction as shown in Figure 3.8 (packet P0 takes routers Q, and then I towards destination). After reaching *InterDest* (router I), if the packet attempts to take X-direction again, then it violates XY-DOR algorithm, which may lead to deadlock. To prevent this situation, a 1-bit flag *Local* is added in the modified packet header (refer Figure 3.4). When *Local* flag is set to 1, the packet after reaching router I is sent to one of the local input port VCs. The demultiplexers placed in north and south input ports will extract the packet and add it to one of the local input port VCs via the multiplexers. From this local input port, the packet can take X-direction towards destination like a newly injected packet. Thus, even though we use both XY-DOR and YX-DOR algorithms by incorporating the local input port VCs, we are able to prevent deadlock in the proposed SAR architecture.

Table 3.2: System configuration

Processor	64 OoO x86 cores
L1 Cache	32KB, 4-way, private, split (instruction and data)
L2 Cache (LLC)	512KB×64 cores, 16-way, shared
Memory Bank	4; one located at each corner
Coherence	MESI distributed directory
NoC	8×8 2D-Mesh topology, 128-bit channel width 3 Virtual Networks (VNs) 4 Virtual Channels (VCs) per VN 1-flit depth control VC, 4-flit depth data VC
Routing	2-stage routers, XY-DOR algorithm VC based wormhole packet-switching
Packets	1-flit for control packet, 5-flit for data packet
Word/Flit/Block	64-bit/128-bit/64B; 2-words/flit, 8-words/block
Benchmarks	SPEC CPU2006 (multi-programmed) PARSEC 3.0 (multi-threaded)

3.4 Performance Evaluation

We consider the following architectures for evaluation:

- **Round-Robin:** Uses traditional Round-Robin policy during VC and switch arbitration.
- **Aergia:** Uses state-of-the-art Aergia [17] policy during VC and switch arbitration.
- **SAR:** Prioritises lower slack packets and re-routes no-slack packets during arbitration.

3.4.1 Simulation Framework and Workloads

We model all the architectures on cycle-accurate trace-driven BookSim simulator [58]. The memory traces are generated by event-driven gem5 simulator [55]. The system configuration is presented in Table 3.2 for reference. We modify the structure of L1 MSHRs in Ruby inside gem5 to include latency-based slack estimation. L1 cache miss requests refer to the modified MSHR entries to get predecessor related information before being injected as a packet in the NoC. We modify BookSim to implement the proposed router microarchitecture that enables slack-aware re-routing policy for priority-based VC and switch arbitration. BookSim driven by the memory traces from gem5 constitutes the simulation framework for evaluation.

To evaluate and analyse the performance, we consider multi-programmed as well as multi-threaded applications. We create multiple workloads using applications (benchmarks)

TH-2538_156101018

Table 3.3: Benchmark characteristics

#	Suite	Benchmark	No-Slack (%)	MPKI
1	SPEC CPU2006	milc	70.12	Low
2		libquantum	68.03	Low
3		gobmk	66.15	High
4		bzip2	52.36	High
5		lbm	52.14	High
6		namd	50.49	Low
7		specrand	45.25	Low
8		povray	32.73	Low
9		soplex	29.82	Low
10		cactusADM	25.47	Low
11	PARSEC 3.0	x264	48.65	High
12		streamcluster	48.28	Low
13		ferret	46.72	High
14		blackscholes	44.59	Low

of varying network-related characteristics. We estimate the percentage of no-slack packets to identify the criticality of each of the benchmarks. We also calculate Misses Per Kilo Instructions (MPKIs) to estimate the network load contributed by the respective benchmarks. Both of these characteristics for the considered benchmarks are presented in Table 3.3.

For multi-programmed workloads, we consider SPEC CPU2006 benchmarks [62] to mimic a modern NoC based TCMP running multiple applications in parallel. We create 7 multi-programmed workload mixes (MP1 - MP7) based on the percentage of no-slack packets and MPKIs, as given in Table 3.4. These mixes run a random combination of 4 different benchmarks from their groups with 16 copies each (4×16: 64). For multi-threaded workloads, we consider PARSEC 3.0 benchmarks [63] to mimic a modern NoC based TCMP running multiple threads of a single application. We identify a mix of 4 computation and communication intensive benchmarks (MT1 - MT4), as given in Table 3.4. They are run individually as a 64-thread workload on all the 64 cores (1 thread/core) of an NoC based TCMP (1×64: 64). Altogether, we have 11 workloads to evaluate the performance, 7 multi-programmed benchmark mixes and 4 multi-threaded benchmarks. For a relative comparison, all the results are normalised with respect to the baseline (Round-Robin) architecture.

TH-2538_156101018

Table 3.4: Workload mixes

Mix	Benchmarks	Copies	Characteristics
MP1	bzip2, lbm, milc, libquantum		50% high MPKI, 50% high no-slack
MP2	bzip2, lbm, cactusADM, soplex		50% high MPKI, 50% low no-slack
MP3	specrand, namd, milc, libquantum		50% low MPKI, 50% high no-slack
MP4	specrand, namd, cactusADM, soplex	4×16: 64	50% low MPKI, 50% low no-slack
MP5	milc, libquantum, cactusADM, soplex		50% high no-slack, 50% low no-slack
MP6	milc, libquantum, gobmk, povray		75% high no-slack, 25% low no-slack
MP7	cactusADM, soplex, gobmk, povray		25% high no-slack, 75% low no-slack
MT1	blackscholes		
MT2	ferret	1×64: 64	Runs with 64 threads
MT3	streamcluster		
MT4	x264		

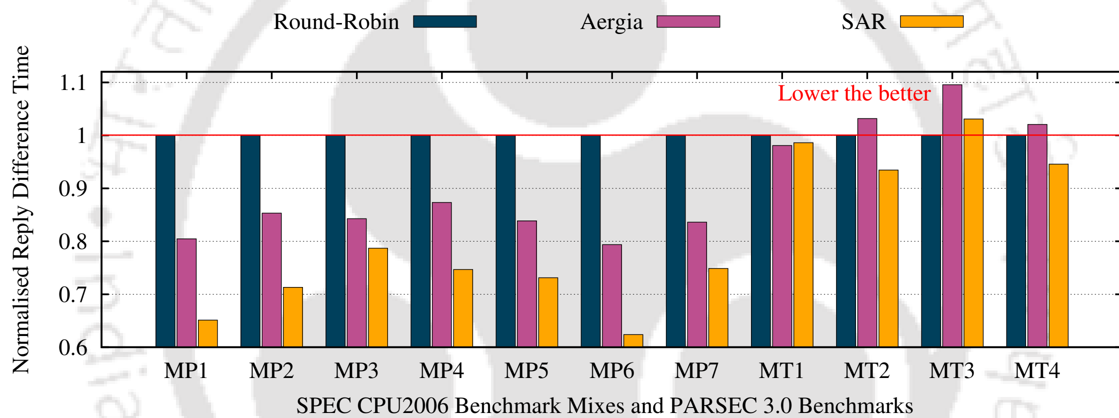


Figure 3.9: Reply difference time

3.4.2 Result Analysis and Discussion

3.4.2.1 Reply Difference Time

It is defined as the number of cycles between the arrival of first and last flits of the reply packet at the requesting core (executing the application). Reply difference time shows how long an application may stall due to the delayed arrival of the remaining flits of a packet after the head flit has arrived. Lower reply difference time implies better performance. Since we consider 128-bit flits and 64B cache blocks, every 1-flit cache miss request packet generates a 5-flit cache block reply packet (refer Figure 3.1 and Table 3.2) to be received by the requesting core. An Application can resume execution only after all the flits of the reply

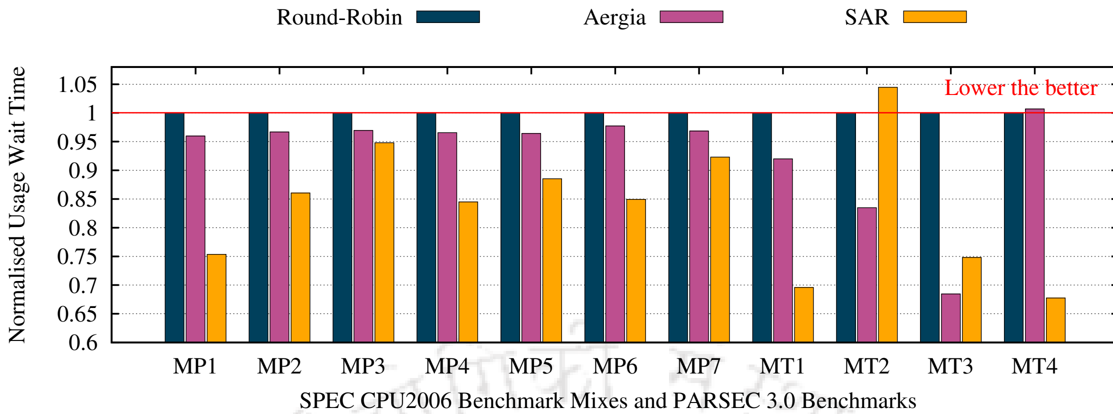


Figure 3.10: Usage wait time

packet reach the requesting core. Our proposed technique facilitates forwarding the body flits of the reply packets as soon as possible using the priority field (S) of the input port VCs.

Figure 3.9 shows the normalised reply difference time with respect to the Round-Robin architecture. In the proposed SAR architecture, reply flits get forwarded without any interleaving. Due to which the body and tail flits reach the requesting core at the earliest without much delay. Even though Aergia architecture also has good performance, SAR achieves an average of 14% reduction in reply difference time over Aergia in multi-programmed workloads. Since there is not much slack diversity in multi-threaded workloads (refer Table 3.3), all the packets are more or less equally critical. Round-Robin performs better than Aergia (or even SAR for MT3) because of the lack of slack diversity. The unnecessary level of slack-based priority and negative effects of batching is the reason for this behaviour.

3.4.2.2 Usage Wait Time

It is defined as the number of cycles a reply packet waits from its arrival at the destination (requesting core) until being used by the executing application. Usage wait time implies how early or late reply packets arrive at the requesting core. Lower usage wait time implies better performance. A reply packet has usage wait time if it reaches the requesting core earlier than necessary. While the packet has reached, at least one of its predecessors is still in the network. This scenario might happen by penalising peer packets during port conflict at the intermediate routers. Another possibility of having a usage wait time is that the packet may have a very high slack which is not fully exploited during port conflicts at the routers.

Figure 3.10 shows the normalised usage wait time with respect to the Round-Robin architecture. In the proposed SAR architecture, a higher slack packet can never delay a no-

TH-2538_156101018

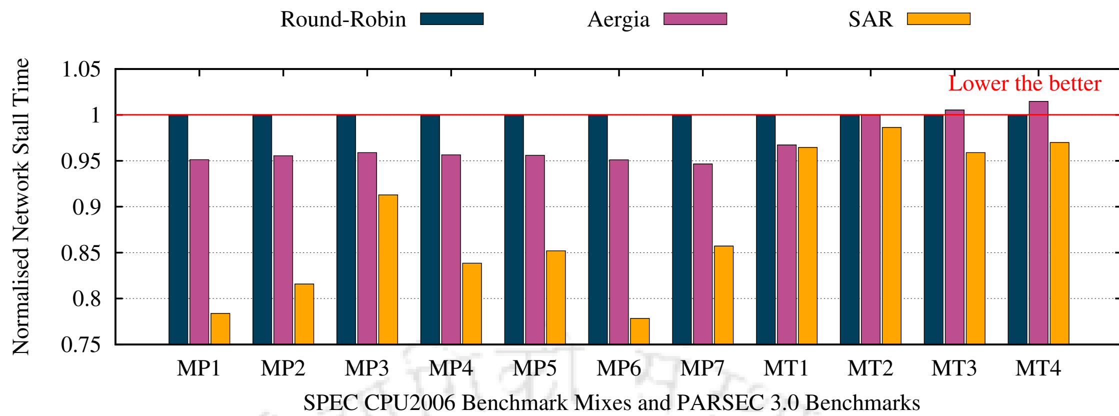


Figure 3.11: Network stall time

slack packet. SAR achieves significant reduction in usage wait time for the multi-programmed workloads. It varies from 5% reduction in MP3 to 25% reduction in MP1 when compared with the Round-Robin. With respect to Aergia architecture, an average of 10% reduction in usage wait time is achieved by the proposed SAR. Even for multi-threaded workloads, SAR perform much better than Aergia, which uses multi-bit slack value with batching. Results show that even with single-bit slack value without batching, we can avoid starvation. In our case, any slack above 0 whether it is between 1 and 5 (low slack) or above 5 (high slack), all are represented as high slack packets. In MT2 workload, we find that low slack packets are over-penalised leading to higher usage wait time than Round-Robin and Aergia. In MT1, MT3 and MT4 workloads, all no-slack and high slack packets are received just in time.

3.4.2.3 Network Stall Time

It is defined as the number of cycles an application stalls waiting for a network packet. We assume all L2 cache accesses are hits as we want to identify the effects of NoC alone. Figure 3.11 shows the normalised network stall time with respect to the Round-Robin architecture. From the reductions in reply difference time and usage wait time, the reduction in network stall time with the proposed SAR architecture is intuitive. As Round-Robin is application oblivious, it delays packets during port conflicts irrespective of their load and criticality. SAR reduces stall time for all the workloads, but significant reduction can be observed for workload mixes of high load and no-slack rich benchmarks. A maximum reduction of 22% over Round-Robin and 18% over Aergia architecture is achieved for MP6 workload as it contains 75% of no-slack rich benchmarks (refer Table 3.4). Similarly, for MP1 workload, we achieve a significant reduction as it is a mix of high load and no-slack rich

benchmarks. Whereas, a reduction of only 5% over Aergia is observed for MP3 workload as it has fewer port contentions due to low rate of packet injection (low MPKI benchmarks).

For multi-threaded workloads, due to the inherent Dynamic Non-Uniform Cache Architecture (D-NUCA) based assignment of L2 cache address space, majority of the L1 cache misses need not travel far to reach the corresponding L2 cache bank. This behaviour results in either no-slack packets or very low slack packets. Hence, there are limited opportunities to apply a slack-aware re-routing technique leading to only a marginal reduction in network stall time with the proposed SAR architecture. Even Aergia could not achieve much improvement.

3.5 Overhead Analysis

3.5.1 Timing Overhead

The proposed SAR microarchitecture is implemented in Verilog and synthesised in Synopsis Design Compiler with TSMC 65nm standard cell library to obtain the timing characteristics. The NoC is operating at 1GHz frequency with an inter-router link delay of 1 cycle. We use the traditional 2-cycle pipelined router with the first cycle for PP and RC units (refer Figure 3.7). Even though PP unit is in the critical path, the combined combinational delay of PP and RC units is 7% lower than the combined combinational delay of VA and SA units that constitute the second cycle. Our LR unit works in parallel to VA and SA units. The experimental observation that VA and SA stage determines the pipeline latency is already established [74]. Hence, SAR routers can be operated with the same pipeline frequency.

3.5.2 Storage, Area and Power Overhead

We use 8 additional bits (as different flags) in the packet header (refer Figure 3.4) to facilitate the working of the proposed SAR architecture. Our NoC uses 128-bit flit channel (refer Table 3.2), but a typical packet header (head flit) uses fewer bits ($\approx$ 64 bits) for its operation. So, we can accommodate the additional 8 bits in the head flit without any storage overhead.

Our additional units incur a router area overhead of 1.70% and static (leakage) power consumption of 2.10%. We compute the dynamic power dissipation estimates of SAR routers using ORION 2.0 [59] at 65nm processor technology. Dynamic power consumption of NoC using SAR is 7.5% lower than using Aergia routers due to effective re-routing and reduction in latency of no-slack packets. This compensates for the minor hardware and area overhead.

TH-2538_156101018

3.6 Related Work

Criticality: Available literature has proposals that target criticality of data and instructions. Inherently, slack is more powerful than just criticality, as it also dictates the degree of criticality for a data/instruction (packet). The concept of slack is originally taken from network analysis [75] and applied to microarchitecture. In one of the first attempts, Casmira and Grunwald proposed using dynamic scheduling slack to control microarchitectures through clustered voltage scaling [76]. Fields et al. [26] claimed to have measured the slack accurately and used it as an input for guiding control policies in processors. Apart from slack, Subramaniam et al. [77] used the observation that a load instruction with more dependents could be on the critical path. They designed a load criticality predictor and proposed six criticality based optimisations for x86 processors. By informing the memory controller about which loads are urgent for the processor, Ghose et al. [78] proposed to make better scheduling decisions and improve performance. Another recent work defines data criticality as the promptness with which an application uses data after it is fetched from the memory [19]. Cache miss criticality is explored with different MLP based proposals [79][80]. Memory scheduling techniques are explored with bank-level parallelism [81][82]. Recently, slack-based criticality using NoC has gained momentum due to the emergence of heterogeneous applications running on modern TCMPs with different QoS requirements. Hence, slack-based criticality is explored again for both performance and power optimisation in NoC based TCMPs [17][72][73]. However, the presence, degree and impact of no-slack packets are not explored earlier.
Prioritisation: Other than the traditional Round-Robin and Aging based prioritisation, available literature also has QoS [83][84][85] and application-aware [16][65][86] prioritisation policies. There are prioritisation proposals based on latency-sensitivity of NoC packets [87][88]. While most of the available proposals aimed for guaranteed service or fairness, our aim is to reduce application-level stall time, which improves the system performance. Furthermore, available proposals assign static priority to improve real-time performance. In contrast, our proposal computes dynamic priority for VC and switch arbitration.

3.7 Chapter Summary

By understanding the diversity and interference of packets, we propose a policy that prioritises critical packets in NoC based TCMPs. We present SAR architecture, a slack-aware re-routing technique that prioritises lower slack packets over higher slack packets and re-routes no-slack packets through alternate minimal path towards the destination. Experimental analysis shows that our policy improves application-level performance over existing policies for both

multi-programmed and multi-threaded workloads. The performance gain is achieved with only a negligible area and power overhead. We believe that the proposed SAR routers can be a good design alternative for modern NoC based TCMPs that run time-critical applications.

A data reply packet carries a cache block containing multiple words. The next chapter goes one level deeper and presents our work on critical word based prioritisation in NoC.

TH-2538_156101018

Chapter 4

Critical Word Prioritisation

This chapter presents our work on critical word prioritisation in NoC based TCMPs. A processor usually requests for a single word called critical word from the memory hierarchy, but an entire block (multiple flits) is fetched. The proposed work prioritises flits carrying the critical words to reach destination at the earliest, which significantly reduces miss penalty.

4.1 Introduction

At any given point in execution, a processor usually requests for a single word, called *critical word* from the cache memory [28]. When the processor encounters a cache miss on the critical word, a data transfer from the next level of memory is triggered. Data transfer between different levels of memory is always in the unit of blocks (multiple words). Block-level transfer exploits spatial locality, avoids large tag arrays and improves DRAM row buffer utilisation [19]. Hence, an entire block containing the critical word is transferred from memory to the cache; for the processor. In conventional uni-core and multi-core processors with bus based communication, a block is transferred as a continuous stream of bytes (words). However, in TCMPs with NoC based communication, the transfer is not continuous. The smallest, discrete unit of transfer is called a *flit* that carries multiple words. Since on-chip data transfer bandwidth is limited to flit size, fetching an entire block requires multiple discrete flits to be transferred from next level of memory to the cache. When all the flits of a block are received, it is constructed, and the critical word is forwarded to the processor to resume execution. The processor is usually stalled during the entire data transfer and can resume execution only after it receives the critical word from the cache. Aggressive OoO processors experience fewer stalls due to MLP. However, simpler OoO processors used in most of the modern TCMPs cannot hide the miss latency beyond an extent, thus suffers from stalls.

TH-2538_156101018

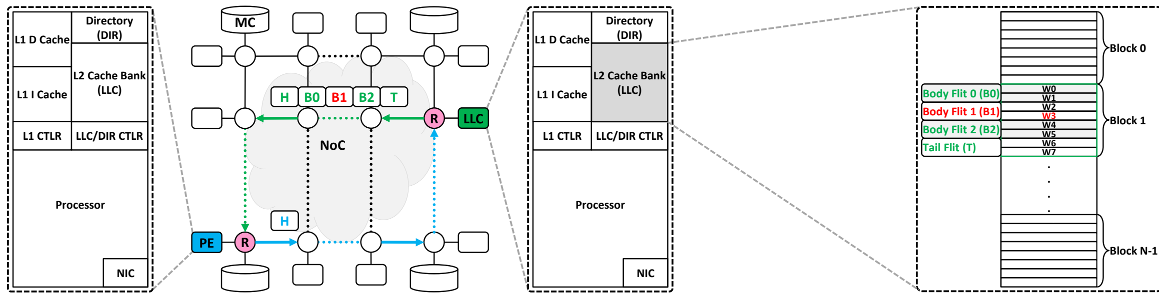


Figure 4.1: Data transfer between different levels of memory. A data request packet consists of a single head flit (H) carrying the header, whereas a data reply packet consists of a head flit followed by multiple body flits and ended by a tail flit (H, B0, B1, B2, T) carrying header and data. The critical word (W3) and the flit carrying it (B1) are shown in *red*.

The most popular memory access optimisations to reduce miss latency (penalty) of the critical word are *Early Restart (ER)* and *Critical Word First (CWF)* [28]. In ER optimisation, as soon as the critical word is received at the cache, it is forwarded to the processor to resume execution; without waiting for the entire block. Whereas in CWF optimisation, the critical word is sent out of order to be received as the first word in the cache to resume processor execution at the earliest. Both of these optimisations are very effective in reducing miss penalty and will be explained with greater details in Section 4.2. In modern NoC based TCMPs where a single block transfer requires multiple flit transfer, these optimisations can yield even better benefits in terms of improved memory access latency. However, both ER and CWF optimisations are NoC oblivious and hence, when employed in NoC based TCMPs neglect the concept of flit based discrete transfer. As shown in Figure 4.1, flits travel through multiple NoC routers to reach their destination and thus experience unknown router delay along the way. Since the transfer is discrete, during NoC congestion, the delay (latency) in arrival of flits of a block in the cache may vary significantly. Hence, a control unit checking for the critical word in ER/CWF optimisation may have to wait indefinitely to resume processor execution. As a result, the effectiveness of these optimisations is less in NoC based TCMPs; quite opposite to the initial intuition. If ER and CWF like memory access optimisations are made aware of the nature of NoC based communication, they may perform better with reduced miss penalty and improved overall system performance.

In this work, we explore the design of NoC aware memory access optimisations to reduce miss penalty of critical word. We start by observing and understanding the pattern of critical words requested from the memory. In a first of its kind critical word based profiling of memory requests, we learn that critical words are usually located at the beginning of the requested blocks. Then we present the pattern of delays experienced by the flits carrying the critical words while travelling from source to their destination. Using the insights from

our observation, we propose a novel NoC aware implementation of the ER optimisation that reduces miss penalty even further and resumes processor execution at the earliest. Since critical words are located at the beginning of the requested blocks, we advocate that CWF optimisation might not be necessary. Our work makes the following primary contributions:

- **Critical Word Based Profiling:** In a unique profiling of L1 cache miss requests, we present the position of critical word within the requested block and corresponding flits for PARSEC 3.0 [63] and SPLASH-2x [64] benchmarks. We also show that flits carrying the critical words may be indefinitely delayed, and the time difference between the arrival of the first and last flit of a block at the requesting core is sufficiently high.
- **Critical Flit Identification:** We define *critical flit* as the flit carrying the critical word for a requested data block. We perform run-time analysis of data requests to identify critical flits in parallel with tag and data access at the cache controller without delay.
- **Critical Flit Prioritisation:** We propose an NoC-aware ER optimisation that prioritises critical flits during routing and arbitration decisions. The prioritisation policy intelligently favours the critical flits to avoid starvation for others. The critical flits reach destination as soon as possible to resume processor execution at the earliest.

4.2 Background

4.2.1 Cache Organisation in NoC based TCMPs

Modern NoC based TCMPs like Intel Xeon Phi Processor (2016) [89], Princeton Piton Processor (2015) [90], MIT Scorpio Processor (2014) [91], and others have private L1 caches and a shared L2 cache. In most of these processors, L2 cache is divided into multiple banks and distributed across all the tiles, as shown in Figure 4.1. Due to limited on-chip caching, L2 cache serves as the LLC and communicates with the MCs for off-chip data transfer. Since the private L1 cache is small, once in a while processor encounters cache miss on a critical word. As shown in Figure 4.1, a data request packet is sent to the appropriate L2 cache bank, as it is distributed. As we already know that data transfer is in the unit of blocks, so an entire block containing the critical word is replied from L2 cache bank to the L1 cache. In NoC based TCMPs, on-chip data transfer bandwidth is limited to flit size (flit $<<$ block). Hence, fetching an entire cache block requires multiple flit transfers from L2 cache bank to the L1 cache. The cost of an L1 cache miss, also known as *miss penalty* (MP_{L1}) is the time required

to replace an existing block with the requested, incoming block. MP_{L1} can be given as:

$$MP_{L1} = t_{Request}^{L1-L2} + T_{Access}^{L2} + t_{Reply}^{L2-L1} \tag{4.1}$$

where

$$T_{Access}^{L2} = \begin{cases} T_{Hit}^{L2} & \text{if L2 Hit} \\ T_{Miss}^{L2} + MP_{L2} & \text{if L2 Miss} \end{cases} \tag{4.2}$$

Here, T_j^i is the time taken by module i to complete a task j whereas, t_k^{i-j} is the time taken by message k to travel from module i to module j through the underlying NoC. For example, T_{Access}^{L2} is the time taken by an L2 cache bank to access a block whereas, $t_{Request}^{L1-L2}$ is the time taken by a cache miss request to travel from L1 cache to the corresponding L2 cache bank.

Designers of memory hierarchies have proposed a number of interesting optimisations for miss penalty reduction in conventional uni-core and multi-core processors [28].

4.2.2 Cache Optimisations for Miss Penalty Reduction

Conceptually, miss penalty is the time a processor is stalled after encountering a miss in the L1 cache. However, in modern OoO processors, the effective miss penalty is no longer the sum of concurrent cache miss times but the non-overlapped time a processor is stalled. The most popular and advance cache optimisations proposed for miss penalty reduction in modern processors are: (1) non-blocking caches, (2) ER and CWF, and (3) hardware and compiler pre-fetching. All of these optimisations have minimum power overhead [28], hence are implemented in almost all the modern conventional processors. However, all of the above optimisations have significant hardware cost and complexity [28]. Also, these optimisations are completely oblivious of the underlying communication infrastructure and simply expects it to be a continuous medium of transfer. But, NoC based TCMPs have a completely different communication infrastructure than bus-based conventional processors, and include indefinite transfer delays. Thus, implementing these optimisations on modern NoC based TCMPs pose a number of additional challenges with diminishing returns in performance. In this work, we explore the possibility of reducing miss penalty of critical words in NoC based TCMPs. Hence, our discussion is only limited to the ER and CWF memory access optimisations.

A cache block B_i where $0 \leq i < N$ can be represented as:

$$B_i = \{W_0^i \mid W_1^i \mid \ldots \mid W_{c-1}^i \mid W_c^i \mid W_{c+1}^i \mid \ldots \mid W_{n-1}^i\} \tag{4.3}$$

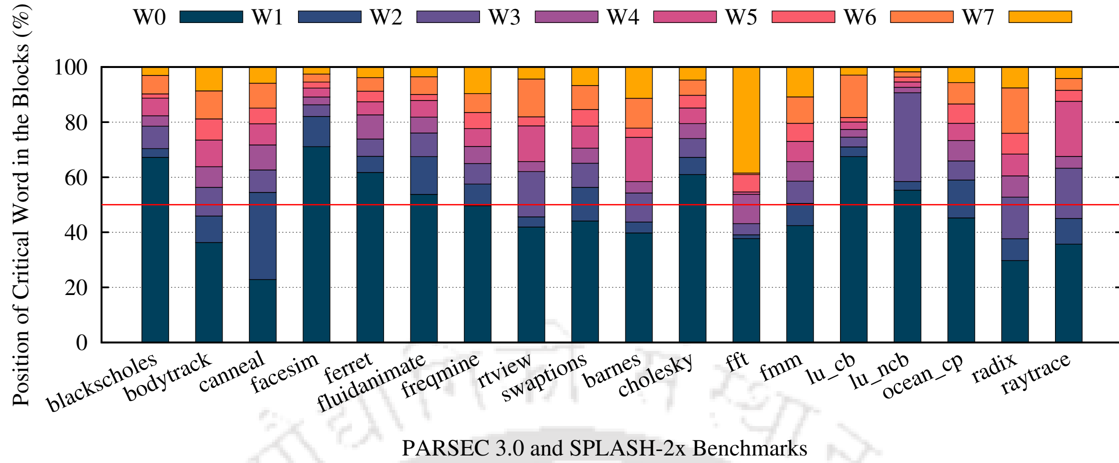


Figure 4.2: Position of critical word in the requested blocks

where W_j^i is one of the many words ($0 \leq j < n$) that constitutes a block B_i (refer Figure 4.1). Usually, each W_j^i is a group of few bytes. Let W_c^i be the word requested by a processor (critical word) that resulted in an L1 cache miss. Nevertheless, the entire block B_i containing the word W_c^i is fetched from L2 cache bank to the L1 cache. Only after the last word W_{n-1}^i of B_i is received at the L1 cache, W_c^i is forwarded to the processor to resume its execution. In Figure 4.1, $W3$ (shown in *red*) of *Block 1* is the critical word requested by the processor. Both, ER and CWF optimisations are impatient: don't wait for the entire block to be received at the L1 cache, forward W_c^i as soon as possible to resume execution and reduce miss penalty.

4.2.2.1 Early Restart (ER)

It proposes to fetch words W_j^is of block B_i in order but forward W_c^i to processor as soon as it is received at the L1 cache. While the processor resumes its execution early, rest of the words ($W_{c+1}^i \mid \ldots \mid W_{n-1}^i$) are fetched in the background in order. In Figure 4.1, as soon as $W3$ is received at the L1 cache, it is forwarded towards the processor (at the PE shown in *blue*) to resume its execution. Words $W4$, $W5$, $W6$ and $W7$ are fetched in the background in order.

4.2.2.2 Critical Word First (CWF)

It proposes to fetch the critical word W_c^i of block B_i first, in out of order to resume the processor execution at the earliest. Hence, W_c^i is the very first word to be received at the L1 cache followed by others ($W_c^i \mid W_0^i \mid \ldots \mid W_{c-1}^i \mid W_{c+1}^i \mid \ldots \mid W_{n-1}^i$). Since the order of words is broken, they need re-ordering when received at the L1 cache. In Figure 4.1, $W3$ is sent first from the L2 cache bank to the L1 cache to resume processor execution. Words $W0$, $W1$, $W2$, $W4$, $W5$, $W6$ and $W7$ are fetched in the background and re-ordered to construct *Block 1*.

4.3 Data Criticality in Applications

Applications running in any computing device can be classified as either multi-programmed or multi-threaded. However, multi-threaded applications utilise the resources of multi-core processors better with multiple tasks (threads) running concurrently and independently using those shared resources. To get an insight about how multi-threaded applications request critical words, we profile data requests of PARSEC 3.0 [63] and SPLASH-2x [64], the two most popular suites of multi-threaded benchmarks. Moreover, to understand why NoC is responsible for 60% to 75% of the miss penalty in multi-threaded applications, we specifically profile those data requests that resulted in an L1 cache miss. These requests travel through the underlying NoC to reach the corresponding L2 cache bank and get data, thus suffers with NoC related delay [9]. To the best of our knowledge, this critical word based profiling is a first of its kind for any multi-threaded applications. Figure 4.2 shows the average position of critical word in the corresponding blocks requested from the L2 cache. For example, during the entire run of *blackcholes* benchmark from PARSEC 3.0 suite, for 67.20% of the time, the first word ($W0$) is the critical word, for 3.17% of the time, the second word ($W1$) is the critical word and so on. There is a very interesting trend: *the first word is the critical word for most of the requested blocks*. This pattern is observed for the majority of the benchmarks of both PARSEC 3.0 and SPLASH-2x suites even though they are from diverse domains, including physics, finance, etc. However, the trend is unusual but not unreasonable, as the literature has proof of critical word regularity [92][93]. Literature states that it is reasonable to expect that data in a given region may be accessed in similar order on multiple occasions.

While explaining the individual memory access patterns for each of the benchmarks is beyond the scope of this work, we give some common characteristics that justify the pattern on the location of a critical word. Benchmarks that traverse through data arrays exhibit critical words near the beginning of the data block, most often to word 0 ($W0$). Also, the benchmarks having strided access with the smallest stride length of 0 have $W0$ as the critical word. Nevertheless, there are also benchmarks like *canneal* and *radix*, where the critical word is somewhat uniformly distributed. Benchmarks whose memory accesses are generated due to pointer chasing exhibits better distribution of the critical word. Based on these observations, specific memory access optimisation can be implemented for a class of applications exhibiting a specific behaviour to reduce the miss penalty of the critical word.

> Critical words are usually located at the beginning of the requested memory blocks.

Table 4.1: Position of critical word in the corresponding flits. Note that head flit (H) does not carry any words of the data block and hence its corresponding column is left empty.

#	Position	Benchmark	Flits of an incoming requested block				
			H	B0	B1	B2	T
1	PARSEC 3.0	blackscholes		70.37	11.94	7.95	9.74
2		bodytrack		45.90	17.91	17.38	18.81
3		canneal		54.49	17.20	13.38	14.93
4		facesim		82.06	7.06	5.40	5.48
5		ferret		67.60	15.04	8.56	8.80
6		fluidanimate		67.47	14.34	8.23	9.96
7		freqmine		57.50	13.67	12.32	16.51
8		rtview		45.59	20.08	16.26	18.07
9		swaptions		56.33	14.21	14.04	15.42
10	SPLASH-2X	barnes		43.70	14.69	19.40	22.21
11		cholesky		67.19	12.27	10.26	10.28
12		fft		39.10	14.69	7.14	39.07
13		fmm		50.46	15.19	13.94	20.41
14		lu_cb		70.99	6.37	4.32	18.32
15		lu_ncb		58.41	34.25	3.70	3.64
16		ocean_cp		58.96	14.35	13.27	13.42
17		radix		37.68	22.81	15.47	24.04
18		raytrace		45.03	22.51	24.03	8.43
Average				56.60	16.03	11.95	15.42

4.4 Motivation

In NoC based TCMPs, everything travels as packets (including the data blocks), which are further divided into multiple flits. A head flit (H) carries the packet header containing the routing information and does not carry any part of the data block. Multiple body flits (Bi), ended with a tail flit (T) carries the data block from source to the destination. Hence, a data block (of 8-words, refer Table 4.2) is transferred as a sequence of head flit, followed by three body flits (of 2-words each) and a tail flit (of 2-words) (H, B0, B1, B2, T). Table 4.1 presents the percentage of the critical words that fall on different flits of a requested data block.

To understand the observation in Table 4.1, when a requested data block in *blackscholes* benchmark is transferred through flits, 70.37% of the time, critical words are in flit B0, 11.94% in flit B1, 7.95% in flit B2 and 9.74% in the tail flit T. It can be seen that the first body flit (B0) contains the critical words most of the time. It was evident from the fact that most of the critical words are the first word of a data block (as given in Figure 4.2), and B0 carries the first two words of the block. Hence, ER like optimisation can be very effective in these kinds of applications. CWF optimisation involves sending the critical word in the first flit by allowing out of order travel. Since by their very nature, the majority of the applications have their critical word in the first flit itself, CWF optimisation might not be required. Avoiding CWF optimisation also brings in the advantage of avoiding the complexity of sending the critical word out of order and then reordering the words at their destination L1 cache.

From Section 4.1, we know that ER and CWF like optimisations are oblivious of the underlying on-chip communication infrastructure. These optimisations were introduced in the era of bus based on-chip communication, where a data block is transferred as a continuous stream of words. So the data block is transferred within a fixed time, and the processor could resume execution at the earliest as per the optimisation. However, modern TCMPs use NoC based on-chip communication where the data block is transferred as multiple flits in a discrete fashion. On their way to the destination, flits experience indefinite router delay due to on-chip congestion. Hence the effectiveness of ER and CWF like optimisations is less.

To understand the delay experienced by incoming flits, we conduct an experimental analysis for PARSEC 3.0 and SPLASH-2x benchmarks, as given in Figure 4.3. We consider a metric called *Reply Difference Time (RDT)* [27], which calculates the time difference between arrival of the first flit (H) and last flit (T) of a data block in the requesting core.

$$T^{P_i}_{RepDiff} = T^{F^i_{Tail}}_{Arrival} - T^{F^i_{Head}}_{Arrival} \tag{4.4}$$

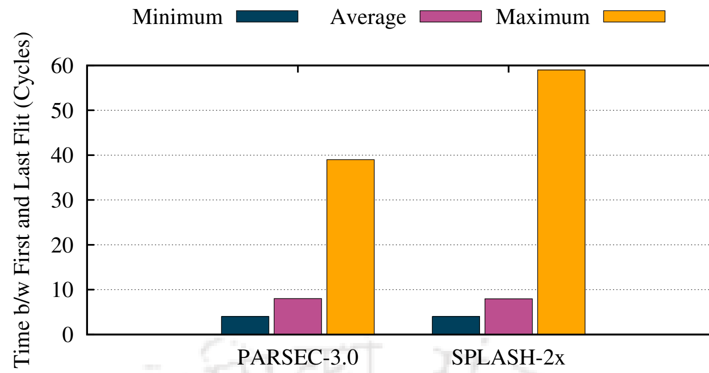


Figure 4.3: Reply difference time

such that

$$T^{P_i}_{RepDiff} = \begin{cases} |P_i| - 1 & \text{in Ideal Case} \\ |P_i| - 1 + \Delta_i & \text{in Actual Case} \end{cases} \tag{4.5}$$

where, $|P_i|$ is the number of flits in packet P_i and Δ_i is the experienced unknown router delay.

The minimum RDT remains 4 for both PARSEC 3.0 and SPLASH-2x benchmarks. It implies that all the flits reach back to back without any delay (ideal case). However, the maximum RDT is surprisingly high (during congestion): 39 for PARSEC 3.0 and 59 for SPLASH-2x benchmarks. Even the average RDT is 8.01 and 7.98, more than the minimum RDT meaning, flits are generally getting delayed. Any of the flits (including the one carrying the critical word) may be indefinitely delayed and hamper the performance. If memory access optimisations are made aware of the pros and cons of the underlying on-chip communication infrastructure, they may yield even better benefits. Based on our observation in Figure 4.2 and 4.3, ER optimisation, if made aware of the NoC infrastructure, can be highly effective in reducing miss penalty for the presented PARSEC 3.0 and SPLASH-2x benchmarks.

Proposed Solution:

Identify flits carrying the critical words at run-time and prioritise them during routing and arbitration decisions to reach destination at the earliest to resume execution.

4.5 NoC-Aware Early Restart Optimisation

This section presents an NoC-aware ER optimisation called *ER-NoC*, for modern TCMPs. ER-NoC is proposed on top of the original ER optimisation with a practical implementation.

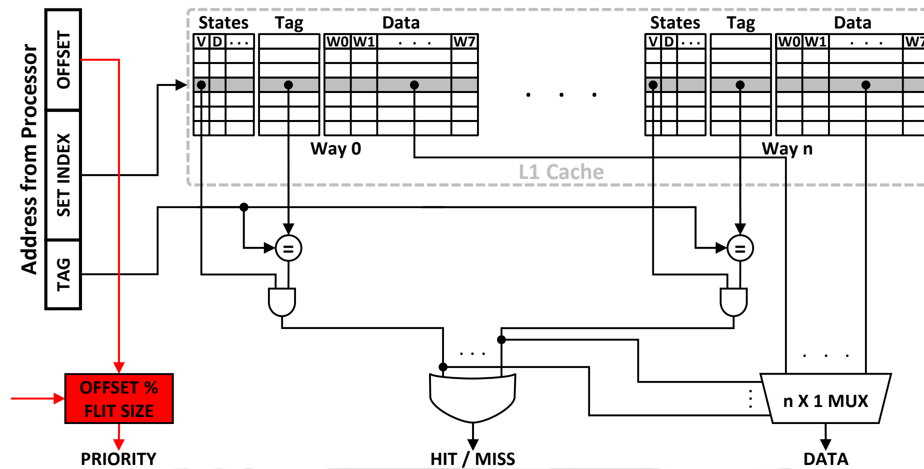


Figure 4.4: Modified L1 cache controller

4.5.1 Critical Flit Identification

When the processor needs a critical word, it sends a request to the L1 Cache Controller (L1 CTLR), giving the address of the data block which contains the critical word. The L1 CTLR maps into the corresponding set using the set index bits of the requested address, as shown in Figure 4.4. After the set is identified, L1 CTLR checks the tag bits for hit/miss detection. The corresponding data lookup is also done in parallel to reduce memory access latency. If the tag checker returns true (cache hit), the requested block is present in the L1 cache. The L1 CTLR identifies the critical word using the offset bits of the address and sends it to the processor for execution. If the tag checker returns false (cache miss), the block needs to be brought from the corresponding L2 cache bank through the underlying NoC. While the block is being fetched, ER optimisation uses the offset bits to check for the arrival of the critical word. As soon as the critical word is fetched at the L1 cache, i.e., the flit carrying the critical word has arrived, it is sent to the processor to resume its execution, without waiting for the arrival of the entire block (remaining flits). While the execution continues, remaining words of the block are fetched in the background (through flits). Since data transfer is still performed at block-level granularity, the underlying cache coherence remains unaffected.

Algorithm 3: Identify critical flit

Input: $Block\ Offset$
Output: $Critical\ Flit\ Identifier\ (CFI)$
Parameters: $Flit\ Size$
Variables: CFI
1 $CFI \leftarrow Block\ Offset\ \%\ Flit\ Size$;
2 **return** CFI

We define *critical flit* as the flit carrying the critical word of the block through the underlying NoC. The proposed ER-NoC optimisation adds a tiny module at L1 CTLR to identify the critical flit, as shown in *red* in Figure 4.4. This module takes as input the offset bits and flit size to return a 2-bit value (00/01/10/11) called *Critical Flit Identifier (CFI)*. The working of the proposed module is given in Algorithm 3. If the CFI value is 00, it means that the critical word will be transferred in the body flit B0, if 01 then B1, if 10 then B2 and if 11 then in the tail flit T. The proposed module added in L1 CTLR to get CFI runs in parallel with tag checker and data lookup modules and hence does not incur any additional delay in memory access latency. If the tag checker returns true, the CFI value is ignored as the requested block is already in the L1 cache, and there will be no flit transfer. However, if the tag checker returns false, the CFI value is added to the message/packet header when the block request is sent to the corresponding L2 cache bank through the underlying NoC.

4.5.2 Critical Flit Prioritisation

Upon receiving the request, the L2 Cache Bank Controller (L2 CTLR) replies with the data block as multiple flits through the underlying NoC. For the proposed ER-NoC optimisation, L2 CTLR puts the corresponding CFI value back into the packet header (H) of the reply. When the head flit traverse through the NoC routers, the CFI values are stored in a counter C in each router, as shown in Figure 4.5. ER-NoC optimisation modifies the traditional Round-Robin based priority policy to use C for priority during routing and arbitration. The motive behind this move is to reduce the router delay for critical flits to reach the destination at the earliest. It is employed after learning about the experienced RDT due to on-chip network congestion (refer Figure 4.3). However, with priority, there is always a risk of starvation for lower priority flits. To minimise such a risk, our policy does not prioritise all the flits of a data block; rather, it just prioritises up to the critical flit. For example, if the critical flit is B1 for a data block, our proposed policy just prioritises B0 and B1. This is done as all the flits preceding the critical flit of the block should reach L1 cache at the earliest, then only the critical flit will reach. Once the critical flit is prioritised, the succeeding flits (B2 and T) can reach the L1 cache at their own pace as they are not required to resume execution. However, due to spatial locality, future data requests can be for that portion of the incoming block which is not yet fetched. In such a case, the effectiveness of ER like memory access optimisations suffers from performance degradation [28]. It is observed that for a critical word W_c^i, a good number of future data requests to the incoming block B_i is for the immediate neighbours of W_c^i, i.e., words W_{c-1}^i and W_{c+1}^i. As ER-NoC optimisation prioritises critical flit, which carries multiple words, W_{c-1}^i and W_{c+1}^i may also be part of it. Thus, proposed ER-NoC optimisation has the scope for handling future data requests to the incoming block.

TH-2538_156101018

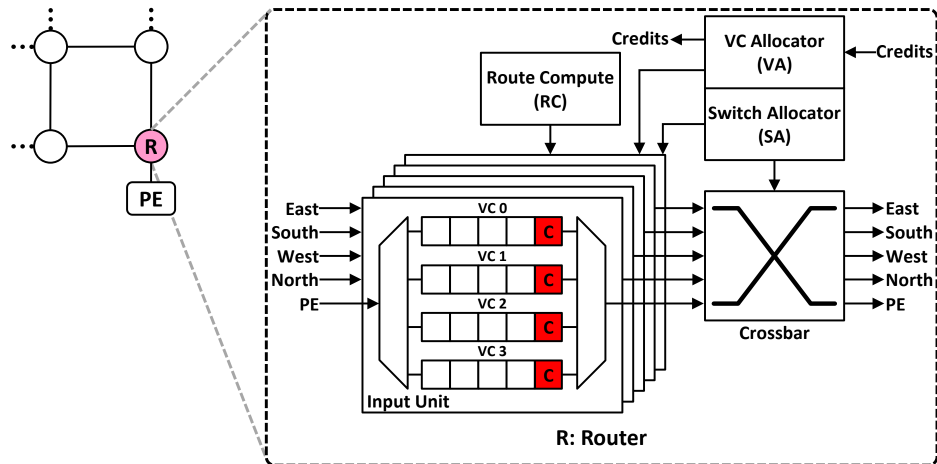


Figure 4.5: Modified router microarchitecture

Algorithm 4: Prioritise critical flit

Input: *Input VCs*
Output: *Winner VC*
Parameters: n : *Number of VCs*, C : *CFI Value*
Variables: i, *winner*

1 $winner \leftarrow VC_0$;
2 **for** $i \leftarrow 0$ **to** $n-1$ **by** 1 **do**
3   **if** $VC_i[C] > winner[C]$ **then**
4         $winner \leftarrow VC_i$;
5 **if** $winner[C] > 0$ **then**
6     $winner[C] \leftarrow winner[C] - 1$;
7   **return** *winner*;
8 **else**
9   **Choose** *winner* based on **Round-Robin**;
10   **return** *winner*;

When two flits of two different data blocks (packets) compete for the same output port at a particular NoC router, one with the lower CFI counter C is prioritised. When a flit wins the arbitration, the corresponding C is decremented by 1. This way, when the critical flit reaches a router, the counter C becomes 0, indicating that this flit has the highest priority. Hence the proposed policy prioritises only and until critical flit of a data block. The working of the proposed priority policy is given in Algorithm 4. Exploiting NoC infrastructure to prioritise based on data criticality reduces miss penalty and improves overall system performance.

Table 4.2: System configuration

Processor	64 OoO x86 cores, 1.3GHz
L1 Cache	32KB, 8-way, private, split (instruction and data)
L2 Cache (LLC)	512KB$\times$64 cores, 16-way, shared
Memory Bank	4; one located at each corner
Coherence	MOESI distributed directory
NoC	8$\times$8 2D-Mesh topology, 128-bit channel width 3 Virtual Networks (VNs) 3 Virtual Channels (VCs) per VN 1-flit depth control VC, 4-flit depth data VC
Routing	2-stage routers (1.54ns), XY-DOR algorithm VC based wormhole packet-switching
Packets	1-flit for control packet, 5-flit for data packet
Word/Flit/Block	64-bit/128-bit/64B; 2-words/flit, 8-words/block
Benchmarks	PARSEC 3.0 and SPLASH-2x (multi-threaded)

4.6 Performance Evaluation

We consider the following architectures for evaluation:

- **Baseline:** Without any optimisation.
- **ER:** Original NoC oblivious Early Restart optimisation.
- **ER-NoC:** Proposed NoC-aware Early Restart optimisation.

4.6.1 Simulation Framework and Workloads

We model all the architectures on event-driven gem5 simulator [55]. Our system configuration is similar to Intel Xeon Phi Processor 7235 [56] with shared and distributed L2 cache (LLC). The system configuration is presented in Table 4.2 for reference. We modify MOESI_CMP_directory protocol in Ruby inside gem5 to implement the proposed cache controllers. We modify GARNET [57] inside gem5 to implement the proposed router microarchitecture. We run the simulation in Full-System (FS) mode to collect the results.

To evaluate and analyse the performance, we use PARSEC 3.0 and SPLASH-2x suite of multi-threaded benchmarks. Each of the benchmarks mimic a modern NoC based TCMP running multiple threads of a single application. We consider a mix of 18 (9 each from PARSEC 3.0 and SPLASH-2x) computation-intensive, communication-intensive and memory-intensive benchmarks. These benchmarks are run individually as a 64-thread workload on all

TH-2538_156101018

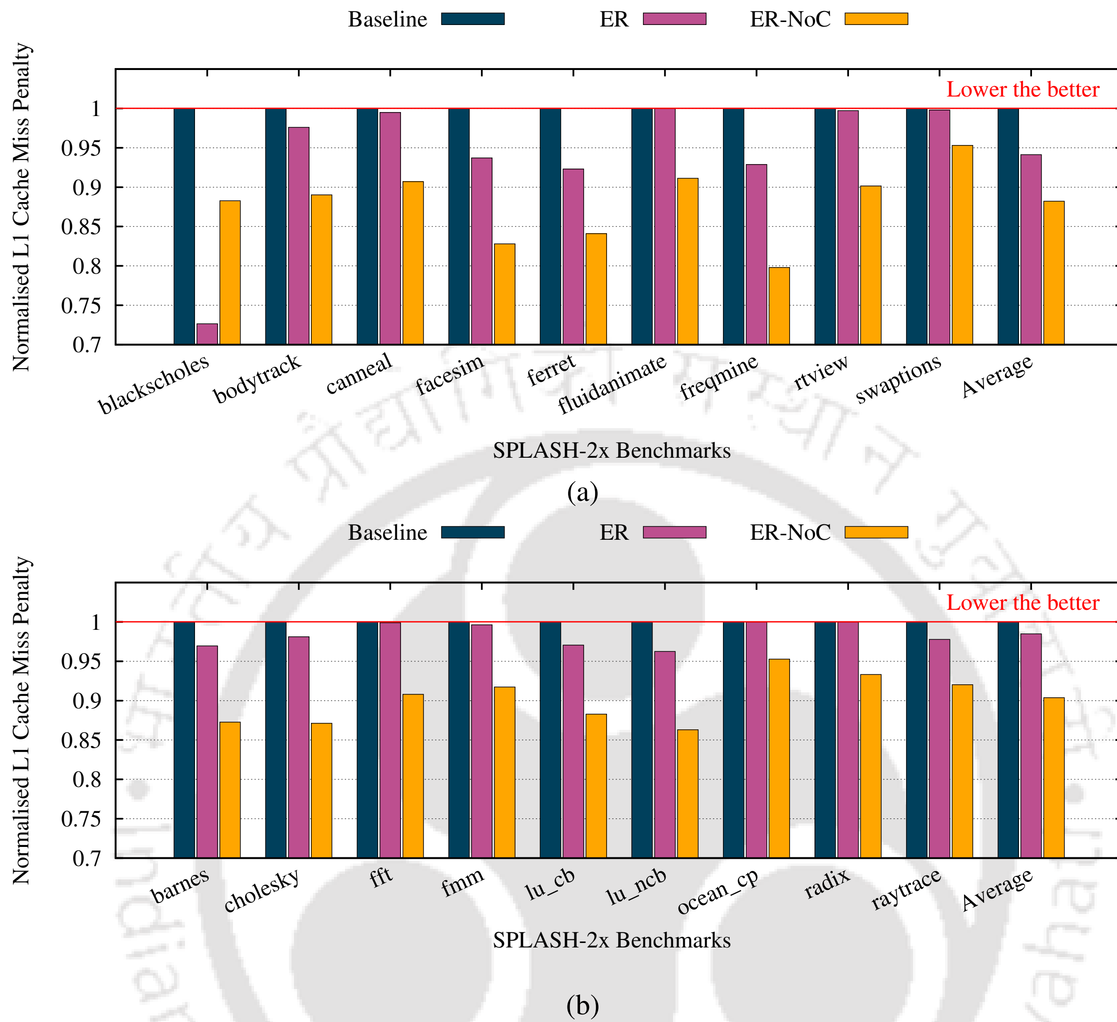


Figure 4.6: L1 cache miss penalty

the 64 cores (1 thread/core) of an NoC based TCMP (1×64: 64). We consider *sim-medium* input set for these benchmarks and report the results for the run of entire RoI. For a relative comparison, all the results are normalised with respect to the baseline architecture.

4.6.2 Result Analysis and Discussion

4.6.2.1 L1 Cache Miss Penalty

For the baseline architecture, it is defined as the number of cycles required to bring a requested data block (containing the critical word) in the L1 cache. In case of ER and ER-NoC architectures, it is defined as the number of cycles required to receive the critical

TH-2538_156101018

word while the requested data block is brought in the L1 cache. L1 cache miss penalty directly reflects the effectiveness of the proposed ER-NoC architecture over others.

Figure 4.6a and 4.6b shows the normalised L1 cache miss penalty for PARSEC 3.0 and SPLASH-2x benchmarks, respectively, with respect to the baseline architecture. Since baseline architecture could resume execution only after the entire data block is received, and the average RDT is quite high (refer Figure 4.3), miss penalty is more. As the experienced RDT is due to on-chip congestion, it also affects the effectiveness of the existing ER architecture. By exploiting the insights about the position of critical word (Figure 4.2 and Table 4.1) and RDT due to congestion (Figure 4.3), our prioritisation scheme in the proposed ER-NoC architecture significantly reduces L1 cache miss penalty. A maximum of 21% (14%) and an average of 12% (10%) reduction in L1 cache miss penalty for PARSEC 3.0 (SPLASH-2x) benchmarks is achieved by the proposed ER-NoC architecture. However, our prioritisation scheme performs poorly for *blackscholes* benchmark from PARSEC 3.0 suite when compared to the ER architecture (refer Figure 4.6a). For more than 70% of the time, critical words are in the first body flit B0 for *blackscholes* (refer Table 4.1). Also, it is one of the simplest benchmarks with a very small working set and negligible communication. Hence, the ambitious scheme to prioritise critical words during on-chip congestion is unnecessary. The presence of the prioritisation scheme in the proposed ER-NoC architecture delays routing and arbitration, thus performing poorly with respect to ER architecture for *blackscholes*. Even for *facesim* benchmark from PARSEC 3.0 suite, critical words are in the first body flit B0 for more than 80% of the time. However, *facesim* has a large working set with sharing, hence the remaining 20% is a large enough number of blocks that may suffer with the delay. This is why proposed ER architecture is able to perform better for *facesim*, unlike *blackscholes*.

4.6.2.2 System Speedup

For multi-threaded workloads, system speedup (S) is given by:

$$S = \frac{ExecTime_{Baseline}}{ExecTime_{Proposed}} \tag{4.6}$$

where $ExecTime_{Baseline}$ and $ExecTime_{Proposed}$ are the total execution time of the baseline and proposed architectures, respectively. We prefer execution time as multi-threaded workloads have synchronisation primitives like locks and barriers, which brings variation in IPC.

Figure 4.7a and 4.7b shows the normalised system speedup for PARSEC 3.0 and SPLASH-2x benchmarks, respectively, with respect to the baseline architecture. As expected, a reduction in L1 cache miss penalty directly translates into the improvement of overall system performance. A maximum of 15% (13%) and an average of 11% (7%) improvement in system

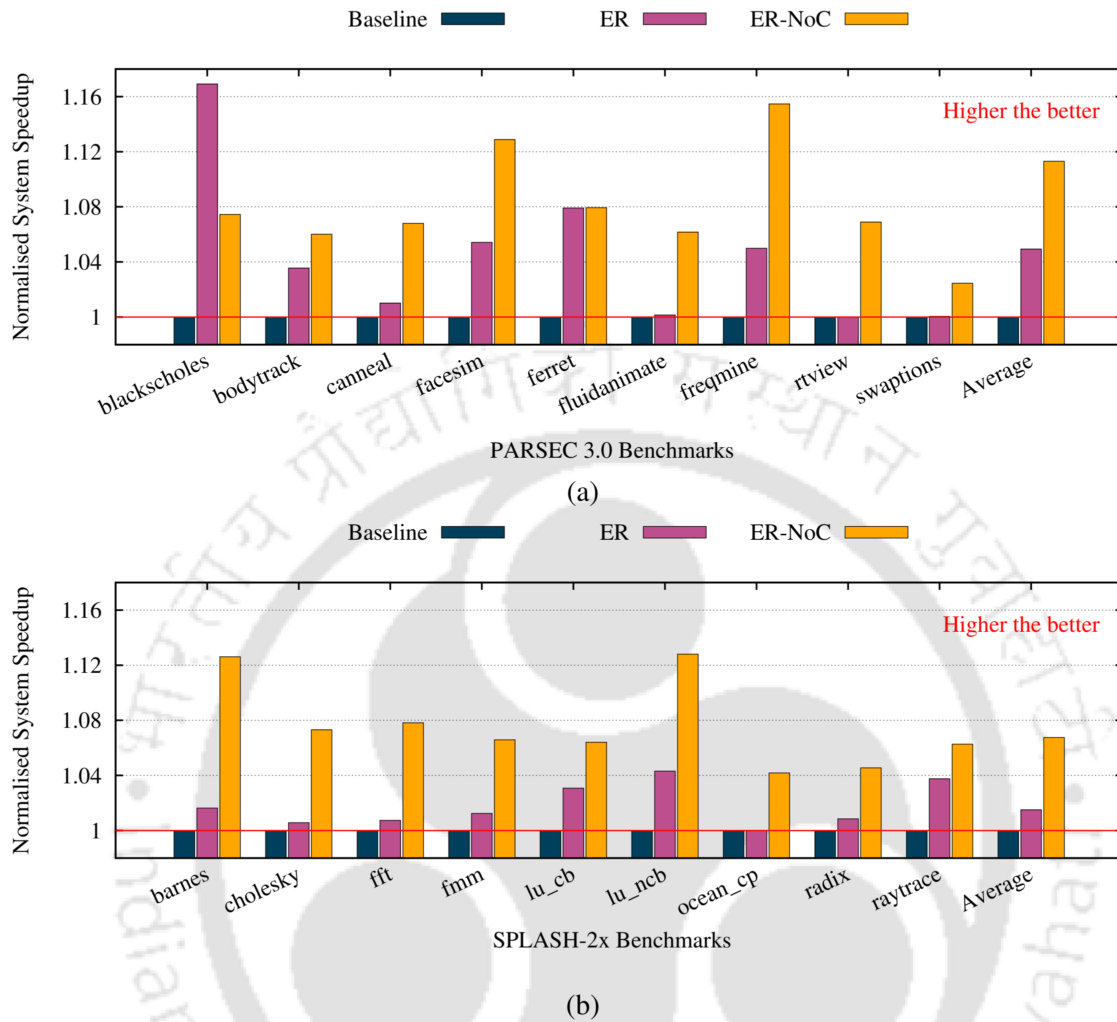


Figure 4.7: System Speedup

speedup for PARSEC 3.0 (SPLASH-2x) benchmarks is achieved by the proposed ER-NoC architecture. Improvement in *blackscholes* is less when compared to the ER architecture.

4.6.3 Comparison with CWF Optimisation

As most of the critical words are located at the beginning of the requested blocks, we feel that CWF optimisation might not be necessary. To validate this intuition, we evaluate **CWF** architecture that implements the original NoC oblivious Critical Word First optimisation. Figure 4.8 shows the comparison of normalised system speedup for CWF and other architectures. While CWF outperforms ER architecture (expected), the improvement is only marginal ($\approx$ 1.5%). On the other hand, ER-NoC has significant improvement with respect to the CWF architecture. There are two main reasons for this outcome, (a) critical word being the first one

TH-2538_156101018

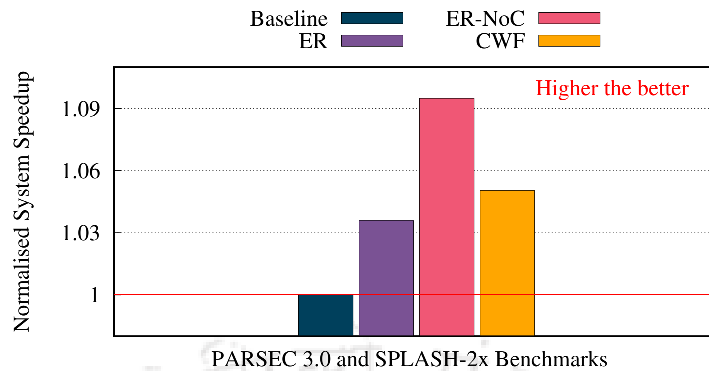


Figure 4.8: Comparison with CWF optimisation

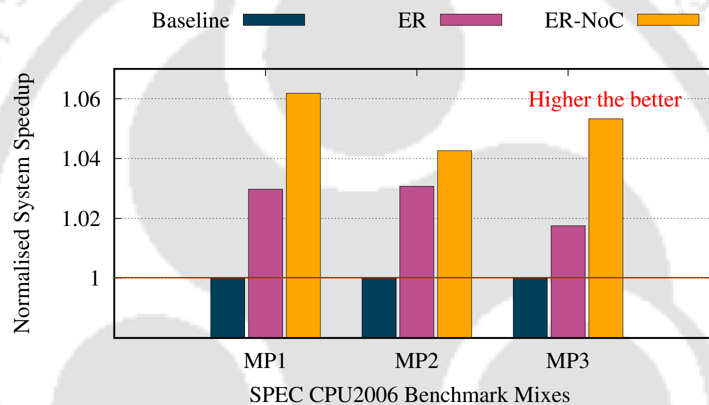


Figure 4.9: Impact on multi-programmed workloads

in the requested blocks, and (b) prioritising critical flits to reach the destination at the earliest. An NoC-aware CWF optimisation (something like CWF-NoC) would probably perform better, but with ER-NoC, we are already able to get significant performance improvement, and at the same time, avoid the implementation complexity and overhead of CWF altogether.

4.7 Sensitivity and Overhead Analysis

4.7.1 Impact on Multi-Programmed Workloads

We consider multi-threaded applications for evaluation, as they put more stress onto the shared resources like NoC and LLC, thus giving a clear picture of the effectiveness of the proposed work. However, one can always evaluate using multi-programmed applications as

TH-2538_156101018

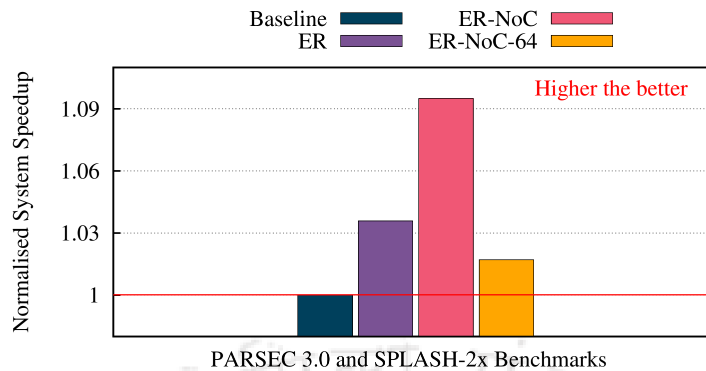


Figure 4.10: Impact of channel width

well. To get an idea, we consider SPEC CPU2006 multi-programmed benchmarks and mimic a modern NoC based TCMP running multiple applications in parallel. Using Misses Per Kilo Instructions (MPKIs), we create three different workload mixes. MP1 and MP2 run 64 copies of the *GemsFDTD* (high MPKI) and *gromacs* (low MPKI) benchmarks, respectively, on all the 64 cores (1×64: 64). Whereas MP3 runs random combination of 4 different benchmarks (*astar*, *h264ref*, *hmmer*, and *omnetpp*) with 16 copies each (4×16: 64). By separately profiling each benchmark, we choose a smaller representative window of instructions to have a tractable simulation time. We use total IPC to compare the system speedup between baseline and the proposed architectures for multi-programmed workloads, as shown in Figure 4.9. In general, due to higher MLP exhibited by the multi-programmed workloads when compared to the multi-threaded workloads, the performance gain with ER like optimisations in the former remains lower. Nonetheless, the proposed ER-NoC beats the existing ER architecture.

4.7.2 Impact of Channel Width

One of the key NoC parameters that can have a significant impact on the proposed ER-NoC architecture is the *channel width (flit size)*. Hence, we conduct a sensitivity analysis by changing the channel width in the proposed ER-NoC architecture from 128-bit (refer Table 4.2) to 64-bit. The new architecture called **ER-NoC-64**, has 9-flit data packets (H, B0, B1, B2, B3, B4, B5, B6, T) and hence should ideally take more time to transfer a data block from L2 cache bank to the L1 cache. Surprisingly, as shown in Figure 4.10, ER-NoC-64 is still able to beat the performance of the baseline architecture that has only 5-flit data packets.

4.7.3 Storage, Area and Power Overhead

We use 2 additional bits (CFI value) in the packet header to facilitate the prioritisation scheme in the proposed ER-NoC architecture. Our NoC uses 128-bit flit channel (refer Table 4.2), but a typical packet header (head flit) uses fewer bits ($\approx$ 64 bits) for its operation. So, we can easily accommodate the additional 2 bits in the head flit without any storage overhead.

We use McPAT [61] at 22nm processor technology and feed the configuration and output files of gem5 [55] to get the area, static (leakage) and dynamic power overheads. While the ER-NoC architecture has a negligible area and leakage power overhead of 1.34% and 3.60%, respectively, dynamic power reduces by 5.85% due to the improvement in performance.

4.8 Related Work

There are specific works on efficient cache [94] and NoC [95] organisation to improve the performance of NoC based TCMPs. The existence of critical word and memory access optimisations to prioritise critical words are available in the classic textbook by Hennessy and Patterson [28]. One of the first attempts to understand criticality for data requests from L1 to L2 cache is by Gieske [92]. He reported that for multi-programmed benchmark suites SPEC CPU2000 and CPU2006, sufficient regularity in critical word exists. Exploiting this criticality information, quite a few optimisations are proposed in the past. However, none of them explicitly studied the behaviour of critical words for multi-threaded benchmark suites.

Prior research efforts target criticality of data and instructions at different levels. Direct optimisations based on critical words are proposed at NoC, cache and main memory levels. Rengasamy and Mutyam exploit packet information (source, destination, address, data) to improve NoC communication by optimising multicasting, saving RC time, saving VCs, etc [96]. In a relatively recent work, Miguel and Jerger defined data criticality as the promptness with which an application uses data after it is fetched from the memory [19]. They find that conventional, criticality-oblivious NoC designs waste up to 37.5% of energy. They propose NoCNoC, a practical, criticality-aware NoC design that achieves up to 60.5% energy savings with no loss in performance. In a subsequent work, the same group proposed a Runahead NoC that considers the first word of a block as critical and send it through a lossy, 1-hop network [20]. At the cache level, there is an interesting work by Huang and Nagarajan, where instead of storing the entire cache blocks, they propose to store only the critical words of the blocks in the cache [97]. In effect, this technique increases the cache capacity and improve performance. Some interesting proposals are available at the main memory level [93][98]. There are also a good number of indirect optimisations like

slack-based criticality with packet prioritisation [17][27], cache miss criticality with memory level parallelism [79][80], memory scheduling with bank-level parallelism [81][82], etc.

4.9 Chapter Summary

In this work, we presented a unique profiling of memory requests to understand how critical words are distributed in the data blocks. Then we found that flits carrying the critical words are indefinitely delayed at the NoC routers, which hampers processor execution. Due to this delay, NoC oblivious memory access optimisations suffers from performance degradation. Hence we proposed an NoC-aware memory access optimisation that prioritises the flits carrying the critical words during routing and arbitration decisions. This way the flits can reach their destination as soon as possible and resume processor execution at the earliest, which significantly reduces miss penalty and improves overall system performance.

The next chapter describes the first part of our proposed work on opportunistic caching.

TH-2538_156101018

Chapter 5

Opportunistic Caching by Exploiting Underutilised Router Buffers

This chapter describes the first part of the proposed work on opportunistic caching, where we exploit underutilised NoC router buffers to store recently evicted cache blocks. Future re-references to the evicted cache blocks can be serviced from the local routers without any delay, which significantly reduces miss penalty and improves overall system performance.

5.1 Introduction

With the ever-increasing cores in modern TCMPs, communication through the underlying NoC usually plays a very significant role in memory (data) access latency. Nevertheless, for data access requests that travel to the off-chip memory banks through the MCs, the role of NoC is limited. Existing literature argues that NoC resource utilisation is very low, with an average injection rate of only around 5% [30][31][32]. Exploiting underutilised NoC resources to possibly improve data access latency for off-chip requests should be explored.

Due to the usage of relatively simpler OoO processing cores, modern TCMPs, including Intel Xeon Phi Processors [89], use only 2-levels of on-chip cache memories. In these processors, L2 serves as the LLC and communicates with the off-chip memory banks for data transfer. On-chip caches are usually small due to their associated cost and hence going to the off-chip memory banks for fetching data is inevitable. With data-driven applications, going off-chip for data is even more common. Current NoC based TCMPs like Intel Xeon Phi Processors [89] have private, write-back L1 caches and a shared, distributed, write-back LLC with directory. When an L1 cache miss occurs, the data request reaches the corresponding LLC bank. When an LLC miss occurs, the data request reaches the corresponding off-chip

TH-2538_156101018

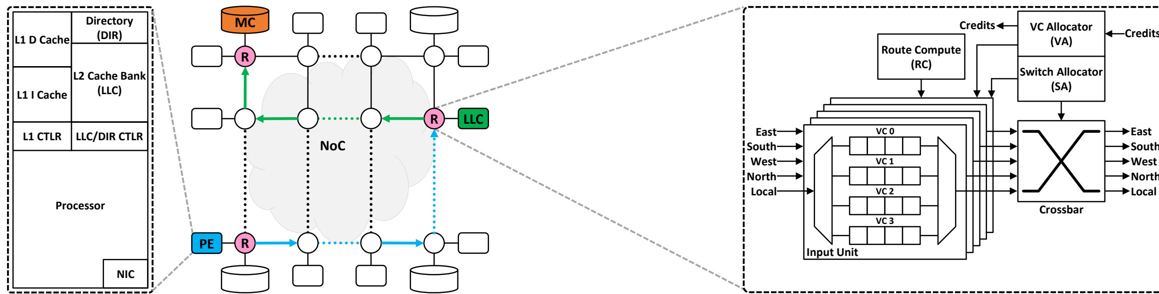


Figure 5.1: Conceptual view of an NoC based TCMP, where data transfer request is travelling from L1 cache to LLC (shown in *blue*) and then from LLC to MC (shown in *green*).

memory bank through the MC, as shown in Figure 5.1. The requested data is fetched from the off-chip memory and forwarded to the corresponding LLC bank and then to L1 cache to resume execution. The entire communication is packet based and is done through the underlying NoC. The time required to replace an existing block in LLC with the requested incoming cache block is called LLC miss penalty. It is also known as *off-chip miss penalty* as the incoming block is fetched from off-chip memory. Since LLC is small in size, it fills up quickly, and all subsequent LLC miss always evicts a valid cache block. As per the coherence directory information, the LLC/Directory Controller (LLC/DIR CTLR) decides whether to discard the block (clean) or send it to off-chip memory for write-back (dirty). However, for applications with good temporal locality, a recently evicted LLC block, if requested again, needs to be re-fetched from the off-chip memory. The LLC miss penalty is generally in order of hundreds of system clock cycles. This severely hampers the overall system performance.

Modern NoC based TCMPs employ input-buffered routers for scalable on-chip bandwidth [99][100]. In-transit packets on their way from source to destination are stored in the buffers of intermediate routers to take part in routing and arbitration decisions. However, an experimental analysis on real application-based workloads shows that the average buffer utilisation of NoC routers is very low except during peak NoC congestion (Section 5.2.2). In this work, we attempt to exploit empty (underutilised) buffers of NoC routers in a way that increases their utilisation and positively impacts overall system performance. When evicted, dirty blocks of an LLC bank reach the local router [1] to travel over the NoC to reach off-chip memory for write back; we propose to delay their travel. We temporarily disable arbitration of such evicted blocks to delay their travel and keep them stored in local router buffers. We also propose to store some of the evicted, clean blocks in local routers, which are otherwise discarded. Since the buffer utilisation is low, these blocks can be kept stored in local routers without inducing any NoC congestion. When a recently evicted LLC block

[1]Local router connects a tile (core) to the underlying NoC

is re-referenced, we propose to arrange a quick reply with the stored block from the local router. These optimisations help to avoid the off-chip miss penalty and improve overall system performance. In this work, we make the following significant contributions:

- **Local Store:** We propose an NoC based TCMP that identifies evicted, dirty LLC blocks in local router buffers and disables their arbitration. We also make some evicted, clean LLC blocks reach the local router and get stored in the available buffers. Both clean and dirty LLC blocks are stored in local router buffers for as long as possible.
- **Local Reply:** We propose an optimisation such that during the time an evicted LLC block (clean or dirty) is stored in the local router buffer, a re-reference request for the same block, by the same core can be locally replied from the router. With local replies, we completely avoid off-chip miss penalties and improve overall system performance.
- **Block Forward and Drop:** We forward the locally stored, dirty LLC blocks for write-back towards their destination using two techniques based on router buffer contention. Whereas we discard the stored, clean LLC blocks as they need not be forwarded for write-back. This way, we make sure that locally storing evicted LLC blocks does not create injection suppression for others, as well as cache coherence is maintained.
- **Case Study for L1 Cache:** Due to their limited size, the number of cache blocks evicted from private L1 caches is much higher than the shared LLC banks. We modify the proposed architecture to store evicted cache blocks for L1 caches. We propose two additional techniques to forward and drop the stored blocks from the local routers.

5.2 Background

5.2.1 LLC Miss Penalty

Most of the modern NoC based TCMPs like Intel Xeon Phi Processor (2016) [89], Princeton Piton Processor (2015) [90], MIT Scorpio Processor (2014) [91], and others use only 2-levels of on-chip cache memories. Data-driven, memory-intensive applications expose this limited on-chip caching when they encounter frequent LLC misses. As we already know that the time required to replace an existing block in LLC with the requested, incoming block is called LLC miss penalty (MP_{LLC}). In NoC based TCMPs, MP_{LLC} can be given as:

$$MP_{LLC} = T_{On-Chip} + T_{Off-Chip} \tag{5.1}$$

TH-2538_156101018

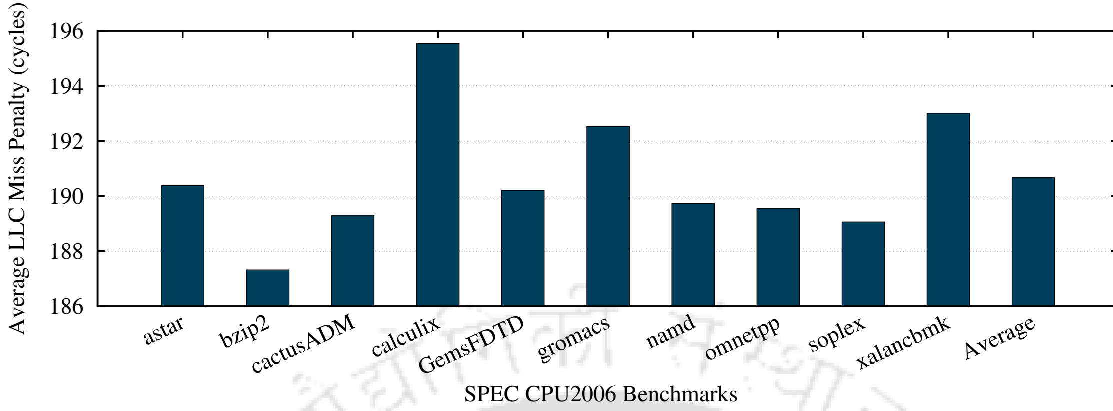


Figure 5.2: LLC miss penalty

where

$$T_{On-Chip} = t_{Request}^{LLC-MC} + t_{Reply}^{MC-LLC} \tag{5.1a}$$

$$T_{Off-Chip} = t_{Request}^{MC-MEM} + t_{Reply}^{MEM-MC} \tag{5.1b}$$

$$T_{On-Chip} << T_{Off-Chip} \tag{5.1c}$$

here, $T_{On-Chip}$ and $T_{Off-Chip}$ are the time taken in on-chip and off-chip travels, respectively. t_k^{i-j} is the time taken by message k to travel from module i to module j. For example, $t_{Request}^{LLC-MC}$ is the time taken by a cache miss request to travel from LLC bank to the corresponding MC.

Figure 5.2 shows the average LLC miss penalties for different SPEC CPU2006 benchmarks. The average LLC miss penalty for the presented benchmarks is around 190 cycles, which is very expensive. As given in Equation (5.1c), LLC miss penalty is dominated by the off-chip data transfer time ($T_{Off-Chip}$) between MC and memory (MEM). Off-chip data transfer time is usually in hundreds of system clock cycles, whereas on-chip data transfer time is only in tens of cycles. Hence, the role of NoC based on-chip data transfer in LLC miss penalty is minimum. In another observation from Figure 5.2, the LLC miss penalty is very similar for all the presented benchmarks. It can be attributed to the fact that after acquiring the shared data bus, off-chip transfer time is almost fixed. Since off-chip data transfer time dominates in LLC miss penalty, it remains similar across benchmarks. On-chip congestion and delay in NoC routers owing to the application behaviour have negligible impact.

LLC (off-chip) miss penalty is very expensive (hundreds of system clock cycles) as well as application oblivious with a limited role of NoC and its available resources.

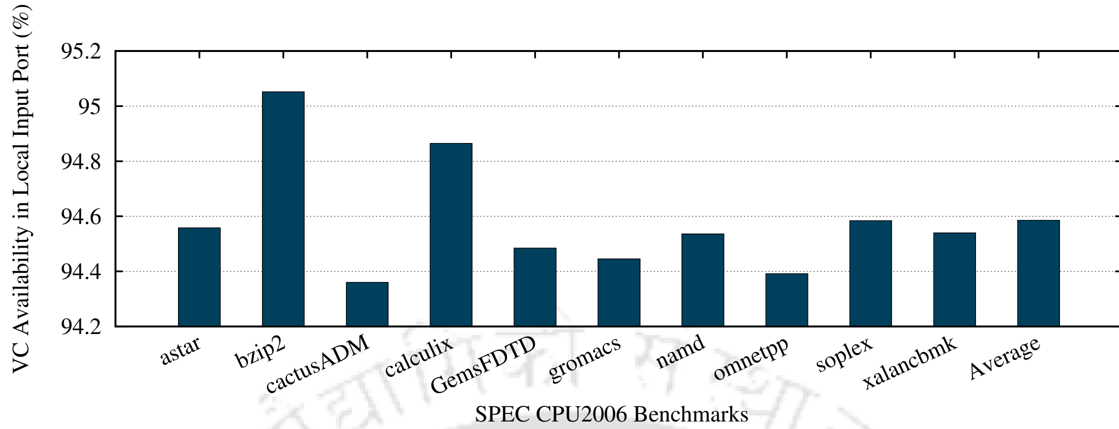


Figure 5.3: VC availability in local input port

5.2.2 VC Availability

NoC based systems use routers to communicate the transfer of packets from source to their destination. NoC routers have three major design alternatives: buffered, minimally-buffered and buffer-less; each with its own set of advantages as well as disadvantages. Most of the modern TCMPs employ input-buffered NoC routers for scalable on-chip transfer bandwidth [99][100]. Router (input port) buffers are further divided into VCs for deadlock-free routing and better utilisation [101]. Packets coming through different input ports (east, south, west, north and local) gets stored in the available VCs and take part in routing and arbitration decisions. VC availability (VCA_n) in NoC based TCMPs can be represented as:

$$VCA_n = \frac{Cycles\ when\ n\ VCs\ are\ Free}{Total\ Execution\ Cycles} \tag{5.2}$$

Figure 5.3 shows the percentage of VC availability in local input port of NoC routers for different SPEC CPU2006 benchmarks. As the average injection rate of these multi-programmed benchmarks is only around 5%, except during peak NoC congestion, at least one VC is always free ($\approx$ 95%). The observation in Figure 5.3 is in sync with the conclusions in the literature about low NoC router buffer utilisation with real applications [30][31][32].

> Modern NoC based TCMPs use input-buffered routers for worst-case performance bandwidth, but buffers (VCs) remain underutilised except during peak NoC congestion.

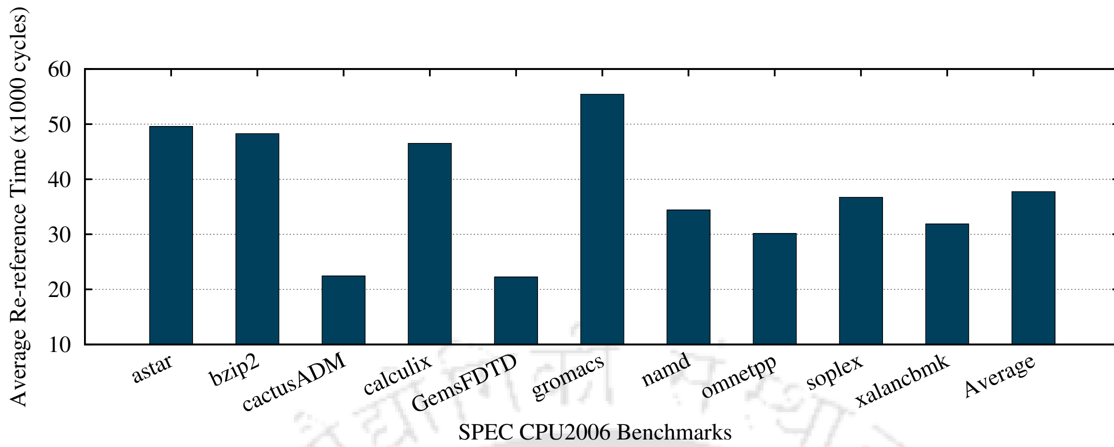


Figure 5.4: Re-reference time of evicted LLC blocks

5.3 Motivation

The concept of caching is governed by the principle of locality of reference; *temporal* and *spatial*. Applications with good temporal locality of reference may request for recently evicted cache blocks in LLC. The duration from the eviction of an LLC block to the request of the same block in future by the same core is called re-reference time. Figure 5.4 shows the average re-reference time for different SPEC CPU2006 benchmarks. For example, in a 64-core NoC based TCMP running a memory-intensive benchmark *GemsFDTD*, an evicted LLC block is re-referenced within an average time of 22213 cycles. Across all benchmarks, on average, within an interval of around 37000 cycles, an evicted LLC block is re-referenced. For all the observations presented in Figures 5.2, 5.3 and 5.4, we run 64 copies of the same benchmark in a 64-core NoC based TCMP (refer Table 5.4 for system configuration).

Evicted, clean LLC blocks are discarded, whereas evicted, dirty LLC blocks are sent over the NoC to the off-chip memory bank for write-back. To reach their destination for write-back, evicted, dirty LLC blocks enter the local router through the local input port as packets. We call them LLC write-back packets. They get stored in the available VCs to take part in routing and arbitration decisions. In this work, we propose to delay the travel of LLC write-back packets and keep them stored in the local NoC router buffers (VCs) for as long as possible. During the time an LLC write-back packet is locally stored, a re-reference request for the same LLC block by the same core, can be locally replied. We also propose to bring some of the evicted, clean LLC blocks to the local router and keep them stored in the available VCs to improve our chances of local reply. As shown in Figure 5.3, there are sufficient free VCs available in the local input port of NoC routers. So, keeping the evicted LLC blocks (clean and dirty) locally stored will not create injection suppression for other

TH-2538_156101018

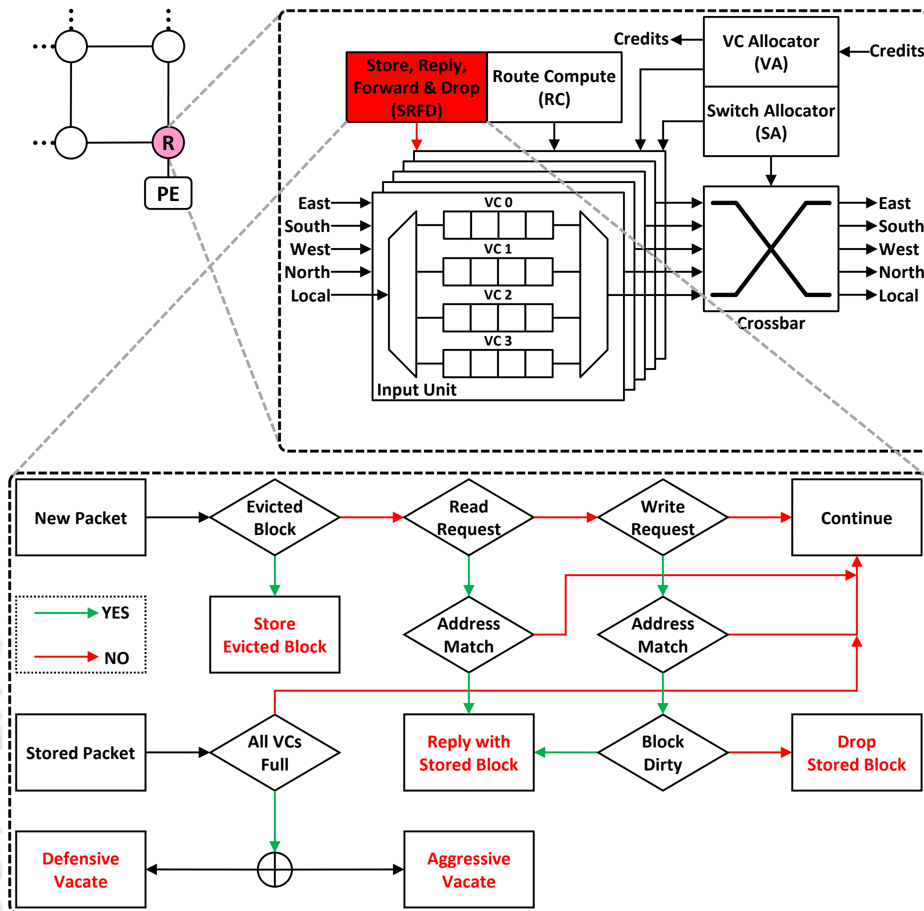


Figure 5.5: Conceptual illustration of the proposed router microarchitecture

packets. As given in Equation (5.1), LLC miss penalty is dominated by the off-chip transfer time, and the role of NoC is limited. With direct reply of LLC blocks from the local routers, we propose to completely avoid off-chip transfer time. Our proposed optimisations have the potential to reduce LLC miss penalty, thereby increasing overall system performance.

Proposed Solution:

Store evicted LLC blocks in underutilised NoC router buffers (VCs) and upon re-reference, generate local replies from the routers to reduce LLC miss penalty.

5.4 Proposed Architecture

In this section, we explain the working of the proposed architecture. We consider MOESI directory protocol with non-inclusive LLC, and the stable states of LLC are presented in

TH-2538_156101018

Src	Dest	Addr	. . .	Store	Dirty	S	X
				1-bit	1-bit	1-bit	1-bit

Figure 5.6: Modified message/packet header

Table 5.1: MOESI Directory Protocol - Stable states of LLC

State	Description
I	Invalid everywhere; LLC and all the private L1 caches
S	Valid, but not exclusive, not owned and not dirty
ILS	Invalid in LLC, but locally held in S state by one or more L1 caches
E	Valid, exclusive, owned but not dirty
ILE	Invalid in LLC, but locally held in E state by one of the L1 caches
O	Valid, owned and potentially dirty but not exclusive
ILO	Invalid in LLC, but locally held in O state by one of the L1 caches
M	Valid, exclusive, owned and potentially dirty.
ILM	Invalid in LLC, but locally held in M state by one of the L1 caches

Table 5.1. Directory is distributed and co-located with the corresponding LLC bank, as shown in Figure 5.1. LLC/DIR CTLRs are responsible for maintaining on-chip coherence. MCs acts as an interface to the off-chip memory banks (MEM) and maintain off-chip coherence (when applicable). A conceptual illustration of the proposed router microarchitecture is given in Figure 5.5, where the additional units and links are shown in *red*. The memory hierarchy dictates that L1 cache controllers (L1 CTLRs) communicate with LLC/DIR CTLRs, which communicate with MCs for data and coherence. An L1 CTLR can not bypass an LLC/DIR CTLR to communicate with an MC directly. We use the table-based method popularised by Nagarajan et al. [53] to explain the working of MOESI directory protocol. Explaining the protocol for all the possible states and events are beyond the scope of this work. Hence, we limit the explanation to only the states and events that are relevant to the proposed work.

5.4.1 Local Store: Keeping Evicted LLC Blocks in Router Buffers

On a replacement, a valid cache block is evicted from an LLC bank most of the time. Table 5.2 presents the transitions and actions performed at an LLC/DIR CTLR on a block replacement. A blank entry in the table indicates that the corresponding transition is either impossible or irrelevant for our discussion. A shaded row i is the modified and proposed version of row $i-1$ to implement our optimisations. For example, in Table 5.2, row 4 is the modified and proposed version of row 3. Within an entry in a shaded row, only the bold actions are additionally performed along with the existing actions to implement the proposed

Table 5.2: LLC/DIR Controller - Block replacement. REPLACEMENT: block replacement from LLC, ACK-PUT: acknowledgement from MC for PUT request, PUTE/PUTO/PUTM: write-back request for E/O/M state, DATA: evicted block, INV: invalidation, REQ: L1 requester, DIR: directory, MEM: off-chip memory connected through MC.

#	State	REPLACEMENT	ACK-PUT
1	S	Remove REQ from Sharers, Store Block State to DIR, Deallocate Block / ILS	
2	E	Send PUTE to MEM / EI^A	
3	EI^A	Stall	Clear Owner, Deallocate Block / I
4	EI^A	Stall	**Add a Store Flag as SET**, **Add a Dirty Flag as RESET**, **Send DATA to MEM**, Clear Owner, Deallocate Block / I
5	O	Send PUTO to MEM / OI^A	
6	OI^A	Stall	Send DATA to MEM, Send INV to All Sharers, Clear Sharers, Clear Owner, Deallocate Block / I
7	OI^A	Stall	**Add a Store Flag as SET**, **Add a Dirty Flag as SET**, Send DATA to MEM, Send INV to All Sharers, Clear Sharers, Clear Owner, Deallocate Block / I
8	M	Send PUTM to MEM / MI^A	
9	MI^A	Stall	Send DATA to MEM, Clear Owner, Deallocate Block / I
10	MI^A	Stall	**Add a Store Flag as SET**, **Add a Dirty Flag as SET**, Send DATA to MEM, Clear Owner, Deallocate Block / I
11	I, ILS, ILE, ILO, ILM: Explanation of transitions is out of scope		

optimisations. A transient state XY^Z denotes that a cache block is transitioning from a stable state X to another stable state Y, and the transition is waiting for an event Z to happen.

With MOESI directory protocol, a valid cache block evicted from an LLC bank can be either clean (S/E) or dirty (O/M). Clean blocks are usually dropped, and dirty blocks are forwarded towards off-chip memory bank for write-back. To evict a dirty block, LLC/DIR CTLR initiates a write-back request (PUTO/PUTM) to send over the NoC towards off-chip memory bank as given in rows 5 and 8 of Table 5.2. After receiving such a request, MC replies with an acknowledgement (ACK-PUT) to receive the evicted block for write-back. When an ACK-PUT message reaches the LLC/DIR CTLR, it evicts the dirty block and sends the block for write-back to the off-chip memory as a DATA message (rows 6 and 9 of Table 5.2). Our first optimisation targets these write-back DATA messages on their way to the destination. These messages are marked with a 1-bit flag *Store* in their header for identification, as shown in Figure 5.6. Related modifications in the LLC/DIR CTLR are given in rows 7 and 10 of Table 5.2. All the messages travel as packets through the underlying NoC to reach their destination. They enter NoC through the local router, which is connected to their tile, get stored in the available buffers of VCs and take part in routing and arbitration decisions. An additional **Store, Reply, Forward & Drop (SRFD)** unit (refer Figure 5.5) in the local router checks the *Store* flag of all the buffered packets. We consider two-stage NoC routers (1: RC, 2: VA and SA) where SRFD unit works in parallel with the Route Compute (RC) unit. So, while SRFD unit checks the *Store* flag of a packet, RC unit calculates the route for the same packet in parallel. If the *Store* flag is found SET for a packet, SRFD unit disables its VC and switch arbitration (VA and SA). This packet is an evicted, dirty LLC block (DATA message of PUTO/PUTM) which remains stored in the local router unless forwarded for write-back (explained in Section 5.4.3). Both LLC/DIR CTLR and MC are unaware of the proposed optimisation of local store. For LLC/DIR CTLR, the DATA message is on its way or already reached the off-chip memory for write-back. On the other side, since MC sent an acknowledgement (ACK-PUT) to receive the evicted LLC block for write-back, it expects the cache block (DATA message) and believes that the DATA message is on its way.

Evicted, clean LLC blocks are usually not sent for write-back as the off-chip memory has the same version of such blocks. If a valid cache block evicted from an LLC bank is clean, it is either in S or E state (refer Table 5.1). To evict a block that is in S state, LLC/DIR CTLR silently updates the directory and deallocates the LLC block without communicating with the MC (row 1 of Table 5.2). Since the LLC is non-inclusive, evicting a block from an LLC bank does not invalidate other sharers of the block. The corresponding state of the block in MC is not required to change (already in S), and hence no intimation from LLC/DIR CTLR is necessary. However, to evict a clean block that is in E state, LLC/DIR CTLR needs

to communicate with the MC. Even though the block is clean, it is a single copy without any sharers. So, deallocating the LLC block requires the state to be updated to I in both, directory and the MC. The LLC/DIR CTLR initiates a write-back request (PUTE) as given in row 2 of Table 5.2. Upon receiving a PUTE message, the MC updates the corresponding state of the block to I and sends an ACK-PUT. When LLC/DIR CTLR receives an ACK-PUT message, it updates the directory and deallocates the LLC block (row 3 of Table 5.2). Since the evicted block is clean, LLC/DIR controller does not send the block for write-back, and the MC never expects it. Our next optimisation proposes to store these evicted, clean LLC blocks that are in E state, in the local router buffers. We initiate a special write-back DATA message of PUTE, set its *Store* flag in the header and send it towards off-chip memory (row 4 of Table 5.2). SRFD unit in the local router identifies the message based on the *Store* flag and keep it stored there for as long as possible. All the DATA messages of PUTE, PUTO and PUTM are also marked with an additional 1-bit flag *Dirty* to facilitate local reply (explained in Section 5.4.2). The working of the additional SRFD unit is presented in Algorithm 5.

TH-2538_156101018

Algorithm 5: Working of Store, Reply, Forward & Drop (SRFD) unit

Input: $Input\ VCs,\ Packet\ Header$
Output: $Local\ Store\ or\ Reply,\ Block\ Forward\ or\ Drop$
Parameters: $n : Number\ of\ VCs$
Variables: $P_i^j : Packet\ in\ VC_i\ where\ j \in \{new,\ stored\}$

```
1  if P_i^new[Store] == SET then
2      /* Local Store of DATA Message of PUTE/PUTO/PUTM [5.4.1] */
3      Disable VA and SA for P_i^new
4  else if P_i^new[S] == SET then
5      /* Local Reply to GETS Message [5.4.2] */
6      for ∀VC_k where VC_k ≠ NULL do
7          if P_k^stored[Addr] == P_i^new[Addr] then
8              P_k^stored[Src] = P_i^new[Dest]
9              P_k^stored[Dest] = P_i^new[Src]
10             Deallocate VC_i to Drop P_i^new
11             Enable VA and SA for P_k^stored
12 else if P_i^new[X] == SET then
13     /* Local Reply to GETX Message [5.4.2] */
14     for ∀VC_k where VC_k ≠ NULL do
15         if P_k^stored[Addr] == P_i^new[Addr] then
16             if P_k^stored[Dirty] == SET then
17                 P_k^stored[Src] = P_i^new[Dest]
18                 P_k^stored[Dest] = P_i^new[Src]
19                 Deallocate VC_i to Drop P_i^new
20                 Enable VA and SA for P_k^stored
21             else
22                 Deallocate VC_k to Drop P_k^stored
23 /* Forward/Drop of DATA Message of PUTE/PUTO/PUTM [5.4.3] */
24 for ∀VC_k, if VC_k ≠ NULL do
25     /* Defensive Vacate */
26     if ∃VC_k where P_k^stored[Store] == SET then
27         if P_k^stored[Dirty] == SET then
28             Enable VA and SA for P_k^stored
29         else
30             Deallocate VC_k to Drop P_k^stored
31     /* Aggressive Vacate */
32     if ∀VC_k, P_k^stored[Store] == SET then
33         if P_k^stored[Dirty] == SET then
34             Enable VA and SA for P_k^stored
35         else
36             Deallocate VC_k to Drop P_k^stored
```

Table 5.3: LLC/DIR Controller - Block miss. GETS: read request from L1, GETX: write request from L1, DATA: shared block from off-chip memory, EX-DATA: exclusive block from off-chip memory, CAN-PUT: cancel PUT request for coherence

#	memory	GETS	GETX	DATA	EX-DATA
1	I	Send GETS to MEM / IS^D	Send GETX to MEM / IM^D		
2	I	**Add an S Flag as SET**, Send GETS to MEM / IS^D	**Add an X Flag as SET**, Send GETX to MEM / IM^D		
3	IS^D	Stall	Stall	Send DATA to REQ, Add REQ to Sharers / ILS	Send EX-DATA to REQ, Clear Sharers, Make REQ the Owner / ILE
4	IS^D	Stall	Stall	**If (Store == SET)** { **Send CAN-PUT to MEM** } Send DATA to REQ, Add REQ to Sharers / ILS	**If (Store == SET)** { **Send CAN-PUT to MEM** } Send EX-DATA to REQ, Clear Sharers, Make REQ the Owner / **ILM**
5	IM^D	Stall	Stall		Send EX-DATA to REQ, Clear Sharers, Make REQ the Owner / ILM
6	IM^D	Stall	Stall		**If (Store == SET)** { **Send CAN-PUT to MEM** } Send EX-DATA to REQ, Clear Sharers, Make REQ the Owner / ILM

TH-2538_156101018

Table 5.3: Continued

#	State	GETS	GETX	DATA	EX-DATA
7	S	Send Data to REQ, Add REQ to Sharers	Send GETX to MEM / SM^D		
8	S	Send Data to REQ, Add REQ to Sharers	**Add an X Flag as SET**, Send GETX to MEM / SM^D		
9	SM^D	Stall	Stall		Send EX-DATA to REQ, Send INV to Sharers except REQ, Clear Sharers, Make REQ the Owner / ILM
10	SM^D	Stall	Stall		**If (Store == SET)** { **Send CAN-PUT to MEM** } Send EX-DATA to REQ, Send INV to Sharers except REQ, Clear Sharers, Make REQ the Owner / ILM
11	ILS	Forward GETS to Owner, Add REQ to Sharers	Send GETX to MEM / SM^D		
12	ILS	Forward GETS to Owner, Add REQ to Sharers	**Add an X Flag as SET**, Send GETX to MEM / SM^D		
13	E, ILE, O, ILO, M, ILM: Explanation of transitions is out of scope				

TH-2538_156101018

5.4.2 Local Reply: Responding to Block Requests from Routers

On a cache miss, the corresponding LLC bank requests the block from the off-chip memory through the MC. Table 5.3 presents the transitions and actions performed at an LLC/DIR CTLR on a cache block miss. Based on the type of miss, LLC/DIR CTLR issues either a read request (GETS) or a write request (GETX) to send over the NoC towards off-chip memory bank, as given in rows 1, 7 and 11 of Table 5.3. After receiving such a data request, MC fetches the requested block from off-chip memory and forwards it to the LLC/DIR CTLR. As per the request, LLC/DIR CTLR may receive the block either in shared (DATA) or exclusive (EX-DATA) state. Our optimisation identifies GETS/GETX messages when they reach the local router to travel over the NoC to their respective destination. We mark GETS by a 1-bit flag *S* and GETX by a 1-bit flag *X* in the message/packet header as shown in Figure 5.6. Related modifications in the LLC/DIR CTLR is given in rows 2, 8 and 12 of Table 5.3.

For a new packet entering the local router with *S* flag SET (GETS message), SRFD unit compares its requested address with the addresses of non-empty VCs. One of the non-empty VCs may have the requested block stored as a packet when it was evicted from the LLC in the past (DATA message of PUTE/PUTO/PUTM). If a match is found, we can generate a local reply to the GETS message with a stored DATA message. SRFD unit swaps the source and destination of the matched DATA message (stored packet) with the GETS message (request packet) and drop the request packet, as given in Algorithm 5. The new destination of the stored packet is the same LLC bank from where it was evicted. The stored packet is now enabled for VC and switch arbitration, which were disabled to keep it stored in the local router. Since the destination LLC bank is connected to the very same router, the stored packet gets ejected out of the router through the local output port. Avoiding off-chip travel to fetch the requested block significantly reduces LLC miss penalty. In essence, we satisfy a GETS message with a matching DATA message of PUTE/PUTO/PUTM stored in the local router.

For a new packet entering the local router with *X* flag SET (GETX message), SRFD unit performs the same steps, but with an additional check. When a match is found for the requested block in the local router, we have to make sure that the matched block (stored packet) is in exclusive state. SRFD unit checks the *Dirty* flag of the stored packet to identify if it is a DATA message of PUTO/PUTM and not PUTE (explained in Section 5.4.4). If the *Dirty* flag is SET, a local reply with the matched DATA message can be generated from the router. If a match is found in the local router but the *Dirty* flag is not SET (DATA message of PUTE), then the GETX message needs to travel off-chip to fetch the requested block. We drop the matched DATA message of PUTE before forwarding the GETX message towards memory to avoid creating a stale copy of the block. SRFD unit takes care of this operation as

given in lines 21-22 of Algorithm 5. We can satisfy a GETX message with a matching DATA message of PUTO/PUTM only and not of PUTE as it may violate cache coherence.

5.4.3 Block Forward and Drop: Releasing Stored Blocks

The evicted LLC blocks can not be kept stored in local routers forever as they might create injection suppression for others during on-chip congestion. Our optimisation is about exploiting underutilised router buffers and not create buffer (VC) unavailability. If a new packet can not be injected into the local router due to VC unavailability, we immediately take a suitable action to vacate one of the VCs where an evicted LLC block is stored. Based on the local input port buffer contention, we implement two ways of vacating a VC occupied by the evicted LLC blocks; *Defensive Vacate* and *Aggressive Vacate*. When all the VCs are full, *Defensive Vacate* dictates that if any one of the VCs contains a stored LLC block (DATA message of PUTE/PUTO/PUTM), a VC needs to be vacated to make room for new packet injection. However, with *Aggressive Vacate*, only when all the VCs are full with stored LLC blocks, a VC is vacated by the SRFD unit (refer Algorithm 5). When multiple VCs have stored LLC blocks, we vacate the oldest of them. After we identify the VC to be vacated, the *Dirty* flag helps us to decide whether to forward or drop the LLC block stored in that VC. If the VC has a dirty LLC block (DATA message of PUTO/PUTM), it needs to be forwarded towards off-chip memory for write-back. We enable the VC and switch arbitration for the stored block, which were disabled when we kept it stored in the VC buffer. If the VC has a clean LLC block (DATA message of PUTE), it is silently dropped as off-chip memory has the same version of the block. Neither the MC sent any acknowledgement to receive the block nor the LLC/DIR CTLR sent the block for write-back. DATA message of PUTE is a special message generated for our optimisation; to give more scope for a local reply.

5.4.4 Maintaining Cache Coherence

When an evicted, dirty LLC block is sent for write-back, the LLC/DIR CTLR invalidates all its sharers (if exists) and clear the directory entry (refer rows 6 and 9 of Table 5.2). Thus, when we keep such blocks (DATA messages of PUTO and PUTM) stored in the local router, no one else has copy of the blocks. MC is expecting these blocks for write-back and believes that they are on the way. As per the memory hierarchy, a new request for any of these blocks reaches the same LLC/DIR CTLR from where they were evicted. LLC/DIR CTLR issues a new request (GETS/GETX) for the block towards the same MC. When the request reaches the local router, and a local reply is generated, the stored block is sent back to the same LLC bank for where it was evicted. While generating local replies with the stored blocks

(DATA messages of PUTO and PUTM), we need to preserve their states as they are dirty. So, irrespective of the type of request (GETS or GETX), a local reply with a stored dirty block is always generated in M state. This makes sure that the block can not be discarded and will be eventually sent for write-back. When LLC/DIR CTLR receives a block in the form of a DATA or EX-DATA message, we perform an additional check of the *Store* flag (rows 4, 6 and 10 of Table 5.3). If the *Store* flag is found SET, LLC/DIR CTLR learns that the block has come from the local router (local reply). MC was expecting the block for write-back, and now we have generated a local reply with the block. So, we make LLC/DIR CTLR send a CAN-PUT message to the MC. With the CAN-PUT message, MC knows that the evicted block will not reach for write-back and updates the state to M to maintain coherence.

When a clean block that is in E state is evicted, we make LLC/DIR CTLR generate a special DATA message towards off-chip memory (row 4 of Table 5.2). Neither the LLC/DIR CTLR sent the message for write-back, nor the MC expects it. The purpose of this message is to keep the evicted, clean LLC block stored in the local router and improve the scope of local reply. Since the block is clean and not required to be sent for write-back, a local reply with such a stored block (DATA message of PUTE) is always generated in S state. Hence, a DATA message of PUTE can generate local replies to only GETS messages and not GETX messages. Upon receiving the local reply, LLC/DIR CTLR issues a CAN-PUT message to the MC. Since the MC was not expecting anything, when a CAN-PUT arrives, it understands that a local reply is generated in S state and updates its record to maintain coherence.

5.5 Performance Evaluation

We consider the following architectures for evaluation:

- **Baseline:** Without any optimisation.
- **DB-Defensive:** Store evicted, dirty LLC blocks in local router buffers and use *Defensive Vacate* for block forward.
- **DB-Aggressive:** Store evicted, dirty LLC blocks in local router buffers but use *Aggressive Vacate* for block forward.
- **CB+DB-Aggressive:** Store both clean and dirty LLC blocks in local router buffers and use *Aggressive Vacate* for block forward and drop.

Table 5.4: System configuration

Processor	64 OoO x86 cores
L1 Cache	16KB, 4-way, private, split (instruction and data)
L2 Cache (LLC)	128KB×64 cores, 8-way, shared
Memory Bank	4; one located at each corner
Coherence	MOESI distributed directory
NoC	8×8 2D-Mesh topology, 128-bit channel width 3 Virtual Networks (VNs), VN0, VN1, VN2 2/4/6 Virtual Channels (VCs) per VN 1-flit depth control VC, 5-flit depth data VC
Routing	2-stage routers, XY-DOR algorithm VC based wormhole packet-switching
Packets	1-flit for control packet, 5-flit for data packet
Word/Flit/Block	64-bit/128-bit/64B; 2-words/flit, 8-words/block
Benchmarks	SPEC CPU2006 (multi-programmed)

5.5.1 Simulation Framework and Workloads

We model all the architectures on event-driven gem5 simulator [55]. Our system configuration is similar to Intel Xeon Phi Processor 7235 [56] with shared and distributed L2 cache (LLC). Due to certain limitations in gem5, we could not exactly model the cache configuration of Intel Xeon Phi Processor 7235. However, our cache configuration is not chosen to give undue advantage to the proposed optimisations. Rather, it challenges the optimisations with a hit rate of around 90% to 95% for all the benchmarks we evaluate. Our system configuration is presented in Table 5.4 for reference. We modify GARNET [57] module in gem5 to implement the proposed router microarchitecture. We modify MOESI_CMP_directory protocol in Ruby inside gem5 to implement and maintain cache coherence between LLC/DIR CTLR and MC.

To evaluate and analyse the performance, we consider SPEC CPU2006 multi-programmed benchmarks to mimic a modern NoC based TCMP running multiple applications in parallel. We create two different types of workloads with varying Misses Per Kilo Instructions (MPKIs) and re-reference interval (refer Figure 5.4) to run on all the 64 cores of the system, as given in Table 5.5. The first type of workload runs 64 copies of the same benchmark on all the 64 cores (1×64: 64). The second type runs a random combination of 4 different benchmarks with 16 copies each (4×16: 64). By separately profiling each benchmark, we choose a smaller representative window of instructions to have a tractable simulation time. For a relative comparison, all the results are normalised with respect to the baseline architecture.

TH-2538_156101018

Table 5.5: Workload mixes

Mix	Benchmarks	Copies
M1	GemsFDTD	1×64: 64
M2	gromacs	
M3	astar, cactusADM, omnetpp, soplex	4×16: 64
M4	astar, calculix, GemsFDTD, soplex	
M5	bzip2, calculix, gromacs, xalancbmk	
M6	cactusADM, calculix, GemsFDTD, omnetpp	
M7	cactusADM, namd, omnetpp, xalancbmk	

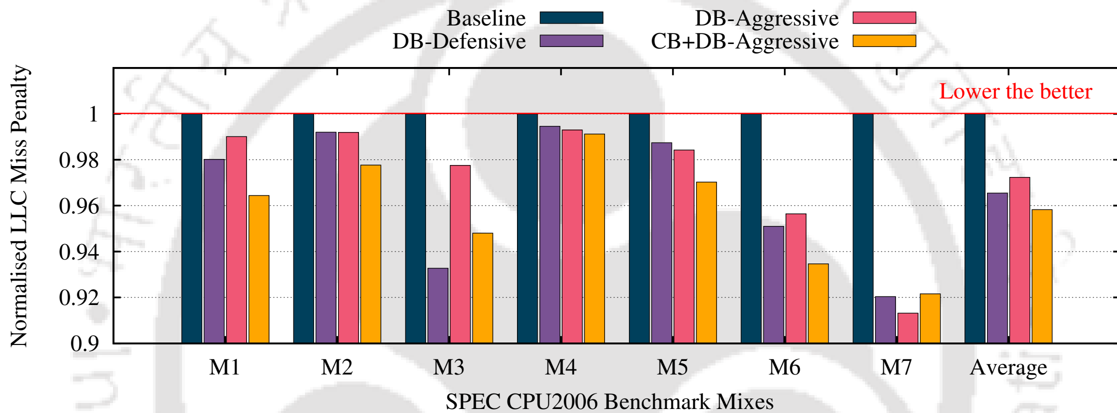


Figure 5.7: LLC miss penalty

5.5.2 Result Analysis and Discussion

5.5.2.1 LLC Miss Penalty

It is defined as the number of cycles required to replace an existing cache block in LLC with an incoming block. LLC miss penalty directly reflects the effectiveness of the proposed local reply optimisation. Figure 5.7 shows the normalised LLC miss penalty with respect to the baseline architecture. With local replies, the proposed architectures reduce LLC miss penalty for all the workload mixes. In general, *CB+DB-Aggressive* architecture performs better, as with more locally stored blocks (both clean and dirty), it has more scope of local replies. However, for mixes M3 and M7, *CB+DB-Aggressive* perform relatively less compared to other proposed architectures. Mixes M3 and M7 contains high MPKI valued benchmarks like *astar*, *soplex* and *xalancbmk* which frequently inject packets into the network. As a result, the evicted LLC blocks are not able to stay stored in the local routers for long. A maximum reduction of 9% in LLC miss penalty is achieved by one of our proposed architectures.

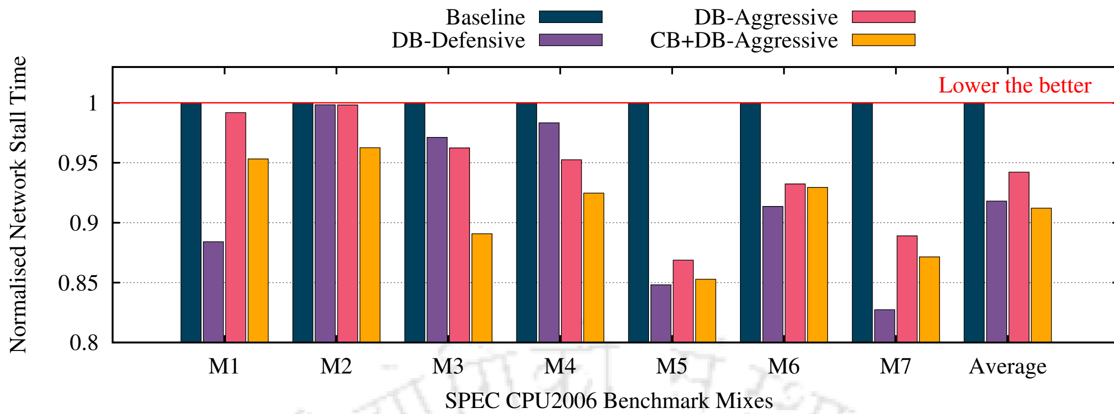


Figure 5.8: Network stall time

5.5.2.2 Network Stall Time

It is defined as the number of cycles the processor stalls waiting for a network packet. Network stall time helps us to understand how storing LLC blocks in local routers impact the NoC communication. Figure 5.8 shows the normalised network stall time with respect to the baseline architecture. As expected, across all the workloads, our proposed architectures significantly reduce network stall time. With local reply from the NoC routers, we avoid both on-chip and off-chip travel time as given in Equation (5.1). Avoiding on-chip travel time indirectly translates into reduced network stall time. However, occupying router buffers for long may affect NoC communication during peak network congestion. To avoid such a scenario, we proposed a *Defensive Vacate* and an *Aggressive Vacate* block forward and drop technique. *DB-Defensive* performs significantly better than *DB-Aggressive* and *CB+DB-Aggressive* architectures for workload mixes M1 and M7. *GemsFDTD* having the highest MPKI value among the presented benchmarks is run on all the 64 cores in M1 workload mix. Since M1 frequently injects network packets from all the 64 cores, the NoC routers are flooded with packets. *DB-Aggressive* and *CB+DB-Aggressive* architectures vacate a VC for new packet injection only when all the VCs are occupied by evicted LLC blocks. As a result, new packets suffer an injection delay that increases the effective network stall time.

5.5.2.3 System Speedup

We use total Instructions Per Cycle (IPC) to compare the system speedup between baseline and the proposed architectures. Figure 5.9 shows the normalised system speedup with respect to the baseline architecture. From the improvements in LLC miss penalty and network stall time, the increase in system speedup with the proposed architectures is intuitive. We achieve

TH-2538_156101018

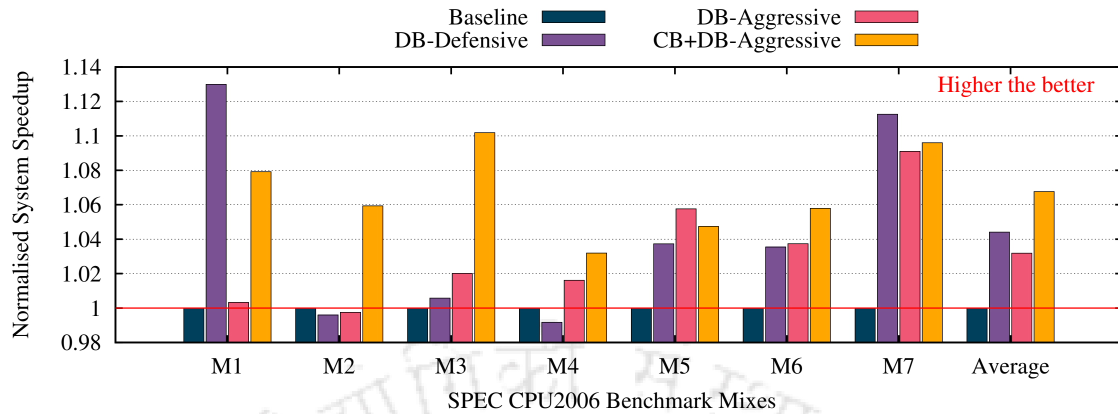


Figure 5.9: System speedup

a maximum system speedup of 13% and an average system speedup of 7% for the presented workloads. In general, *CB+DB-Aggressive* performs well across all the workload mixes with a few exceptions. Mix M1 performs the best with *DB-Defensive* architecture among all the workloads. M1 running *GemsFDTD* benchmark has a very small re-reference interval of only around 22000 cycles (refer Figure 5.4). However, frequent packet injection and the delay due to *Aggressive Vacate* technique employed in the other two architectures, *DB-Defensive* performs better. Mix M2 runs a low MPKI valued benchmark *gromacs*, that has the largest re-reference interval. Since *CB+DB-Aggressive* stores maximum evicted LLC blocks for the longest duration, infrequent LLC block re-references by M2 are also locally replied. The other two architectures suffer performance degradation, as mix M2 being a light workload, can not reap the benefits of local replies. One of the main reasons for mix M7 to perform well across all the architectures is the composition of benchmarks with low re-reference intervals. For example, *cactusADM* benchmark has one of the lowest (quickest) re-reference intervals.

5.5.3 Case Study: Proposed Architecture for L1 Cache

Due to their limited size, the number of cache blocks evicted from L1 caches is much higher than the LLC banks. Intuitively, the proposed architecture should perform better if implemented for the L1 caches. Hence, keeping everything else unchanged (including the system configuration), we modify the proposed architecture to store evicted cache blocks of L1 caches in the local routers. We consider the following architectures for evaluation:

- **Baseline:** Without any optimisation.

Table 5.6: Workload mixes

Mix	Benchmarks	Copies
same	astar, cactusADM, GemsFDTD, gromacs, h264ref, hmmer, lbm, leslie3d, namd, omnetpp, perlbench, sjeng, soplex, sphinx, xalancbmk	1×64: 64
low	gromacs, GemsFDTD, hmmer, astar, h264ref	4×16: 64
med	sphinx, perlbench, cactusADM, omnetpp, soplex	
high	xalancbmk, namd, sjeng, leslie3d, lbm	

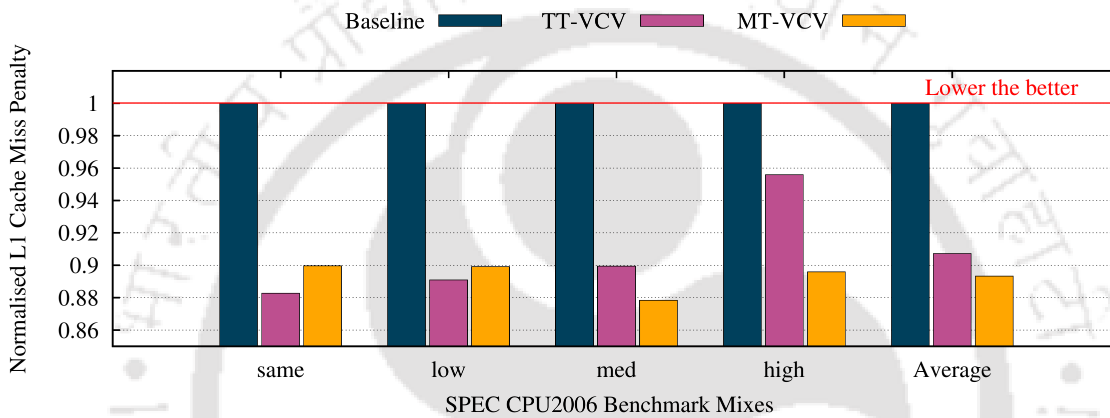


Figure 5.10: L1 cache miss penalty with the proposed architecture for L1 cache

- **TT-VCV:** Store evicted L1 cache blocks in local router till the expiry of a specific time threshold. With each VC, a threshold counter (τ_i) is attached in the local input port.
- **MT-VCV:** Store evicted L1 cache blocks in local router till they are requested by others in the LLC/DIR CTLR.

Workload mixes considered for the evaluation are given in Table 5.6 with similar characteristics as that of Table 5.5. All the results are normalised with respect to the baseline architecture. Figures 5.10, 5.10 and 5.12 shows the normalised L1 cache miss penalty, network stall time and system speedup, respectively, when the proposed architecture is implemented for L1 cache. As expected, the proposed architecture for L1 cache is able to achieve a maximum of up to 12% and an average of 10% overall system speedup for the presented workload mixes. Since the performance is better for L1 cache, this extension is explored in the next chapter.

TH-2538_156101018

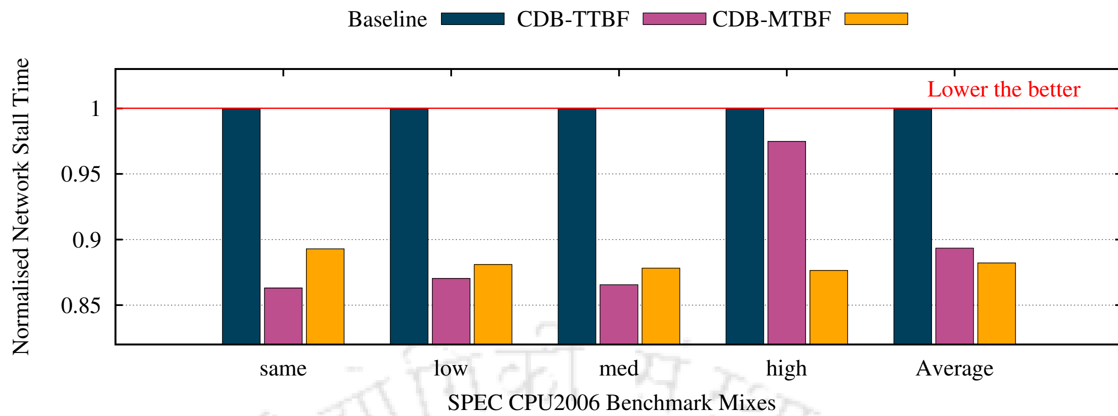


Figure 5.11: Network stall time with the proposed architecture for L1 cache

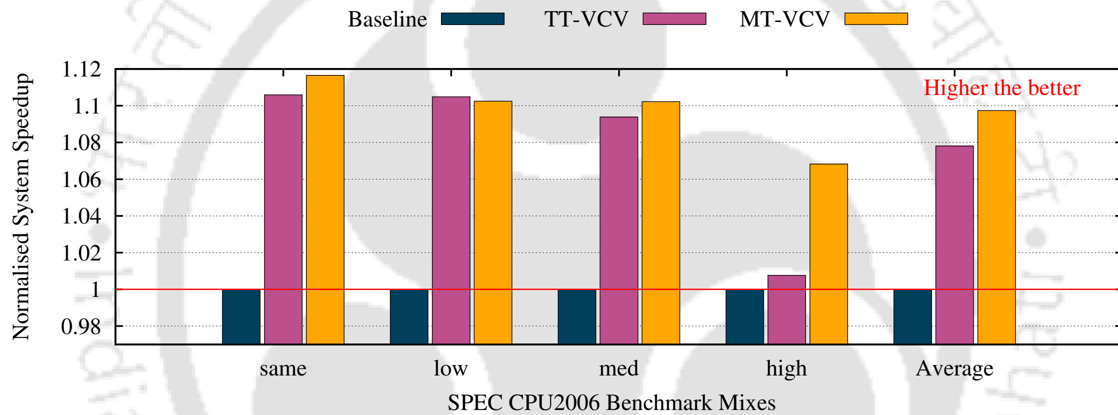


Figure 5.12: System speedup with the proposed architecture for L1 cache

5.6 Sensitivity and Overhead Analysis

5.6.1 Impact of Number of VCs

As our optimisation is based on the storage of evicted LLC blocks in local input port VCs, we also explore the impact of number of VCs/VN in the proposed architectures. For all the results discussed so far, we have considered a number of 4 VCs/VN (as presented in Table 5.4). However, in Figure 5.13, we show that varying the number of VCs/VN will have different implications on the overall system performance. Based on the available number of VCs/VN in an NoC based TCMP, our optimisations achieve appropriate performance.

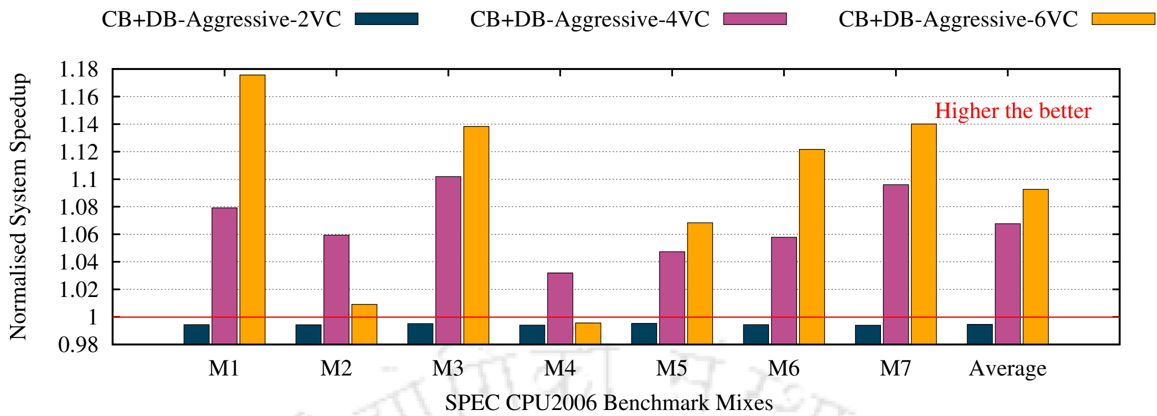


Figure 5.13: Impact of number of VCs

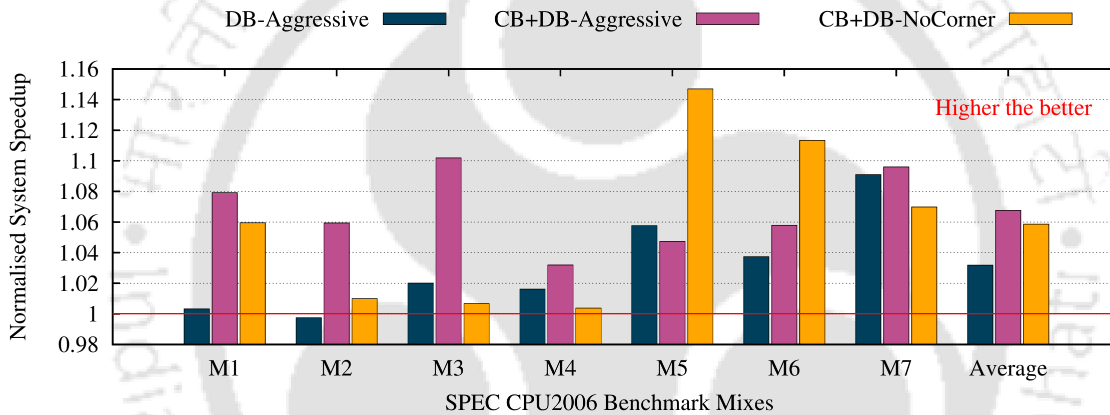


Figure 5.14: Without corner router VCs

5.6.2 Without Corner Router VCs

While understanding the VC availability in local input port (refer Figure 5.2.2), we find that the corner routers have fewer free VCs compared to the others. The unavailability of free VCs can be attributed to the presence of MCs in the corners servicing continuous requests. Since our optimisation is about exploiting free VCs, we do not benefit much from the corner routers. To explore further, we design a heterogeneous architecture called *CB+DB-NoCorner* where the corner routers do not participate in local store and reply optimisation. Figure 5.14 shows the comparison of system speedup for *CB+DB-NoCorner* with other architectures.

5.6.3 Storage, Area and Power Overhead

We use 4 additional bits (*Store*, *Dirty*, *S* and *X*) in the packet header (refer Figure 5.6) to facilitate the working of the SRFD unit. Our NoC uses 128-bit flit channel (refer Table 4.2), but a typical packet header (head flit) uses fewer bits ($\approx$ 64 bits) for its operation. So, we can easily accommodate the additional 4 bits in the head flit without any storage overhead.

We use DSENT [60], integrated with gem5 to get the area, static (leakage) and dynamic power overheads of the proposed routers. In DSENT, we use 22nm processor technology at 1GHz operating frequency. The addition of the SRFD unit in the routers incurs a negligible area overhead of 1.69% and a leakage power overhead of 1.92% compared to the baseline. However, due to significant improvement in overall system performance, we achieve a 4.21% reduction in dynamic power compared to the baseline routers. Since SRFD unit works in parallel to the RC unit (refer Section 5.4.1), it is not in the critical path of execution.

5.7 Related Work

One of the first works by Mizrahi et al. [11] attempted to change the abstraction of NoC from communication to storage. They proposed to migrate the data cache entirely in the NoC routers. Going forward in the same line, Eisley et al. [12] decoupled data and coherence and proposed to keep only the coherence directories in the routers. Yanamandra et al. [13] combined the goodness of both and proposed to keep frequently used cache blocks along with the coherence directories. There is another work by Wang et al. [102] which is also focused on in-network cache and coherence. However, almost all the proposed optimisations require additional storage in the NoC routers. Recently, Jindal et al. [103] proposed to reuse a Design-for-Debug (DFD) storage hardware as extended VCs in the NoC routers.

Existing literature argues that NoC router buffers remain underutilised. For example, SPLASH-2 benchmarks have an average router utilisation of less than 20% [10]. The era of power-aware NoC designs explored buffer-less and minimally-buffered routers [104][105]. Nowadays, the trend is more towards application and computation aware NoC designs [27][106]. Irrespective of the promising design alternatives, input-buffered routers are employed in modern NoC based TCMPs for scalable on-chip bandwidth [99]. In a new and promising attempt, Das et al. [36] proposed to exploit idle buffers of NoC routers to store dirty blocks evicted from the private L1 caches. We have explored using these buffers to store dirty blocks evicted from shared LLC in a preliminary version of this work [107]. Whereas the proposed work can be thought of as an implementation of victim caching [108] for both LLC as well as L1 cache (based on implementation), i.e., using underutilised VCs to reduce miss penalty.

TH-2538_156101018

5.8 Chapter Summary

In this work, we proposed a set of NoC based TCMP architectures to reduce LLC miss penalty. We used the underutilised buffers of NoC routers to keep the evicted LLC blocks stored there for as long as possible. Future references to these evicted blocks, now stored in local routers, are directly replied from the routers. Local reply avoids off-chip travel to fetch a block and significantly reduces LLC miss penalty. We also proposed two approaches to forward/drop the locally stored LLC blocks in due time and avoid injection suppression. Finally, we explored implementing the proposed architecture to store evicted cache blocks from L1 caches. Since the number of evicted cache blocks are way more than what can be accommodated in local routers, we explore nearby/additional storage in the future work.

The next chapter completes our discussion of the proposed work on opportunistic caching.

TH-2538_156101018

Chapter 6

Opportunistic Caching by Exploiting Unused Trace Buffers

This chapter completes the discussion of the proposed work on opportunistic caching, where we exploit underutilised router buffers as well as unused trace buffers to store all the evicted cache blocks. Accommodating more blocks increases the chances of local reply from NoC routers, which reduces miss penalty even further and improves overall system performance.

6.1 Introduction

Since L1 caches are small and frequently accessed, an L1 cache miss almost always evicts a valid block. Based on whether an evicted, valid block is clean or dirty (modified), it is either discarded or sent to the L2 cache for write-back. However, due to temporal and/or spatial locality, if a recently evicted block from the L1 cache is re-referenced, it needs to be fetched again. As the L2 cache is distributed, the re-referenced block is fetched from the corresponding L2 cache bank, which can be anywhere, in the nearest, to the farthest core. In the worst case, when the re-referenced block is not present in the L2 cache (L2 cache miss), it is fetched from the off-chip memory. Increasing the size of L1 cache may delay the block eviction and avoid cache miss penalty up to an extent. However, it is not feasible, as increasing the cache size may impact its hit time and affect instruction pipeline. Increasing the cache size is also not feasible due to the on-chip area and associated cost constraints. In any way, experiencing cache miss penalties on re-reference of recently evicted blocks is inevitable. Frequent cache miss penalties severely hamper the application execution time.

Packet-based NoC use routers to establish on-chip communication between the available cores in a TCMP. Most of the modern NoC based TCMPs employ input-buffered routers for

TH-2538_156101018

scalable on-chip bandwidth [99][100]. Packets on their way from source to the destination are temporarily stored in the input port buffers of NoC routers. Stored packets take part in routing and arbitration decisions and get forwarded towards destination as soon as they get the desired output port. However, due to the low packet injection rate, input port buffers in NoC routers are underutilised. Experimental analysis with standard applications shows that buffer utilisation of routers is very low, except during peak NoC congestion (Section 6.2.2).

Due to the design complexity of modern NoC based TCMPs, post-silicon debug is usually practised to validate a proposed design before going into the production. To aid post-silicon debug and validation, DfD hardware are embedded across various modules and cores of a TCMP [109]. An important phase of the debug involves validating the on-chip interaction between different cores. *Trace buffers* are DfD hardware embedded in NoC routers to record their state for post-silicon debug and validation. However, when a TCMP design goes into production, most of the DfD hardware become non-functional. Since the usage of DfD hardware, including that of the trace buffers, is sporadic and rare after the production, most of them are left unused. While DfD hardware are power-gated, they still occupy chip area.

In this work, we exploit the underutilised storage space available in NoC routers to store evicted L1 cache blocks. If an evicted L1 cache block is dirty, it is sent to the corresponding L2 cache bank for write-back. Such blocks enter the local router [1] as packets, gets stored in local input port buffers, and take part in routing and arbitration to reach their destination for write-back. We propose to disable the arbitration of such blocks and keep them stored in the local router buffers for as long as possible. Since buffer utilisation is low, evicted, dirty L1 cache blocks can be kept stored in the local routers without inducing any NoC congestion. If an evicted L1 cache block is clean, it is discarded and not sent for write-back as the corresponding L2 cache bank has the same copy of the block. We propose to send such blocks to the local router and keep them stored in the unused embedded trace buffer. Now, when a recently evicted L1 cache block is re-referenced, we propose to arrange a quick reply with the stored block from the local router. It is possible as the recently evicted cache block might be present (stored) either in the local input port buffer or the embedded trace buffer. These optimisations can significantly reduce the L1 cache miss penalty and improve overall system performance. In this work, we make the following major contributions:

- **Reply with Stored Dirty Blocks:** We identify evicted, dirty L1 cache blocks when they enter the local routers to travel for write-back towards their destination. We disable arbitration of such blocks to keep them stored in local input port buffers. Future re-reference to the stored blocks are replied by the local router to reduce miss penalty.

[1] Local router connects a tile (core) to the underlying NoC.

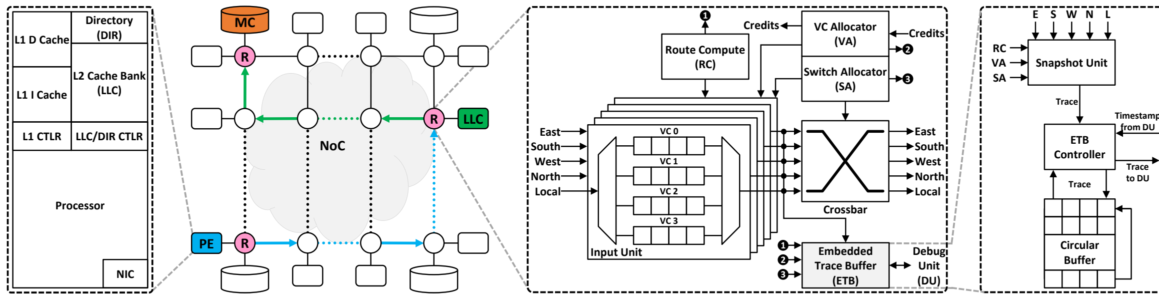


Figure 6.1: Conceptual view of an NoC based TCMP, where data transfer request is travelling from L1 cache to LLC (shown in *blue*) and then from LLC to MC (shown in *green*).

- **Reply with Stored Clean Blocks:** To increase the chances of local reply, we propose to keep the evicted, clean L1 cache blocks (which are usually discarded) in the unused, embedded trace buffer. Local router can reply to the future re-references from local input port buffers as well as trace buffer, which reduces miss penalty even further.

- **Forward/Drop of Stored Blocks:** We propose two techniques to forward dirty L1 cache blocks stored in the local input port buffers. A time-triggered technique based on a certain time threshold, and a message-triggered technique based on a request from the L2 cache bank. We also propose a technique to drop clean L1 cache blocks stored in the embedded trace buffer and inform the corresponding L2 cache bank.

- **Maintain Cache Coherence:** To preserve the states of evicted L1 cache blocks, we propose a new coherence message. Since the L2 cache is shared, we make sure that the proposed optimisations of local store and reply do not violate the cache coherence.

6.2 Background

The conceptual view of a modern NoC based TCMP is shown in Figure 6.1 for reference as we explain the necessary background for the proposed work on opportunistic caching.

6.2.1 L1 Cache Miss Penalty

Since the L1 caches are small, a cache miss is likely to occur against a requested block. On an L1 cache miss, the requested block is fetched from the next level of memory (L2 cache). In NoC based TCMPs, the L2 cache is usually divided into multiple banks and distributed across all the cores, as shown in Figure 6.1. Hence, the requested block needs to be fetched from the corresponding L2 cache bank, which can be anywhere, in the nearest, to the farthest

TH-2538_156101018

core. If the requested block is not present in the L2 cache bank (L2 cache miss), it is fetched from the next level of memory. Latest NoC based TCMPs like Intel Xeon Phi Processor (2016) [89], Princeton Piton Processor (2015) [90], MIT Scorpio Processor (2014) [91], and others, use only 2-levels of on-chip caching. In these systems, L2 serves as the LLC, and a cache miss in the L2 requires the block to be fetched from the off-chip memory through the MCs. Hence, in the worst case, L1 cache miss on a requested block requires the block to be fetched from the off-chip memory, which is very time expensive. The time required to replace an existing block in L1 with the requested, incoming cache block is called L1 cache miss penalty. In modern NoC based TCMPs, L1 cache miss penalty (MP_{L1}) can be given as:

$$MP_{L1} = t_{Request}^{L1-LLC} + T_{Access}^{LLC} + t_{Reply}^{LLC-L1} \tag{6.1}$$

where

$$T_{Access}^{LLC} = \begin{cases} T_{Hit}^{LLC} & \text{if LLC Hit} \\ T_{Miss}^{LLC} + MP_{LLC} & \text{if LLC Miss} \end{cases} \tag{6.2}$$

Here, T_j^i is the time taken by module i to complete a task j whereas, t_k^{i-j} is the time taken by message k to travel from module i to module j through the underlying NoC. For example, T_{Access}^{LLC} is the time taken by an LLC bank to access a block whereas, $t_{Request}^{L1-LLC}$ is the time taken by a cache miss request to travel from the L1 cache to the corresponding LLC bank.

As given in Equation (6.1), L1 cache miss penalty is dominated by the on-chip transfer time (t_k^{i-j}). In the worst case of an LLC miss, the dominance of this transfer time is even more since the miss request now travels to MC and come back with a reply from off-chip memory. The corresponding MC can be located in any of the corner routers (refer Figure 6.1). During on-chip congestion, the transfer time can get longer due to unknown router delays along the way. Hence, the underlying NoC plays an important role in L1 cache miss penalty.

6.2.2 VC Availability

NoC based systems employ routers, which have three design alternatives: input-buffered, minimally-buffered and buffer-less, each with different pros and cons. To meet the worst-case performance bandwidth, modern TCMPs prefer input-buffered NoC routers [99][100]. Input buffers are further divided into VCs for deadlock-free routing and better utilisation [101]. As shown in Figure 6.1, packets entering a router through different input ports (east, south, west, north and local) get temporarily stored in the available VCs and take part in routing

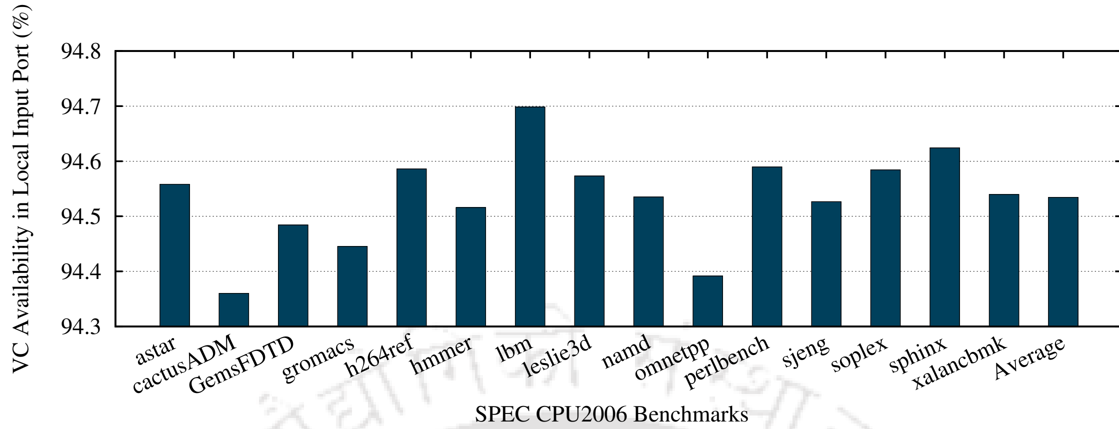


Figure 6.2: VC availability in local input port

and arbitration decisions. VC availability (VCA_n) in NoC based TCMPs can be given as:

$$VCA_n = \frac{Cycles\ when\ n\ VCs\ are\ Free}{Total\ Execution\ Cycles} \tag{6.3}$$

Figure 6.2 shows the VC availability in local input port of NoC routers for a set of standard multi-programmed benchmarks (SPEC CPU2006 [62]). As the average injection rate of these benchmarks is only around 5%, except during peak NoC congestion, at least one VC is always free ($\approx$ 95%). A similar observation is expected for standard multi-threaded benchmarks as well (PARSEC 3.0 [63]), where the average injection rate is even lower. The observation in Figure 6.2 is in sync with the conclusions in the existing state-of-the-art [30][31][32].

> Modern NoC based TCMPs use input-buffered routers for worst-case performance bandwidth, but buffers (VCs) remain underutilised except during peak NoC congestion.

6.2.3 Embedded Trace Buffer

Pre-silicon validation is a standard practice in the process of any hardware system design. It involves theory-based formal verification of the design for functional correctness and simulation-based verification of the RTL description [110]. Increasing core counts and the need for an efficient and scalable communication increase the design complexity of modern NoC based TCMPs. For such systems, theory-based formal verification suffers from state space explosion problem. Furthermore, simulation-based verification is very slow. Hence, exhaustively exploring the entire design space of an NoC based TCMP is not feasible in a time-bound pre-silicon validation. Thus, post-silicon debug and validation is necessary.

TH-2538_156101018

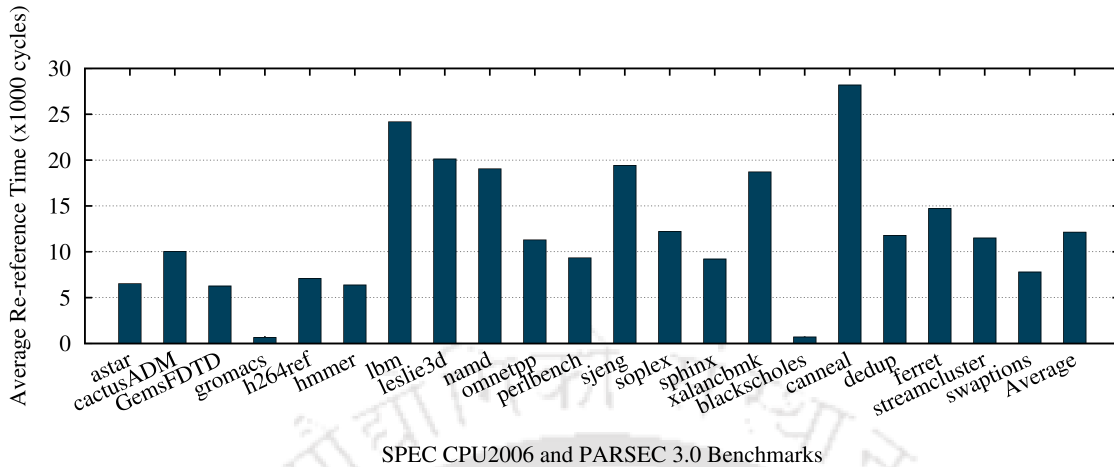


Figure 6.3: Re-reference time of evicted L1 cache blocks

Post-silicon debug and validation begins when the first few silicon prototypes of the proposed design are available. Longer tests are run on actual hardware (prototype) in native speed for thorough validation. Hence, post-silicon validation can expose functional bugs that might have been missed during pre-silicon validation. Key to an effective post-silicon debug and validation lies in the observability and controllability of internal signals when the tests are run. To facilitate debug and validation, DfD hardware are embedded across various modules and cores of a TCMP [109]. Among many other things, DfD hardware can trace internal signals, dump contents of registers and memory, patch microcode and firmware, create user-defined triggers and interrupts, etc. An important phase of debugging the NoC based TCMPs is to validate the on-chip interaction between different cores. *Trace buffers* are DfD hardware embedded in the NoC routers to record their state for post-silicon debug and validation. Trace buffer and Embedded Trace Buffer (ETB) are used interchangeably throughout the text, thus should not be confused with. Trace buffers periodically take snapshot of the NoC router and store it as a compressed trace in a circular memory storage, as shown in Figure 6.1. Size of trace buffers typically varies between 2KB to 8KB in different modules and can roughly store 10K to 30K lines of compressed trace. After successful debug and validation, the silicon prototype goes for mass production. Thereafter, most of the DfD hardware, including the trace buffers, become non-functional. Since the usage of DfD hardware is sporadic and rare after the production, most of them are left unused. Even though the DfD hardware are power-gated, their area footprint remains in the routers (chip) without any benefit.

Embedded Trace Buffers (ETBs) facilitate post-silicon debug and validation of NoC routers, but they are left unused after production, leaving a storage and area footprint.

6.3 Motivation

Since the L1 caches are small and frequently accessed, an L1 cache miss almost always evicts a valid block. However, due to temporal locality, a recently evicted L1 cache block may be re-referenced. Since a cache block contains multiple words, even for spatial locality, a recently evicted L1 cache block may be re-referenced. The duration from the eviction of an L1 cache block to the request of the same block in future, by the same core is called re-reference time. Re-reference time (RT_i^j) in modern NoC based TCMPs can be given as:

$$RT_i^j = |Request(B_i^j)|_{T_y} - |Eviction(B_i^j)|_{T_x} \tag{6.4}$$

where at time T_x, cache block B_i was evicted from core j and in the future at time T_y, the same cache block B_i is requested again by the same core j. Figure 6.3 shows the average re-reference time for different SPEC CPU2006 and PARSEC 3.0 benchmarks. For example, in a 64-core NoC based TCMP running a multi-programmed benchmark *astar*, an evicted L1 cache block is re-referenced within an average time of 6532 cycles. Across all the presented benchmarks, on average, within a small interval of around 12000 cycles, an evicted L1 cache block is re-referenced. This interval indirectly indicates how frequently applications running in NoC based TCMPs suffer from L1 cache miss penalty due to unfortunate block evictions.

In this work, we explore ways to reduce L1 cache miss penalty in NoC based TCMPs. Evicted, clean L1 cache blocks are discarded, whereas dirty L1 cache blocks are sent over the NoC to the corresponding L2 cache bank for write-back. To reach their destination for write-back, evicted, dirty L1 cache blocks enter the local router through the local input port as packets. They temporarily get stored in the available VCs and take part in routing and arbitration decisions to get the desired output port towards their destination. In this work, we propose to disable the arbitration of such evicted, dirty L1 cache blocks while they are stored in the local input port VCs. Without taking part in the arbitration, these evicted, dirty cache blocks can not get the desired output port and leave the local router. From the observation in Figure 6.2, we know that local input port VCs remain underutilised (free) most of the time. Hence, we can keep the evicted, dirty L1 cache blocks stored in the local router for as long as possible without creating injection suppression for other packets. During the time an evicted, dirty L1 cache block is locally stored, a re-reference request for the same block by the same core can be replied from the local router. Hence, we propose to generate direct reply from the local router if a requested block is present in the local VCs. From Equation (6.1) and Section 6.2.1, we know that L1 cache miss penalty involves on-chip travel and may also suffer from NoC communication delay. Local reply to L1 cache miss requests from the NoC routers can avoid the on-chip travel altogether and get significant reduction in miss penalty.

Unlike dirty blocks, clean blocks are discarded after eviction from the L1 cache since a write-back is not necessary. However, the number of clean blocks evicted from L1 cache is much more than the number of dirty blocks. From the observation in Figure 6.3, we are aware that an evicted L1 cache block (clean or dirty) is re-referenced within a small interval of around 12000 cycles. Hence, to improve the chances of local reply, we propose to bring evicted, clean L1 cache blocks to the local routers and keep them stored in the local input port VCs. Nevertheless, the underutilised local VCs are already employed to store evicted, dirty L1 cache blocks. Making the evicted, clean and dirty blocks compete against each other for a place in the local VCs kill the purpose of local store. From the conclusion in Section 6.2.3, we know that a DfD storage infrastructure called trace buffer, embedded in NoC routers is left unused. In this work, we re-purpose the unused trace buffer in NoC routers to store evicted, clean L1 cache blocks; which are normally discarded. Until a block is stored in the trace buffer, a re-reference to the same block by the same core can be serviced from the local router. With the evicted, dirty L1 cache blocks stored in local VCs and the evicted, clean L1 cache blocks now stored in the trace buffer, our chances local replies increase many-fold. Our proposed optimisations to generate immediate local reply from the NoC routers can significantly reduce L1 cache miss penalty, thereby improving overall system performance.

Proposed Solution:

Store evicted L1 cache blocks in underutilised NoC router buffers (VCs) and unused trace buffers (ETBs). Upon re-reference, generate direct reply from the local routers.

6.4 Opportunistic Caching in NoC

A conceptual view of the router microarchitecture that implements our proposed optimisations is given in Figure 6.4. We consider 2-levels of on-chip caching with MOESI distributed directory coherence protocol. Keeping Figure 6.4 and MOESI protocol as a reference, we explain the detailed working of the proposed architecture in the following sub-sections.

TH-2538_156101018

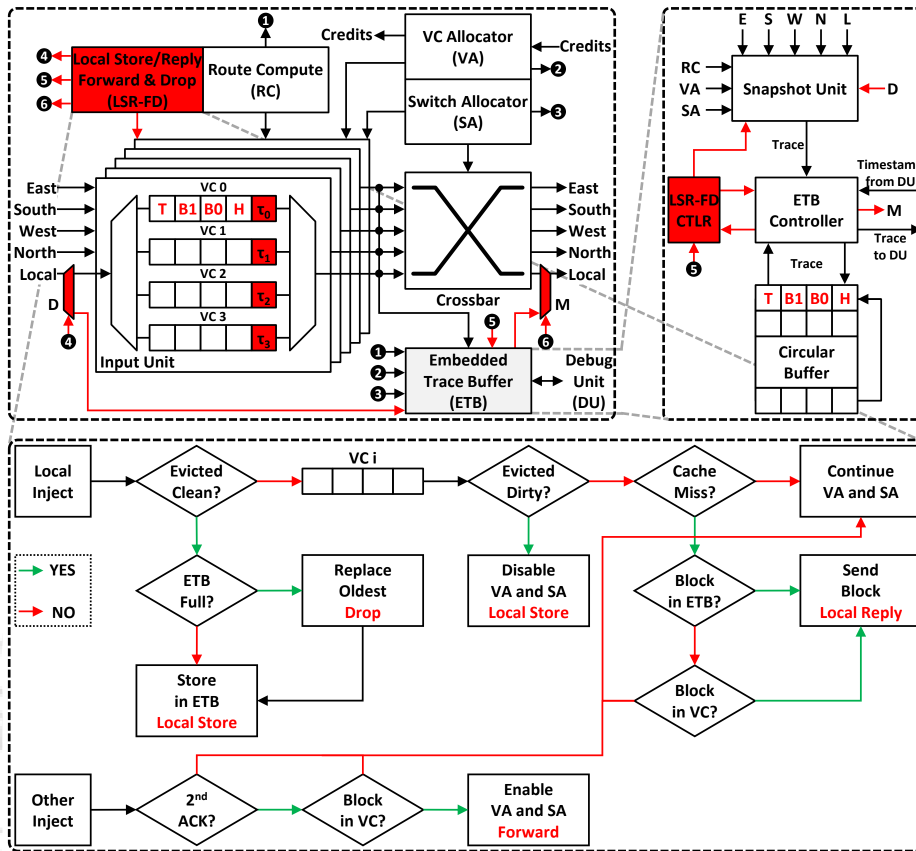
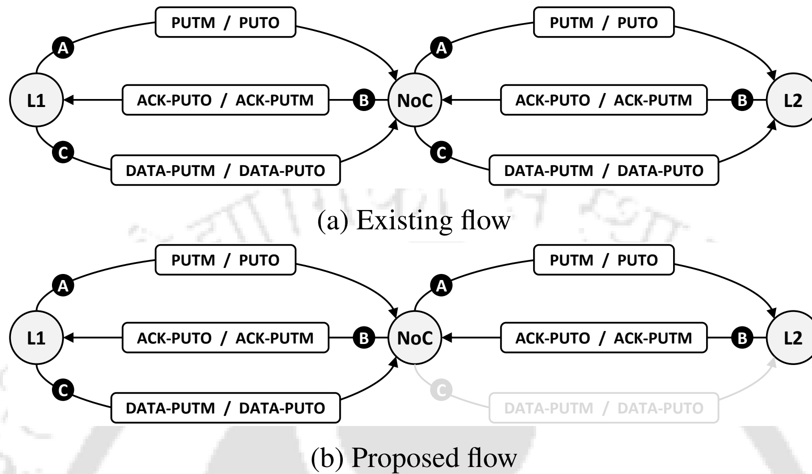


Figure 6.4: Conceptual view of the proposed router microarchitecture. All the additional units and links are shown in *red*. Evicted L1 cache blocks enter the NoC as packets and get divided into multiple smaller units called flits (H, B0, B1, T). Based on whether a block is clean or dirty, the corresponding flits get stored in either the trace buffer or the local VCs.

In MOESI distributed directory coherence based shared cache organisation, a block can be in **Modified (M):** Possibly different from memory and only copy, **Owned (O):** Possibly different from memory and possibly shared, **Exclusive (E):** Same as memory and only copy, **Shared (S):** Same as memory/owner and possibly shared, or **Invalid (I):** Invalid copy state. Exclusive (E) state can be considered as a subset of Owned (O) state. Our discussion includes the following coherence messages from the protocol. GETS/ GETX: Read/Write request, DATA-GETS/DATA-GETX: Shared/Exclusive data, PUTS/PUTO/PUTM: Write-back request for Shared/Owned/Modified data, ACK-PUTS/ACK-PUTO/ACK-PUTM: Acknowledgement for PUTS/PUTO/PUTM write-back, DATA-PUTS/DATA-PUTO/DATA-PUTM: Shared/Owned/Modified data for write-back, UNBLOCK: Intimation for DATA-PUTS drop.

Src	Dest	Addr	. . .	Evicted	Clean	Miss	Forward
				1-bit	1-bit	1-bit	1-bit

Figure 6.5: Modified message/packet header



(a) Existing flow

(b) Proposed flow

Figure 6.6: Eviction of a dirty L1 cache block

6.4.1 Block Store in Router Buffers

A valid block evicted from an L1 cache can be either clean (shared) or dirty (owned/modified). Clean cache blocks are discarded and dirty cache blocks are sent for write-back. For evicted, dirty cache blocks, a PUTO/PUTM write-back request is initiated from the L1 Cache Controller (L1 CTLR) to the corresponding L2 cache bank. As shown in Figure 6.6a, such requests travel through the underlying NoC and reach their destination (Ⓐ). After receiving a request, the corresponding L2 Cache Bank Controller (L2 CTLR) replies with an acknowledgement (ACK-PUTO/ACK-PUTM) to receive the evicted, dirty cache block for write-back (Ⓑ). As soon as L1 CTLR receives an acknowledgement, the evicted block is sent towards the L2 cache bank as a DATA-PUTO/DATA-PUTM message(Ⓒ). All the data and control messages enter the local NoC router as packets, gets stored in the available VCs and take part in routing and arbitration decisions to reach their respective destination. For our optimisations, all the evicted L1 cache blocks (both clean and dirty) are marked with a 1-bit flag (*Evicted*) in their message/packet header for identification, as shown in Figure 6.5.

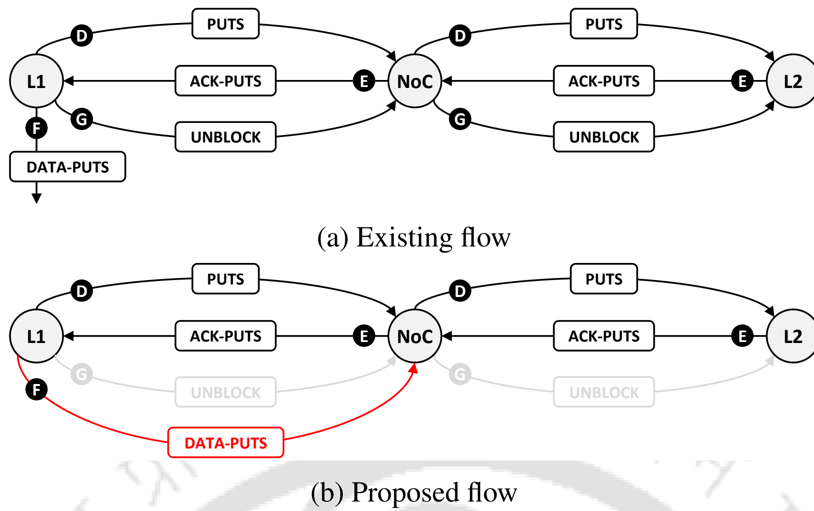
Our first optimisation targets DATA-PUTO/DATA-PUTM write-back data messages on their way to the destination. When any new packet enters the local router and gets buffered in the VC for routing and arbitration, *Evicted* flag is checked by the additional **Local Store/Reply, Forward & Drop (LSR-FD)** unit as shown in Figure 6.4. If the *Evicted* flag is SET, we know that the corresponding packet is actually an evicted, dirty L1 cache block

(DATA-PUTO/DATA-PUTM), which is on its way to the L2 cache bank for write-back. Even though the *Evicted* flag is set for both clean and dirty cache blocks, the identified block in the router can not be clean as they are dropped after eviction. We consider two-stage NoC routers (stage-1: RC, stage-2: VA and SA) where LSR-FD unit works in stage-1 in parallel with the Route Compute (RC) unit. While a check is performed by the LSR-FD unit to identify an evicted block (packet), route for the packet is also computed in parallel by the RC unit. If the *Evicted* flag is found SET for a packet (DATA-PUTO/DATA-PUTM) in stage-1, LSR-FD unit disables stage-2, i.e. VC and switch arbitration for the packet (VA and SA). Without arbitration, such packets can not leave the local router, as shown in Figure 6.6b. This way, we keep all the evicted, dirty L1 cache blocks stored in the local input port VCs of local router for as long as possible (explained in Section 6.4.4). Since VCs are underutilised due to low packet injection rate, keeping the evicted, dirty L1 cache blocks stored in local routers do not usually create any injection suppression. Both L1 and L2 caches are unaware of the proposed optimisation. For L1 CTLR, the evicted, dirty block is on its way or already reached the corresponding L2 cache bank for write-back. On the other side, since the L2 CTLR already sent an acknowledgement to receive the block, it believes that the block is on its way.

6.4.2 Block Store in Trace Buffers

If an evicted L1 cache block is clean (shared), it is simply discarded since a write-back is not necessary. As shown in Figure 6.7a, L1 CTLR initiates a PUTS write-back request towards the corresponding L2 cache bank (Ⓓ). Upon receiving the request, L2 CTLR sends an acknowledgement (ACK-PUTS) to the L1 cache (Ⓔ). The acknowledgement from L2 CTLR serves as the permission to discard the evicted, clean block (DATA-PUTS) in the L1 cache. Accordingly, L1 CTLR drops the evicted cache block (Ⓕ) and intimate the L2 cache bank with an UNBLOCK message (Ⓖ). After receiving the UNBLOCK message, the L2 CTLR removes the L1 cache entry from the corresponding sharer list of that cache block.

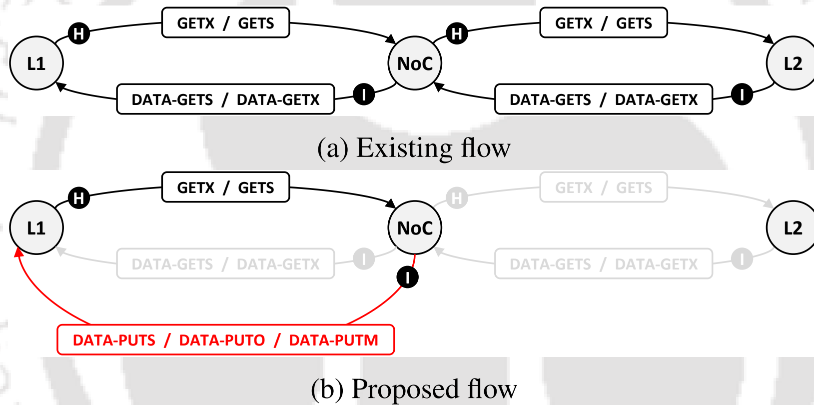
Our second optimisation targets clean L1 cache blocks that are discarded after eviction (DATA-PUTS). Since clean blocks are more frequently evicted, we propose to keep them stored to increase our chances of local reply (explained in Section 6.4.3). As shown in Figure 6.7b, instead of dropping DATA-PUTS, we redirect the message towards NoC (Ⓕ). We also prohibit the transfer of UNBLOCK message towards the L2 cache bank. Since an acknowledgement is already sent, L2 CTLR believes that the corresponding block is discarded, and the UNBLOCK message is on the way. Now, the challenge is to accommodate the evicted, clean L1 cache blocks in local routers, which are normally discarded. Though we advocate that modern NoC based TCMPs use input-buffered routers and buffers (VCs) are underutilised, but VCs are limited. With the first optimisation in Section 6.4.1, underutilised



(a) Existing flow

(b) Proposed flow

Figure 6.7: Eviction of a clean L1 cache block



(a) Existing flow

(b) Proposed flow

Figure 6.8: L1 cache miss on a requested block

VCs are already employed to store evicted, dirty L1 cache blocks when they enter the local router to travel for write-back. Making the clean blocks compete with dirty blocks for the limited VCs available in local input port defeats the purpose of accommodating more blocks. Thus, we consider storing the evicted, clean L1 cache blocks in the unused trace buffer of the local routers. To facilitate the optimisation, all the evicted, clean L1 cache blocks are marked with a 1-bit flag (*Clean*) in their packet header as shown in Figure 6.5. When a new packet enters the local router, a 1:2 DEMUX (D) checks the *Clean* flag and if found SET, routes the packet towards the ETB (refer Figure 6.4). These packets are actually evicted, clean L1 cache blocks sent to NoC by our optimisation to be locally stored. We have re-purposed the ETB with the help of LSR-FD unit to accommodate such incoming packets. The detailed working of LSR-FD unit is presented in Algorithm 6. All the evicted, clean L1 cache blocks are now stored in the local routers to facilitate local replies when a possibility appears.

TH-2538_156101018

Algorithm 6: Working of Local Store/Reply, Forward & Drop (LSR-FD) unit

Input: $Input\ VCs,\ ETB,\ Packet\ Header$
Output: $Local\ Store\ or\ Reply,\ Block\ Forward\ or\ Drop$
Parameters: $n : Number\ of\ VCs,\ m : Number\ of\ ETB\ Entries,$
$\tau_i : Time\ Threshold\ of\ VC_i \mid 0 \leq i < n$
Variables: $P^{new}_{local} : Packet\ Entered\ through\ Local\ Input\ Port,$
$P^{stored}_{ETB_i} : Packet\ Stored\ in\ ETB_i \mid 0 \leq i < m,$
$P^{j}_{VC_i} : Packet\ in\ VC_i \mid 0 \leq i < n,\ j \in \{new,\ stored\}$

```
1  if P^new_local[Clean] == SET then
2  |  /* Local Store of DATA-PUTS [6.4.2] */
3  |  Enqueue ETB_i to Store P^new_local
4  |  Increment i for Next Store
5  else
6  |  if P^new_VCi[Evicted] == SET then
7  |  |  /* Local Store of DATA-PUTO/DATA-PUTM [6.4.1] */
8  |  |  τ_i = 64 ∨ 128 ∨ ... ∨ 1024
9  |  |  Disable VA and SA for P^new_VCi
10 |  else if P^new_VCi[Miss] == SET then
11 |  |  /* Local Reply of GETS/GETX [6.4.3] */
12 |  |  for ∀ETB_j | ETB_j ≠ NULL do
13 |  |  |  if P^stored_ETBj[Addr] == P^new_VCi[Addr] then
14 |  |  |  |  Dequeue ETB_j to Send P^stored_ETBj
15 |  |  |  |  Deallocate VC_i to Drop P^new_VCi
16 |  |  for ∀VC_k | τ_k > 0 do
17 |  |  |  if P^stored_VCk[Addr] == P^new_VCi[Addr] then
18 |  |  |  |  τ_k = 0
19 |  |  |  |  P^stored_VCk[Src] = P^new_VCi[Dest]
20 |  |  |  |  P^stored_VCk[Dest] = P^new_VCi[Src]
21 |  |  |  |  Enable VA and SA for P^stored_VCk
22 |  |  |  |  Deallocate VC_i to Drop P^new_VCi
23 /* Defensive Vacate of DATA-PUTO/DATA-PUTM [6.4.4] */
24 for ∀VC_i | VC_i ≠ NULL do
25 |  if ∃VC_i | P^stored_VCi[Evicted] == SET then
26 |  |  τ_i = 0
27 |  |  P^stored_VCi[Evicted] = RESET
28 |  |  Enable VA and SA for P^stored_VCi
29 /* TT-BF of DATA-PUTO/DATA-PUTM [6.4.4.1] */
30 for ∀VC_i | τ_i > 0 do
31 |  τ_i = τ_i − 1
32 |  if τ_i == 0 then
33 |  |  P^stored_VCi[Evicted] = RESET
34 |  |  Enable VA and SA for P^stored_VCi
35 /* MT-BF of DATA-PUTO/DATA-PUTM [6.4.4.2] */
36 if P^new_VCi[Forward] == SET then
37 |  for ∀VC_j | τ_j > 0 do
38 |  |  if P^stored_VCj[Addr] == P^new_VCi[Addr] then
39 |  |  |  τ_j = 0
40 |  |  |  P^stored_VCj[Evicted] = RESET
41 |  |  |  Enable VA and SA for P^stored_VCj
42 |  |  |  Deallocate VC_i to Drop P^new_VCi
```

6.4.3 Block Reply from Routers

On a cache miss, the L1 CTLR issues a GETS/GETX request to the corresponding L2 cache bank (Ⓗ), as shown in Figure 6.8a. Based on the received request, L2 CTLR replies with the block either in shared (DATA-GETS) or exclusive (DATA-GETX) state (Ⓘ). Since the L2 cache is distributed, based on the location of the corresponding L2 cache bank and the underlying NoC congestion, reply takes an indefinite time to reach the requesting L1 cache. In the worst case of an L2 cache miss, the reply message can take even longer time.

Our next optimisation identifies GETS/GETX messages and attempts local reply with the stored DATA-PUTS/DATA-PUTO/DATA-PUTM messages from the local NoC routers (Ⓘ), as shown in Figure 6.8b. All the data request messages (GETS and GETX) are marked with a 1-bit flag (*Miss*) in their packet header, as shown in Figure 6.5. When a new packet enters the local router with its *Miss* flag SET, the LSR-FD unit attempts to generate a local reply if possible. These packets are actually GETS/GETX request messages carrying the address of a requested cache block. The LSR-FD unit compares the requested address with the addresses of the stored cache blocks in the trace buffer. One of the entries may have the requested block since the stored blocks were evicted from the same L1 cache in the recent past. If a match is found, we can generate a local reply to the cache miss request with a stored DATA-PUTS message (Ⓘ), as shown in Figure 6.8b. The matched block (packet) is forwarded from the trace buffer to the local output port (refer Figure 6.4). A 2:1 MUX (M) checks the *Clean* flag and if found SET, knows that the packet is a local reply and has come from the embedded trace buffer. Such packets are given priority to take the local output port for their destination.

If the requested address is not found in the trace buffer, the LSR-FD unit compares it with the addresses of all the stored blocks in the non-empty local input port VCs. A match is possible since the stored blocks in local input port VCs are recently evicted, dirty L1 cache blocks. If a match is found, we can generate a local reply to the cache miss request with a stored DATA-PUTO/DATA-PUTM message (Ⓘ), as shown in Figure 6.8b. LSR-FD unit swaps the source and destination of the matched block (packet) with the request packet (GETS/GETX) and drop the GETS/GETX packet as given in Algorithm 6. The new destination of the matched block (packet) is the same L1 cache from where it was evicted. The stored packet is now enabled for VC and switch arbitration, which were disabled earlier to keep it stored in the local router. Since the destination (L1 cache) is connected to the very same local router, such packets get ejected through the local output port. Avoiding on-chip travel (also off-chip travel in case of an L2 cache miss) to fetch a requested block (DATA-GETS/DATA-GETX) significantly reduces L1 cache miss penalty. In realisation, the proposed optimisations satisfy a GETS/GETX request with a matching DATA-PUTS/DATA-PUTO/DATA-PUTM message stored in the local router (VCs or ETB).

6.4.4 Block Forward and Drop from Routers

As we store evicted L1 cache blocks in the underutilised VCs and unused ETB of the local NoC routers, we face two key challenges. First, at a time during NoC congestion (high packet injection rate), keeping the VCs occupied with stored blocks may create VC unavailability for incoming packets. Second, an evicted block that is now stored in the local router may be requested by others in the L2 cache bank resulting in the delay of their execution. Since our work is all about opportunistic caching, we take all the necessary steps to make sure that the proposed local store and reply does not hamper the usual NoC communication for others.

If a new packet can not be injected into the local router due to VC unavailability, we employ a *Defensive Vacate* technique to identify one of the VCs to be vacated where an evicted block is stored. When all the VCs are full, *Defensive Vacate* dictates that if any one of the VCs contains a stored block, that VC needs to be vacated. When multiple VCs have stored blocks, the oldest of them is vacated. *Defensive Vacate* is given in lines 23-28 of Algorithm 6. The identified VC contains an evicted, dirty L1 cache block since clean blocks are stored in the ETB. To vacate the identified VC, we must forward the stored, dirty block towards its destination for write-back. LSR-FD unit simply enables the VC and switch arbitration for the stored block, which were disabled when we kept it stored in the VC. This action ensures that the corresponding VC will be free in subsequent cycles and hence make room for new injection. NoC congestion can create scenarios like hotspots and Head-of-Line (HoL) blocking. In such cases, vacating all the VCs instead of just one, having stored blocks may revive the network. However, run-time detection of scenarios like HoL blocking is difficult [111]. Nevertheless, *Defensive Vacate* can be modified accordingly when required.

Even in the absence of injection pressure, the status of a stored cache block (both clean and dirty) may be expected in the L2 cache bank by other requesters to continue their execution. Since L2 CTLR is expecting a reply (UNBLOCK/DATA-PUTO/DATA-PUTM), it makes all the requesters wait for the status. Trace buffer accommodates evicted, clean L1 cache blocks in a small circular queue (refer Figure 6.4) and hence such blocks do not stay stored for very long. When the trace buffer is full, the oldest clean cache block is replaced by an incoming block. When the oldest clean cache block is replaced (dropped), an UNBLOCK message is sent to the corresponding L2 cache bank for necessary action (explained in Section 6.4.5). To make sure that the wait for evicted, dirty cache blocks in the corresponding L2 cache bank is not too long, we propose the following two techniques.

6.4.4.1 Time-Triggered Block Forward (TT-BF)

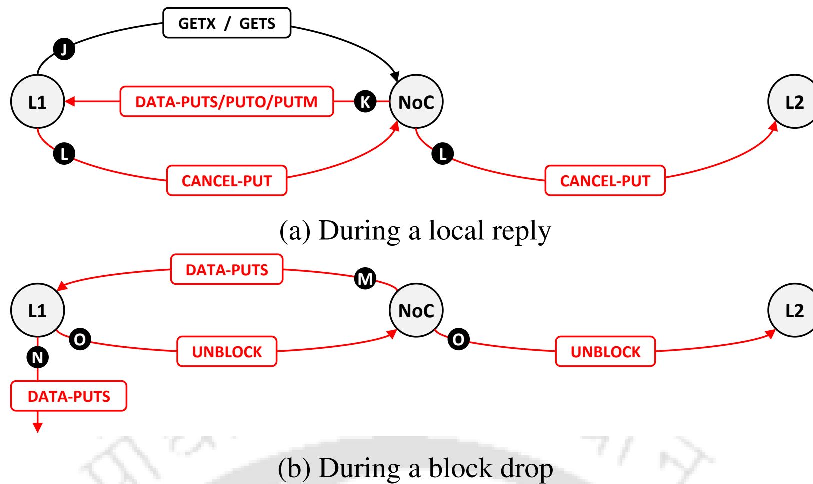
An evicted, dirty L1 cache block is stored in the local input port VC of a local NoC router until a certain time threshold which is decided based on the re-reference time of the evicted blocks (refer Figure 6.3). We add a threshold counter (τ_i) correspond to each VC of the local input port, as shown in Figure 6.4. When a counter reaches the threshold, the stored cache block (packet) in the corresponding VC is enabled for VC and switch arbitration. This action triggers forwarding of the stored, dirty cache block towards its destination for write-back. Till the time a cache block is stored in the local router, access requests for the block by other requester is delayed by a time equal to the threshold in the corresponding L2 cache bank.

6.4.4.2 Message-Triggered Block Forward (MT-BF)

An evicted, dirty L1 cache block is stored in the local input port VC of a local router until it is requested by someone in the corresponding L2 cache bank. In such a case, we make the corresponding L2 CTLR resend an acknowledgement (ACK-PUTO/ACK-PUTM) for the requested block. Such acknowledgements are marked with a 1-bit flag (*Forward*) in their packet header, as shown in Figure 6.5. An acknowledgement was already sent, and the L2 CTLR is now waiting for the block for write-back. We send the second acknowledgement to inform that the cache block is requested by someone. When the second acknowledgement arrives at the destination router (local router of the stored cache block), the corresponding stored block is enabled for VC and switch arbitration. LSR-FD unit takes the decision when it finds the *Forward* flag SET for an incoming packet, as given in lines 35-42 of Algorithm 6. Then, the second acknowledgement reaches the L1 CTLR, where it is simply ignored.

6.4.5 Cache Coherence

As the shared L2 cache is involved in the proposed optimisations, we have to make sure that cache coherency is maintained. After sending an acknowledgement (ACK-PUTO/ACK-PUTM), the L2 CTLR waits for the corresponding DATA-PUTO/DATA-PUTM message to initiate write-back. Since the evicted, dirty L1 cache block is coming for write-back, it must be the only copy in the entire system. Therefore, as long as evicted, dirty L1 cache blocks are stored in local routers, no special action is needed to maintain coherence in the system. A write-back is not necessary for evicted, clean L1 cache blocks. Hence, after sending an acknowledgement (ACK-PUTS), the L2 CTLR waits for an UNBLOCK message to remove the corresponding entry from the sharer list. While the L2 CTLR waits for the incoming UNBLOCK message, new requests for the corresponding clean block may be serviced. A new request for shared access (GETS) to the block is granted, while a request for exclusive



(a) During a local reply

(b) During a block drop

Figure 6.9: Messages to maintain cache coherence

access (GETX) is put on wait. Though multiple copies of the block may exist in the system, all of them are clean, and hence the cache block is coherent. Thus, no special action is needed for coherence when an evicted, clean L1 cache block is kept stored in the local router.

When we attempt a local reply of GETS/GETX message with the stored DATA-PUTS/DATA-PUTO/DATA-PUTM message from the local NoC router, we need to maintain coherence. As shown in Figure 6.9a, when a local reply reaches the L1 CTLR (Ⓚ), a special coherence message CANCEL-PUT is initiated towards the L2 cache bank (Ⓛ). With that CANCEL-PUT message, the L2 CTLR learns that the corresponding UNBLOCK/DATA-PUTO/DATA-PUTM message will not come. Hence, the L2 CTLR rolls back the state of the corresponding cache block as if the eviction never happened. This way, we preserve the state of an evicted L1 cache block to maintain coherence. However, there can be a scenario where a write request (GETX) that requires a cache block with exclusive access is locally replied by a stored block that has shared access (DATA-PUTS). In that case, we make sure that the L1 CTLR take permission from the corresponding L2 CTLR before granting the write request.

When a VC is to be vacated, the dirty block stored in that VC is forwarded for write-back, hence no coherence violation. When an entry in the trace buffer needs to be deleted, the stored clean block is dropped; but the corresponding L2 cache bank needs to be intimated to maintain coherence. Hence, to drop a clean block stored in the trace buffer, we send the block (DATA-PUTS) back to the same L1 cache from where it was evicted (Ⓜ), as shown in Figure 6.9b. After receiving the DATA-PUTS message, L1 CTLR drops the block (Ⓝ) and generates an UNBLOCK message for the L2 cache bank (Ⓞ). With the arrival of the UNBLOCK message, L2 CTLR removes the sharer and completes the block eviction process.

Table 6.1: System configuration

Processor	64 OoO x86 cores
L1 Cache	16KB, 4-way, private, split (instruction and data)
L2 Cache (LLC)	128KB$\times$64 cores, 8-way, shared
Memory Bank	4; one located at each corner
Coherence	MOESI distributed directory
NoC	8$\times$8 2D-Mesh topology, 128-bit channel width 3 Virtual Networks (VNs), VN0, VN1, VN2 2/4/6 Virtual Channels (VCs) per VN 1-flit depth control VC, 5-flit depth data VC
Routing	2-stage routers, XY-DOR algorithm VC based wormhole packet-switching
Packets	1-flit for control packet, 5-flit for data packet
Word/Flit/Block	64-bit/128-bit/64B; 2-words/flit, 8-words/block
Trace Buffer (ETB)	2KB/4KB/8KB per router
Benchmarks	SPEC CPU2006 (multi-programmed) PARSEC 3.0 (multi-threaded)

6.5 Performance Evaluation

We consider the following architectures for evaluation:

- **Baseline:** Without any optimisation.
- **DB-TTBF:** Store evicted, dirty L1 cache blocks in local router VCs and use TTBF.
- **DB-MTBF:** Store evicted, dirty L1 cache blocks in local router VCs and use MTBF.
- **CDB-TTBF:** Store evicted, clean as well as dirty L1 cache blocks in local router VCs. Use TTBF for dirty blocks and drop clean blocks.
- **CDB-MTBF:** Store evicted, clean as well as dirty L1 cache blocks in local router VCs. Use MTBF for dirty blocks and drop clean blocks.
- **CDB-ETB-TTBF:** Store evicted, clean L1 cache blocks in ETB and dirty blocks in local router VCs. Use TTBF for dirty blocks and drop clean blocks.
- **CDB-ETB-MTBF:** Store evicted, clean L1 cache blocks in ETB and dirty blocks in local router VCs. Use MTBF for dirty blocks and drop clean blocks.

TH-2538_156101018

Table 6.2: Workload mixes

Mix	Benchmarks	Copies
same	astar, cactusADM, GemsFDTD, gromacs, h264ref, hmmer, lbm, leslie3d, namd, omnetpp, perlbench, sjeng, soplex, sphinx, xalancbmk	1×64: 64
low	gromacs, GemsFDTD, hmmer, astar, h264ref	4×16: 64
med	sphinx, perlbench, cactusADM, omnetpp, soplex	4×16: 64
high	xalancbmk, namd, sjeng, leslie3d, lbm	4×16: 64
bla	blackscholes; runs with 64 threads	1×64: 64
can	canneal; runs with 64 threads	1×64: 64
ded	dedup; runs with 64 threads	1×64: 64
fer	ferret; runs with 64 threads	1×64: 64
str	streamcluster; runs with 64 threads	1×64: 64
swa	swaptions; runs with 64 threads	1×64: 64

6.5.1 Simulation Framework and Workloads

We model all the architectures on event-driven gem5 simulator [55]. Our system configuration is similar to Intel Xeon Phi Processor 7235 [56] with shared and distributed L2 cache (LLC). Due to certain limitations in gem5, we could not exactly model the cache configuration of Intel Xeon Phi Processor 7235. However, our cache configuration is not chosen to give undue advantage to the proposed optimisations. Rather, it challenges the optimisations with a hit rate of around 90% to 95% for all the benchmarks we evaluate. Our system configuration is presented in Table 5.4 for reference. We modify GARNET [57] module in gem5 to implement the proposed router microarchitecture. We modify MOESI_CMP_directory protocol in Ruby inside gem5 to implement and maintain cache coherence between L1 CTLR and L2 CTLR.

To evaluate and analyse the performance, we consider multi-programmed as well as multi-threaded applications. For multi-programmed workloads, we consider SPEC CPU2006 benchmarks to mimic a modern NoC based TCMP running multiple applications in parallel. We create different workload mixes based on the re-reference time of the benchmarks (refer Figure 6.3), as given in Table 6.2. *same* is a homogeneous workload mix that runs 64 copies of the same benchmark on all the 64 cores (1×64: 64). *low*, *med* and *high* workload mixes are created by grouping benchmarks with low, medium and high re-reference times, respectively. These mixes run a random combination of 4 different benchmarks from their groups with 16 copies each (4×16: 64). By separately profiling each benchmark, we choose a smaller representative window of instructions to have a tractable simulation time. We create a total of 45 workload mixes (15 homogeneous, i.e. *same*, and 10 each for *low*, *med* and *high*).

TH-2538_156101018

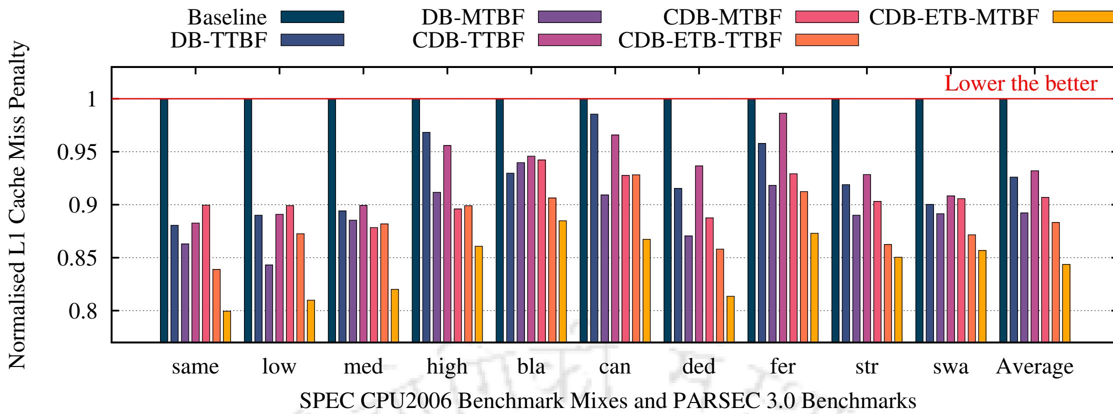


Figure 6.10: L1 cache miss penalty

For multi-threaded workloads, we consider PARSEC 3.0 benchmarks to mimic a modern NoC based TCMP running multiple threads of a single application. We identify a mix of 6 computation-intensive, communication-intensive and memory-intensive benchmarks, as given in Table 6.2. For example, *dedup* benchmark has huge working sets (computation-intensive) whereas the working set for *streamcluster* can be varied. *blackscholes* has negligible communication whereas *ferret* is very communication-intensive. *canneal* has the most demanding memory behaviour (memory-intensive) and so on. These benchmarks are run individually as a 64-thread workload on all the 64 cores of the TCMP (1 thread/core). We consider *sim-medium* input set of PARSEC 3.0 and evaluate the performance on region-of-interest. Altogether, we have 10 workloads to evaluate and analyse the performance, 4 multi-programmed benchmark mix and 6 multi-threaded benchmarks. For a relative comparison, all the results are normalised with respect to the baseline architecture.

6.5.2 Result Analysis and Discussion

6.5.2.1 L1 Cache Miss Penalty

It is defined as the number of cycles required to replace an existing cache block in L1 with an incoming block. L1 cache miss penalty directly reflects the effectiveness of the proposed local store and reply optimisation. Figure 6.10 shows the normalised L1 cache miss penalty with respect to the baseline architecture. With local replies, the proposed architectures reduce the L1 cache miss penalty for all the simulated workload mixes. In general, *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures perform better compared to others. With more blocks (both clean and dirty) locally stored in more space (ETB and VCs), *CDB-ETB-TTBF* and *CDB-ETB-MTBF* has more scope for local reply (hits) on re-reference. *CDB-TTBF* and

TH-2538_156101018

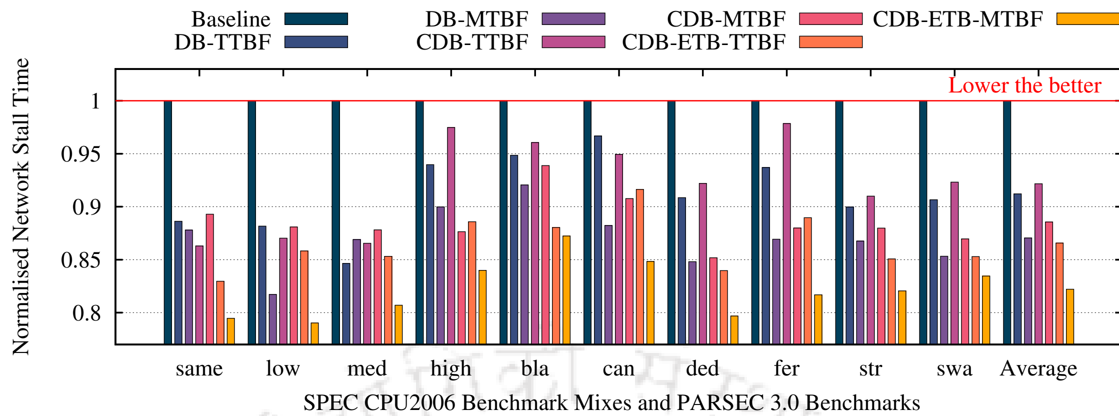


Figure 6.11: Network stall time

CDB-MTBF architectures also store both clean and dirty blocks in local NoC routers, but the storage space is limited to VCs. As a consequence, blocks are frequently moved in and out of the VCs, which reduces the chance of local hits. A maximum reduction of 21% and an average reduction of 16% in L1 cache miss penalty is achieved by our proposed architectures.

Among the multi-programmed workloads, *same* and *low* are outperforming *med* and *high* mixes as they have benchmarks with low re-reference time of evicted L1 cache blocks (refer Figure 6.3). Whereas the miss penalty reduction in multi-threaded workloads is relatively less when compared with the multi-programmed counterparts. It is due to the frequent sharing of data among the participating threads of a particular workload. Keeping evicted L1 cache blocks stored in local routers for long increases the miss penalty of other threads waiting in the corresponding L2 cache bank. In general, TTBF architectures (*DB-TTBF*, *CDB-TTBF* and *CDB-ETB-TTBF*) perform poorly as they keep evicted, dirty blocks stored for a certain time threshold (τ) even if there are other requesters waiting in the L2 cache bank. On the other hand, MTBF architectures (*DB-MTBF*, *CDB-MTBF* and *CDB-ETB-MTBF*) can forward stored blocks as and when a request is received from the L2 cache bank. For example, *can* and *fer* workloads suffer the most while running in the TTBF architectures as they have the most demanding memory and communication behaviour, respectively.

6.5.2.2 Network Stall Time

It is defined as the number of cycles the processor stalls waiting for a network packet. Network stall time helps us to understand how storing evicted L1 cache blocks in local routers impact the overall NoC communication latency. We prefer network stall time over *Packet Latency/Network Latency* as the former is a more appropriate metric to evaluate

network-related slowdown in NoC based TCMPs [22]. Figure 6.11 shows the normalised network stall time with respect to the baseline architecture. As expected, across all the simulated workload mixes, our proposed architectures significantly reduce network stall time. With local reply from the routers, we avoid both on-chip travel and NoC communication delay as given in Equation (6.1). Saving on-chip travel time indirectly translates into reduced network stall time. A maximum of 21% and an average of 18% reduction in network stall time is achieved by our proposed architectures. In TTBF architectures, a stored dirty cache block is forwarded for write-back only after the time threshold (τ) expires. However, with MTBF architectures, a stored dirty block is forwarded either after the time threshold (τ) expires or even earlier if a request from the corresponding L2 cache bank is received. As a result, all the proposed MTBF architectures experience less network stall time in general.

Usually, network stall time reduction is relatively less for communication-intensive workload mixes when compared to others. Since these workload mixes frequently inject packets in the network, evicted cache blocks can not be stored in the routers for long. As a consequence, their chances of local reply reduce. Additionally, frequent store and forward of evicted cache blocks increase the network latency for others. For example, the network stall time reduction in *high* is minimum when compared with other multi-programmed workloads. This is because *high* workload contains *leslie3d* and *lbm* benchmarks which have very high packet injection rate. Similarly, being the most communication-intensive multi-threaded workload, *fer* experience less reduction in network stall time. Interestingly, *bla* experiences the lowest reduction in network stall time even though it has a negligible communication pattern. This is due to the fact that *bla* is not able to get the benefit of local store and reply. Even though the average re-reference time of *bla* is one of the lowest (refer Figure 6.3), the number of re-references are low. So, evicted blocks just stay in the local router for some time and then get forwarded or dropped. To mitigate the negative effect of occupying VCs, Dynamically Allocated Multiple Queue (DAMQ) buffering schemes can be explored [112].

6.5.2.3 System Speedup

We use total Instructions Per Cycle (IPC) to compare the system speedup between baseline and the proposed architectures for multi-programmed workloads (SPEC CPU2006). Whereas, for multi-threaded workloads (PARSEC 3.0), we use execution time to compare the system speedup. We prefer execution time for multi-threaded workloads as they have synchronisation primitives like locks and barriers, which brings variation in the IPC. Figure 6.12 shows the normalised system speedup with respect to the baseline architecture. From the improvements in L1 cache miss penalty and network stall time, the increase in system speedup with the proposed architectures is intuitive. We achieve a maximum system speedup of 19% and

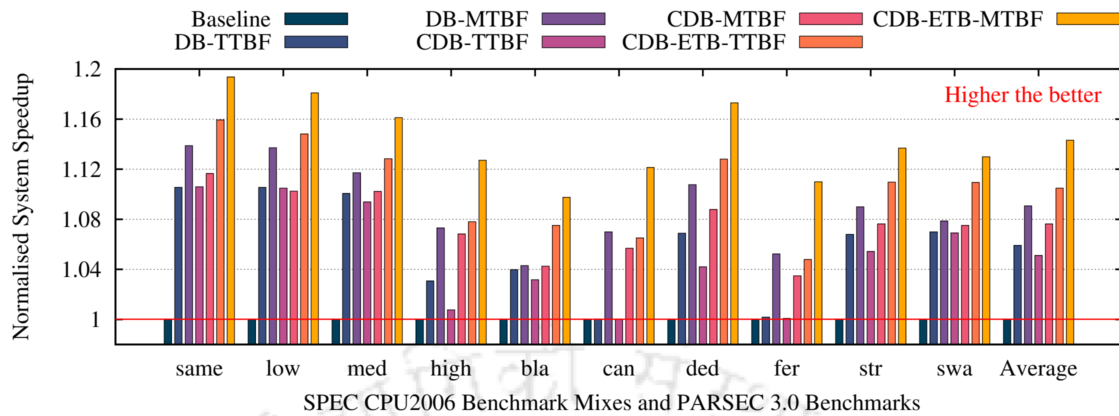


Figure 6.12: System speedup

an average system speedup of 14% for the presented workloads. Usage of trace buffers in the proposed *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures significantly improve the overall system performance with frequent local replies from the local NoC routers.

6.5.3 Qualitative Comparison with An Existing Work

Jindal et al. [113][114] proposed to reuse ETB in the processor as a victim cache [108] to improve system performance. The key idea is to re-purpose the ETB as a set-associative cache called *VCache* to hold recently evicted blocks of L1 data cache. VCache indirectly increases the size of L1 data cache as they are mutually exclusive. A block requested by the processor is simultaneously searched in both the L1 data cache and VCache. If the requested block is not found in L1 data cache but the VCache, it is brought into the L1 data cache by swapping out another block into the VCache. The authors learn that in Simultaneous Multithreading (SMT), competing threads may flush cache blocks of each other from the VCache resulting in poor performance. Thus, they propose two techniques to partition the VCache among the participating threads, which promotes cooperation. These two techniques attempt to increase VCache utilisation and improve performance. The concept of VCache and our proposed TTBF/MTBF architectures are complementary in nature and can be implemented together. However, there are a few important differences between VCache and the TTBF/MTBF:

- VCache does not differentiate between clean and dirty cache blocks and flushes them immediately with the incoming blocks. Whereas TTBF/MTBF segregates clean and dirty cache blocks in such a way that dirty blocks are cached in VCs until a certain time threshold. This optimisation delays/avoids expensive writes to the next level

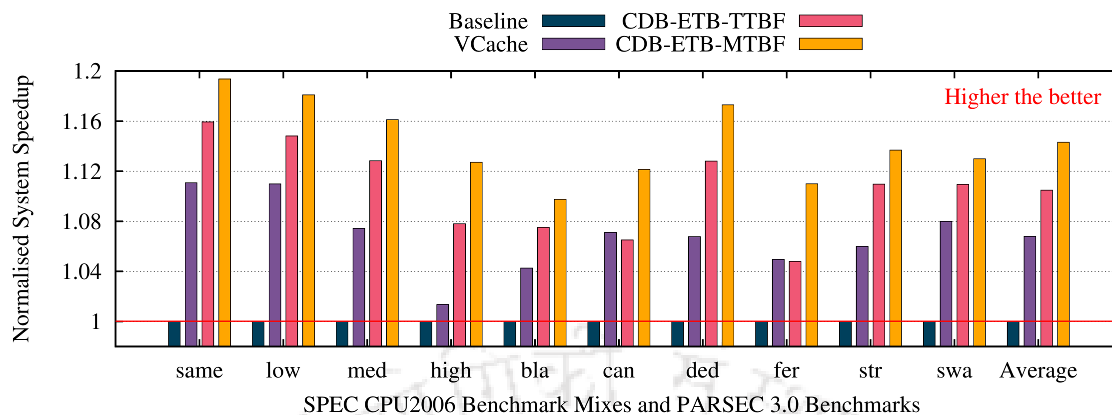


Figure 6.13: Comparison with *VCache* architecture

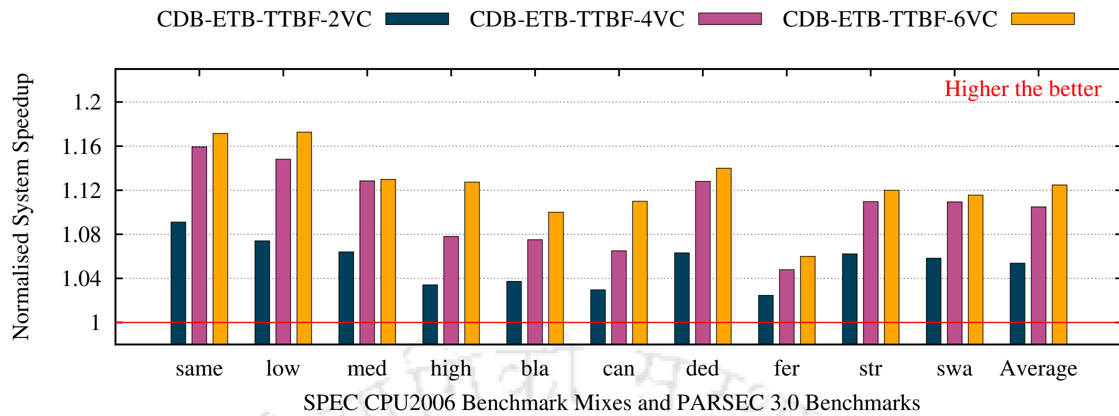
of memory (L2 cache bank). Hence, TTBF/MTBF indirectly uses VCs like a write buffer [115] and ETB like a victim cache to improve overall system performance.

- VCache does not talk about cache coherence. Cache blocks stored in VCache can be either in a shared or exclusive state. Frequent data sharing among different cores (threads) is more prevalent in multi-threaded applications. VCache is evaluated only for multi-programmed applications (SPEC CPU2006), and the discussion about shared memory and the associated coherence is not included. Whereas TTBF/MTBF provides a detailed discussion about how coherence is maintained during local store, reply, forward and drop of evicted L1 cache blocks. TTBF/MTBF also adds a new coherence message to make sure that the states of the evicted L1 cache blocks are preserved.

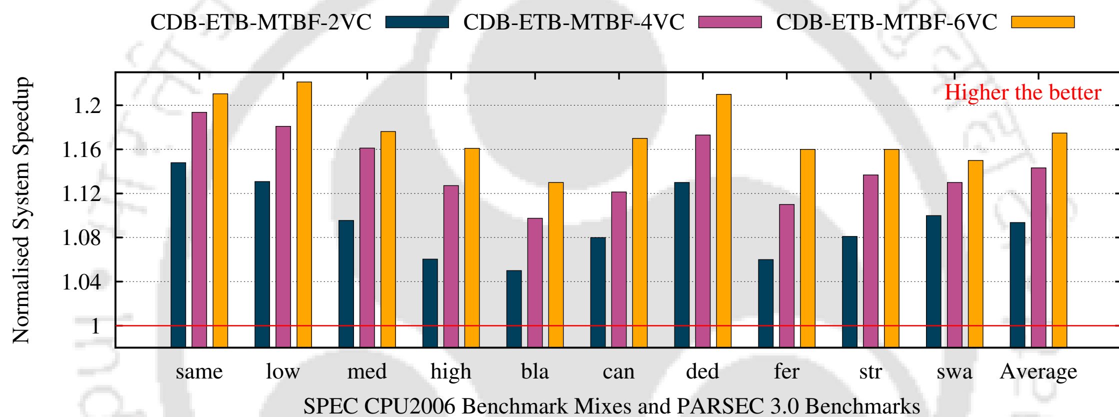
- VCache is set-associative where multiple evicted L1 data cache blocks will map into the same set. This may result in a frequent flush of stored cache blocks from VCache, which hampers the system performance. On the other hand, TTBF/MTBF uses VCs and ETB on NoC routers like a fully-associative cache. Hence, there are less conflicts and blocks can be retained longer, which improves the chances of local replies.

- VCache works as an extension of the L1 data cache, and hence it only stores evicted data blocks. Whereas TTBF/MTBF stores all the evicted blocks, both of data and instruction L1 caches. Additionally, since TTBF/MTBF use VCs as well as ETB of NoC routers to store evicted L1 cache blocks, they can hold more blocks simultaneously.

VCache is originally proposed for LEON3 Processor [116] which can be realised for up to 16 CPU cores. To make a fair comparison with TTBF/MTBF architectures, we faithfully



(a)



(b)

Figure 6.14: Impact of number of VCs

model a 64-core equivalent of the VCache architecture. Figure 6.13 shows the overall system performance comparison of existing VCache and the proposed TTBF/MTBF architectures.

CDB-ETB-TTBF and *CDB-ETB-MTBF* performs better than VCache in almost all the workloads. Simultaneously holding more evicted L1 cache blocks, fewer conflicts during block store, and longer retention of stored blocks are the key factors. However, VCache performs better than *CDB-ETB-TTBF* for *can* and *fer* workloads. With the most demanding memory and communication behaviour, *can* and *fer* suffers in *CDB-ETB-TTBF* that keeps evicted, dirty L1 cache blocks stored for a certain time threshold (τ) even if other requesters are waiting in the L2 cache bank. Compared to VCache, an average of 4% and 8% improvement in system speedup is seen for *CDB-ETB-TTBF* and *CDB-ETB-MTBF*.

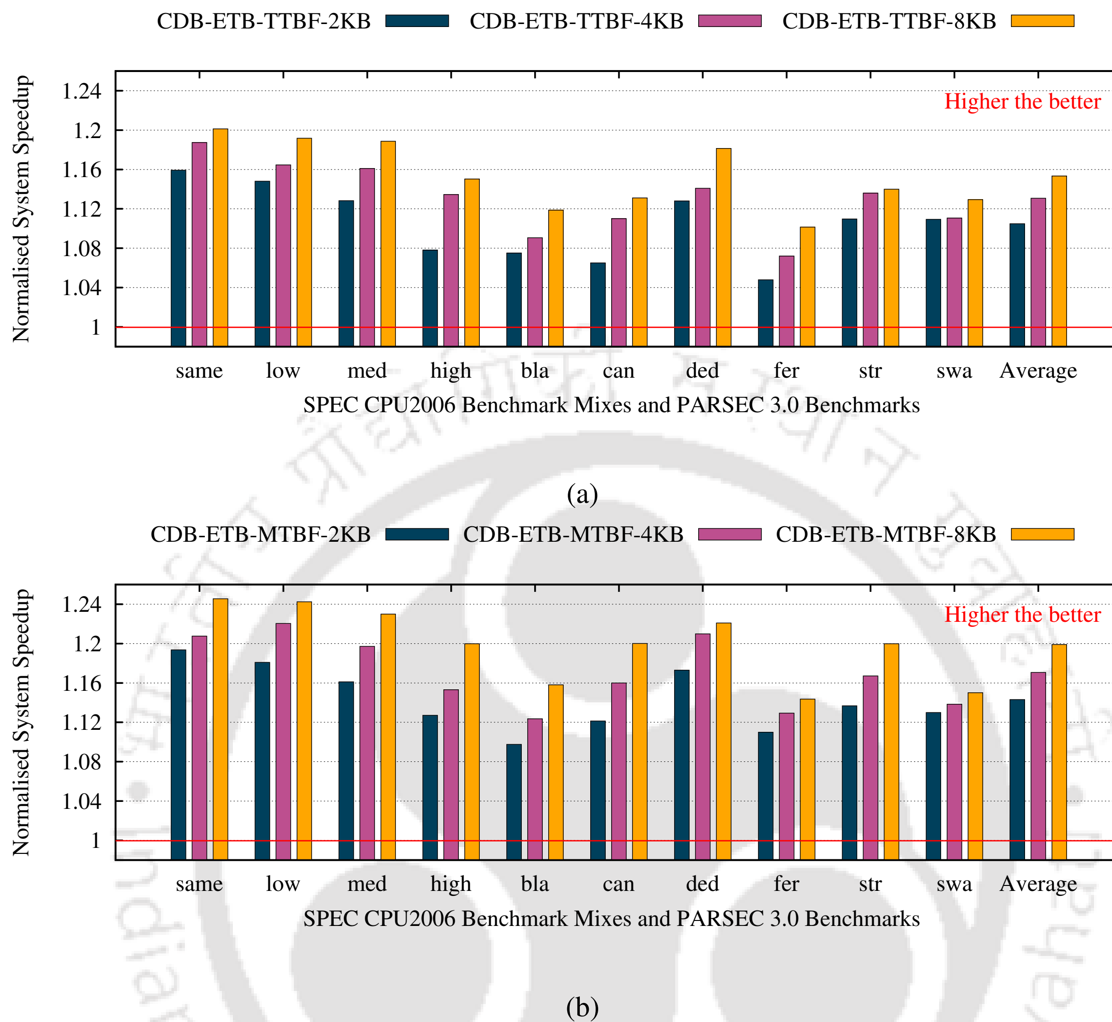


Figure 6.15: Impact of trace buffer size

6.6 Sensitivity and Overhead Analysis

6.6.1 Impact of Number of VCs

Our first optimisation requires evicted, dirty L1 cache blocks to be stored in VCs of the local input port. Hence, we explore the impact of the number of VCs/VN in the proposed architectures. For all the results discussed so far, we have considered 4 VCs/VN (as presented in Table 6.1). However, Figures 6.14a and 6.14b show how varying the number of VCs/VN impact the overall system performance. We have given the results of only *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures as they have the best performance in their respective groups (TTBF and MTBF groups). It is trivial that increasing the number of VCs/VN will improve performance. However, an interesting observation is the performance of architectures

TH-2538_156101018

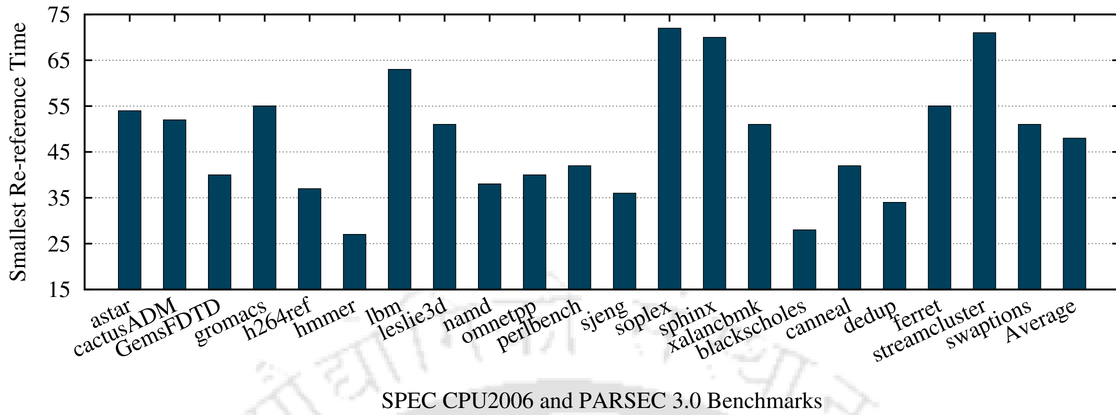


Figure 6.16: Smallest re-reference time

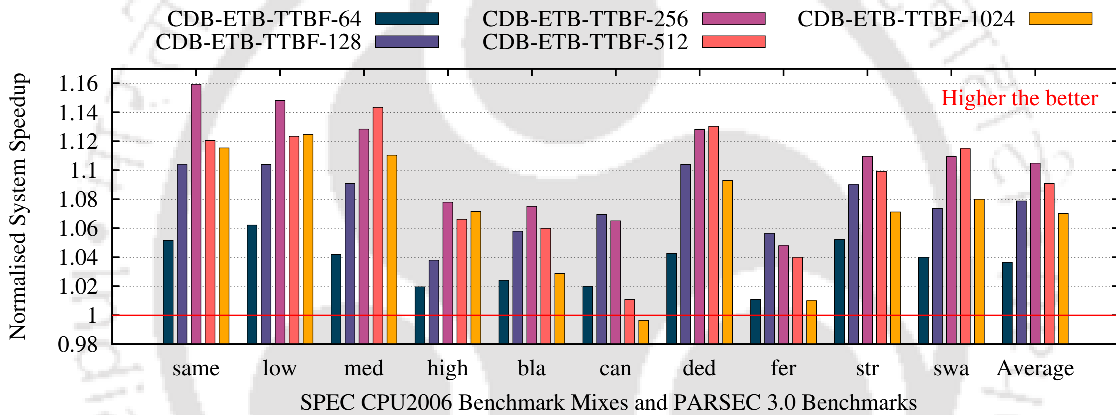


Figure 6.17: Impact of time threshold

with only 2 input port VCs/VN (*CDB-ETB-TTBF-2VC* and *CDB-ETB-MTBF-2VC*). The main reason for a performance gain despite having only 2 VCs/VN in the local input port is the presence of ETB. A good number of re-references are locally replied with the stored clean blocks from ETB that contributes to the overall system performance improvement.

6.6.2 Impact of Trace Buffer Size

Our second optimisation requires evicted, clean L1 cache blocks to be stored in the ETB of local NoC routers. Hence, we explore the impact of ETB size in the proposed architectures for performance. For all the results discussed so far, we have considered size of ETB as 2KB (usually the minimum size). However, Figure 6.15a and 6.15b shows how varying the size of ETB impact the overall system performance. With our optimisation, ETB in the local NoC

routers can be viewed as a cache that holds recently evicted, clean L1 cache blocks. As we increase the size of the ETB, we get appropriate improvement in the system performance.

6.6.3 Impact of Time Threshold

To make sure that the evicted, dirty L1 cache blocks are not stored for too long and penalise others, we proposed TTBF that uses a time threshold (τ). For all the results discussed so far with TTBF architectures (*DB-TTBF*, *CDB-TTBF* and *CDB-ETB-TTBF*), we have considered τ as 256 cycles. Intuitively, the optimal value of τ should be 12150 cycles, equal to the average re-reference time of evicted L1 cache blocks (refer Figure 6.3). However, keeping a block stored for such a long duration delays execution of others who are expecting the block in the corresponding L2 cache bank. This scenario is more prevalent in multi-threaded workloads, where a lot of data sharing happens among the participating cores. Hence, we perform an empirical study to find the optimal value of τ. We use an incremental approach and begin from the smallest re-reference time of evicted L1 cache blocks. Figure 6.16 shows that the smallest re-reference time of all the benchmarks is under 80 with an average of 48 cycles. So, we begin with the value of τ as 64 cycles and incrementally change it to 128, 256, 512 cycles and more. Figure 6.17 shows how varying the value of τ impact overall system performance. Based on the observation, we considered τ as 256 for our evaluation.

However, for all the results with MTBF architectures (*DB-MTBF*, *CDB-MTBF* and *CDB-ETB-MTBF*), we kept τ as 16384 cycles; smallest power of 2 which is large enough for the average re-reference time (12150 cycles). This is not optimal rather an intuitive time threshold to increase our chances of a local reply. Now, an evicted, dirty L1 cache block is forwarded towards destination either after getting a second acknowledgement (refer Section 6.4.4.2) or after 16384 cycles, whichever is earlier. MTBF architectures optimise performance by triggering a block forward based on a message as well as a time threshold.

6.6.4 Storage, Area and Power Overhead

We use 4 additional bits (*Evicted*, *Clean*, *Miss* and *Forward*) in the packet header (refer Figure 6.5) for the working of the LSR-FD unit. Our NoC uses 128-bit flit channel (refer Table 4.2), but a typical packet header (head flit) uses fewer bits ($\approx$ 64 bits) for its operation. So, we can accommodate the additional 4 bits in the head flit without any storage overhead.

Since LSR-FD unit works in parallel to the RC unit (refer Section 6.4.1), it is not in the critical path of execution. In Algorithm 6, lines 1-22, 23-28, 29-34 and 35-42 can execute in parallel to complete the working of LSR-FD unit in time to avoid the critical path. However, the addition of LSR-FD unit in NoC routers contributes to the area and power overhead. As

Table 6.3: Overhead compared to the baseline

Overhead	CDB-ETB-TTBF	CDB-ETB-MTBF
Area	↑ 2.23%	↑ 2.58%
Static (Leakage) Power	↑ 3.71%	↑ 3.94%
Dynamic Power	↓ 5.06%	↓ 6.12%

we have 4 VCs/VN and the local input port requires to have a time threshold counter (τ_i) for each VC, we add binary down counters[2]. Among the 3 VNs (refer Table 6.1), VN2 carries evicted cache blocks. Hence, we add only four 8-bit binary down counters (1 counter/VC for VN2) in TTBF architectures to count from 255 down to 0. Whereas, 14-bit binary down counters are added in MTBF architectures to count from 16383 down to 0. As a result, the addition of these counters also contributes to the area and power overhead. The MUX-DEMUX pair of M and D (refer Figure 6.4) and the connecting links also contribute to a negligible area and power overhead. Embedded trace buffer (ETB) was always present in NoC routers in power-gated mode. So, ETB does not contribute to the area but only to the power overhead. We use McPAT [61] at 22nm processor technology and feed the configuration and output files of gem5 [55] to get the area, static (leakage) and dynamic power overheads. We present the percentage increase/decrease in overhead for *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures compared to the baseline in Table 6.3. While we get negligible area and leakage power overheads due to the additional circuits, dynamic power is reduced due to the significant improvements in overall system performance.

During post-silicon debug and validation, ETB is typically used for functional test. ETB requires to monitor internal signals in real time for functional bugs. Hence, ETB operates at full system clock frequency, and there is no additional delay while using it in our optimisation. However, even though the usage of ETB post production is very rare, but it is possible. In such a scenario, during the time the ETB is used for debug and validation, *CDB-ETB-TTBF* and *CDB-ETB-MTBF* architectures will behave like *CDB-TTBF* and *CDB-MTBF*, respectively.

6.7 Related Work

Existing literature explored different possibilities for efficient utilisation of NoC resources. Since our work is about using NoC as a storage, our discussion is limited to the related works where NoC is projected in that light. One of the first works that attempted to change the abstraction of NoC from communication to storage is by Mizrahi et al. [11]. They advocated

[2] An N-bit binary down counter counts from $2^N - 1$ to 0.

that NoC can be included in the memory hierarchy by placing a small cache in the routers. Going forward in the same line, Eisley et al. [12] decoupled cache from coherence and proposed to keep only the coherence directories in NoC routers. Using the stored directories, they designed a coherence protocol within the NoC routers. Yanamandra et al. [13] combined the advantages of both and proposed to keep frequently used cache blocks along with the coherence directories in NoC routers. There are other significant works that are focused towards NoC aware cache and coherence implementations [117][14][118]. However, almost all of the proposed optimisations employed additional storage in the NoC routers.

A new direction has gained popularity where the unused DfD hardware that were added for post-silicon debug and validation are viewed as a storage. For example, Jindal et al. [113][114] have re-purposed DfD hardware for improving the cache performance by using them as a victim cache. DfD hardware are also used to store critical information for run-time verification and system security [119][120]. Specifically, in the context of NoC, embedded trace buffers in routers are used as extended input port VCs to improve communication [121][103]. However, extending VCs might not be beneficial in input-buffered NoC routers, where the existing VCs are already underutilised [30][31][32].

Sanchez et al. [9] provided a key insight that NoC is responsible for 60% to 75% of the miss latency in NoC based TCMPs. They argued that as NoC based TCMPs continue to scale, considering NoC and memory hierarchy together is the way forward. In a promising new attempt, Das et al. [36][33] recently proposed to exploit underutilised VCs of local NoC routers to store some evicted cache blocks. They attempted to reply future references to such blocks from the local routers and reduce miss penalty. A similar work on NoC based consumer electronics is also available [34]. However, none of these works considered all the evicted L1 cache blocks. In this work, we extend [36] to store evicted, dirty L1 cache blocks in VCs and clean L1 cache blocks in the trace buffer of local NoC routers. Future references to recently evicted L1 cache blocks are replied either from the VCs or the trace buffer.

6.8 Chapter Summary

In this work, we explored opportunities to store recently evicted L1 cache blocks in NoC to reduce cache miss penalty. We identified underutilised input port buffers and unused trace buffers as potential storage space in NoC routers. We proposed multiple architectures to store recently evicted cache blocks in NoC routers and facilitate direct reply when such blocks are re-referenced. We also proposed two techniques to forward stored, dirty cache blocks for write-back and a technique to drop stored, clean cache blocks. To preserve the state of evicted cache blocks and maintain coherence, we also propose an additional coherence

TH-2538_156101018

message. We experimentally validated that the proposed architectures have the potential to reduce cache miss penalty and improve overall system performance. Since the proposed optimisations are on NoC, they can be easily integrated into any optimisation in the memory.

The next chapter concludes the dissertation and discusses future research directions.

TH-2538_156101018

TH-2538_156101018

Chapter 7

Conclusions and Future Work

As TCMPs continue scaling to meet the ever-growing demand of data-driven applications from domains like Artificial Intelligence (AI), Machine Learning (ML), Deep Learning (DL), etc., the role of NoC will continue to be prominent in determining the performance. This dissertation shows that designing data-aware NoC for future TCMPs can improve resource utilisation, reduce memory access latency and improve overall system performance. This chapter summarises our main contributions along with the possible future research directions.

7.1 Dissertation Summary

Figure 7.1 summarises the contributions towards designing data-aware NoC for performance. NoC plays a significant role in memory access latency, yet most of the existing memory access techniques and optimisations are NoC oblivious. In this dissertation, we propose four different techniques along with other optimisations to address the interaction gap. We validate that considering NoC and memory hierarchy together while designing techniques and proposing optimisations for TCMPs is the way forward towards optimal performance.

Being a shared infrastructure, multiple cores (processors) can compete for a given NoC resource at the same time. For example, multiple packets can request the same output port of a particular router. In that case, the arbitration policy employed in the router uses some criteria to select a winner from the competitors. However, different packets travelling through the underlying NoC can have a differential impact on the overall system performance. For example, some packets may be more critical and can not be delayed as they stall execution in the processor, whereas some other packets may tolerate delay as their latency is hidden by the outstanding latency of their predecessors. We believe this scenario will be pervasive in future NoC based TCMPs as they run more and more diverse applications. Hence, our first contribution makes a case for critical packet prioritisation in NoC based TCMPs. We consider

TH-2538_156101018

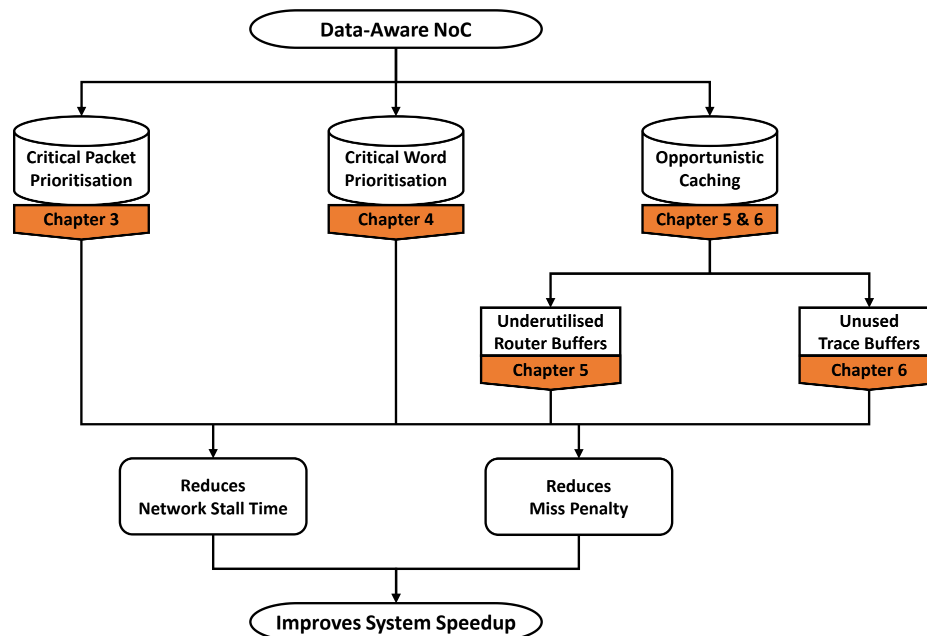


Figure 7.1: Summary of contributions in the dissertation

a metric from the literature called *slack* that represents the relative importance (criticality) of packets. Lower slack packets are more critical than higher slack packets, whereas no-slack packets are most critical. We obtain the slack of travelling packets at run-time by interacting with the on-chip cache controllers and use it as a priority metric during routing and arbitration decisions at NoC routers. We propose a novel technique and few optimisations to prioritise lower slack packets over their higher counterparts and find an alternate minimal path for no-slack packets. The proposed prioritisation policy reduces application-level stall time by up to 22% over traditional *Round-Robin* and 18% over state-of-the-art *Aergia* policies.

A data reply packet carries the requested cache block containing multiple words. However, at any given point in execution, a processor usually requests for a single word called *critical word* from the memory hierarchy. Hence, our next contribution goes one level deeper and proposes critical word based prioritisation in NoC based TCMPs. Since data transfer between different levels of memory is in the unit of blocks, even though the processor requests a single word, an entire data block (containing the critical word) is brought from the next level of memory. Nevertheless, transfer in NoC is discrete and transfer bandwidth is limited to the channel width called *flit*, such that flit $<<$ packet. A data block (packet) is divided into multiple flits and sent in sequence to the requester. The critical word can be in any of the incoming flits which experience unknown router delay along the way. With a first of its kind profiling of outstanding memory requests, we find that the first word is the critical word for most of the requested blocks. We propose an NoC-aware memory access optimisation where

TH-2538_156101018

the flits carrying the critical words are prioritised to reach their destination as soon as possible. The proposed technique involves dynamic cooperation between L1 cache controllers, NoC and the LLC controller to propagate the information about the critical word in a block. It achieves a maximum and an average system speedup of 15% and 9%, respectively.

Data-driven applications with large memory footprints encounter frequent cache misses and suffer from recurring miss penalties when they re-reference recently evicted cache blocks. On the other hand, due to the very low packet injection rate of only around 5% in modern applications, a lot of NoC resources remain underutilised, or even worse, unused. Some of these NoC resources can not be avoided as they provide worst-case performance bandwidth and used for debug and validation. Our third and fourth contributions on opportunistic caching exploit these resources to accommodate evicted cache blocks in NoC routers. Future re-references to the evicted cache blocks can be serviced from the routers without any delay.

Our third contribution proposes to store evicted cache blocks in underutilised VCs without hampering the usual NoC transfer. An evicted cache block can either be *clean* or *dirty*, where the former is discarded while the latter is sent to the next level of memory for write-back. We propose to store the evicted, dirty cache blocks in the local input port VCs on their way for the write-back. When a recently evicted cache block is re-referenced, we arrange a local reply with the matching stored block from the routers. Local reply completely avoids the on-chip travel, which significantly reduces miss penalty and improves overall system performance. We also bring some evicted, clean cache blocks to the routers (which are otherwise discarded) and store them in the local input port VCs to increase our chances of local reply. A maximum of up to 12% and an average of 10% overall system speedup is achieved by this technique.

Input port VCs are underutilised, but they are also limited in numbers; hence making the clean blocks compete with the dirty blocks defeats the purpose of accommodating more evicted cache blocks for local reply from the routers. *Trace buffers* are DfD hardware embedded in NoC routers to record their state for post-silicon debug and validation. However, when a TCMP design goes into production, most of the DfD hardware become non-functional. Since the usage of DfD hardware (trace buffers) is sporadic and rare after production, most of them are left unused. Our fourth and final contribution proposes to store evicted, dirty cache blocks in input port VCs and evicted, clean cache blocks in trace buffers, thereby accommodating all the evicted cache blocks. This technique significantly increases our chances of local reply from the NoC routers with a maximum system speedup of 19%.

TH-2538_156101018

7.2 Future Research Directions

All the contributions made in this dissertation have considered a general-purpose, homogeneous NoC based TCMP. However, the future is moving towards domain-specific, heterogeneous architectures. NoC will be required to smoothly integrate specific Intellectual Property (IP) cores of diverse domains together. One way of looking into improving our packet prioritisation technique will be to get the criticality metric from the IP cores, as they might have specific QoS requirements. Hence, a hardware-software co-design may be explored. Another possible way to extend our technique is by considering different routing algorithms for different VCs within the same input port. This will reduce the conflict between packets.

The number of processing cores is massively increasing in Domain-Specific Architectures (DSAs), and they have frequent memory interactions. If processing cores only need the critical word to continue their execution, NoC channel bandwidth can be divided among competitors to transfer multiple critical words to different cores rather than sending the entire block. This could be one possible way of extending our critical word prioritisation technique for DSAs. Another possible extension could be in the line of NoC traffic compression.

Our work on opportunistic caching can be extended to use the buffers of other input ports as well as buffers of neighbouring routers. Another way of extending this technique is to store pre-fetched blocks to avoid cache pollution. Opportunistic caching may also be explored for Non-Volatile Memories (NVMs) to reduce expensive writes and increase their lifetime. As we advocate that one of the upcoming areas to work on is the design of data-aware NoC, emerging on-chip communication technologies like wireless and photonic NoCs provides even better scope. One of the examples is the design of coherence protocols by taking advantage of the wireless NoC for future TCMPs. Neuromorphic architectures like Spiking Neural Network (SNN) uses NoC for scalability. These architectures exhibit higher memory interactions, where designing data-aware NoC might become a game changer.

TH-2538_156101018

References

[1] G. E. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, pp. 1–4, 1965.

[2] R. H. Dennard, F. H. Gaensslen, H. N. Yu, V. L. Rideout, E. Bassous, and A. R. LeBlanc, "Design of Ion-Implanted MOSFET's with Very Small Physical Dimensions," *IEEE Journal of Solid-State Circuits (JSSC)*, vol. 9, no. 5, pp. 256–268, 1974.

[3] D. W. Wall, "Limits of Instruction-Level Parallelism," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 1991, pp. 176–188.

[4] (2015) International Technology Roadmap for Semiconductors (ITRS). [Online]. Available: https://tinyurl.com/2015-itrs-report

[5] (2017) Intel Xeon Phi Processors. [Online]. Available: https://tinyurl.com/intel-xeon-phi-processors

[6] (2021) AMD EPYC Processors. [Online]. Available: https://tinyurl.com/amd-epyc-processors

[7] (2021) Ampere Altra Processors. [Online]. Available: https://tinyurl.com/ampere-altra-processors

[8] W. J. Dally and B. Towles, "Route Packets, Not Wires: On-Chip Inteconnection Networks," in *Proceedings of the Design Automation Conference (DAC)*, 2001, pp. 684–689.

[9] D. Sanchez, G. Michelogiannakis, and C. Kozyrakis, "An Analysis of On-Chip Interconnection Networks for Large-Scale Chip Multiprocessors," *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 7, no. 1, pp. 1–28, 2010.

[10] H. Farrokhbakht, H. M. Kamali, and S. Hessabi, "SMART: A Scalable Mapping and Routing Technique for Power-Gating in NoC Routers," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2017, pp. 1–8.

[11] H. E. Mizrahi, J. L. Baer, E. D. Lazowska, and J. Zahorjan, "Introducing Memory into the Switch Elements of Multiprocessor Interconnection Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1989, pp. 158–166.

[12] N. Eisley, L. S. Peh, and L. Shang, "In-Network Cache Coherence," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2006, pp. 321–332.

TH-2538_156101018

[13] A. Yanamandra, M. J. Irwin, V. Narayanan, M. Kandemir, and S. H. K. Narayanan, "In-Network Caching for Chip Multiprocessors," in *Proceedings of the International Conference on High-Performance Embedded Architectures and Compilers (HiPEAC)*, 2009, pp. 373–388.

[14] L. Huang, "Leveraging On-Chip Networks for Efficient Prediction on Multicore Coherence," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2014, pp. 1–4.

[15] Y. Yao and Z. Lu, "iNPG: Accelerating Critical Section Access with In-Network Packet Generation for NoC Based Many-Cores," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2018, pp. 15–26.

[16] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Application-Aware Prioritization Mechanisms for On-Chip Networks," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2009, pp. 280–291.

[17] R. Das, O. Mutlu, T. Moscibroda, and C. R. Das, "Aérgia: Exploiting Packet Latency Slack in On-Chip Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2010, pp. 106–116.

[18] R. Das, R. Ausavarungnirun, O. Mutlu, A. Kumar, and M. Azimi, "Application-to-Core Mapping Policies to Reduce Memory System Interference in Multi-Core Systems," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2013, pp. 107–118.

[19] J. S. Miguel and N. E. Jerger, "Data Criticality in Network-on-Chip Design," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2015, pp. 1–8.

[20] Z. Li, J. San Miguel, and N. E. Jerger, "The Runahead Network-on-Chip," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2016, pp. 333–344.

[21] A. Glew, "MLP yes! ILP no!" in *Proceedings of the ASPLOS Wild and Crazy Ideas Session (WACI)*, 1998, p. 1.

[22] O. Mutlu and T. Moscibroda, "Stall-Time Fair Memory Access Scheduling for Chip Multiprocessors," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2007, pp. 146–160.

[23] C. Kim, D. Burger, and S. W. Keckler, "An Adaptive, Non-Uniform Cache Structure for Wire-Delay Dominated On-Chip Caches," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2002, pp. 211–222.

[24] D. Lo, L. Cheng, R. Govindaraju, L. A. Barroso, and C. Kozyrakis, "Towards Energy Proportionality for Large-Scale Latency-Critical Workloads," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014, pp. 301–312.

TH-2538_156101018

[25] L. A. Barroso, J. Clidaras, and U. Hölzle, *The Datacenter as a Computer: An Introduction to the Design of Warehouse-Scale Machines*, 2nd ed. Morgan & Claypool Publishers, 2013.

[26] B. Fields, R. Bodik, and M. D. Hill, "Slack: Maximizing Performance Under Technological Constraints," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2002, pp. 47–58.

[27] A. Das, S. Babu, J. Jose, S. Jose, and M. Palesi, "Critical Packet Prioritisation by Slack-Aware Re-routing in On-Chip Networks," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2018, pp. 1–8.

[28] J. L. Hennessy and D. A. Patterson, *Computer Architecture: A Quantitative Approach*, 6th ed. Elsevier, 2017.

[29] A. Das, J. Jose, and P. Mishra, "Data Criticality in Multi-Threaded Applications: An Insight for Many-Core Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)*, vol. 29, no. 9, pp. 1675–1679, 2021.

[30] N. Barrow-Williams, C. Fensch, and S. Moore, "A Communication Characterisation of Splash-2 and Parsec," in *Proceedings of the International Symposium on Workload Characterization (IISWC)*, 2009, pp. 86–97.

[31] P. Gratz and S. W. Keckler, "Realistic Workload Characterization and Analysis for Networks-on-Chip Design," in *Proceedings of the Workshop on Chip Multiprocessor Memory Systems and Interconnects (CMP-MSI)*, 2010, pp. 1–10.

[32] R. Hesse, J. Nicholls, and N. E. Jerger, "Fine-Grained Bandwidth Adaptivity in Networks-on-Chip using Bidirectional Channels," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2012, pp. 132–141.

[33] A. Das, A. Kumar, and J. Jose, "Reducing Off-Chip Miss Penalty by Exploiting Underutilised On-Chip Router Buffers," in *Proceedings of the International Conference on Computer Design (ICCD)*, 2020, pp. 230–238.

[34] A. Das, A. Kumar, J. Jose, and M. Palesi, "Revising NoC in Future Multi-Core based Consumer Electronics for Performance," *IEEE Consumer Electronics Magazine (CEM)*, 2021.

[35] B. Vermeulen and S. K. Goel, "Design for Debug: Catching Design Errors in Digital Chips," *IEEE Design & Test of Computers (D&T)*, vol. 19, no. 3, pp. 37–45, 2002.

[36] A. Das, A. Kumar, J. Jose, and M. Palesi, "Exploiting On-Chip Routers to Store Dirty Cache Blocks in Tiled Chip Multi-Processors," in *Proceedings of the Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 147–152.

[37] A. Das, A. Kumar, J. Jose, and M. Palesi, "Opportunistic Caching in NoC: Exploring Ways to Reduce Miss Penalty," *IEEE Transactions on Computers (TC)*, vol. 70, no. 6, pp. 892–905, 2021.

[38] J. Duato, S. Yalamanchili, and L. Ni, *Interconnection Networks: An Engineering Approach*, 1st ed. Morgan Kaufmann, 2003.

TH-2538_156101018

[39] W. J. Dally and B. P. Towles, *Principles and Practices of Interconnection Networks*, 1st ed. Elsevier, 2004.

[40] N. E. Jerger, T. Krishna, and L. S. Peh, *On-Chip Networks*, 2nd ed. Morgan & Claypool Publishers, 2017.

[41] C. J. Glass and L. M. Ni, "The Turn Model for Adaptive Routing," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1992, pp. 278–287.

[42] P. Kermani and L. Kleinrock, "Virtual Cut-Through: A New Computer Communication Switching Technique," *Computer Networks (CN)*, vol. 3, no. 4, pp. 267–286, 1979.

[43] W. J. Dally and C. L. Seitz, "The Torus Routing Chip," *Distributed Computing (DC)*, vol. 1, no. 4, pp. 187–196, 1986.

[44] W. J. Dally, "Virtual-Channel Flow Control," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1990, pp. 60–68.

[45] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for A Teraflops Processor," *IEEE Micro*, vol. 27, no. 5, pp. 51–61, 2007.

[46] M. Galles, "Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip," in *Proceedings of the International Symposium on High Performance Interconnects (HOTI)*, 1996, pp. 141–146.

[47] L. S. Peh and W. J. Dally, "A Delay Model and Speculative Architecture for Pipelined Routers," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2001, pp. 255–266.

[48] R. Mullins, A. West, and S. Moore, "Low-Latency Virtual-Channel Routers for On-Chip Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2004, pp. 188–197.

[49] T. Krishna, "Enabling Dedicated Single-Cycle Connections over A Shared Network-on-Chip," Ph.D. dissertation, Massachusetts Institute of Technology (MIT), 2014.

[50] B. Jacob, D. Wang, and S. Ng, *Memory Systems: Cache, DRAM, Disk*, 1st ed. Morgan Kaufmann, 2007.

[51] B. Jacob, *The Memory System: You Can't Avoid It, You Can't Ignore It, You Can't Fake It*, 1st ed. Morgan & Claypool Publishers, 2009.

[52] R. Balasubramonian, N. P. Jouppi, and N. Muralimanohar, *Multi-Core Cache Hierarchies*, 1st ed. Morgan & Claypool Publishers, 2011.

[53] V. Nagarajan, D. J. Sorin, M. D. Hill, and D. A. Wood, *A Primer on Memory Consistency and Cache Coherence*, 2nd ed. Morgan & Claypool Publishers, 2020.

[54] P. Sweazey and A. Smith, "A Class of Compatible Cache Consistency Protocols and their Support by the IEEE Futurebus," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1986, pp. 414–423.

TH-2538_156101018

[55] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 Simulator," *ACM SIGARCH Computer Architecture News (CAN)*, vol. 39, no. 2, pp. 1–7, 2011.

[56] (2017) Intel Xeon Phi Processor 7235. [Online]. Available: https://tinyurl.com/intel-7235-processor

[57] N. Agarwal, T. Krishna, L. S. Peh, and N. K. Jha, "GARNET: A Detailed On-Chip Network Model inside a Full-System Simulator," in *Proceedings of the International Symposium on Performance Analysis of Systems and Software (ISPASS)*, 2009, pp. 33–42.

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

[59] A. B. Kahng, B. Li, L. S. Peh, and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2009, pp. 423–428.

[60] C. Sun, C. H. O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L. S. Peh, and V. Stojanovic, "DSENT - A Tool Connecting Emerging Photonics with Electronics for Opto-Electronic Networks-on-Chip Modeling," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2012, pp. 201–210.

[61] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An Integrated Power, Area, and Timing Modeling Framework for Multicore and Manycore Architectures," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2009, pp. 469–480.

[62] J. L. Henning, "SPEC CPU2006 Benchmark Descriptions," *ACM SIGARCH Computer Architecture News (CAN)*, vol. 34, no. 4, pp. 1–17, 2006.

[63] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC Benchmark Suite: Characterization and Architectural Implications," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2008, pp. 72–81.

[64] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 Programs: Characterization and Methodological Considerations," *ACM SIGARCH Computer Architecture News (CAN)*, vol. 23, no. 2, pp. 24–36, 1995.

[65] N. C. Nachiappan, A. K. Mishra, M. Kademir, A. Sivasubramaniam, O. Mutlu, and C. R. Das, "Application-Aware Prefetch Prioritization in On-Chip Networks," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2012, pp. 441–442.

TH-2538_156101018

[66] L. R. Hsu, S. K. Reinhardt, R. Iyer, and S. Makineni, "Communist, Utilitarian, and Capitalist Cache Policies on CMPs: Caches as a Shared Resource," in *Proceedings of the International Conference on Parallel Architectures and Compilation Techniques (PACT)*, 2006, pp. 13–22.

[67] Y. Li, D. Feng, and Z. Shi, "Heterogeneous-Aware Cache Partitioning: Improving the Fairness of Shared Storage Cache," *Parallel Computing (PC)*, vol. 40, no. 10, pp. 710–721, 2014.

[68] S. W. Keckler, "Rethinking Caches for Throughput Processors: Technical Perspective," *Communications of the ACM (CACM)*, vol. 57, no. 12, pp. 90–90, 2014.

[69] Y. Kim, D. Han, O. Mutlu, and M. Harchol-Balter, "ATLAS: A Scalable and High-Performance Scheduling Algorithm for Multiple Memory Controllers," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2010, pp. 1–12.

[70] T. Pimpalkhute and S. Pasricha, "NoC Scheduling for Improved Application-Aware and Memory-Aware Transfers in Multi-Core Systems," in *Proceedings of the International Conference on VLSI Design (VLSID)*, 2014, pp. 234–239.

[71] L. Subramanian, D. Lee, V. Seshadri, H. Rastogi, and O. Mutlu, "BLISS: Balancing Performance, Fairness and Complexity in Memory Access Scheduling," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 27, no. 10, pp. 3071–3087, 2016.

[72] J. Zhan, N. Stoimenov, J. Ouyang, L. Thiele, V. Narayanan, and Y. Xie, "Optimizing the NoC Slack Through Voltage and Frequency Scaling in Hard Real-Time Embedded Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 33, no. 11, pp. 1632–1643, 2014.

[73] B. Sudev, L. S. Indrusiak, and J. Harbin, "Network-on-chip Packet Prioritisation based on Instantaneous Slack Awareness," in *Proceedings of the International Conference on Industrial Informatics (INDIN)*. IEEE, 2015, pp. 227–232.

[74] C. Nicopoulos, S. Srinivasan, A. Yanamandra, D. Park, V. Narayanan, C. R. Das, and M. J. Irwin, "On the Effects of Process Variation in Network-on-Chip Architectures," *IEEE Transactions on Dependable and Secure Computing (TDSC)*, vol. 7, no. 3, pp. 240–254, 2008.

[75] A. Battersby, *Network Analysis for Planning and Scheduling: Studies in Management*, 1st ed. Macmillan International Higher Education, 1970.

[76] J. Casmira and D. Grunwald, "Dynamic Instruction Scheduling Slack," in *Proceedings of the MICRO Kool Chips Workshop*, 2000, pp. 1–7.

[77] S. Subramaniam, A. Bracy, H. Wang, and G. H. Loh, "Criticality-Based Optimizations for Efficient Load Processing," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2009, pp. 419–430.

TH-2538_156101018

[78] S. Ghose, H. Lee, and J. F. Martínez, “Improving Memory Scheduling via Processor-Side Load Criticality Information,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2013, pp. 84–95.

[79] M. K. Qureshi, D. N. Lynch, O. Mutlu, and Y. N. Patt, “A Case for MLP-Aware Cache Replacement,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2006, pp. 167–178.

[80] M. Moreto, F. J. Cazorla, A. Ramirez, and M. Valero, “Dynamic Cache Partitioning Based on the MLP of Cache Misses,” in *Proceedings of the International Conference on High-Performance Embedded Architectures and Compilers (HiPEAC)*, 2011, pp. 3–23.

[81] Y. Kim, V. Seshadri, D. Lee, J. Liu, and O. Mutlu, “A Case for Exploiting Subarray-Level Parallelism (SALP) in DRAM,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2012, pp. 368–379.

[82] X. Tang, M. Kandemir, P. Yedlapalli, and J. Kotra, “Improving Bank-Level Parallelism for Irregular Applications,” in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2016, pp. 1–12.

[83] J. W. Lee, M. C. Ng, and K. Asanovic, “Globally-Synchronized Frames for Guaranteed Quality-of-Service in On-Chip Networks,” in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2008, pp. 89–100.

[84] B. Li, L. S. Peh, L. Zhao, and R. Iyer, “Dynamic QoS Management for Chip Multiprocessors,” *ACM Transactions on Architecture and Code Optimization (TACO)*, vol. 9, no. 3, pp. 1–29, 2012.

[85] H. Zhu and M. Erez, “Dirigent: Enforcing QoS for Latency-Critical Tasks on Shared Multicore Systems,” in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016, pp. 33–47.

[86] T. Pimpalkhute and S. Pasricha, “An Application-Aware Heterogeneous Prioritization Framework for NoC based Chip Multiprocessors,” in *Proceedings of the International Symposium on Quality Electronic Design (ISQED)*, 2014, pp. 76–83.

[87] E. Bolotin, Z. Guz, I. Cidon, R. Ginosar, and A. Kolodny, “The Power of Priority: NoC based Distributed Cache Coherency,” in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2007, pp. 117–126.

[88] W. Dai, H. An, Q. Li, G. Li, B. Deng, S. Wu, X. Li, and Y. Liu, “A Priority-Aware NoC to Reduce Squashes in Thread Level Speculation for Chip Multiprocessors,” in *Proceedings of the International Symposium on Parallel and Distributed Processing with Applications (ISPA)*, 2011, pp. 87–92.

[89] A. Sodani, R. Gramunt, J. Corbal, H. S. Kim, K. Vinod, S. Chinthamani, S. Hutsell, R. Agarwal, and Y. C. Liu, “Knights Landing: Second-Generation Intel Xeon Phi Product,” *IEEE Micro*, vol. 36, no. 2, pp. 34–46, 2016.

TH-2538_156101018

[90] J. Balkind, M. McKeown, Y. Fu, T. Nguyen, Y. Zhou, A. Lavrov, M. Shahrad, A. Fuchs, S. Payne, X. Liang, M. Matl, and D. Wentzlaff, "OpenPiton: An Open Source Manycore Research Framework," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2016, pp. 217–232.

[91] B. K. Daya, C. H. O. Chen, S. Subramanian, W. C. Kwon, S. Park, T. Krishna, J. Holt, A. P. Chandrakasan, and L. S. Peh, "SCORPIO: A 36-Core Research Chip Demonstrating Snoopy Coherence on a Scalable Mesh NoC with In-Network Ordering," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2014, pp. 25–36.

[92] E. J. Gieske, "Critical Words Cache Memory," Ph.D. dissertation, University of Cincinnati (UC), 2008.

[93] N. Chatterjee, M. Shevgoor, R. Balasubramonian, A. Davis, Z. Fang, R. Illikkal, and R. Iyer, "Leveraging Heterogeneity in DRAM Main Memories to Accelerate Critical Word Access," in *Proceedings of the International Symposium on Microarchitecture (MICRO)*, 2012, pp. 13–24.

[94] J. Huh, C. Kim, H. Shafi, L. Zhang, D. Burger, and S. W. Keckler, "A NUCA Substrate for Flexible CMP Cache Sharing," *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, vol. 18, no. 8, pp. 1028–1040, 2007.

[95] Y. Xue and P. Bogdan, "User Cooperation Network Coding Approach for NoC Performance Improvement," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2015, pp. 1–8.

[96] P. V. Rengasamy and M. Mutyam, "Using Packet Information for Efficient Communication in NoCs," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2014, pp. 143–150.

[97] C. C. Huang and V. Nagarajan, "Increasing Cache Capacity via Critical-Words-Only Cache," in *Proceedings of the International Conference on Computer Design (ICCD)*, 2014, pp. 125–132.

[98] B. P. Lilly, J. M. Kassoff, and H. Chen, "Critical Word Forwarding with Adaptive Prediction," Apr. 29 2014, US Patent 8,713,277.

[99] B. K. Daya, L. S. Peh, and A. P. Chandrakasan, "Quest for High-Performance Bufferless NoCs with Single-Cycle Express Paths and Self-Learning Throttling," in *Proceedings of the Design Automation Conference (DAC)*, 2016, pp. 1–6.

[100] G. Michelogiannakis, D. Sanchez, W. J. Dally, and C. Kozyrakis, "Evaluating Bufferless Flow Control for On-Chip Networks," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2010, pp. 9–16.

[101] W. Dally and C. Seitz, "Deadlock-Free Message Routing in Multiprocessor Interconnection Networks," *IEEE Transactions on Computers (TC)*, vol. 36, no. 5, pp. 547–553, 1987.

TH-2538_156101018

[102] J. Wang, Y. Xue, H. Wang, and D. Wang, "Network Caching for Chip Multiprocessors," in *Proceedings of the International Performance Computing and Communications Conference (IPCCC)*, 2009, pp. 341–348.

[103] N. Jindal, S. Gupta, D. P. Ravipati, P. R. Panda, and S. R. Sarangi, "Enhancing Network-on-Chip Performance by Reusing Trace Buffers," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, vol. 39, no. 4, pp. 922–935, 2019.

[104] T. Moscibroda and O. Mutlu, "A Case for Bufferless Routing in On-Chip Networks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 2009, pp. 196–207.

[105] C. Fallin, G. Nazario, X. Yu, K. Chang, R. Ausavarungnirun, and O. Mutlu, "MinBD: Minimally-Buffered Deflection Routing for Energy-Efficient Interconnect," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2012, pp. 1–10.

[106] K. Sangaiah, M. Lui, R. Kuttappa, B. Taskin, and M. Hempstead, "SnackNoC: Processing in the Communication Layer," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2020, pp. 461–473.

[107] A. Kumar, "Exploiting NoC Buffers as Victim Cache between Last Level Cache and Main Memory," Master's thesis, Indian Institute of Technology Guwahati (IITG), 2020.

[108] N. Jouppi, "Improving Direct-Mapped Cache Performance by the Addition of a Small Fully-Associative Cache and Prefetch Buffers," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*, 1990, pp. 364–373.

[109] B. Vermeulen, "Design-for-Debug to Address Next-Generation SoC Debug Concerns," in *Proceedings of the International Test Conference (ITC)*, 2007, pp. 1–1.

[110] R. Abdel-Khalek and V. Bertacco, "Post-Silicon Platform for the Functional Diagnosis and Debug of Networks-on-Chip," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 13, no. 3s, pp. 1–25, 2014.

[111] J. Duato, I. Johnson, J. Flich, F. Naven, P. Garcia, and T. Nachiondo, "A New Scalable and Cost-Effective Congestion Management Strategy for Lossless Multistage Interconnection Networks," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*, 2005, pp. 108–119.

[112] J. Liu and J. G. Delgado-Frias, "DAMQ Self-Compacting Buffer Schemes for Systems with Network-On-Chip," in *Proceedings of the International Conference on Computer Design (CDES)*, 2005, pp. 97–103.

[113] N. Jindal, P. R. Panda, and S. R. Sarangi, "Reusing Trace Buffers as Victim Caches," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)*, vol. 26, no. 9, pp. 1699–1712, 2018.

TH-2538_156101018

[114] N. Jindal, P. R. Panda, and S. R. Sarangi, "Reusing Trace Buffers to Enhance Cache Performance," in *Proceedings of the Design, Automation and Test in Europe Conference (DATE)*, 2017, pp. 572–577.

[115] M. A. Mills Jr and L. M. Crudele, "Write Buffer," Feb. 14 1989, US Patent 4,805,098.

[116] (2017) LEON3 Processor. [Online]. Available: https://tinyurl.com/leon3-processor

[117] C. Fensch, N. Barrow-Williams, R. D. Mullins, and S. Moore, "Designing A Physical Locality Aware Coherence Protocol for Chip-Multiprocessors," *IEEE Transactions on Computers (TC)*, vol. 62, no. 5, pp. 914–928, 2012.

[118] W. Shu and N. F. Tzeng, "NUDA: Non-Uniform Directory Architecture for Scalable Chip Multiprocessors," *IEEE Transactions on Computers (TC)*, vol. 67, no. 5, pp. 740–747, 2017.

[119] A. Basak, S. Bhunia, and S. Ray, "Exploiting Design-for-Debug for Flexible SoC Security Architecture," in *Proceedings of the Design Automation Conference (DAC)*, 2016, pp. 1–6.

[120] N. Jindal, S. Chandran, P. R. Panda, S. Prasad, A. Mitra, K. Singhal, S. Gupta, and S. Tuli, "Dhoom: Reusing Design-for-Debug Hardware for Online Monitoring," in *Proceedings of the Design Automation Conference (DAC)*, 2019, pp. 1–6.

[121] S. S. Rout, M. Badri, and S. Deb, "Reutilization of Trace Buffers for Performance Enhancement of NoC based MPSoCs," in *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC)*, 2020, pp. 97–102.

Indian Institute of Technology Guwahati

List of Publications

From Dissertation

1. **Abhijit Das**, Sarath Babu, John Jose, Sangeetha Jose, and Maurizio Palesi, "Critical Packet Prioritisation by Slack-Aware Re-routing in On-Chip Networks," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2018, pp. 1-8.

2. **Abhijit Das**, Abhishek Kumar, John Jose, and Maurizio Palesi, "Exploiting On-Chip Routers to Store Dirty Cache Blocks in Tiled Chip Multi-Processors," in *Proceedings of the Annual Symposium on VLSI (ISVLSI)*, 2020, pp. 147-152. [Best Paper Candidate]

3. **Abhijit Das**, Abhishek Kumar, and John Jose, "Reducing Off-Chip Miss Penalty by Exploiting Underutilised On-Chip Router Buffers," in *Proceedings of the International Conference on Computer Design (ICCD)*, 2020, pp. 230-238.

4. **Abhijit Das**, Abhishek Kumar, John Jose, and Maurizio Palesi, "Opportunistic Caching in NoC: Exploring Ways to Reduce Miss Penalty," *IEEE Transactions on Computers (TC)*, vol. 70, no. 6, pp. 892-905, 2021.

5. **Abhijit Das**, John Jose, and Prabhat Mishra, "Data Criticality in Multi-Threaded Applications: An Insight for Many-Core Systems," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems (TVLSI)*, vol. 29, no. 9, pp. 1675-1679, 2021.

6. **Abhijit Das**, Abhishek Kumar, John Jose, and Maurizio Palesi, "Revising NoC in Future Multi-Core based Consumer Electronics for Performance," *IEEE Consumer Electronics Magazine (CEM)*, 2021. [Accepted, in Press]

7. **Abhijit Das**, John Jose, and Prabhat Mishra, "Critical Word Prioritisation in TCMPs: NoC Aware Implementation and Optimisation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD)*, 2021. [Under Submission]

TH-2538_156101018

Outside Dissertation

8. Aswathy N. S., Reshma Raj R. S., **Abhijit Das**, John Jose, and Josna V. R., "Adaptive Packet Throttling Technique for Congestion Management in Mesh NoCs," in *Proceedings of the International Symposium on VLSI Design and Test (VDAT)*, 2017, pp. 337–344.

9. Reshma Raj R. S., **Abhijit Das**, and John Jose, "Implementation and Analysis of Hotspot Mitigation in Mesh NoCs by Cost-Effective Deflection Routing Technique," in *Proceedings of the International Conference on Very Large Scale Integration (VLSI-SoC)*, 2017, pp. 1–6.

10. John Jose, and **Abhijit Das**, "An Adaptive Deflection Router with Dual Injection and Ejection Units for Mesh NoCs," in *Proceedings of the International Conference on VLSI Design (VLSID)*, 2018, pp. 374–379.

11. Manju R., **Abhijit Das**, John Jose, and Prabhat Mishra, "SECTAR: Secure NoC using Trojan Aware Routing," in *Proceedings of the International Symposium on Networks-on-Chip (NOCS)*, 2020, pp. 1–8.

12. Manju R., **Abhijit Das**, John Jose, and Prabhat Mishra, "Trojan-Aware Network-on-Chip Routing," *Network-on-Chip Security and Privacy*, Springer, pp. 277-307, 2021. [Book Chapter]

Indian Institute of Technology Guwahati

TH-2538_156101018

Vita

Abhijit Das is currently a PhD Scholar in the Department of Computer Science and Engineering (CSE) at Indian Institute of Technology (IIT) Guwahati, Assam, India. Prior to that, he received an MTech degree in CSE from National Institute of Technology (NIT) Silchar, Assam, India in 2015, a BTech degree in CSE from North Eastern Regional Institute of Science and Technology (NERIST), Arunachal Pradesh, India in 2013, and a Diploma in Computer Science and Technology from Tripura University, Tripura, India in 2010. An extended abstract of his PhD dissertation received 3rd Best Poster Award at VLSI-SoC PhD Forum 2018, and is accepted at VLSID PhD Forum 2018 and DAC PhD Forum 2021. The first work of his dissertation is invited for a presentation at IRISS (now ARCS), ACM India Annual Event 2019. The second work of his dissertation is a finalist at Qualcomm Innovation Fellowship (QIF) 2019. And, the third work of his dissertation is a best paper candidate at ISVLSI 2020. His primary research interests include data and security aware NoC and memory systems, Deep Neural Network (DNN) accelerators and neuromorphic architectures. He works with the gem5 simulator (open-source), and his tutorials have one of the highest number of views (22K+) on YouTube among all the gem5 simulator based tutorial videos.

TH-2538_156101018

TH-2538_156101018