A Study of Interconnection Network for Many-Core Processors Based on Traffic Analysis

Yuri Nishikawa

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

DOCTOR OF PHILOSOPHY

School of Science for Open and Environmental Systems
Graduate School of Science and Technology
Keio University

March 2011

Preface

Improvements in VLSI technology has enabled integration of an increasing number of logic blocks such as processor cores, caches and I/O modules on a single chip, and multi- to many-core structure has become the mainstream of processor architectures. There are two approaches to effectively exploit such technology improvement. One is to simply increase homogeneous processor cores, which is a typical approach for fabricating commercial general-purpose multi-core processors. Another approach is to integrate many-core processors with non-uniform caches (NUCA), graphic processing units (GPUs) and video image processing units in a single chip to meet multimedia application requirements. Meanwhile, many-core architecture called "tile architecture" is also expected as a new form of a processor. By introducing the ideas of heterogeneous processors and tile architectures, a viable solution of a future processor is a combination of NUCA-based tile architecture and media processing unit such as SIMD accelerators integrated in a single chip.

To connect such increasing logic blocks, various interconnection methodologies have been widely studied, including shared buses and network-on-chips (NoCs). As system size increases, these interconnects play a more significant role in the performance and cost of the processor. One concern of such heterogeneous NUCA tile architecture is an increase of transactions among the tiles with various traffic characteristics. For example, it is expected that burst point-to-point traffic between certain processing core and GPU cores would increase for media processing off-loading, which requires interconnection network with large bandwidth. Contrary, cache coherence protocol signals would increase, whose message size is short. Therefore, current and future on-chip interconnection networks need to deal with many IP cores on a single chip with various traffic characteristics. Because of such characteristic diversities, it is expected that processor in the near future would introduce heterogeneous network that combine multiple NoCs and global buses in order to meet latency-oriented and bandwidth-oriented traffic requirements. For the above aim, this thesis explores interconnection network including NoCs and shared buses for many-core chip multiprocessors and SIMD accelerators in terms of packet routing and network analysis.

As a packet routing exploration, a non-minimal fully- adaptive routing mechanism for single-flit-structured packet transfers called "Semi-deflection" is proposed. An NoC of chip multiprocessors (CMPs) usually forwards a message that can be packed into a single flit, generated by cache coherence transfer. Semi-deflection routing guarantees deadlock-free packet transfer without use of virtual channels by allowing non-blocking transfer between specific pairs of routers. Evaluation results show that a router that supports Semi-deflection routing is almost equal to the hardware amount compared with those for existing deterministic and adaptive routing. As the result of throughput evaluation, Semi-deflection routing provides 3.17 times higher throughput.

As network analysis explorations, end-to-end latency and throughput of a simple shared-bus and a one-dimensional ring structure for SIMD many-core processors is analyzed and modeled in order to study its feasibility with increasing number of processor cores. To study precise and practical network performance characteristics of modern SIMD many-core processors, we analyze Swazzle and ClearConnect, which are a one-dimensional NoC and a global bus for ClearSpeed's CSX600, an SIMD processor consisting of 96 Processing Elements (PEs). Evaluation and analysis results shows that (1) the number of used PEs, (2) the size of transferred data, and (3) data alignment of a shared memory are dominant factors of network performance of CSX600. Based on these observations, we modeled the end-to-end latency and throughput. Using the model, we estimate the best- and the worst-case latencies for traffic patterns taken from several parallel application benchmarks, and confirm that actual latencies of the benchmarks fit in between the best- and the worst-case values.

Finally, we summarize our proposed routing technique for CMPs and network analysis results of SIMD processors. Based on the above, we make a discussion on the structure of interconnection networks for many-core general-purpose tile processors, accelerators, and their combinations.

Acknowledgments

There are a number of people without whom my doctoral study might not have been accomplished, and to whom I am greatly indebted.

Most of all, I would like to express my deepest gratitude to my supervisor, Professor Hideharu Amano. His constant, patient and generous guidance, stimulating suggestions and hopeful encouragements have been my endless source to pursue my work for the six years spending in his laboratory. He has always given me depthful words whenever I had a difficulty in my research life, and reviewed every paper that I have written with constructive comments.

This thesis has not been possible without elaborate, extensive and integral comments by my doctoral committee members, Professor Fumio Teraoka, Associate Professor Hiroshi Shigeno, and Associate Professor Hiroaki Nishi. They spared their valuable year-end and new-year holidays to review my thesis, and pointed out the shortcomings of my work with mighty hearts.

I wish to express my reverence and heartfelt gratitude to Professor Kenichi Miura at National Institute of Informatics. He has shared his great experience to enlarge my knowledge on computer architecture fields, and also provided me with the crucial evaluation environment for this study.

I am greatly indebted to Associate Professor Michihiro Koibuchi at National Institute of Informatics, who has been my primary collaborator, and at the same time, a symbol of my admiration as a researcher. He also assisted me until very last minutes to submit my papers. Definitely without his constant support, this thesis have not been accomplished.

I'd like to convey my very deep gratitude to Assistant Professor Masato Yoshimi at Doshisha University. His extraordinary yet rational ideas were such an incentive for me, and it was him who have opened my door to the doctoral course. It is beyonds words to express my thanks for his passionate support over a prolonged period.

I owe a deep dept of gratitude to Doctor Hiroki Matsutani at Tokyo University. Being a charismatic leading expert of Network-on-Chip, he frankly guided me to his research field with magnetic leadership, and attentively reviewed many of my papers even in the middle of extreme schedules.

I would like to thank all my GCC group members. Akihiro Shitara's devoting attitude toward his research all day and night has been motivating. Toshiaki Kamata would not probably believe how much I have felt secured by working and "tweeting" with him. Masahiro Yamada has been a brilliant spark of the group. Tetsuya Nakahama has taken on many difficulties and soundly worked for solution. Persevering but humorous spirits of Ryota Nishida and Nobuhiro Ohnishi were the

crutches of the group. I am also grateful to all the members of ASAP group. Takaaki Miyajima - a big brother and also skilled in controversy, Kenta Inakagata - a “random seed” presence of the group, Takaaki Yokoyama - tightened our group with his unselfconscious loftiness, Naoko Yamada - earnest and reserved, yet dignified and bright, and Takayuki Akamine - naturally gingered up our group. I thank all h_superusers and all the “hlab” members for offering daily inspirations and so much fun for me. Yuto Hirata has companionably provided me with technical supports and significant knowledge. Masayuki Kimura’s profound comprehension and “true engineering mind” have been thought-provoking. Talks with Nobuaki Ozaki have been spicy pleasures. Last but not least, I am obliged to our secretary Eriko Ninoi for her careful and firm stewardship.

I would like to express my gratitude to my seniors for all their troubles, especially Dr. Tomohiro Ohtsuka (Information and Technology Center, Keio University), Dr. Konosuke Watanabe (Toshiba), Dr. Yasunori Osana (Seikei University), Dr. Yasuki Tanabe (Toshiba), Dr. Yohei Hasegawa (Toshiba), and Dr. Akira Tsuji (NEC). I also would like to thank my argumentative and cheerful juniors, especially Yoshiki Saito (Toshiba), Yu Kojima (Sony), Keisuke Imada (IBM), Hirokazu Morishita (Sony) and Yosuke Sakai (Denso) for sharing their keen insights with me occasionally even after their graduations. It has been exciting to notice the similarities and diversities in the lookouts of many “AALAB” members. The impact when I first met Dr. Hirotaka Osawa in my freshman year is unforgettable, and he was my catalyst to enroll this department. Reflective yet ingenuous disposition and immense expertise presented by Kazutoshi Suito have furnished me with fresh visions. Hiroyuki Chishiro’s nifty and attentive mindset has been my congenial stimulation.

My gratitude extends to Toshiro Sawaki and Yukari Furihata at Information and Technology Center, and Yuko Suga at the office room of this department. They have assisted me with their administrations, and also have provided a comfortable place for me. I am grateful to all my ICS- and CS-root members for sustaining supports.

My research spirits have been kept up by Miho Hatanaka at the Department of Chemistry, an exclusive colleague as a Ph.D. candidate. I genuinely thank Ayumi Takano, who is my alumni in many schools including this university, for perpetually vitalizing me with her enthusiasm.

I am seized with special gratitude to Professor Nobutoshi Yamazaki at the Department of Mechanical Engineering. He is my former professor who has addressed evocative words on every milestone in my university life here at Keio.

My loving thanks are due to my parents Tariko and Mineki, and all my grandparents. My gratitude for their tangible and intangible supports, and deep understanding toward my uncertain Ph.D. life are more than words can say.

Finally, my unbounded thanks to all my friends.

Yuri Nishikawa
Yokohama, Japan
March 2011

Contents

Preface i

Acknowledgments iii

1 Introduction 1

1.1 Background 1

1.2 Objective 2

1.3 Contribution 3

1.4 Thesis Organization 4

2 On-chip interconnection technologies 6

2.1 Design issues of on-chip interconnection architectures 6

2.2 Buses 8

2.2.1 Bus topology 8

2.2.2 Bus communication schemes 9

2.2.3 Overview of actual on-chip buses 11

2.3 NoCs 17

2.3.1 NoC topology 18

2.3.2 Routing 20

2.3.3 Router architecture 28

3 Many-core processor architectures and its network traffic characterizations 33

3.1 Examples of recent CPU architectures 34

3.1.1 Intel's Nehalem Xeon processor 34

3.1.2 General-purpose multi-core processor integrating a GPU 34

3.2 Examples of tile architectures 35

3.2.1 Tilera's TILE64 and TILE-Gx100 36

3.2.2 Intel's 80-Tile NoC 37

3.2.3 Intel's 48-core SCC processor architecture 38

3.2.4 Trips Processor 39

3.2.5 Raw Processor 41

3.2.6 aSOC Architecture 43
3.2.7 Xpipe Architecture 45
3.3 Examples of SIMD many-core accelerators 46
3.3.1 Cell Broadband Engine 47
3.3.2 Larrabee 48
3.3.3 IMAP processors 48
3.4 Network analysis and traffic characterizations of existing many-core processors . . 50
3.4.1 Cache coherency protocol signals and their traffic patterns 50
3.4.2 Application-specific traffic 51
3.4.3 Analysis of interconnection network using an actual processor 53
3.4.4 Analytical traffic modeling and characterization 54
3.4.5 Summary for analytical studies of interconnection network 54
3.5 Chapter summary 55

4 Routing methodology for chip multiprocessors **57**
4.1 Semi-deflection routing 57
4.1.1 Turn model 58
4.1.2 Deadlock removal 58
4.1.3 Router structure 61
4.1.4 Semi-deflection routing mechanism 62
4.2 Evaluation 63
4.2.1 Simulation conditions 63
4.2.2 Number of non-waiting ports 64
4.2.3 Hardware amount 64
4.2.4 Throughput 65
4.2.5 Average hop count 67
4.3 Chapter summary 70

5 Network analysis of a many-core SIMD accelerator **74**
5.1 Many-core SIMD processor with one-dimensional array topology 74
5.1.1 Target architecture of the study 74
5.1.2 ClearSpeed's CSX600 75
5.2 Exploratory evaluation of Swazzle interconnect and ClearConnect bus 78
5.2.1 Evaluation endpoints and environments 79
5.2.2 Swazzle interconnect 79
5.2.3 Network performance of ClearConnect 80
5.3 Analysis and performance modeling of Swazzle interconnect and ClearConnect bus 82
5.3.1 Analysis and modeling policy 85

5.3.2 Analysis of Swazzle interconnect 85
5.3.3 Analysis of ClearConnect 86
5.4 Model verification using parallel benchmarks 89
5.4.1 Overview and implementation of benchmarks 89
5.4.2 Amount of data transfer 91
5.4.3 Estimated and measured latencies 92
5.5 Chapter summary 94

6 Conclusions 97
6.1 Discussion 97
6.1.1 Routing schemes for chip multiprocessors 97
6.1.2 Bus and one-dimensional ring architecture for SIMD processors 98
6.2 Concluding Remarks 100

Abbreviations and Acronyms 102

Bibliography 104

Publications 111

A Livelock arguments of the Chaos router 118

B Specifications of ClearSpeed's CSX600 memory architecture and ClearConnect bus 120
B.1 Memory architecture 120
B.2 ClearConnect Bus (CCB) 122

List of Tables

2.1 Overview of bus features 16

2.2 Channel bisection B_c 19

2.3 Average hop count H_{ave} 20

2.4 Number of routers R 20

2.5 Total unit-length of links L (1-unit = distance between neighboring two cores) . . . 21

2.6 Deadlock-free routing algorithms 24

3.1 Interconnect architectures of representative many-core processors 55

4.1 Evaluation environments 63

4.2 Number of Non-waiting ports 64

4.3 Hardware structure of designed router 64

4.4 Area of a router for 8×8 mesh network. 65

4.5 Average hop count (4x4 mesh) 68

4.6 Average hop count (8x8 mesh) 69

4.7 Average hop count (16x16 mesh) 70

5.1 Specifications of CSX600 78

5.2 Throughput of Swazzle Interconnect 80

5.3 Data size and number of cycles for Swazzle interconnect 85

5.4 Data transfer latency between Mono and Poly memories 87

5.5 Problem size of target benchmarks 91

5.6 MG (Class A): Number of Data Transfer (Mono-to-Poly) 92

5.7 MG (Class A): Number of Data Transfer (Poly-to-Mono) 92

5.8 Himeno Benchmark (Size L): Number of Data Transfer 93

5.9 MG(Class A): Breakdown of Data Transfer (Mono to Poly, Active number of PEs = 96, $M = 512$) 94

List of Figures

1.1 NUCA-based tile processors with SIMD accelerators 2
1.2 Thesis organization 5

2.1 Shared-medium network 7
2.2 Switched-medium network 7
2.3 Decreasing signal transfer distance per clock with advancing process technology . . 7
2.4 Typical bus topologies 8
2.5 AMBA based system architecture 11
2.6 A system based on CoreConnect bus 12
2.7 STBus interconnect 13
2.8 Point-to-Point Connection of VCI 15
2.9 Two VCI connections to realize a bus connection 15
2.10 Network-on-Chip 17
2.11 Two-dimensional layouts of typical topologies 18
2.12 Taxonomy of routing algorithms 21
2.13 The adaptivity property 22
2.14 The different paths property 22
2.15 Taxonomy of routing implementations 23
2.16 Prohibited turn sets of three routing algorithms in Turn-Model 25
2.17 Prohibited turn set in Odd-Even turn model 26
2.18 Deadlocks of packets 27
2.19 Wormhole router architecture 30
2.20 Router pipeline structure (4-cycle) 30
2.21 Router pipeline structure (3-cycle) 31
2.22 Router pipeline structure (2-cycle) 31

3.1 Intel's Nehalem Xeon 7500 processor 34
3.2 Architecture of Intel's Sandy Bridge and AMD's Fusion APU 35
3.3 Tile-structured Many-core Processor 36
3.4 Tirela's 100-Core TILE-Gx100 37

3.5 NoC block diagram and architecture of Intel's 80-tile 38
3.6 Block diagram and tile architecture of Intel's SCC 39
3.7 Trips processor core architecture 40
3.8 Global microarchitecture of Trips processor 41
3.9 Interconnection networks on Raw microprocessor 42
3.10 aSOC architecture 43
3.11 Communication scheduling of aSOC 44
3.12 Output port of Xpipes switch 46
3.13 Network interface of Xpipe architecture 47
3.14 EIB Diagram - the Command Bus 48
3.15 Ring bus of Intel's Larrabee 49
3.16 IMAP-VISION 50
3.17 IMAP-VISION2 51

4.1 Prohibited turns and non-waiting ports 59
4.2 Non-waiting ports for North-last and West-first turn models 60
4.3 Throughput and traffic load without throttle mechanism (8x8 mesh) 66
4.4 Throughput and traffic load without throttle mechanism (16x16 mesh) 66
4.5 Number of used input port and traffic load without throttle mechanism (8x8 mesh) . . 66
4.6 Number of used input port and traffic load without throttle mechanism (16x16 mesh) 66
4.7 Interpretation of throughput evaluation figures 67
4.8 Uniform traffic (4x4 mesh) 71
4.9 Uniform traffic (8x8 mesh) 71
4.10 Uniform traffic (16x16 mesh) 71
4.11 Matrix transpose (4x4 mesh) 71
4.12 Matrix transpose (8x8 mesh) 71
4.13 Matrix transpose (16x16 mesh) 71
4.14 Bit reversal (4x4 mesh) 72
4.15 Bit reversal (8x8 mesh) 72
4.16 Bit reversal (16x16 mesh) 72
4.17 Uniform traffic (versus 4-flit and 8-flit WH) 72
4.18 Matrix transpose (versus 4-flit and 8-flit WH) 72
4.19 Bit reversal (versus 4-flit and 8-flit WH) 72
4.20 Target 8-core CMP architecture for throughput evaluation 73
4.21 NPB CG (Class=W) 73
4.22 NPB MG (Class=W) 73

5.1 Block diagram of one-dimensional SIMD processor 75
5.2 MTAP architecture of CSX600 76

5.3 Structure of PEs (numbers inside arrows indicate the width of data paths) 77
5.4 Total throughput of Swazzle interconnect 80
5.5 Traffic Patterns 81
5.6 Timing of synchronization 81
5.7 Mono-to-Poly throughput 83
5.8 Poly-to-Mono throughput 83
5.9 Mono-to-Poly throughput with misalignment 84
5.10 Poly-to-Mono throughput with misalignment 84
5.11 Measured execution cycle and estimated cycles for Mono-to-Poly transfer 88
5.12 Measured execution cycle and estimated cycles for Poly-to-Mono transfer 89
5.13 Data partitioning and transfer sequence 90
5.14 Data Transfer Time in MG Benchmark Class=A 96
5.15 Data Transfer Time of Himeno Benchmark Size=L 96

6.1 Throughput of ClearConnect with various number of PEs 99

B.1 Memory unit block diagram of CSX600 121

Chapter 1

Introduction

1.1 Background

The International Technology Roadmap for Semiconductors (ITRS) [1] predicts that 32% yearly increase in the number of cores is necessary to keep up with the applications' needs, and assesses that number of cores in a single chip will exceed 100 in 2012. As the CMOS technology improved, the number of intellectual property (IP) cores integrated on a single chip has steadily increased.

There are two approaches to effectively exploit such CMOS technology improvement. One is to simply increase the number of homogeneous processor cores, which is a typical approach for fabricating commercial general-purpose multi-core chips. Another approach is to integrate graphic processing units (GPUs) and video image processing units on a single die to meet multimedia application requirements.

Reflecting the above two approaches, indeed there are two trends in state-of-the-art multi- to many-core processors. AMD's commercial general-purpose processor Opteron 6000 series have 12 homogeneous cores, and Intel empirically fabricates 80-Tile processor with 80 homogeneous computation "tiles" connected in a form of grid, or 2-D mesh. The same approach follows for SIMD-based processors such as NVIDIA's Tesla C1060 GPU and GRAPE-DR, both with 512 cores [2, 3]. In addition, there are several ongoing researches on tile architectures, which is expected as a new form of chip multiprocessors by enabling integration of many processing elements (tiles) with heterogeneous features such as computation cores, caches, registers, and dedicated multimedia processing units to be arranged in a grid form. Contrary, Intel's Sandy Bridge [4] and AMD's Fusion APU [5], both of which came on to the market in January 2011, integrate multiple processor cores and a graphic core in a single chip. STI's Cell Broadband Engine (Cell/B.E.) is a popular heterogeneous multi-core processor with PowerPC-based core and eight computation elements suitable for media processing.

It is likely that this architectural trend is going to continue; that is a combination of a processor and media processing unit such as SIMD accelerators to be integrated on a single chip. Since wiring delay with process technology improvement has prolonged, especially NUCA-based tile ar-

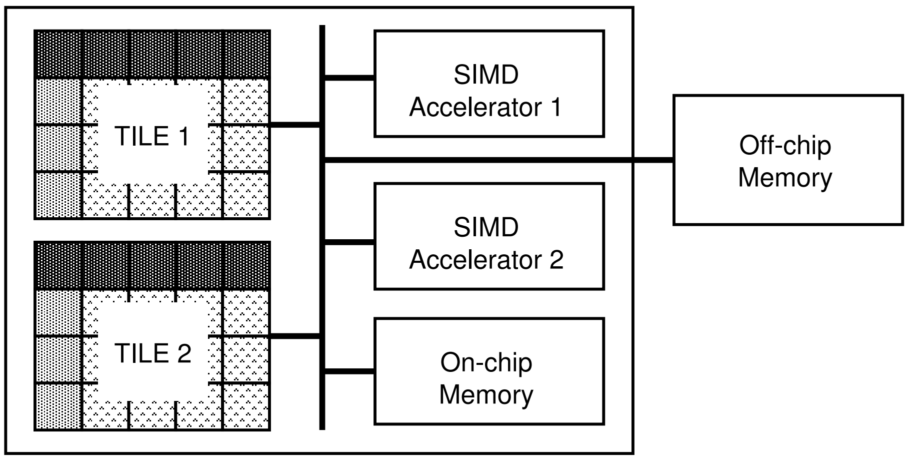


Figure 1.1: NUCA-based tile processors with SIMD accelerators

chitectures are expected as solutions to achieve scalability with increasing number of processor cores and caches. Also, they are advantageous in terms of re-utilizing existing IPs. On the other hand, multimedia processing also have new computation requirements such as real-time processing and encryption/decryption for higher security. Thus, heterogeneous NUCA-based tile architecture with dedicated SIMD accelerator is regarded as a viable solution as future many-core processor, as shown in Figure 1.1.

One concern of such processor architecture is an increase of transactions among the processor with various traffic characteristics. For example, it is likely that cache coherence protocol signals would increase, whose message sizes are short. At the same time, burst point-to-point traffic between certain processing core and GPU cores would also increase for media processing off-loading, which requires interconnection network with high bandwidth. Therefore, current and future on-chip interconnection networks need to deal with many IP cores on a single chip with various traffic characteristics. Because of such characteristic diversities, it is expected that processor in the near future would introduce heterogeneous network that combine multiple NoCs and global buses in order to meet latency-oriented and bandwidth-oriented traffic requirements.

1.2 Objective

The objective of this thesis is to study interconnection network architectures and routing mechanisms suitable for tile-based many-core processors and accelerators. The study is based on the viewpoint that future many-core architectures would integrate NUCA-based tile processors and media processing unit such as SIMD accelerators integrated in a single chip.

As mentioned in the previous section, study towards the practicality of tile architectures is one of the prime tasks in many-core designs. As tile architectures equip on-chip cache tiles, major data transfer in existing chip multiprocessor (CMP) on-chip network are signal transfer for cache coherence protocols, whose message size is very small. Thus, the majority of messages fits into a single-flit packet. This characteristic well suits for deadlock-free adaptive routing [6–8], which uses a flexible routing algorithm. However, most current NoCs apply deterministic routing and applies wormhole switching. Deterministic routing takes a single path between hosts, and it guarantees in-order packet delivery between the same pair of hosts [9]. Contrary, adaptive routing dynamically selects the route of a packet in order to make the best use of bandwidth in interconnection networks. In adaptive routing, when a packet encounters a congested path, different path can be selected. Since this allows better network load distribution, adaptive routing can improve throughput and latency especially for local traffic. Moreover, wormhole switching requires a packet structure with header flit succeeded by body flits, so it requires at least two flits. However, since signals for cache coherence protocols are very small and fit into a single-flit, wormhole routing is costly in terms of router area, and inefficient for delivering short messages. Thus, in this work, we propose a non-minimal fully-adaptive routing for single-flit packets which utilizes customized single-cycle pipelined routers.

On the other hand, there are studies on traffic patterns for accelerators, and several typical interconnection network for them have already been fixed, including graphic processors. According to existing studies on communication requirements for existing massively-parallel computations, most point-to-point communications are stable and sparse, and only use a fraction of available communication paths through a fully-connected network switch. Thus, for applications that do not exhibit special communication pattern, a lower-degree interconnect topology provide better efficiency and space and power. For these reasons, SIMD accelerators often employ simple linear bus architectures or network topology dedicated for collective communications such as scatter and gather operations. As one concern for such interconnection networks for accelerators, they face increasing delay of data transfers, and also a difficulty in making the best use of total link bandwidth. Although there are many researches on NoC architectures such as topology or NoC prototypes [10–13], only a few work investigates performance factors of an end-to-end communication of NoCs using actual SIMD processors. Unlike MIMD multi-core processors including SPMD (Single Program, Multiple Data) whose program threads run on each PE independently, an SIMD processor synchronously executes instructions among all PEs. Thus, communication latency between components such as PEs and shared memories can drastically affect execution time of programs. Thus, estimating the communication overhead is very important to make the best use of SIMD processors.

1.3 Contribution

In this thesis, we study interconnection network architectures, routing mechanism and network performance for tile-based many-core processors and accelerators. To provide interconnection network for such architectures, one concern is an increase of transactions among the tiles with various traffic characteristics. Especially, efficient routing mechanism is essential for delivering increasing cache coherence protocol signals, whose message size is short, however, current many-tile architectures adopt deterministic routing mechanism with low flexibility, as addressed in Chapter 2. On the other hand, media processing deal with large data packets, and many stable interconnection techniques have been established. Based on the observations of traffic characterizations, in this study, we focus on flexible packet routing mechanism for efficiently delivering short messages. On the other hand, we focus on network analysis and network modeling for many-core SIMD processors by running parallel benchmark on an actual processor, and verify the model in order to contribute to the study of designing efficient interconnection design for media processing.

1) As packet routing explorations, we introduce a non-minimal fully-adaptive routing called Semi-deflection routing which does not require virtual channels in virtual-cut-through networks. Semi-deflection routing allows non-blocking transfer among certain routers in order to guarantee deadlock and livelock freedom. Since non-blocking transfer means that a packet that arrived in a router must secure an output port in minimum transfer time, it does not necessarily choose the shortest path to the destination. This is a similar approach with deflection (hot potato) routing [14, 15] and chaos router [16], which are routing methodologies that forces each incoming packet to select an output port every cycle in a non-blocking manner, so as to avoid packet blocking. Although packets may take non-minimal paths, the studies already found that the proposed routing mechanisms were deadlock- and livelock-free. Unlike these existing work, in our proposed Semi-deflection routing, some packets can wait for the ports to their destinations when they are occupied by other packets (Chapter 4).

2) As network analysis explorations, we study network performance of one-dimensional many-core SIMD processors in order to understand the behavior of NoCs in an existing modern many-core SIMD system. Performance characterization of interconnection network, such as impact of the packet contention, hops, and transmission overhead, is rarely illustrated using a real chips. Consequently, based on throughput evaluation of an actual processor, profiling of communication latency, and observation of data packets, we will introduce an analytical model for estimating communication latency. Using the model, it can also estimate communication latency and network throughput with various number of PEs and various data size. To concrete the evaluation, we studied performance of NoCs called ClearConnect and Swazzle of ClearSpeed's CSX600 [17], which is a typical SIMD processor that consists of 96 Processing Elements (PEs) connected in a form of a one-dimensional array topology (Chapter 5).

Finally, we summarize our proposed routing technique for CMPs and network analysis results of SIMD processors. Based on the above, we make a discussion on the structure of interconnection networks for many-core general-purpose tile processors, accelerators, and their combinations.

1.4 Thesis Organization

The thesis is organized as follows, and the thesis organization is illustrated in Figure 1.2. Chapter 2 explains current interconnection network technologies as well as their examples. Chapter 3 surveys

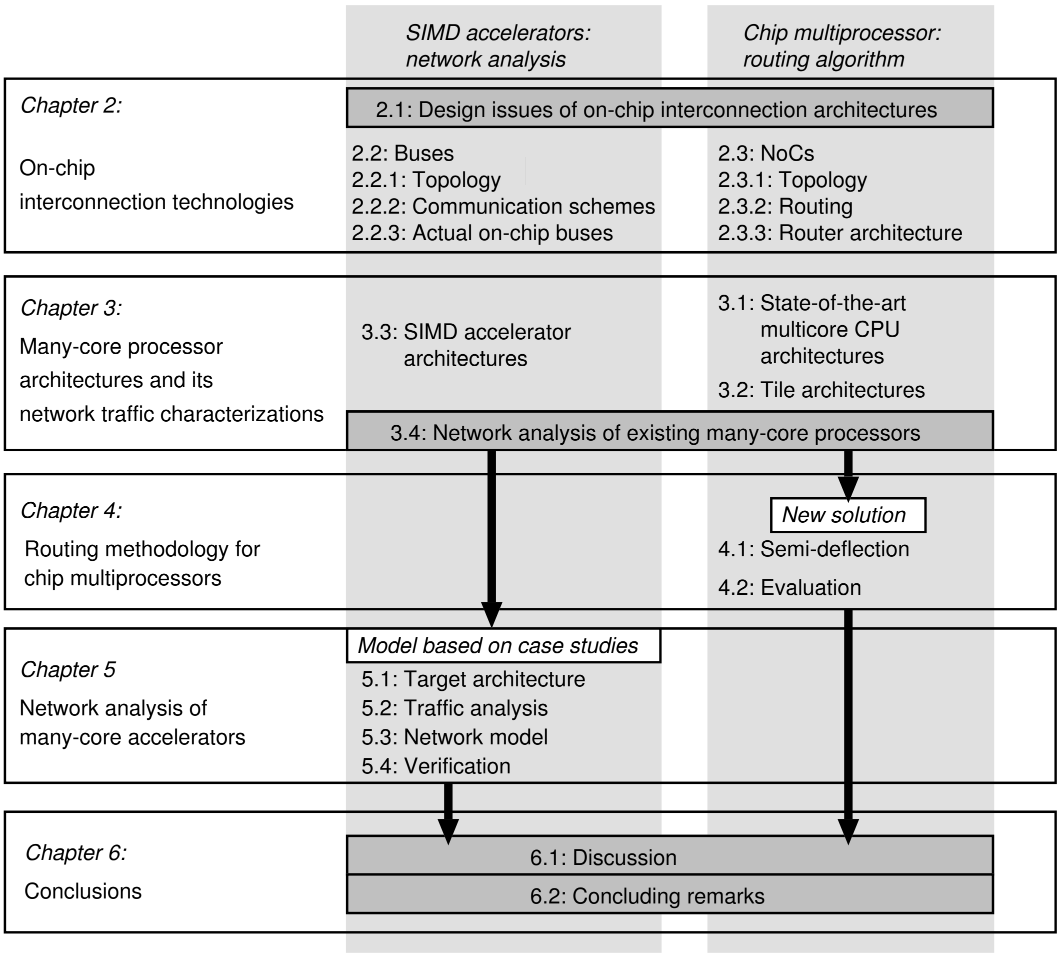

SIMD accelerators: network analysis
Chip multiprocessor: routing algorithm
Chapter 2:
On-chip interconnection technologies
2.1: Design issues of on-chip interconnection architectures
2.2: Buses
2.2.1: Topology
2.2.2: Communication schemes
2.2.3: Actual on-chip buses
2.3: NoCs
2.3.1: Topology
2.3.2: Routing
2.3.3: Router architecture
Chapter 3:
Many-core processor architectures and its network traffic characterizations
3.3: SIMD accelerator architectures
3.1: State-of-the-art multicore CPU architectures
3.2: Tile architectures
3.4: Network analysis of existing many-core processors
Chapter 4:
Routing methodology for chip multiprocessors
New solution
4.1: Semi-deflection
4.2: Evaluation
Chapter 5
Network analysis of many-core accelerators
Model based on case studies
5.1: Target architecture
5.2: Traffic analysis
5.3: Network model
5.4: Verification
Chapter 6:
Conclusions
6.1: Discussion
6.2: Concluding remarks

Figure 1.2: Thesis organization

interconnection architectural examples of recent versatile processors, tile architectures and SIMD accelerators, and also studies their network traffic characteristics. Chapter 4 proposes a non-minimal, fully-adaptive packet routing methodology for MIMD CMPs called Semi-deflection routing. Chapter 5 studies network performance of one-dimensional many-core SIMD processors which can also communicate with an external shared memory. This study models end-to-end latency and throughput of their shared-bus and NoC structures. Chapter 6 summarizes and discusses the above studies, and then concludes this thesis.

Chapter 2

On-chip interconnection technologies

On-chip interconnection networks are implemented either by using buses or NoCs. This section explains design issues and research topics for them. First, we show the classification of the design issues in Section 2.1. In Sections 2.2 and 2.3, we explain each research topic regarding to on-chip bus architectures and NoCs.

2.1 Design issues of on-chip interconnection architectures

Interconnection networks can be classified into shared-medium networks or switched-medium networks [18]. A shared-medium network transfers data on a network medium (i.e., link) shared by all connected nodes, as shown in Figure 2.1. Shared bus network falls into this category. On the other hand, a switch-medium network consists of switch fabrics (routers) and point-to-point links. A router has several input ports and output ports, and it dynamically establishes a connection between a set of an input port and an output port. A switch-medium network can be formed by connecting routers using point-to-point links, as shown in Figure 2.2.

A typical example of on-chip shared-medium network is an on-chip bus that connects IP cores via a shared wire link on a single chip. On-chip buses have been widely used as traditional on-chip interconnects [19, 20]. Benefits of buses are efficiency in transferring short packets such as signals, and various techniques to improve the performance of on-chip buses [21–23] have been studied. Contrary, buses are considered to be inadequate to handle large communication bandwidth demands when they connect a large number of IP cores, and also inefficient in terms of area. In addition, increase of wiring delay with technology advancement as shown in Figure 2.3 is also a quite severe defect to maintain scalability of buses.

To enhance bandwidth bottleneck and area inefficiency which on-chip bus structures create, on-chip switch-medium networks called Networks-on-Chips (NoCs) have been studied as on-chip interconnects for the next generation [13, 24–26]. NoC architectures are similar to those used in parallel computers and System Area Networks (SANs) [27, 28]. In these networks, source nodes (i.e., cores) generate packets, on-chip routers transfer them through connected links, and destination nodes de-

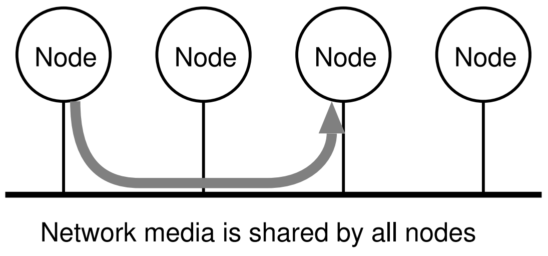


Figure 2.1: Shared-medium network

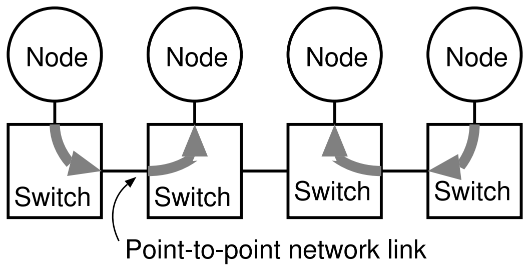


Figure 2.2: Switched-medium network

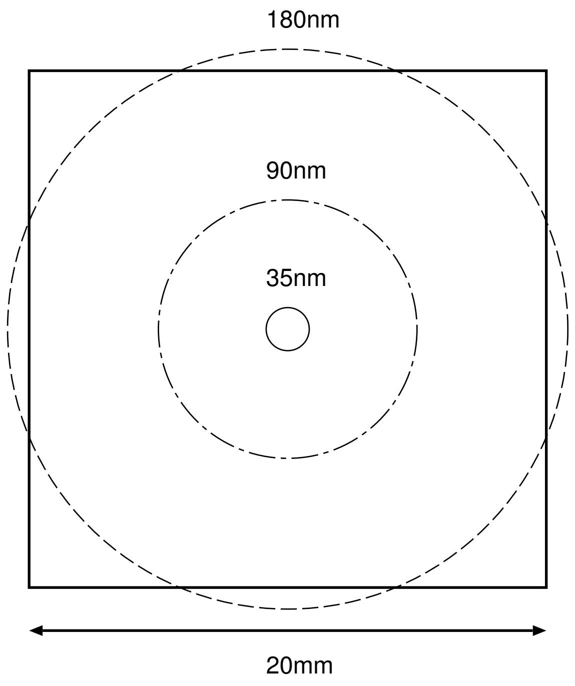


Figure 2.3: Decreasing signal transfer distance per clock with advancing process technology

compose them. Since different packets can be simultaneously transferred on multiple links, bandwidth of NoCs is much larger than that of buses. In addition, the wire-delay problem is resolved, since each flit of a packet is transferred on limited-length point-to-point links, and buffered in every router along the routing path.

The design of on-chip interconnection addresses the following three issues [29]:

- **Network topology** - defines the physical structure of the interconnection network. Number of topologies exist for both buses and NoCs. In case of buses, the applicable topologies varies from single shared bus to complex architectures such as hierarchical bus, crossbar or token rings. Topology variations are even wider for NoCs.

- **Communication scheme** - specifies the exact manner in which communication transaction occurs. These schemes include arbitration mechanism (such as round robin access, priority-

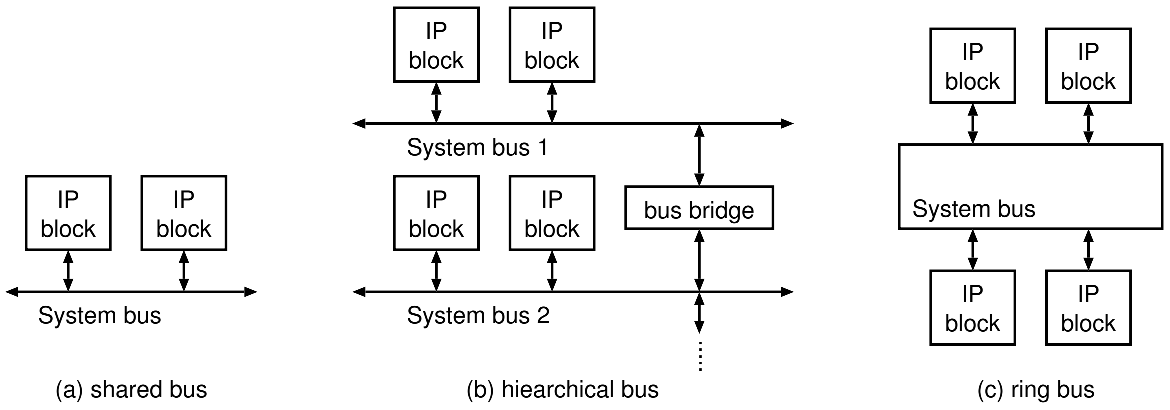


Figure 2.4: Typical bus topologies

based selection). Especially for NoCs, routing mechanism also must be specified.

- **Communication mapping** - refers to the process of associating abstract system-level communication with physical communication paths of buses and on-chip router architectures.

The rest of the chapter introduces each study topics of buses and NoCs based on the above topic categorization.

2.2 Buses

As mentioned previously, research topics related to on-chip bus architecture can be classified into definition of bus topology, selection of communication scheme, and its mapping onto actual hardware. Subsections 2.2.1, 2.2.2, and 2.2.3 survey each topic respectively.

2.2.1 Bus topology

Shared bus The system bus (Figure 2.4 (a)) is the simplest example of a shared communication architecture topology, and is commonly found in many commercial SoCs. Several components can be connected. A *bus arbiter* periodically examines requests from multiple interfaces, and grants access to one of them using arbitration mechanism explained in the next subsection. Increased load on a global bus line limits the bus bandwidth. The advantages of shared bus architecture include simple topology, extensibility, low area cost, simplicity in hardware mapping and implementation. The disadvantages of shared bus are larger load per data bus line, long delay for data transfer, large energy consumption, and low bandwidth. The above advantages with with the exception of bandwidth can be coped by using a low-voltage swing signaling technique.

Hierarchical bus This architecture consists of several shared buses interconnected by bridges to form a hierarchy, as shown in Figure 2.4 (b). Components of buses are placed at certain level in the hierarchy according to the performance they require; components with low performance requirements are placed on lower performance buses, and bridged to higher performance buses not to burden higher performing components. Commercial examples of such architectures include AMBA bus [30], which is described in Subsection 2.2.3. Transactions across the bridge involve additional overhead, and during the transfer both buses remain inaccessible to other components. Hierarchical buses offer large throughput improvements over the shared buses due to:

- decreased load per bus,
- potential for transactions to proceed in parallel on different buses, and
- multiple ward communications can be preceded across the bridge in a pipelined manner.

Ring Idea of a ring bus is shown in Figure 2.4 (c). Each node component communicates using a ring interface, and the bus usually controlled by a token-ring protocol. In numerous applications, ring based application are used, such as network processors, and ATM switches [31,32].

2.2.2 Bus communication schemes

Communication protocols deal with different types of resource management algorithms used for determining access right to shared communication channels. From this point of view, in the rest of this subsection, we will give an introduction about the main feature of the existing communication protocols.

2.2.2.1 Arbitration mechanisms

Static-priority Static-priority employs an arbitration technique of the bus. It is used in shared-bus communication architectures. A centralized arbiter examines accumulated requests from each component connected on the bus, and grants access to the requesting component that has the highest priority. Transactions may be of non-preemptive or preemptive type. AMBA and CoreConnect buses use this protocol [30,31].

Time Division Multiple Access (TDMA) TDMA is the arbitration mechanism based on a timing wheel with each slot statically reserved for unique component of the bus. Special techniques are used to alleviate the problem of wasted slots.

Lottery A centralized lottery manager accumulates request for ownership of shared communication resources from one or more component, each of which has, statically or dynamically, assigned a number of “lottery tickets”.

Token passing Token passing protocol is used in ring based architectures. A special data word called “token” circulates on the ring, and an interface that receives the token is allowed to initiate a transaction. When the transaction completes, the interface releases the token and sends it to the neighboring interface.

Code Division Multiple Access (CDMA) CDMA protocol has been proposed for sharing on-chip communication channel. In a sharing medium, it provides better resilience to noise/interference and has an ability to support simultaneous transfer of data streams. But this protocol requires implementation of complex special direct sequence spread spectrum coding schemes, and energy/battery inefficient systems such as pseudo-random code generators, modulation and demodulation circuits at the component bus interfaces, and differential signaling [33].

2.2.2.2 Bus organization and efficiency

Here, we point out several interconnect issues. These topics have direct impact on bus organization, mapping and its efficiency.

Latency Latency of the bus is caused by the following two factors: a) the access time to the bus, which is the time until the bus is granted; and b) the latency introduced by the bus to transfer data.

Data format Data format is defined by separate wire groups for the transaction type, address, write data, read data, and return acknowledgments or errors.

Programming model Programming model which determines load and store operations are implemented as a sequence of primitive bus transactions. Modules issuing requests are called “masters”, and those serving requests are called “slaves”.

Split bus and non-split buses If there is a single arbitration for a request-response pair, the bus is called “non-split”. In this case, the bus remains allocated to the master of the transaction until the response is delivered. Alternatively, in a “split” bus, the bus is released after the request to allow transaction from different masters to be initiated.

Transaction ordering Usually, all transactions on a bus are ordered. However, on a split bus, a total ordering of transactions on a single master may cause performance degradation. This situation is typical when slaves respond to different speed. To solve this problem, recent extensions to bus protocols allow transactions to be performed on connection.

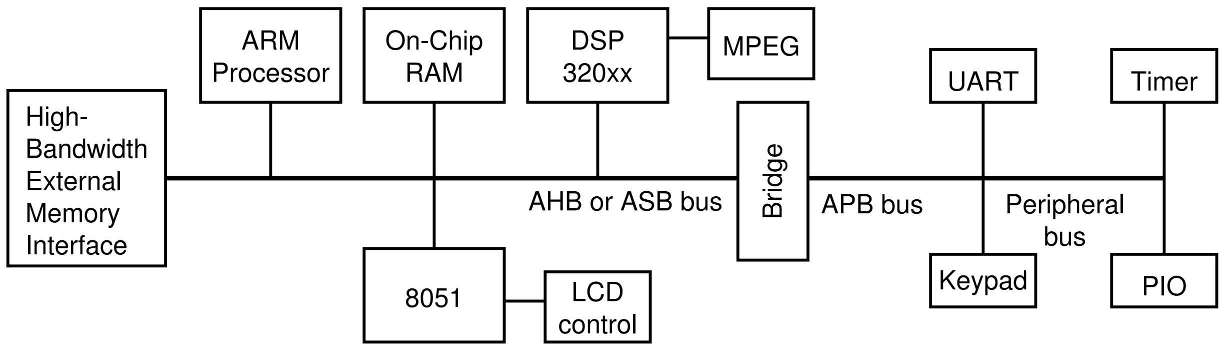


Figure 2.5: AMBA based system architecture

Atomic chains of transactions This represents a sequence of transactions initiated by a single master that is executed on a single slave exclusively. During this activity, other masters are denied to access that slave until the end of the first transaction. This mechanism is standardly used to implement synchronization mechanisms between masters by use of semaphores.

Media arbitration Bus master modules access the bus, and the arbiter grants the access. Arbitration is usually centralized, so there is only one arbiter component connected to a bus. It is also global, since all requests as well as the state of the bus must be visible to the arbiter. When a grant is given, the complete path from the source to the destination is exclusively reserved.

Destination name and routing Command address and data are broadcasted on the bus. They reach every destination, only one of each activates, based on the broadcasted address, and executes the requested command.

2.2.3 Overview of actual on-chip buses

This subsection gives an overview of several popular on-chip bus-based interconnection networks. Here, we especially focus on describing the distinctive features of each of them.

2.2.3.1 AMBA bus

AMBA (Advanced Microcontroller Bus Architecture) [30] is a bus standard devised by ARM. Nowadays, AMBA is one of the leading on-chip bus system used in high performance SoC design. AMBA is hierarchically organized into two segments: system-bus and peripheral-bus (Figure 2.5), mutually connected via bridge that buffers data and operations between them. Standard bus protocols for connecting on-chip components generalized for different SoC structures, independent of the processor type. AMBA does not define method of arbitration, but instead it allows the arbiter to be designed to suit the application requirements.

The three distinct buses in AMBA are the following:

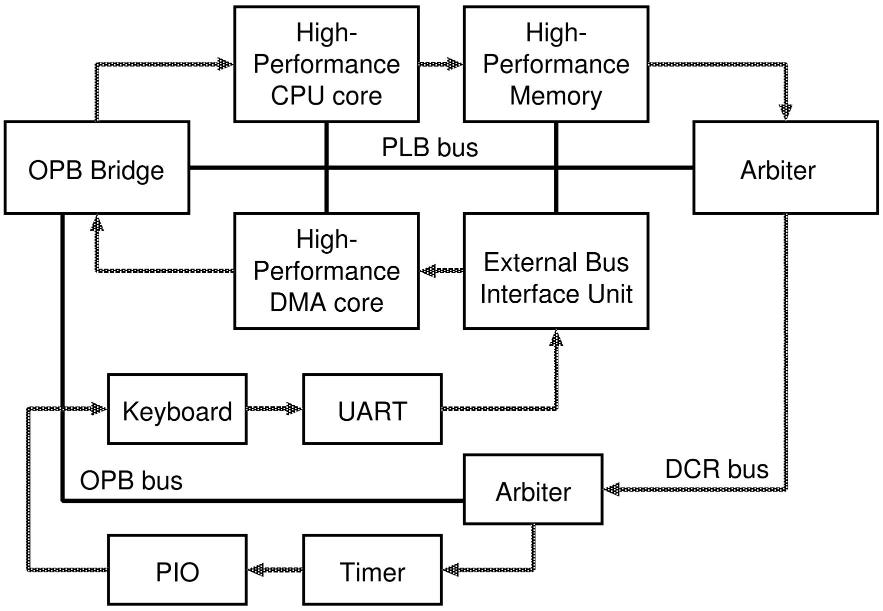


Figure 2.6: A system based on CoreConnect bus

- ASB (Advanced System Bus) - First generation of AMBA system bus used for simple cost-effective designs that support burst transfer, pipelined transfer operation, and multiple bus masters.
- AHB (Advanced High-performance Bus) - As a later generation of AMBA bus is intended for high performance, high-clock synthesizable designs. It provides high-bandwidth communication channel between embedded processors, and high-performance peripherals or hardware accelerators,
- APB (Advanced Peripheral Bus) is used to connect general-purpose, low-speed and low-power devices. The bridge is peripheral bus master, while all bus devices (Timer, UART, etc) are slaves. APB is a static bus that provides a simple addressing with latched addresses and control signals for easy interfacing.

Recently, two new specifications for AMBA bus called Multi-Layer AHB and AMBA AXI are defined. Especially AMBA AXI is based on the concept of point-to-point connection.

2.2.3.2 CoreConnect bus

CoreConnect [31] is an IBM-developed on-chip bus. As shown in Figure 2.6, CoreConnect is a hierarchically organized architecture, comprised of three buses that provide an efficient interconnection of cores, library macros and custom logic within an SoC.

PLB (Processor Local Bus) is the main system bus. It is synchronous, multi-master, central-arbitrated bus. Separate address and data buses support concurrent read and write transfers. PLB

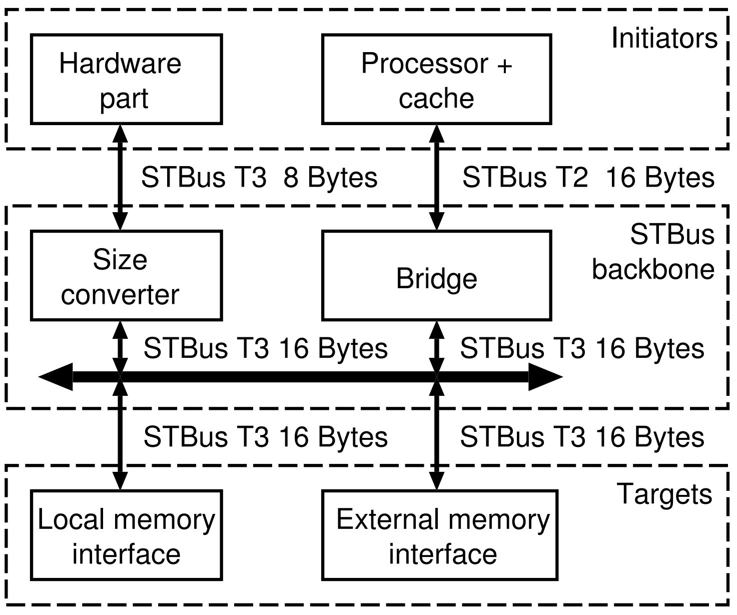


Figure 2.7: STBus interconnect

macro, as glue logic, is used to interconnect various master and slave macros. Each PLB master is attached to the PLB through separate addresses, read-data and write-data buses, and other control signals. PLB slaves are attached to PLB through shared, decoupled address, read data and write data buses.

OPB (On-chip Peripheral Bus) is optimized to connect lower speed, low throughput peripherals such as serial and parallel port, UART, etc. Crucial features of OPB are: fully-synchronous operation, dynamic bus sizing, separate address and data buses, multiple OPB bus masters, single cycle transfer of data between bus masters, and single cycle transfer of data between OPB bus master and OPB slaves. OPB is implemented as multi-master, arbitrated buses. Instead of tri-state drivers, OPB uses distributed multiplexer. PLB masters gain access to the peripherals on the OPB bus through the OPB bridge macro. The OPB bridge acts as a slave device on the PLB and a master on the OPB.

DCR bus (Device Control Register bus) is a single master bus mainly used as an alternative, relatively low-speed datapath to they system for following features: (a) passing status and setting configuration information into the individual device-control registers between the processor core and others components, and (b) design for testability purposes. DCR is a synchronous bus whose topology is ring, and implemented as distributed multiplexer across the chip. It consists of a 10-bit address bus and a 32-bit data bus. CoreConnect implements arbitration based on a static priority, with programmable priority fairness.

2.2.3.3 STbus

STBus is an on-chip bus protocol developed by STMicroelectronics [34]. It represents a set of protocol, interfaces and architectural specifications intended to implement the communication network of digital systems. The STBus interfaces and protocols are closely related to the Virtual Component Interface (VCI) industry standard.

STBus implements both the protocols definition and the bus components. The following protocols are used:

- Type I (Peripheral protocol) is a simple synchronous handshake protocol with limited set of available command types, suitable for register access and slow peripherals. No pipelining is applied. It acts as a request-grant protocol. Only limited operational code and length are supported.

- Type II (Basic protocol) is a more efficient protocol compared to Type I due to supporting split transactions and additional pipelining features. The transaction set includes read/write operation with different sizes, and also specific operations such as Read-Modify-Write and Swap. Transactions may also be grouped into chunks to ensure allocation of slave and to ensure no interruption of the data stream. This protocol is typically suited for External Memory controllers. A limitation of this protocol is that the traffic must be ordered and transactions must be symmetric, that is, the number of the requesting components must be equal to the number of responding components.

- Type III (Advanced protocol) is the most efficient, as it supports split transactions, out-of-order executions and asymmetric communications, that is number of requesting and responding components may differ. This type is mainly used by CPUs, multi-channel DMAs and memory controllers.

The STBus is modular and allows master and slaves of any protocol type and data size to communication. A wide variety of arbitration policies is also applicable, such as bandwidth limitation, latency arbitration, LRU, or priority-based arbitration.

The components interconnected by the STBus can be either initiators (CPUs or ASICs) or targets (memories, registers or dedicated peripherals). Initiators can load or store data through the bus backbone as shown in Figure 2.7.

STBus includes following components:

- Switch or node: they arbitrate and route requests and responses, according to their arbitration mechanism

- Converter or bridge: they convert the request from one protocol to another (from basic to advanced protocol, for example).

- Size converter: it is used between two buses of same type of different widths. It includes buffering capacity.

STBus can instantiate different bus topologies. Single shared bus is suitable for simple low-performance implementations when wiring area is limited. Full crossbar can be applied in case of high-performance

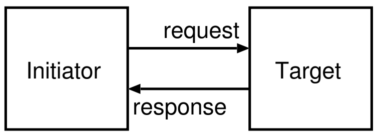


Figure 2.8: Point-to-Point Connection of VCI

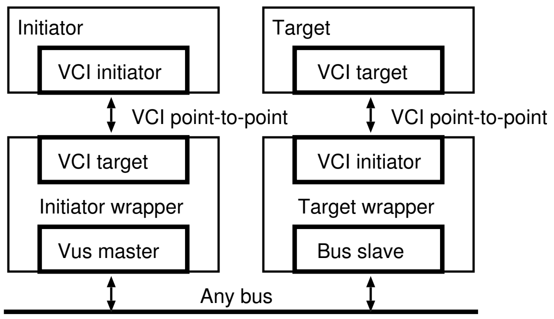


Figure 2.9: Two VCI connections to realize a bus connection

systems, and partial crossbar can also be applied to make good compromise between performance and area.

2.2.3.4 VCI (Virtual Component Interface)

The Virtual Component Interface (VCI) [35] is an interface which specifies the following:

- A request-response protocol
- A protocol for the transfer of requests and responses
- The contents and coding of these requests and responses

The VCI is not in charge of bus allocation schemes.

There are three complexity levels for the VCI: Peripheral VCI (PVCI), Basic VCI (BVCI), and Advanced VCI (AVCI). The PVCI provides a simple interface for applications with minimal requirements. The BVCI defines an interface that is suitable for most applications. The AVCI adds more sophisticated features such as threads to support high-performance applications. PVCI is a subset of BVCI, and BVCI is a subset of AVCI. ClearConnect of ClearSpeed's CSX600 described later adopts the AVCI.

BVCI and AVCI make use of a split protocol. Thus, the timing of the request and the response are fully separate. The initiator can issue as many requests as need without waiting for the response. The protocol does not prescribe any connection between issuing requests and arrival of the corresponding responses. The only specified item is the order of responses corresponding to the order of requests. In the AVCI, requests may be tagged with identifiers, which allow interleaving request threads so that they response to arrive in a different order. Responses bear the same tags issued with the corresponding requests, such that the relation can be restored upon the reception of a response.

As an interface, the VCI can be used as a point-to-point connection between two units called the initiator and the target, where the initiator issues a request and the target response, as shown in Figure 2.8.

Table 2.1: Overview of bus features

Name	Topology	Synchronous or Asynchronous	Arbitration	Transfers
AMBA	Hierarchical	Synchronous	(1)-(6)	(a)-(d)
CoreConnect	Ring, Hierarchical	Synchronous	(1)	(a)-(d)
VCI	n/a	Synchronous	(1)-(6)	(a)-(c)

Supporting arbitration mechanisms: (1) static priority, (2) TDMA, (3) lottery, (4) Round-robin (5) Token passing, (6) CDMA

Supporting transfer protocols: (a) handshaking, (b) split transfer, (c) pipelined transfer, (d) burst transfer

The VCI can be used as the interface to a wrapper, which means a connection to a bus. This is how the VCI allows some virtual component to be connected to any bus. An initiator is connected to that bus by using a bus initiator wrapper. A target is connected to that bus by using a bus target wrapper. Once the wrapper for the bus have been designed, any IPs can be connected to the bus, as shown in Figure 2.9.

2.2.3.5 Summary of commercial on-chip bus standards

Table 2.1 shows the main features of the prevalent existing SoC buses, including supporting network topology, synchronization mechanism, bus arbitration method, and types of data transfer. The atomic operations are implemented is one of the following two ways: a) the central arbiter locks the bus for exclusive use by the master requesting the atomic chain; and b) the central arbiter does not grant access to a locked slave. AMBA and CoreConnect use a mechanism of locking the slave and the bus for a short time. On the other hand, VCI does not lock a bus, so it can still be used by other modules, at the price of a longer locking duration of the slave.

For buses with centralized arbitration, the access time is proportional to the number of masters connected to the bus. The latency itself is constant and relatively low, because modules are linked directly. However, the speed of transfer is limited by bus speed.

All AMBA, CoreConnect and VCI allow pipelined transactions. This means that concurrent address transfer of read transaction is possible, and the data of previous write transaction can be sent, and the data from even earlier read transaction can be received.

2.2.3.6 Advantages and disadvantages of on-chip buses

Advantages In the bus-based design approach, bus components communicate through one or more buses, usually interconnected by bus bridges. Since the bus specification can be standardized, libraries of components whose interfaces directly match this specification can be developed, and for components that do not directly match the specification, wrappers can be built by making use of rich components of libraries, specialized development and simulation environments that companies

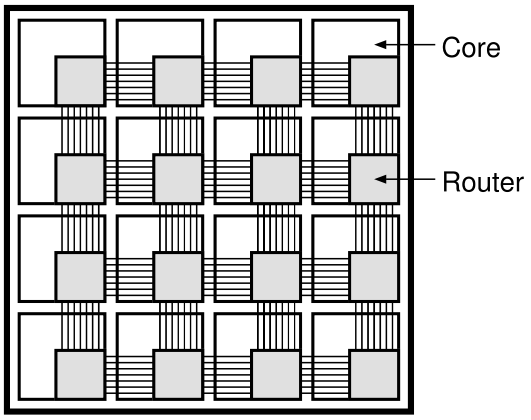


Figure 2.10: Network-on-Chip

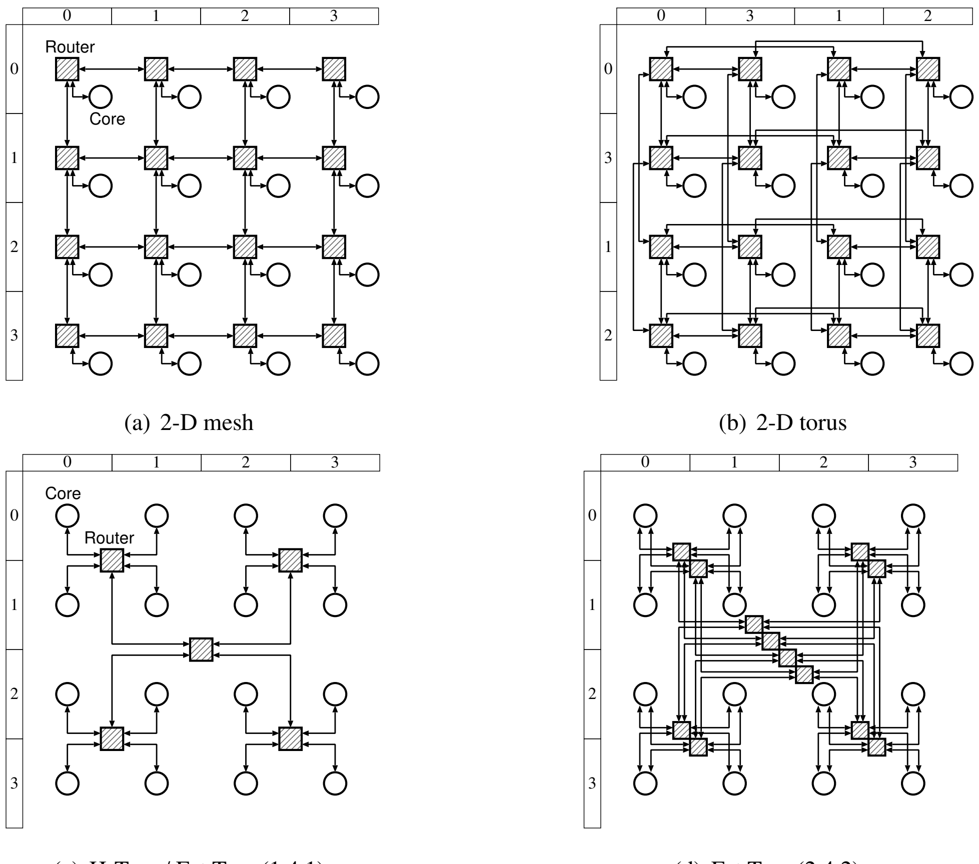
offer for designing systems around their buses. In summary, on-chip bus design methodologies are integration approaches that depend on standardized component or bus interfaces.

Disadvantages and proposals Bus-based design rely on shared communication resources and arbitration mechanism that is in charge of serializing bus access requests. This unfortunately suffers from power and performance scalability limitations, The bus is occupied by a single communication even if multiple communications could operate simultaneously on different portions on the bus. Therefore, a lot of effort has been devoted to the development of advanced bus topologies, such as partial or full crossbar, bridged buses, and protocols for better support of routing flexibility. Therefore, a systematic way of designing networks with possibly arbitrary topology is gaining the importance.

2.3 NoCs

NoCs have been utilized for both high-performance microarchitecture and cost-effective embedded devices mostly used in consumer equipments. Figure 2.10 shows an example of NoC that consists of sixteen tiles each of which has a processing core and a router. NoCs can exploit the enormous wire resources, unlike inter-chip interconnects whose bandwidth is usually limited by the pin-count limitation problems outside the chip. Assuming a 0.1μm CMOS technology with a 0.5μm minimum wire pitch, for example, a 3mm $\times$ 3mm tile can exploit up to 6,000 wires on each metal layer as illustrated in [24]. Finding the on-chip networks that effectively use large numbers of wires for low latency and high throughput communication with a modest silicon resources is thus essential for rapidly evolving devices.

Research topics related to on-chip network architecture can be classified into three topics: network topology, packet routing, and router architecture. Subsections 2.3.1, 2.3.2, and 2.3.3 survey packet routing schemes, on-chip router architectures, and network topologies, respectively.



(a) 2-D mesh

(b) 2-D torus

(c) H-Tree / Fat Tree (1,4,1)

(d) Fat Tree (2,4,2)

Figure 2.11: Two-dimensional layouts of typical topologies

2.3.1 NoC topology

In this subsection, we introduce the various network topologies which define the connection pattern between routers and cores in a network. Then we analyze their characteristics.

2.3.1.1 Interconnection Topologies

Figure 2.11 shows typical on-chip networks. where a white circle represents a processing core and a shaded square represents a router connecting other routers or cores.

Two-dimensional mesh (2-D mesh) Figure 2.11(a) shows a two-dimensional mesh topology. Mesh topologies can be represented as k-ary n-mesh, where k is the number of nodes in a dimension (i.e., array size) and n is the number of dimensions. The degree of the normal node is five, while that of the edge node is four and that of the corner node is three, resulting an asymmetric form.

Two-dimensional torus (2-D torus) Figure 2.11(b) shows a two-dimensional torus topology, which has wrap-around channels that connect nodes in top edge and bottom edge or nodes in left edge and right edge. Thus, a torus is a mesh with wrap-around channels. Torus topologies can be represented as k-ary n-cube, where k is the array size and n is the number of dimensions. Since all nodes in a torus have five ports, a torus is a symmetric topology, unlike mesh that does not have wrap-around ports. A torus has twice bisection bandwidth of a same-sized mesh due to its wrap-around channels. In addition, the average hop count of a torus is shorter than that of the same-sized mesh.

H-Tree On the other hand, constant attention has been focused on tree-based topologies, because of their relatively short hop-count, which enables a lower latency communication than mesh and torus. The simplest tree-based topology is H-Tree in which each router (except for the top-rank router) has one upward and four downward connections (Figure 2.11(c)). Thus, the number of ports is five in every router except the root. However, the links and routers around the root of the tree are frequently congested due to its poor bisection bandwidth.

Fat Tree To mitigate the congestion around the root of the tree, Fat Tree enhances the number of connections toward the root [36]. As stylized in [36, 37], various forms of Fat Tree can be created, and they can be expressed with a triple (p, q, c), where p is the number of upward connections, q is the number of downward connections, and c is the number of upward connections that each core has. Figure 2.11(d) shows a typical Fat Tree (2,4,2), in which each router (except for top-rank routers) has two upward and four downward connections, and each core has two upward connections. This is the network architecture used in CM-5 [38]. Note that a Fat Tree (1,4,1) is identical to the H-Tree.

2.3.1.2 Topological Properties

Ideal Throughput The ideal throughput of a network is the data acceptance rate that would result from a perfectly balanced routing and a flow control with no idle clock cycles; it is calculated as [28]

$$\Theta_{ideal} \leq \frac{2bB_c}{N}, \tag{2.1}$$

where N is the number of cores, b is the channel bandwidth, and B_c is the channel bisection of the network. Table 2.2 shows the channel bisection of typical networks. Again, the number of cores is $N = 2^n \times 2^n$; therefore the number of ranks in an N-core tree is $\log_4(N) = \log_4(4^n) = n$.

Table 2.2: Channel bisection B_c

	N-core	16-core	64-core	256-core
2-D Mesh	2^{n+1}	8	16	32
2-D Torus	2^{n+2}	16	32	64

Average Hop Count The number of source-destination pairs in an N-core network is $N^2 - N$; thus the average hop count in the network is

$$H_{ave} = \frac{1}{N^2 - N} \sum_{x,y \in N} H_{(x,y)}, \tag{2.2}$$

Table 2.3: Average hop count H_{ave}

	16-core	64-core	256-core
2-D Mesh	4.67	7.33	12.67
2-D Torus	4.13	6.06	10.03

Table 2.4: Number of routers R

	N-core	16-core	64-core	256-core
2-D Mesh	N	16	64	256
2-D Torus	N	16	64	256

where $H_{(x,y)}$ is the hop count from core-x to core-y. Table 2.3 shows the average hop count of typical networks for uniform random traffic, in which each source sends equally to each destination. The average hop count depends on whether the routing includes non-minimal paths. Note that the average hop count varies depending on the traffic pattern and task mapping.

Number of Routers The number of routers in a chip affects the network logic area and the implementation cost. In the case of mesh and torus topologies, the number of routers is obvious as shown in Table 2.4.

Total Unit-Length of Links We present the wire length of typical topologies in the case of 2-D layout. Assuming that the distance between two neighboring cores aligned in a 2-D grid square is 1-unit, we define L as the total unit-length of links in a given network.

As for the mesh and torus, we ignore the links between the core and router, which will increase the total unit-length, for the sake of simplicity.

2.3.1.3 Summary of NoC topology

As introduced in this subsection, a large number of network topologies have been proposed so far. However, those employed in practical systems are limited to some well-known topologies, such as 2-D mesh and torus, because their grid-based regular arrangements are considered to match the two-dimensional VLSI layout. Therefore, we mainly compare with 2-D mesh and torus in the rest of this section.

2.3.2 Routing

The packet routing decides routing paths for a given source and destination nodes, and it is crucial to make the best use of topological bandwidth, and prevent packets from being discarded between any pair of nodes.

Table 2.5: Total unit-length of links L (1-unit = distance between neighboring two cores)

	N-core	16-core	64-core	256-core
2-D Mesh	$2(N - 2^n)$	24	112	480
2-D Torus	$4(N - 2^n)$	48	224	960

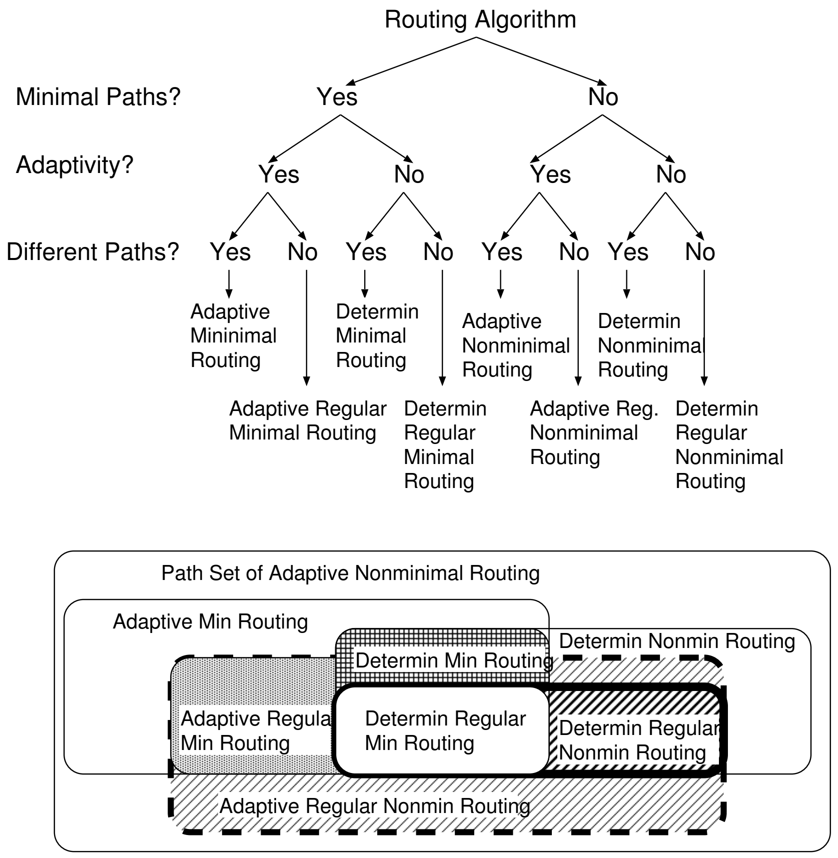

Routing Algorithm
Minimal Paths?
Adaptivity?
Different Paths?
Yes
No
Adaptive Mininimal Routing
Determin Minimal Routing
Adaptive Nonminimal Routing
Determin Nonminimal Routing
Adaptive Regular Minimal Routing
Determin Regular Minimal Routing
Adaptive Reg. Nonminimal Routing
Determin Regular Nonminimal Routing
Path Set of Adaptive Nonminimal Routing
Adaptive Min Routing
Determin Min Routing
Determin Nonmin Routing
Adaptive Regular Min Routing
Determin Regular Min Routing
Determin Regular Nonmin Routing
Adaptive Regular Nonmin Routing

Figure 2.12: Taxonomy of routing algorithms

2.3.2.1 Performance factors of routing protocols

A number of performance factors are involved in designing deadlock- and livelock-free routing algorithms in terms of throughput, amount of hardware, and energy. The implementation of routing algorithms depends on the complexity of routing algorithms, and it affects the amount of hardware for the router and/or network interface. Figures 2.12 and 2.15 show the taxonomy of various methods.

From a view point of path hops, routing algorithms can be classified into a minimal routing and a non-minimal routing. A minimal routing algorithm always assigns topological minimal paths to a given source and destination pair, while non-minimal routing algorithm can take both minimal and non-minimal paths. The performance of a routing algorithm strongly depends on two factors, average path hop and path distribution. Adaptivity allows alternative paths between the same pair of source and destination nodes. This property provides fault tolerance, because it usually enables to select

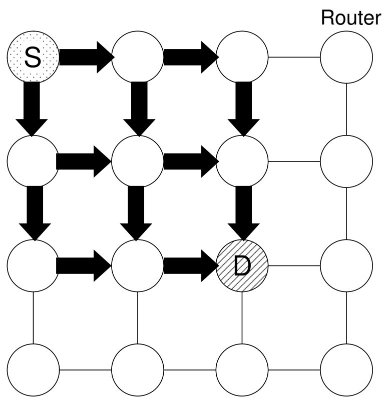


Figure 2.13: The adaptivity property

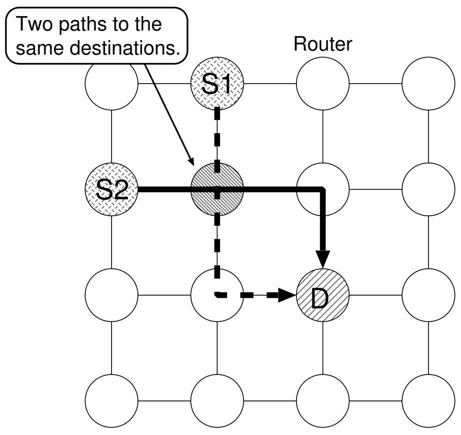


Figure 2.14: The different paths property

a path that avoids faulty network components. Different-path property bears some resemblance to adaptivity. Unlike that of adaptivity, though, the different-path properties enable a choice to be made between different paths to a single destination node depending on the input channel, or source node in Figure 2.14. The different paths property affects the routing table format at a router, and its entry number.

A routing algorithm that has adaptivity is called an "adaptive routing", while the other one that does not have adaptivity is called a "deterministic routing". In deterministic routing, all routing paths between any source and destination pairs are statically fixed and never changed during their flight. In adaptive routing, on the other hand, routing paths are dynamically changed during their flight in response to network conditions. Basis of the adaptive routing is adaptive routing algorithm and output selection function (OSF) [39–41]. First, adaptive routing algorithm defines available set of deadlock-free paths, and OSF actually determines actual output channel from the alternatives defined by the algorithm.

Deterministic routing has the following advantages:

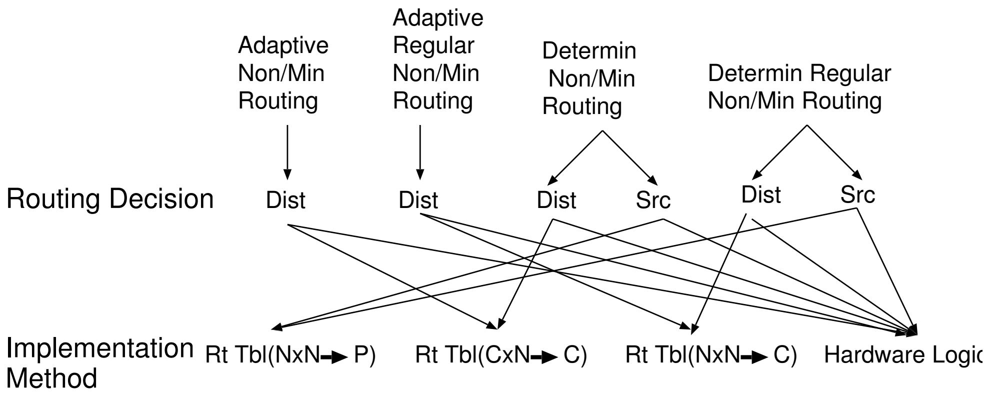


Figure 2.15: Taxonomy of routing implementations

1. Simple switch without selecting an output channel dynamically from alternative channels can be used
2. In-order packet delivery is guaranteed, as a communication protocol often requires.

On the other hand, adaptive routing has the following advantages:

1. Physical channels are efficiently used because multiple paths between a same pair of nodes are provided.
2. Routing paths are dynamically changed during their flight in response to network conditions such as congestion or faulty links.

System software including lightweight communication library, and the implementation of parallel applications are sometimes optimized on the assumption that in-order packet delivery is guaranteed in the network protocol. However, adaptive routing that provides multiple paths between a same pair of nodes introduces out-of-order packet delivery. In the case of using adaptive routing, an additional sorting mechanism at the destination node is needed for guaranteeing in-order packet delivery at the network protocol layer.

As shown in Figure 2.12, the path set of an adaptive non-minimal routing can include that of an adaptive minimal routing, because non-minimal path set consists of minimal paths and non-minimal paths. Similarly, the path set of an adaptive regular non-minimal routing can include that of a deterministic regular minimal routing. For example, the path set of West-First Turn Model (adaptive regular non-minimal routing) includes that of dimension-order routing (deterministic regular minimal routing).

Figure 2.15 shows the taxonomy of routing algorithm implementations. Routing implementation can be classified into a source routing (Src) and a distributed (Dist) routing according to where their routing decisions are made. In source routing, routing decisions are made by the source node prior to the packet injection to the network. The routing information calculated by the source node is stored

Table 2.6: Deadlock-free routing algorithms

Routing algorithm	Type	Topology	Minimum number of VCs
DOR	determin regular min	k-ary n-cube	1 (mesh) or 2 (torus)
Turn-Model Family	adaptive regular nonmin	k-ary n-cube	1 (mesh) or 2 (torus)
Duato's Protocol	adaptive regular min	k-ary n-cube	2 (mesh) or 3 (torus)

in the packet header, and intermediate nodes forward the packet according to the routing information stored in the header. Since a routing function is not required for each router, source routing has been widely used for NoCs in order to reduce the size of on-chip routers. In distributed routing, routing decisions are made by every router along the routing path.

There are three routing-table formats for distributed routing, and these routing-table formats affect the amount of routing information. The simplest routing format (function) directly associates routing (destination) addresses to paths, and it is based on $N(source) \times N(destination) \mapsto P$ routing relation (all-at-once) [28], where N and P are the node set and the path set, respectively. Since a routing address corresponds to a path in this relation, the routing address stored in a packet can be used to detect a source node at a destination node. The other routing functions provide information only for routing, and it is based on $N \times N \mapsto C$ routing relation that only takes into account the current and destination nodes [27], or $C \times N \mapsto C$ routing relation, where C is the channel set. In the $N \times N \mapsto C$, and the $C \times N \mapsto C$ routing relations, the destination nodes can not identify the source nodes from the routing address, and these routing relations cannot represent all complicated routing algorithms [28], unlike the $N \times N \mapsto P$ routing relation. However, their routing address can be smaller than that of the $N \times N \mapsto P$ routing relation.

2.3.2.2 Routing algorithm

Table 2.6 shows typical (deadlock-free) routing algorithms, and their features. Existing routing algorithms are usually dedicated to the target topology, such as k-ary n-cube topology. Since the set of paths on tree-based topologies such as H-tree is naturally deadlock-free, we omit discussion of routing algorithms on tree-based topologies.

Dimension-order routing (DOR) A simple and popular deterministic routing is dimension-order routing (DOR), which uses y-dimension channels after using x-dimension channels in 2-D tori and meshes. Dimension-order routing uniformly distributes minimal paths between all pairs of nodes, and it is sometimes referred to as "xy routing" or "e-cube routing". Dimension-order routing is usually implemented with a simple combination logic on each router; thus, routing tables that require register files or memory cells for storing routing paths are not used.

Turn model Assume that a packet moves to its destination in a 2-D mesh topology and routing decisions are implemented as a distributed routing. The packet has two choices in each hop. That is, it decides to go straight or turn to another dimension at every hop along the routing path until it reaches its final destination. However, several combinations of turns can introduce cyclic dependencies that cause deadlocks. Glass and Ni analyzed special combinations of turns that never introduce deadlocks [6]. Their model is called Turn-Model [6].

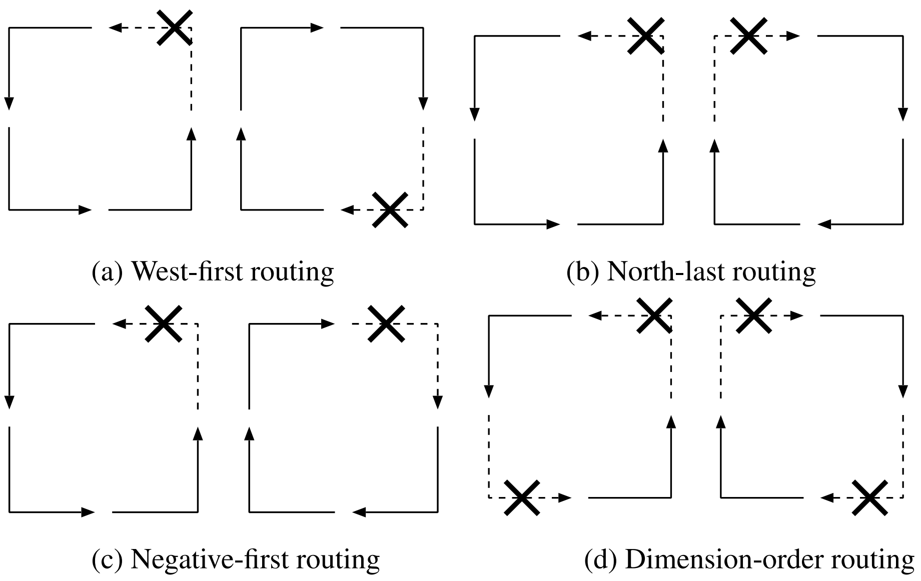


Figure 2.16: Prohibited turn sets of three routing algorithms in Turn-Model

In addition, for a 2-D mesh topology, they proposed three deadlock-free routing algorithms by restricting the minimum sets of prohibited turns that may cause deadlocks. These routing algorithms are called West-first routing, North-last routing, and Negative-last routing [6]. Figure 2.16 shows the minimum sets of prohibited turns in these routing algorithms. In West-first routing, for example, turns from the north or south to the west are prohibited. The authors of [6] proved that deadlock freedom is guaranteed if these two prohibited turns are not used in a 2-D mesh.

Turn-model view can also demonstrate the deadlock-freedom of dimension-order routing. Figure 2.16.(d) shows a set of prohibited turns in dimension-order routing. The prohibited turn-set in dimension-order routing includes that of West-first routing. That is, the restriction of dimension-order routing is a super set of West-first routing's one. Dimension-order routing additionally prohibits two turns, which are not prohibited in West-first routing; thus the routing diversity of West-first routing is better than that of dimension-order routing. However, a routing algorithm that has high routing diversity does not always outperform one with lower diversity.

In fact West-first routing algorithm may provide a routing path set with a very poor traffic distribution for hot-spotted traffic. On the other hand, dimension-order routing provides a routing path set which always uniformly distribute the traffic and achieve good performance in the cases of uniform traffic, even though its path diversity is poor.

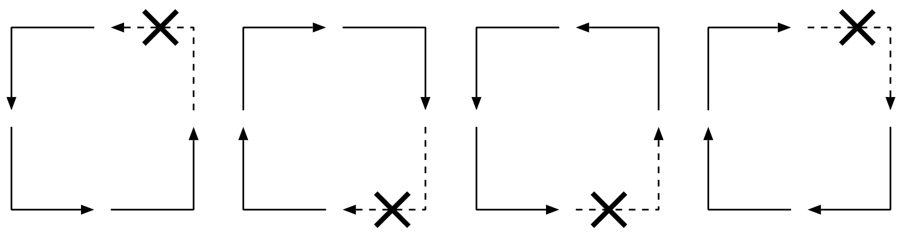

(a) Odd column (b) Even column

Figure 2.17: Prohibited turn set in Odd-Even turn model

Chiu extended the turn model into Odd-Even Turn-Model [42], in which nodes in odd columns and even columns prohibit different sets of turns. Figure 2.17.(a) shows a prohibited turn-set for nodes in odd columns, while Figure 2.17.(b) shows one for nodes in even columns. As reported in [42], the routing diversity of Odd-Even Turn-Model is better than those of the original turn models proposed by Glass and Ni. Odd-Even Turn-Model has an advantage over the original ones, especially in terms of higher path diversity.

Turn models can guarantee deadlock-freedom in a 2-D mesh, but they cannot remove deadlocks in rings and tori that have wrap-around channels in which cyclic dependencies can be formed. A virtual channel mechanism is typically used to cut such cyclic dependencies. That is, packets are first transferred using virtual-channel number *zero* in tori, and the virtual-channel number is then increased when the packet crosses the wrap-around channels.

2.3.2.3 Output selection functions (OSF)

In previous section, we introduced some adaptive routing algorithms. Here, we provide some examples of OSF, which is another factor of adaptive routing.

Random selection function Random selection function (RSF) [7] literally selects output physical and virtual channels randomly from alternative set of possible pair of input and output channels defined by the routing algorithm. Despite its simplicity, it is effective to disperse traffic loads.

Dimension order selection function Dimension order selection function is an OSF which targets dimensional topology such as mesh or torus. It always selects possible physical output channel with the lowest dimension. For example, when output channels of x and y direction are both vacant, it selects the x direction output in 2-D mesh topology.

ZigZag selection function ZigZag selection function [39] also targets dimensional topology such as mesh or torus. When there are multiple alternatives of output channels, it selects the direction with the furthest remaining distant. For example, when a packet is sent in a 2-D mesh network

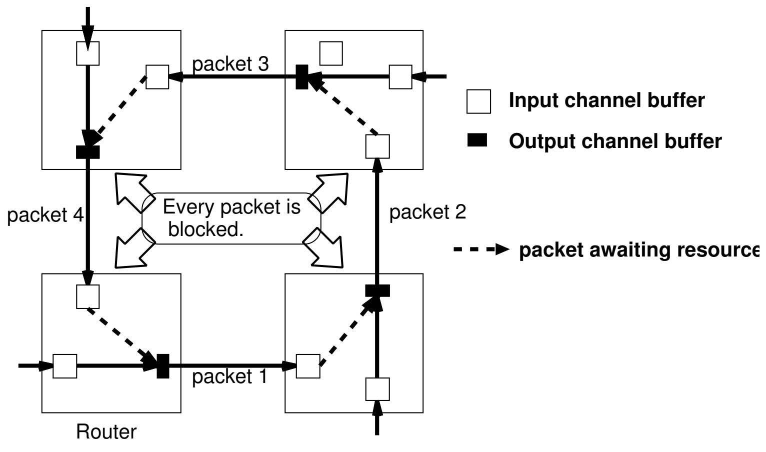


Figure 2.18: Deadlocks of packets

from $s\,(x_s, y_s)$ to $d\,(x_d, y_d)$ provided that x and y output channels are both vacant, output with larger dimension among $|x_d - x_s|$ and $|y_d - y_s|$ is selected.

2.3.2.4 Deadlocks and livelocks

Routing should resolve the deadlock problem. Figure 2.18 shows a situation where every packet is blocked by each other, and they cannot be permanently forwarded. Such a cyclic dependency is called “deadlock”. Once deadlock occurs, at least a single packet within a network must be discarded, and resent. To avoid deadlocks, “deadlock-free routing” algorithms that never introduce deadlocks on paths have been widely researched.

Besides the deadlock-free property, routing must have a livelock-free property to prevent packets from being discarded. Packets would not arrive at their destination node if they take non-minimal paths that go away from destination nodes. In such case, they would be permanently forwarded within NoCs. This situation is called “livelock”.

The deadlock- and livelock-free properties are not strictly required to routing algorithms in the case of traditional LAN and WAN. This is because Ethernet usually employs a spanning tree protocol that limits the topology to a tree whose structure does not cause deadlocks of paths. Moreover, the Internet protocol has the time-to-live field that limits the maximum number of forwarding transfers. Thus, the NoC routing cannot simply apply popular techniques used in commodity LANs and WANs, and new research fields have grown up dedicated to NoCs similar to those in parallel computers.

2.3.2.5 Duato’s protocol

Duato gave a general theorem defining a criterion for deadlock freedom and used the theorem to develop a fully adaptive, profitable, progressive protocol [8], called Duato’s protocol or *-channel. Since the theorem states that by separating virtual channels on a link into escape and adaptive partitions, a fully adaptive routing can be performed and yet be deadlock-free. This is not restricted to a particular topology or routing algorithm. Cyclic dependencies between channels are allowed, provided that a connected channel subset exists which is free of a cyclic dependency. A simple description of Duato's protocol is as follows.

a. Provide an escape path in which every packet can always find a path toward its destination whose channels are not involved in cyclic dependencies.

b. Guarantee that every packet can be sent to any destination node using an escape path and the other path on which cyclic dependency is broken by the escape path (fully adaptive path).

By selecting these two minimal routes (escape path and fully adaptive path) adaptively, deadlocks can be prevented. Duato's protocol can be extended to arbitrary topologies by allowing more routing restrictions, and non-minimal paths [43].

2.3.3 Router architecture

The router architecture has been studied for several decades for off-chip interconnects, and various architectural innovations of routers such as wormhole switching, pipelined packet transfer, and virtual channel mechanism were developed for massively parallel computers in 1980s and 90s. Although early NoCs have simply imported these architectures for on-chip purposes, various unique techniques intended to on-chip purposes are recently proposed [44–46].

2.3.3.1 Switching technique

In the cases of switch-medium networks, packets are transferred to their destination through multiple routers along the routing path in a hop-by-hop manner. Each router forwards an incoming packet to the next router until the packet gets the final destination. The packet switching techniques decide when the router forwards the incoming packet to the neighboring router. They can be classified into three techniques: store-and-forward switching, virtual cut-through switching, and wormhole switching. They affect the network performance and buffer size needed for each router.

1) Store-and-Forward (SAF) switching The straightforward technique for the packet switching is SAF switching, in which each router stores an entire packet in its buffer, and then it forwards the packet to the next node. In another words, each router cannot forward a packet before receiving the entire packet, and it must have enough buffers that can always store the largest packet. The maximum number of clock cycles required to transfer a packet to its destination can be represented as $(Fh + Fb) \times D$, where D is the diameter of a given network, Fh is the number of header flits, and Fb is the number of body flits. SAF switching requires relatively large buffers in each router and its communication latency is larger than the other switching techniques described below. Thus, SAF switching is rarely used in NoCs.

2) Virtual-Cut Through (VCT) switching In VCT switching, each router stores only fractions of a packet (i.e., flit(s)) in its buffer, and then it forwards the fractions to the next node. Consequently, each router can forward flits of a packet before receiving the entire packet. Thus, VCT switching has advantages in the communication latency compared to SAF switching, since the maximum number of clock cycles required to transfer a packet is $Fh \times D + Fb$, which is much smaller than that of SAF switching when D and/or Fb become large.

As well as SAF switching, VCT switching equips enough buffers to always store an entire packet in each router, so all fractions of a packet can be stored in a single node if the packet header cannot progress due to the conflicts. Although the communication latency of VCT switching is better than that of SAF switching, VCT switching requires large buffers for each node. As a result, VCT switching is widely used in off-chip interconnection networks like SAN.

3) Wormhole (WH) switching The most common switching technique used in NoCs is WH switching. It requires small buffers for each router so as to at least store a header flit in each hop, and each router can forward flits of a packet before receiving the entire packet. Thus, WH switching has advantages in the communication latency compared to SAF switching as well as VCT switching, though its buffer requirement is much smaller than that of VCT switching. This is the reason why WH switching is widely used in NoCs.

In WH switching, some flits in a packet are often stored across multiple routers along the routing path, and their movement much looks like a “worm”.

Again, fractions of a packet can be stored across different nodes along the routing path; so the single packet often occupies the buffers in multiple routers along the path when the header of the packet cannot progress due to the conflicts. Such situation is referred to as “Head-of-Line (HoL) blocking”. The occupied buffers due to HoL blocking prevent other packet transfers that go through these paths. The problem of WH switching is the performance degradation due to the frequent HoL blocking.

4) WH switching with virtual channels To equip multiple buffers in a single physical channel is the most common way to mitigate the HoL blocking of WH switching. In this case, each of multiplexed buffers in a physical channel acts as a “virtual” channel [47], and incoming packets would progress if more than one virtual channels in a physical channel are not occupied. Therefore, the occurrence of HoL blocking will be reduced as the number of virtual channels multiplexed in a physical channel increases.

2.3.3.2 Router components

Figure 2.19 illustrates a conventional virtual-channel wormhole router. This router consists of a crossbar switch (XBAR), an arbitration unit (ARB), and five input physical channels (or ports), and

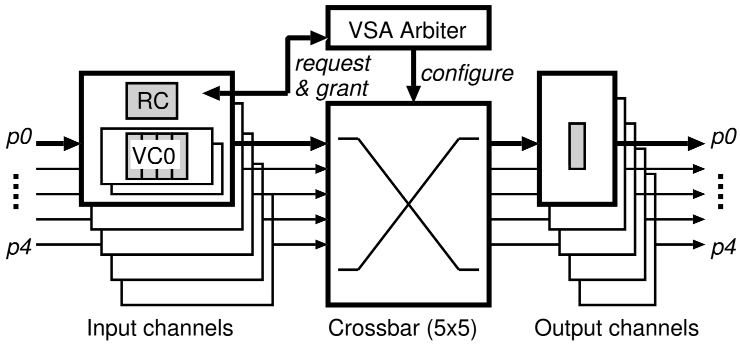


Figure 2.19: Wormhole router architecture

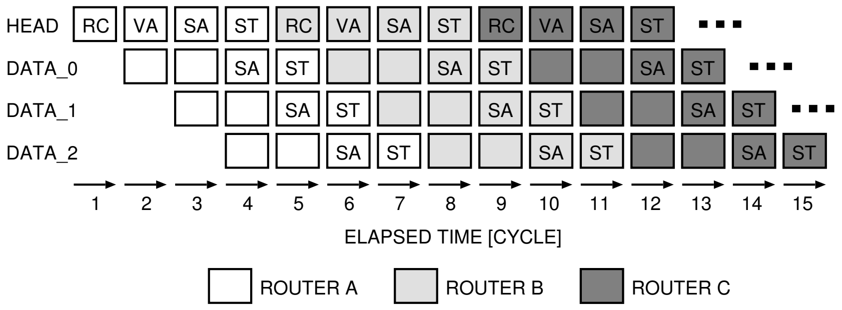


Figure 2.20: Router pipeline structure (4-cycle)

it is likely to keep components to a minimum for packet switching. We use it as a baseline router architecture for ease of understanding, and we explain each component in this section.

Physical channel Each physical channel consists of a routing computation (RC) unit and virtual channels, each of which has a FIFO buffer for storing four flits.

Input buffer The router has buffers at only its input channels. These FIFO buffers can be implemented with either SRAMs or registers, depending on the depth of the buffers, not the width.

Crossbar switch To mitigate the HoL blocking problem, a virtual-channel design sometimes employs a $pv \times pv$ full crossbar, where p is the number of physical channels and v is the number of virtual channels. However, the crossbar complexity significantly increases when using the $pv \times pv$ crossbar. In addition, its performance improvement will be limited because the data rate out of each input port is limited by its bandwidth [47]. Therefore, an on-chip router usually employs a small $p \times p$ crossbar by just duplicating the buffers.

2.3.3.3 Pipeline processing

Conventional router We explain a pipeline structure of the baseline router shown in Figure 2.19. A packet is transferred by using a pipeline processing technique which can be simply split into four

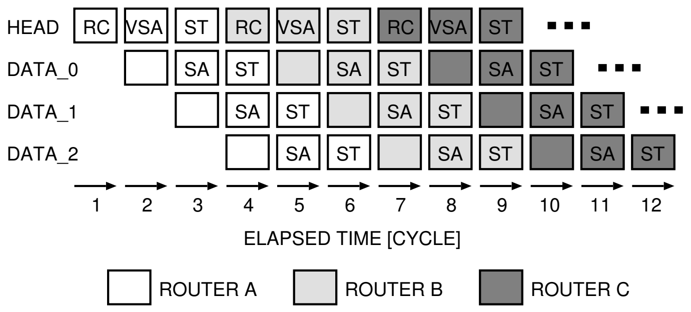


Figure 2.21: Router pipeline structure (3-cycle)

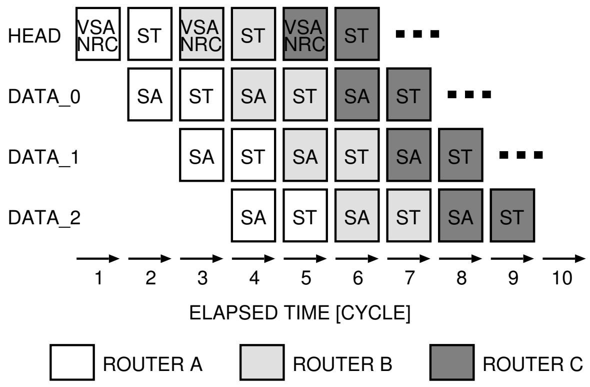


Figure 2.22: Router pipeline structure (2-cycle)

stages in the router. The 4-cycle pipeline [28] as shown in Figure2.20 is a typical pipeline architecture. In this router, a header flit is transferred through four pipeline that consist of the following stages:

- Routing Computation (RC) stage,
- Virtual channel Allocation (VA) stage for output channels,
- Switch Allocation (SA) stage for allocating the time-slot of the crossbar switch to the output channel, and
- Switch traversal (ST) stage for transferring flits through the crossbar.

Various router architectures have recently been developed to improve the communication latency and throughput, of interconnection networks. A speculative router speculatively performs different pipeline stages of a packet transfer in parallel [48]. The look-ahead routing technique removes the control dependency between the routing computation and switch allocation in order to perform them in parallel. In addition, some aggressive low-latency router architectures that skip one or more pipeline stages based on a static or single bypassing policy have been proposed [49, 50].

An on-chip router with a small number of pipeline stages has to perform a complicated operation in each stage compared to a deeper-pipeline router. Thus, while the operating frequency usually decreases, its throughput-per-cycle tends to be better. As the number of pipeline stages decrease, the operating frequency of the router gracefully decreases by speculatively executing multiple pipeline stages in parallel, instead of a simple cascading of adjacent stages that drastically decreases the operating frequency.

Usually, the pipeline depth of existing routers is fixed at design time so as to meet the required operating frequency and are never changed at run-time. However, various applications are running on a multi- or many-core processor, and their traffic loads are not constant. For this aim, a dynamic optimization of on-chip routers' pipeline structure in response to a given traffic pattern has also been proposed [51]. According to this work, pipeline stages are consolidated in the following manner:

- **3-cycle pipeline (Figure 2.21):**
  [RC] [VA/SA] [ST]
- **2-cycle pipeline (Figure 2.22):**
  [NRC/VA/SA] [ST]
- **1-cycle pipeline:**
  [NRC/VA/SA,ST]

Here, [X] denotes that task X is performed in a single cycle. [X,Y] denotes that tasks X and Y are performed sequentially within a clock cycle, while [X/Y] denotes a parallel execution.

On-chip routers of state-of-the-art tile processors such as Trips processor [52] introduce simple 1-cycle routers which performs the above operations in a single cycle. Such router architecture policy has been adopted based on the observation that number of small packets that fit into one flit dominates large proportion of the entire traffic due to increase of cache coherence protocol request/acknowledgment signals. The detail of such tile architectures and traffic analyses will be addressed in Section 3.2.

2.3.3.4 Summary

Packet-based network, which relies on routers, is more complex than that of on-chip bus. The complexity of a router introduces the difficulty in providing low latency, low power, and high reliability. Thus, advanced on-chip routers have been widely researched for improving them.

Chapter 3

Many-core processor architectures and its network traffic characterizations

There are two approaches effectively exploit the increasing transistors. One is to simply increase homogeneous processors cores, which is a typical approach for fabricating commercial general-purpose multi-core processors. Another approach is to integrate graphic processing units (GPUs) and video image processing units on a single die to meet multimedia application requirements. It is likely that this architectural trend is going to continue. Meanwhile, many-core architecture called tile architecture is also expected as a new form of processors. By introducing the ideas of heterogeneous processors and tile architectures, a viable solution of future processor is a combination of NUCA-based processor and media processing unit such as SIMD accelerators integrated on a single chip,

As current and future processors need to deal with many IP cores on a single chip, there are several ongoing studies of chip multiprocessors prototyping. Section 3.1 introduces architectures of latest homogeneous and heterogeneous multi-core processors as well as their interconnection technologies . Section 3.2 introduces representative tile architectures which are introduced in CMPs, and survey their interconnection topologies and routing mechanisms.

On the other hand, simplicity of SIMD's control mechanism allows integration of many computation processing elements (PEs) on a single chip. Vectoralizable applications such as multimedia applications, financial computations based on Monte-Carlo methods, many-body problems, fluid dynamics, or matrix problems, may take advantage of SIMD processors. Since instruction/data broadcasting and simultaneous memory accesses are necessary to take advantage of SIMD processing, shared memory architecture is often introduced. As inter-PE or PE-memory interconnection techniques, one-dimensional bus, ring, scatter-and-gather interconnect, and systolic architectures are major design options. Section 3.3 introduces some typical architectural examples of commercial many-core processors based on SIMD techniques. We also explore the availability of their interconnects by discussing their data transfer characteristics of each main target application in Section 3.4

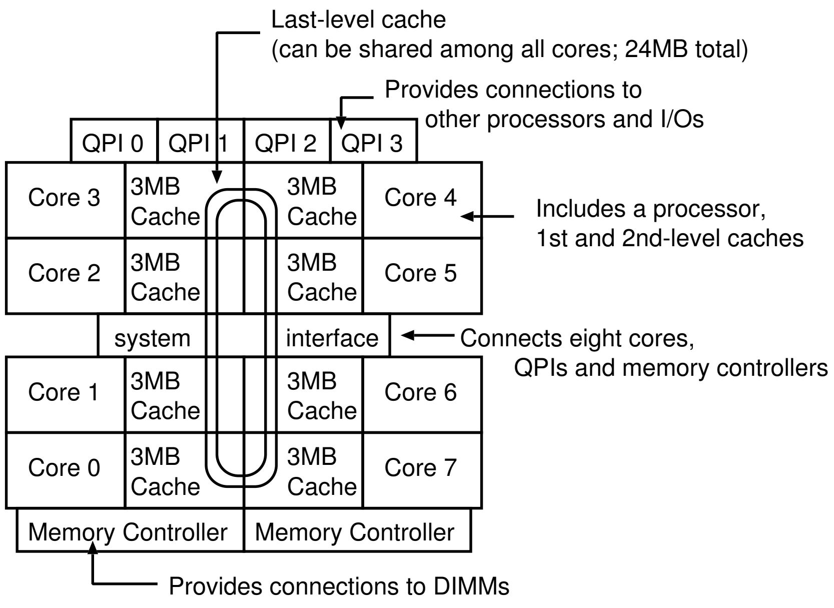

Last-level cache
(can be shared among all cores; 24MB total)
Provides connections to other processors and I/Os
QPI 0
QPI 1
QPI 2
QPI 3
Core 3
3MB Cache
3MB Cache
Core 4
Includes a processor, 1st and 2nd-level caches
Core 2
3MB Cache
3MB Cache
Core 5
system
interface
Connects eight cores, QPIs and memory controllers
Core 1
3MB Cache
3MB Cache
Core 6
Core 0
3MB Cache
3MB Cache
Core 7
Memory Controller
Memory Controller
Provides connections to DIMMs

Figure 3.1: Intel's Nehalem Xeon 7500 processor

3.1 Examples of recent CPU architectures

First, let us show the system architecture of recent representative multi-core general-purpose CPUs. This section introduces Intel's Nehalem (Xeon 7500), Sandy Bridge and AMD's Fusion APU.

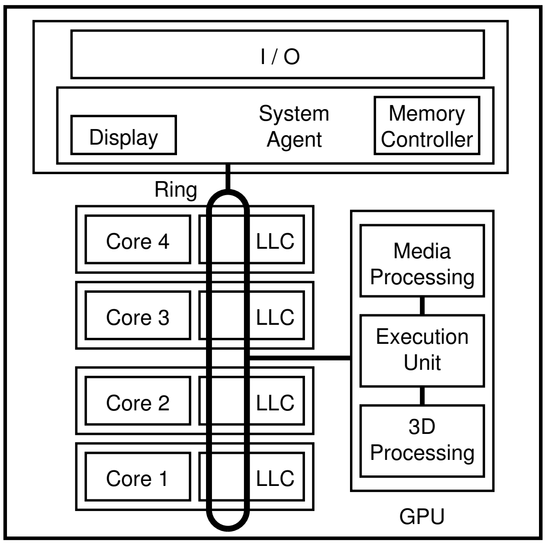
3.1.1 Intel's Nehalem Xeon processor

Nehalem is a basis architecture of 64-bit CPU under the brand name Intel Core i7 [53], fabricated with 45nm process. Figure 3.1 depicts Nehalem-EX, which contains eight processor cores, eight 3MB L3 caches, memory controllers and QPI channels.

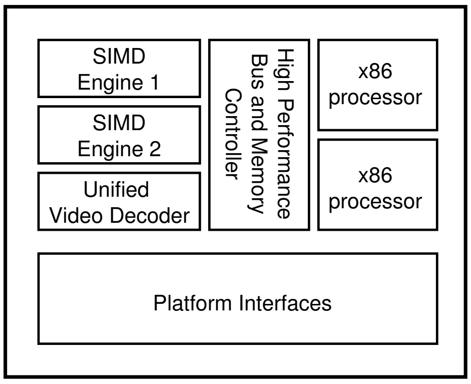
There are one cache slice per core, but all processor cores share eight caches via high bandwidth ring interconnects. The ring interconnect has two counter rotating rings, whose average latency is one half of unidirectional ring, and bandwidth is four times of unidirectional ring. Each data ring is 32-byte in each direction. Simple "rotary" arbitration mechanism is adopted, where traffic on the ring wins the arbitration. Ring stops tagged as "even" or "odd" polarities, which is similar to Cell/B.E. or Larrabee described later in Section 3.3.

3.1.2 General-purpose multi-core processor integrating a GPU

This subsection introduces Intel's Sandy Bridge [4] and AMD's Fusion APU [5], both of which integrate multiple processor cores and graphic cores on a same die. Figure 3.2 depicts overview architectures of the two processors.



(a) Intel's Sandy Bridge



(b) AMD's Fusion APU

Figure 3.2: Architecture of Intel's Sandy Bridge and AMD's Fusion APU

Intel's Sandy Bridge Figure 3.2 (a) depicts microarchitecture of Sandy Bridge, which came to market in January 2011. Fabricated with Intel's 32nm process technology, Sandy Bridge unifies four processor cores, four last-level caches (LLCs), memory controller and a graphic core. Fast access from processor cores or graphics to shared data in the last-level cache accelerates graphics processing.

Like Nehalem, Sandy Bridge processor also introduces a ring architecture. The ring connects CPU cores, LLCs, system agent and the graphics controller. The ring is physically located over the memory caches, and it provides access from any processor cores to any LLCs. There are four rings: data ring, request ring, acknowledge ring and snoop ring, all running at the same clock rate as CPU internal clock. Ring always choose the shortest path to the destination.

AMD's Fusion APU The AMD's Fusion APU (Accelerated Processing Unit) is a processor that integrates general-purpose scalar processors and vector processors on a 75 mm^2 die. Figure 3.2 (b) illustrates the arrangement of an APU, containing x86 cores, SIMD engines and a Unified Video Decoder (UVD) attached directly to a high-speed bus. The processor can be programmed with OpenCL and DirectCompute, which are two major development tools for multi-thread data-parallel software development.

3.2 Examples of tile architectures

In many-core processor architecture, it is not realistic for all the cores to share a single cache. Major approach is to connect cores and caches to a packet-switching network as shown in Figure 3.3.

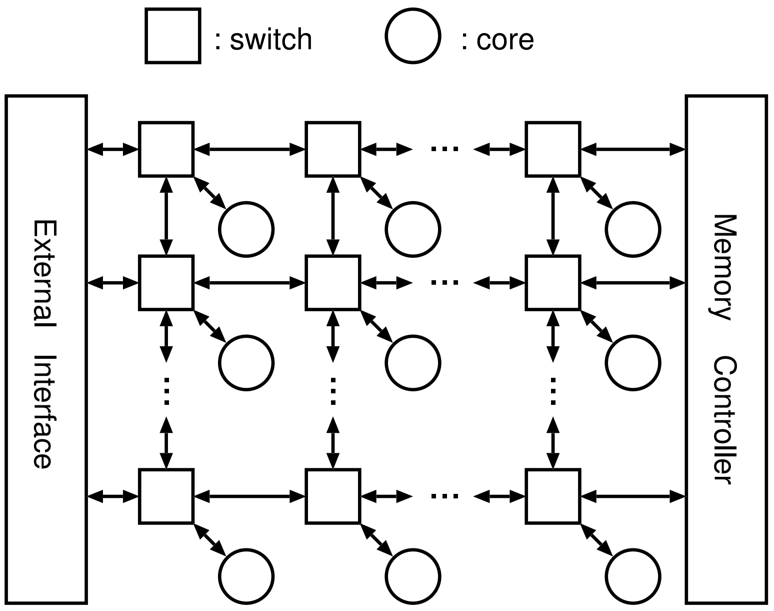


Figure 3.3: Tile-structured Many-core Processor

Currently, maximum number of processor cores in general-purpose processors is 12, and they are connected by ring buses. However, it is expected that bus-based architecture can handle up to 16 cores in maximum, and 2-D mesh topology-based many-core architectures is regarded as viable solution. For the aim of exploratory evaluations of future many-core processors, several tile-based architectures have been proposed, and their network traffic have been analyzed.

Let us show how the NoC technologies introduced in this chapter are used in representative tile architectures. In this section, we introduce Tilera's TILE architecture, Intel's 80-Tile and SCC processors, Trips processor, Raw processor, aSOC architecture, and Xpipes architecture.

3.2.1 Tilera's TILE64 and TILE-Gx100

Tilera's Tile processor is a tiled multi-core architecture. It is a MIMD machine consisting of a 2-D grid of homogeneous, general-purpose computing elements called "tiles", and is inspired by MIT's Raw processor described in Subsection 3.2.5. Their first implementation are called TILE64 with 64-core processor implemented in 90-nm technology [54]. Currently, there are processor series called TILE-Gx, TILEPro64, TILEPro36, with 100, 64, and 36 identical cores, respectively, with support of original dynamic distributed cache (DDC) system for fully coherent cache across the tiles [55]. Especially, TILE-Gx shown in Figure 3.4 offers System-on-a-Chip features including DDR3 memory controllers with ECC and system interfaces such as SGMII (Serial Gigabit Media Independent Interface) ports, USB ports, etc.

The Tile processor provides on-chip interconnect bandwidth through five low-latency mesh networks called user dynamic network (UDN), I/O dynamic network (IDN), static network (STN), memory dynamic network (MDN) and tile dynamic network (TDN). Each network connects five

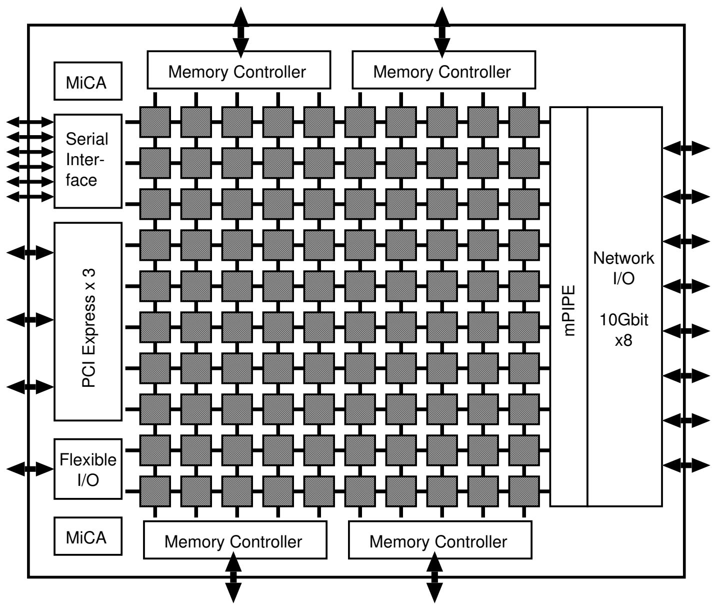


Figure 3.4: Tirela's 100-Core TILE-Gx100

directions (north, south, east, west and local tile processor). Each link consists of two 32-bit unidirectional links. Each tile uses a fully connected crossbar. With exception of STN, the networks are dynamic. Each packet contains a header denoting the x and y destination location along with packet's length, up to 128 words per packet.

The dynamic networks are dimension-ordered and wormhole-routed. The latency of each hop through the network is one cycle when packets are going straight, and one extra cycle for route calculation when a packet must make a turn. Because the networks are wormhole-routed, they use minimal in-network buffering. The network preserve ordering of messages between any two nodes, but do not guarantee ordering between sets of nodes.

The Tile architecture contains no unified bus for communication. Instead, I/O devices connect to the network just as other processor tiles do via IDN. Also, STN is a userland network which programmers can freely map communication channels onto. Thus, it allows a low-latency, high-bandwidth custom network, which is especially suitable for streaming data.

3.2.2 Intel's 80-Tile NoC

Intel's 80-Tile [13, 56] is a network-on-chip architecture containing 80 tiles arranged as 8×10 2-D mesh array of floating-point cores and packet switched routers (Figure 3.5), both designed to operate ad 4GHz. The on-chip 2-D mesh network provides a bisection bandwidth of 2 Terabits/s. In a 65-nm CMOS process, their first silicon achieves over 1.0 TFLOPS while dissipating less than 100W. Each tile consists of a processing engine (PE) connected to and 5-port wormhole router with mesochronous interfaces (MSINT), which forwards packets between the tiles. A router interface block (RIB) handles packet encapsulation between the PE and router. The fully symmetric architecture allows any

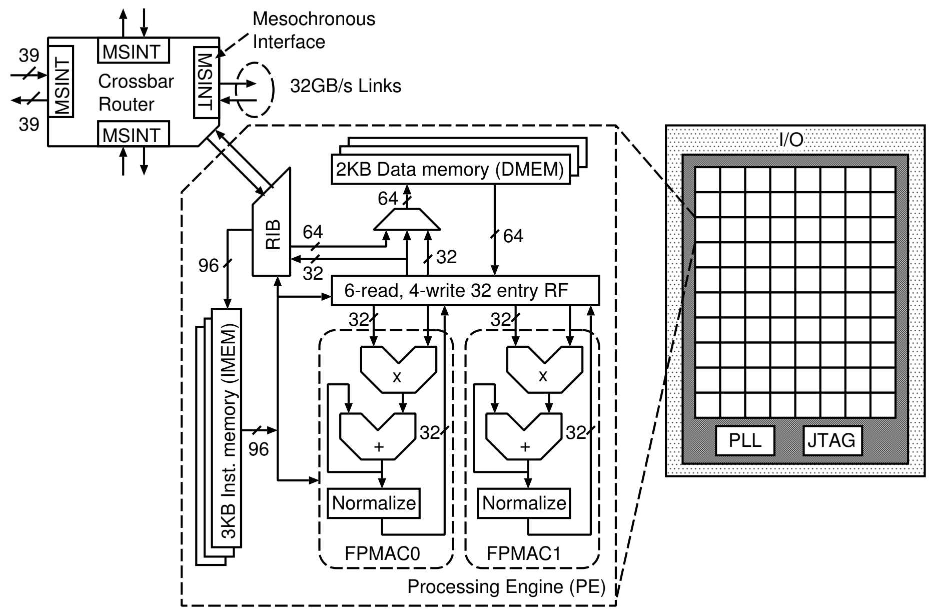


Figure 3.5: NoC block diagram and architecture of Intel's 80-tile

PE to send or receive instruction and data packets to or from any other tile.

Packet structure for 80-tile NoC is subdivided into flits. Each flit contains six control signals and 32 data bits. The packet header allows a flexible source-routing scheme, where a 3-bit destination ID field specifies the router exit port. This field is updated at each hop. Flow control and buffer management between routers is debit-based using *almost-full* bits, which the receiver queue signals via two flow control bits when its buffers reach a specified threshold. Each header supports a maximum of 10 hops. The minimum packet size required by the protocol is two flits, and the router architecture places no restriction on the maximum packet size.

A 4GHz five-port two-lane pipelined packet-switched router core with mesochronous links form the communication fabric for the 80-tile NoC. Each port has two 39-bit unidirectional point-to-point links. The input-buffered wormhole-switched router uses two logical lanes for deadlock-free routing and a fully non-blocking crossbar switch with a total bandwidth of 80 GB/s (32-bit×4GHz×5-ports). Each lane has a 16 queue for storing 16 flits, arbiter and flow control logic. The router uses a 5-stage pipeline with a 2-stage round-robin arbitration scheme that binds an input port to an output port in each lane and then selects a pending flit from one of the two lanes. A shared datapath architecture allows crossbar switch re-use across both lanes on a per-flit basis.

3.2.3 Intel's 48-core SCC processor architecture

Intel's 48-core SCC (Single-chip Cloud Computer) processor [57,58] integrates 48 Pentium class IA-32 cores on a 6×4 2-D mesh network of tile core clusters (Figure 3.6). The processor is optimized to support a message-passing-programming model whereby cores communicate through shared memory. Additionally, an 8-byte bidirectional system interface (SIF) provides 6.4GB/s of I/O bandwidth.

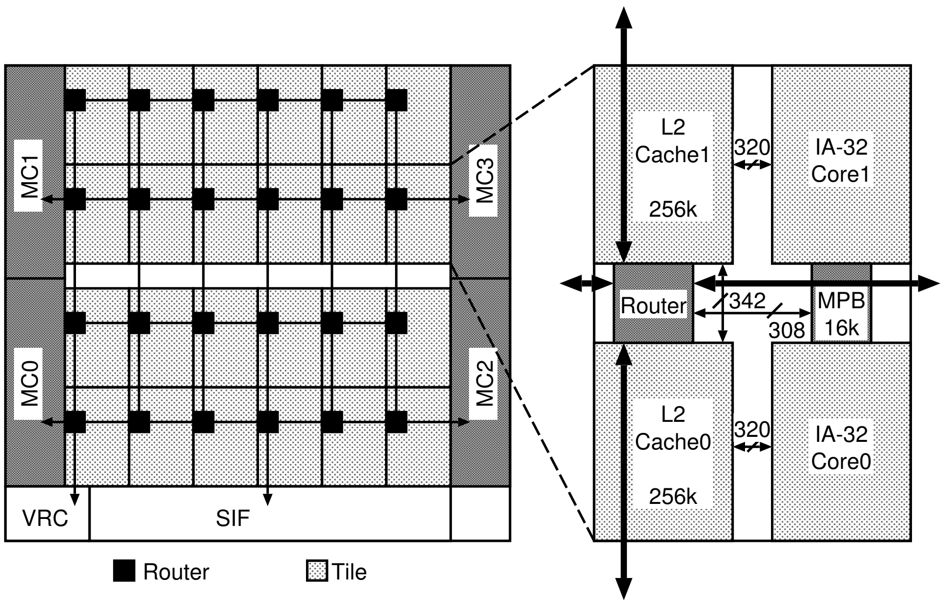


Figure 3.6: Block diagram and tile architecture of Intel's SCC

The design is organized in a 6×4 2-D array of tiles to increase scalability. Each tile is a cluster of two enhanced IA-32 cores sharing a router for inter-tile communication. The 5-port virtual cut-through router used to create the 2-D mesh network employs a credit-based flow-control. Router ports are packet switched, have 16-byte data links, and can operate at 2GHz. Each input port has five 24-entry queues, a route pre-computation unit, and a virtual-channel (VC) allocator. Router pre-computation for the output port of the next router is done on queued packets. An XY dimension ordered routing algorithm is strictly followed. Deadlock-free routing is maintained by allocating eight virtual channels between two message classes on all packets. Input and output port arbitration are done concurrently using a wrapped wave front arbiter. Crossbar switch allocation is done in a single clock cycle on a packet granularity. Router latency is four clock cycles, including link traversal. Individual routers offer 64GB/s interconnect bandwidth, enabling the total network to support 256GB/s of bisection bandwidth.

3.2.4 Trips Processor

A tile architecture named Trips microprocessor has been developed in Texas University [52, 59]. In 2005, a prototype chip was fabricated with a 130nm CMOS technology. Its operating frequency is up to 500MHz, as reported in [52].

A Trips chip has two processor cores. Figure 3.7(a) shows the Trips core architecture. The Trips core can issue sixteen instructions in parallel in an out-of-order manner. The capacity of its instruction buffer is 16×64 instructions; so it can deal with up to 64 instructions in parallel. As shown in Figure 3.7(a), each Trips processor has sixteen execution nodes, and they are connected with a 4×4 2-D mesh network. Figure 3.7(b) shows architecture of the execution node. Each

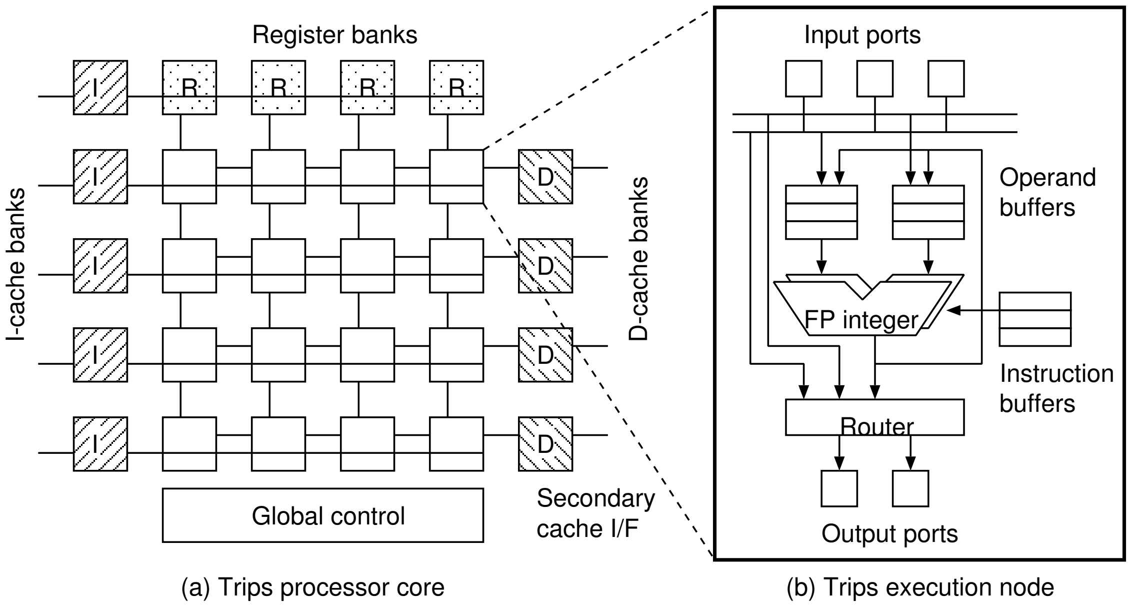


Figure 3.7: Trips processor core architecture

execution node is a simple processor that has an ALU, an FPU, and an instruction buffer that can store up to 64 instructions (8 instructions × 8 blocks). Execution nodes are connected to the network via on-chip routers. As shown in Figure 3.7(a), a Trips processor has four register banks, and they are placed in the top edge of the network that connects execution nodes mentioned above. Similarly, a data cache and an instruction cache are partitioned into several banks, and they are placed in the right and left edge of the execution node network, respectively. These cache banks are connected to the external L2 cache. A total of 2-Megabyte cache memory is distributed on a Trips chip. The L2 cache is divided into 32 banks and they are connected via on-chip networks.

Also, Trips support two NoCs for memory and operand traffic. The Trips on-chip network (OCN) replaces a traditional memory bus, and the operand network (OPN) replaces a traditional operand bypass and level 1 cache buses. Each of these networks is customized for its specific design constraints and traffic characteristics. Figure 3.8 shows the global architecture of Trips processor. The figure shows the OCN connecting the two processor cores two the level 2 cache banks and I/O units, including chip-to-chip network interface unit that extends the OCN off chip. Within each processor core, the OPN connects the different tiles as shown in 3.7.

Because of the different purposes between OCN and OPN, two networks differ in packet length and width, dimensions, and router design. A primary requirement of the OCN is low latency by introducing an OCN wormhole router which has only one pipeline stage, thus single-cycle-per-hop can be accomplished. In OCN, YX dimension order routing is adopted. The channel width is 138-bit, with 10-bit used for flow control and the remaining 128-bit for payload. Because cache fill requests and spill acknowledgment replies require approximately 12-bytes and occupy one half of all packets, channel width is set to the closest even divisor of a cache line size so that small packets can fit into

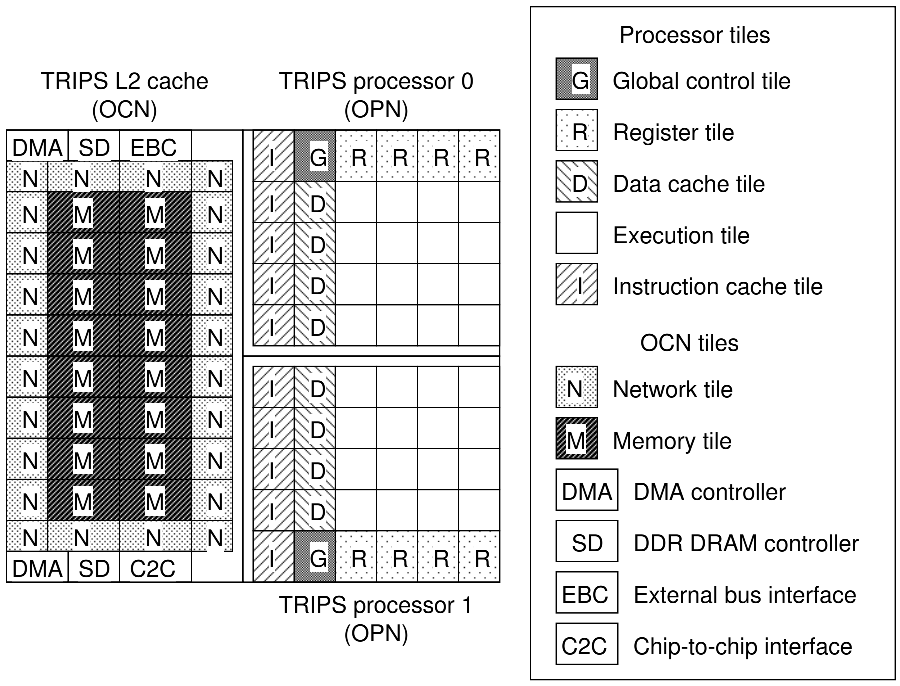


Figure 3.8: Global microarchitecture of Trips processor

one flit.

Contrary, the OPN delivers operands between instructions on different execution tiles with a latency of one cycle per hop. In addition, all OPN messages are fixed to 138-bits in length, packed into a single-flit.

3.2.5 Raw Processor

Raw microprocessor has been developed in MIT since the late 1990s [10,60]. As reported in [60], a Raw microprocessor is built with 122 million transistors and it can execute sixteen different instructions such as load instruction, store instruction, ALU instructions, and FPU instructions in a single cycle. In addition, a total 2-Megabyte L1 cache memory is distributed over the chip. The neighboring two tiles can communicate with each other within a single cycle.

The silicon area of the Raw microprocessor chip is divided into sixteen areas, each of which is referred to as a programmable tile. Each tile has the following modules:

- A single on-chip router for static communication
- Two on-chip routers for dynamic communication
- A single MIPS-like processor
- 32-KByte data cache
- 96-KByte instruction cache

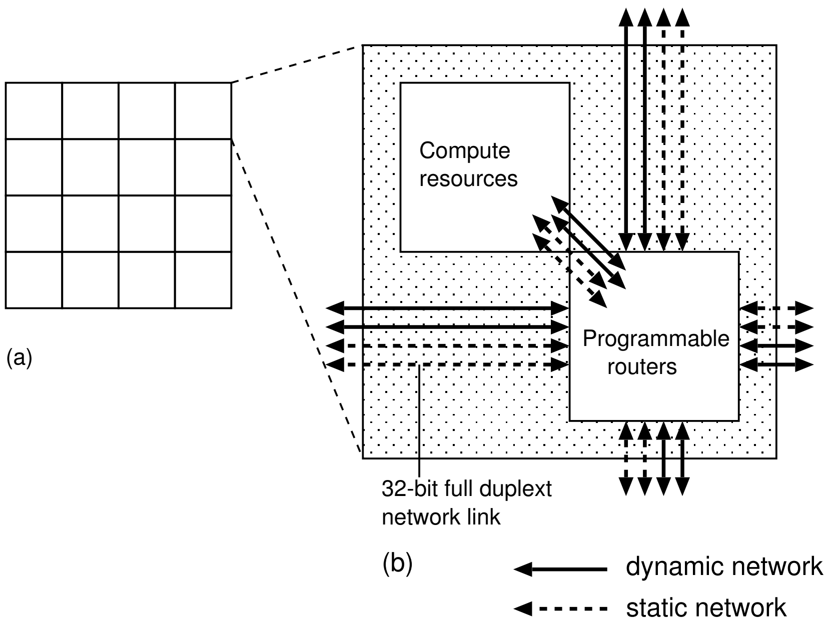


Figure 3.9: Interconnection networks on Raw microprocessor

Authors in [60] suggest the future possibilities of integrating more than a hundreds tiles.

Figure 3.9 shows the interconnect architecture of Raw microprocessor. A Raw microprocessor consists of sixteen tiles (Figure 3.9(a)). As shown in Figure 3.9(b), each tile has its own computational resources and it is connected to four neighboring tiles (i.e., north, south, west, and east) via on-chip routers. Four 32-bit full-duplex links are used for router-router links and router-core links, respectively. Up to 12,500 wires are required for the network links; thus the interconnection area consumes the largest area in the Raw chip.

The Raw architecture has four different networks. Two of four networks are used as static networks, in which all packet transfers are statically scheduled at compile time of the applications. On the other hand, the other two networks are called dynamic networks, in which packets are dynamically generated according to the input data or computational results. Each tile (except for tiles in chip edges) is connected to four neighboring tiles via point-to-point links. These links are the longest wires that introduce the largest wire delay in the Raw microprocessor. Thus the largest wire delay is predictable, and it is highly possible to increase the number of tiles integrated on a chip.

Static routers control two static networks. They are used to transfer single-word operands and data streams in an in-order manner. These data transfers are completely pre-scheduled at the compile time of applications. Static routers are fully pipelined with five pipeline stages. Since there are two static networks, each static router in a tile has two crossbar switches. And the crossbar switch is connected to another crossbar in the same static router, a MIPS-like processor in the same tile, and four neighboring tiles.

Dynamic networks deal with message transfers that cannot be predicted at the compile time, such as packet transfers induced by cache misses and interrupts. The dynamic networks employ

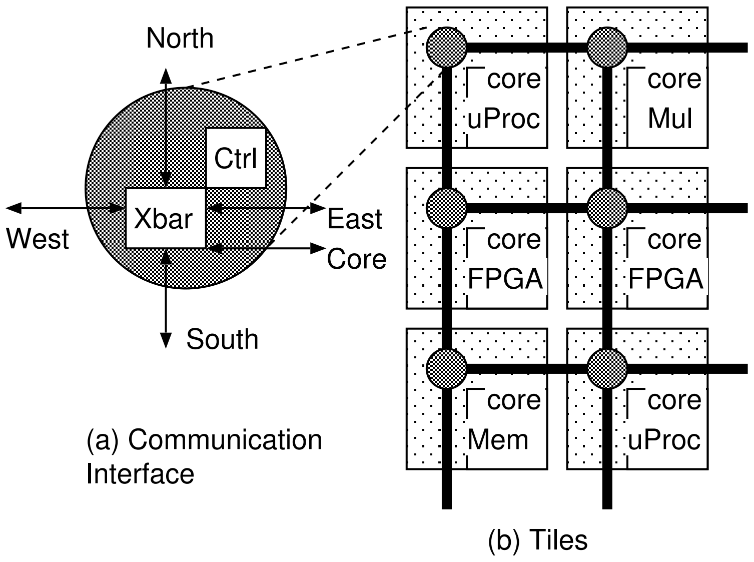


Figure 3.10: aSOC architecture

dimension-order routing and wormhole switching, as a routing algorithm and a switching technique, respectively. A packet consists of up to 31 flits including a header flit in which its destination address is stored. A router takes one cycle to forward a single flit when the packet goes straight via the router, while it takes two cycles to forward it when it turns to another dimension at the router. Two cycles are also required to forward a flit from (to) computational resource in the same tile. Therefore, communication latency required to transfer a single flit from a source tile to a destination tile is $(2 + x + 1 + y + 2)$ cycles, where x and y are the hop counts in x- and y-dimension, respectively.

3.2.6 aSOC Architecture

In 2004, Liang *et. al.* proposed an NoC architecture called adaptive System-On-a-Chip (aSOC) for stream processing [61]. They also produced an application development environment called AppMapper for aSOC. The concept of aSOC is to employ a simple tile architecture with a sophisticated compiler (aSOC compiler) that optimizes the communications between tiles. The prototype chip of aSOC architecture was fabricated with a 180nm CMOS technology. Its operating frequency is up to 400MHz. Various applications such as MPEG-2 encoder and Doppler radar signal analysis were implemented and evaluated on the aSOC chip. The evaluation results show that the aSOC architecture outperforms the traditional hierarchical buses.

Figure 3.10 shows the aSOC architecture. As shown in Figure 3.10(a), computational tiles are connected via a 2-D mesh on-chip network in an aSOC chip. Each tile consists of a processing core and a communication interface. The communication interface can be customized depending on the types of processing cores, the data width, and the operating frequency. Neighboring two communication interfaces are connected via a point-to-point link, as shown in Figure 3.10(b). The packet transfer is fully pipelined.

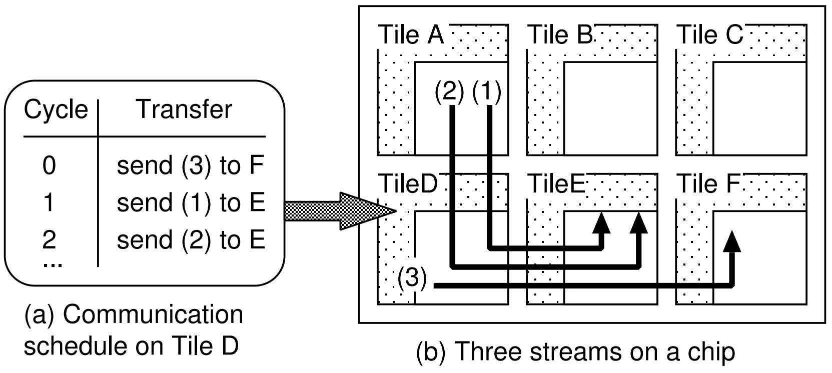


Figure 3.11: Communication scheduling of aSOC

The aSOC communication architecture deals with both static and dynamic communications. The static communications are scheduled and optimized at the compile time, while the dynamic communications are dynamically generated.

Static Communication Scheduling The static communication is fully optimized by AppMapper. AppMapper is used for the code optimization, code partitioning, and communication scheduling. AppMapper calls specific tools to compile and synthesize the source codes of target applications in response to the types of processing cores. This flow is fully automated, but it is possible to manually optimize the source codes. AppMapper integrates various simulators such as SimpleScalar in order to simulate heterogeneous cores. Using AppMapper, designers can simulate the behavior of mapped applications and evaluate their performance before implementing the application on the aSOC chip.

The target applications are described with a high-level language like C. AppMapper first partitions the application into several tasks, and these tasks are mapped onto the physical tiles of the aSOC chip. AppMapper employs a heuristic algorithm for task mapping. Then, the communication pattern between these tiles is analyzed in order to estimate the timing and amount of all data transfers. Based on the estimation, the link bandwidth between tiles are reserved in a cycle-accurate manner.

Figure 3.11 shows an example of communication scheduling of the aSOC architecture that statically reserves a necessary link bandwidth. Assume that data streams are generated on the aSOC, as shown in Figure 3.11(b). As you can see, data streams (1) and (2) are generated from tile A to tile E, and data stream (3) is generated from tile D to tile F. Figure 3.11(a) shows a cycle-level communication scheduling at tile D. The communication schedule shows that stream (3) is forwarded to tile E in cycle 0, stream (1) is forwarded to tile E in cycle 1, and stream (2) is forwarded to tile E in cycle 2. Based on the cycle-level communication scheduling, the necessary link bandwidth is allocated for the stream applications.

Dynamic Communication Scheduling In addition to the predictable communications, stream applications typically include the data dependent processing that cannot be pre-scheduled at the compile time. In this aSOC architecture, each tile always reserves a small amount of extra bandwidth for dynamic data transfers.

The extra bandwidth for all source-destination pairs is not allocated in every cycle, but it is allocated in different time slots in a round-robin manner. That is, a source tile can send a dynamic data stream to a destination tile during a time-slot reserved for the communication with the destination. Otherwise, the source tile must wait for the time-slot before sending the data to the destination.

Both Raw microprocessor and aSOC architecture can deal with dynamic data transfer, but they employ different approaches. Raw microprocessor has sophisticated hardware components such as dynamic routers for communications, while aSOC architecture just reserves a small amount of network bandwidth for communications without adding extra routers.

3.2.7 Xpipe Architecture

Bologna University has developed an NoC architecture called Xpipe and its development environment for multi-processor systems on a chip [62, 63]. The development environment provides soft macro libraries including switch, network interface, and network link for Xpipe architecture. These soft macros can be customized according to the target application. In addition, a synthesis tool called XpipesCompiler is provided to automatically generate the complete NoC systems by combining these soft macros.

Xpipe architecture can be customized so as to fit the target application by changing various parameters listed below:

- Network-specific parameters: flit width, redundancy of error control circuits, address space of the cores, diameter of the network, and bit width used for the flow control purposes
- Block-specific parameters: types of interfaces (e.g., master, slave, or both), size of output buffers, routing table, bit width used for address and data fields, and maximum number of flits in a single burst transfer
- Switch architecture parameters: number of virtual channels in a physical channel, size of buffers, length of a link, and a number of repeater stages in a link

A packet consists of a header flit and a series of payload flits. Each flit contains Flit_Type field in order to identify its flit type (e.g., header or payload). Xpipe architecture employs wormhole switching and street-sign routing, as a switching technique and a deterministic routing algorithm. Street-sign routing is implemented as a distributed routing, and its routing information (e.g., forwarding directions of incoming packets) is stored in the routing table in each switch. The reason why Xpipe architecture selects a deterministic routing, not an adaptive routing, is to simplify the switches and reduce their hardware cost.

The switch architecture is fully pipelined, and its latency is twelve cycles. Figure 3.12 shows architecture of an output controller. Each switch has four physical channels, each of which has two

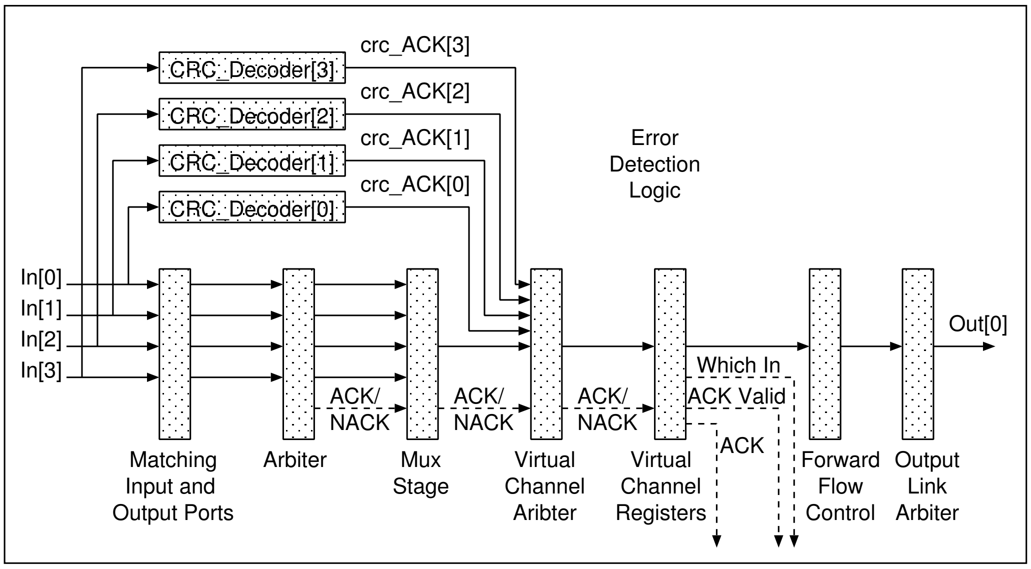


Figure 3.12: Output port of Xpipes switch

virtual channels. Assuming that the number of link pipeline stages is N and the number of switch pipeline stages is M, each virtual channel needs register files that can store $(2N + M)$ flits. In order to improve the reliability of data transfer, a CRC decoder is implemented as an error correction code for each link (Figure 3.12). Although such link level error correction consumes a larger hardware resources than the end-to-end level error correction does, the link-level correction offers a better reliability than the end-to-end level correction does.

Figure 3.13 shows architecture of a network interface in Xpipe architecture. The network interface in Xpipe architecture is compliant with Open Core Protocol (OCP) [64], in which a core and a network interface are regarded as a master device and a slave device, respectively.

As for network topology, Xpipe architecture also supports designing application-specific custom topologies in addition to regular topologies such as meshes. With the Xpipes environment, Bertozzi *et.al.* implemented a Video Object Plane Decoder (VOPD) consisting of twelve tiles onto both mesh and custom topologies [63]. The comparison of the both experimental results showed that the number of necessary switches is reduced in the custom topology, and achieves significant area and power improvements [62, 63].

3.3 Examples of SIMD many-core accelerators

We would like to show interconnection technologies for many-core processors that adopt SIMD control mechanism. In this section, we introduce two generations of NEC's IMAP processor, Intel's Larrabee processor, and STI's Cell/B.E. processor.

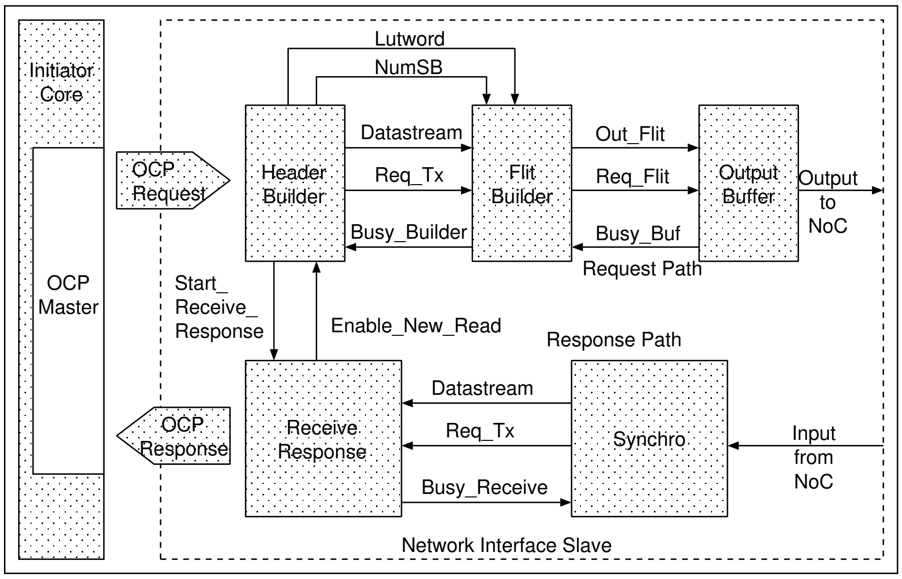


Figure 3.13: Network interface of Xpipe architecture

3.3.1 Cell Broadband Engine

The Cell Broadband Engine (Cell/B.E.) architecture [11,65] shown in Figure 3.14 is a heterogeneous multi-core processor consisting of 12 core elements connected through a multi-ring bus called Element Interconnect Bus (EIB). It integrates one controlling core called Power Processing Element (PPE), eight computation cores called Synergistic Processing Elements (SPEs) one Memory Interface Controller (MIC), and bus controllers (IOIF0, IOIF1).

Both direct memory access (DMA) and memory-mapped input/output (MMIO) are supported by the PPE. DMA is the only method for moving data between Local Store of SPE and system memory.

On-chip network of the Cell/B.E. architecture plays a crucial role, since it must support communication among twelve components. In addition, each component is capable of 51.2 Gbytes/s aggregate throughput. Thus, data rate requirement of its NoC is quite high, especially in contrast with Intel's and AMD's multi-core processors whose data rate is around 15 Gbytes/s per core. Also, it needs to support coherent data transfers, whose packet size may vary.

STI proposed the EIB as the solution to above requirements. It consists of the following:

- four 16-byte data rings (two in each directions),
- a shared command bus, and
- a central data arbiter.

The command bus distributes the commands, whose controllers are at the center (Figure 3.14), sets end-to-end transactions and handles memory coherence. Each ring can handle up to three concurrent non-overlapping transfers, allowing EIB to support twelve concurrent data transfers. The address

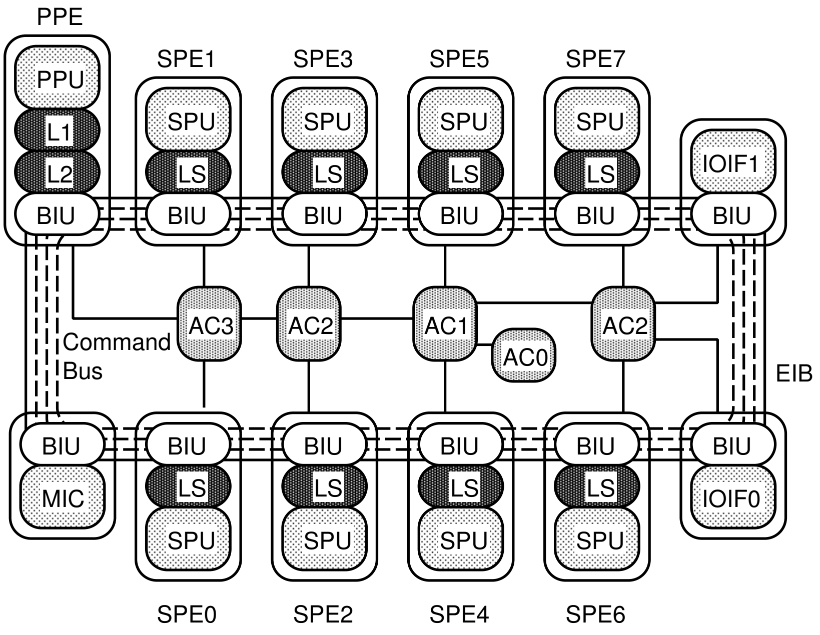


Figure 3.14: EIB Diagram - the Command Bus

concentrators (AC0 -AC3) handle collision detection and prevention, and they provide fair access to the command bus by round-robin mechanism.

3.3.2 Larrabee

Figure 3.15 shows architectural overview of Intel's Larrabee, a graphic processor whose architecture was announced in 2008 [66]. It is considered as a hybrid architecture between a multi-core CPU and a GPU by maintaining x86 architecture compatibility along with SIMD Vector Also, it will feature cache coherency across all of its computation cores.

As interconnection features, Larrabee has 16 CPU core blocks and peripheral circuits connected with bi-directional high-speed ring network, as Figure 3.15 shows. This network supports communication with external I/Os and also coherent transfers for L2 cache. Each direction are 512-bit wide, and can transfer data to adjacent units per two clock cycles. All routing decisions are made before injecting messages into the network. For example, each core can accept a message from one direction on even clocks and from the other direction on odd clocks. This simplifies the routing logic and meas that no storage is required in the routers once the message is in the network so as to provide high bandwidth with minimal contention at low cost.

3.3.3 IMAP processors

IMAP-VISION is a SIMD processor array suitable for image processing [67]. It integrates 32 8-bit processors and connects them in series to form a one-dimensional array. Figure 3.16 shows its first and second generation architectural block diagram. As the figure shows, IMAP-VISION supports 64

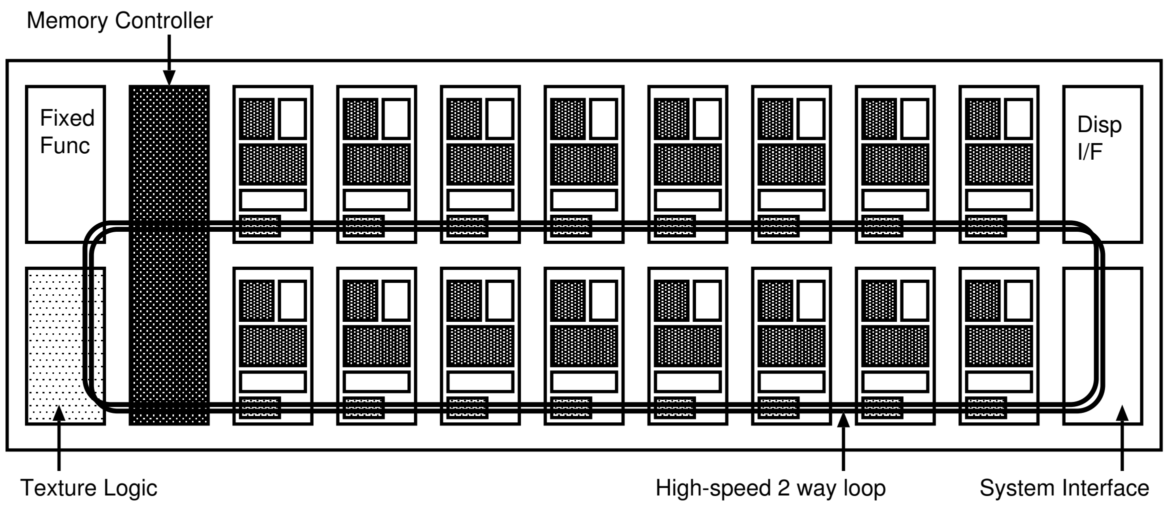


Figure 3.15: Ring bus of Intel's Larrabee

Kbyte/processor access to external memory in a 16 Mbit Synchronous DRAM. Data transfer can be issued asynchronously and concurrently with computations.

The third generation of IMAP processor [68] is shown in Figure 3.17. The chip is designed based on the following observations of preceding IMAP processors:

- Poor performance in global operations using 2-D mesh network: distant pixel require time for communication in proportion to the distance
- Low flexibility to attain good trade-off between granularity (number of pixels an each PE can process) and computation capability of each PE

To address the above problems, the processor introduces an two-dimensional array topology with a global bus. Each PE is connected to the neighboring up, down, left and right PEs and performs inter-PE communication with them. For every row and column, a common global bus is provided and data are supplied to the buses from outside via shift registers. Scalar values are outputted as the processing result via column adders connected to the right-end PEs.

In terms of computation capability, the processor supports a feature called "chained PE" operations to perform high-speed cumulative additions and logical operations, that is an output of one ALU is given to a neighboring ALU for computation. Such chained PEs can be regarded as one large PE. In chained PEs, data broadcasting using cumulative logical OR can be performed freely. Also, the grain size of PE and network structure is dynamically reconfigurable to attain good trade-offs between PE capability and granularity according to the application that runs on IMAP.

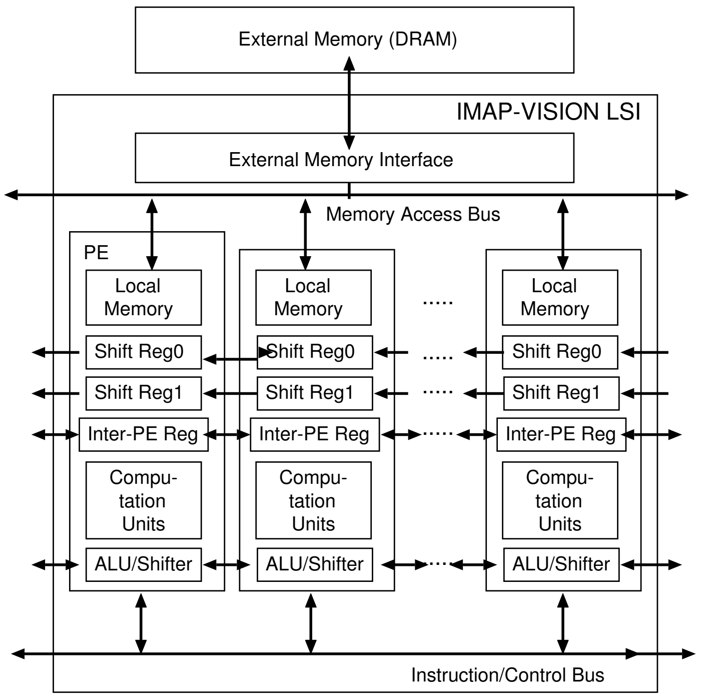


Figure 3.16: IMAP-VISION

3.4 Network analysis and traffic characterizations of existing many-core processors

This section gives overview of network traffic characteristics in recent many-core processors, including that of cache coherence protocols and parallel applications. Next, we introduce some approaches of network analysis for existing many-core processors.

3.4.1 Cache coherency protocol signals and their traffic patterns

First, this subsection gives brief overview of a cache coherence protocol and its traffic characteristics. This knowledge is one of the key factors for designing interconnection network for NUCA-based tile architectures,

The most basic coherence protocol is called MSI (Modified-Shared-Invalidate) protocol. Each cache line has a "tag" to identify the original memory address, and there is "state" bits in each tag. In uniform-cache processor which allows only one cache to be accessible from a processor, only two states are required for cache functions: "invalidate" bit and "modified" bit. In MSI protocol, in addition to the above two states, "shared" state is introduced. Shared state is the case when both invalid and modified bits are zeros.

Like single-cache architectures, invalid bit is set when cache line is empty, and unset when it is written as well as modified bit is unset, which is identical to the shared state. Modified bit is set when the processor write data to the cache line, In MSI protocol, before a processor writing data to its local

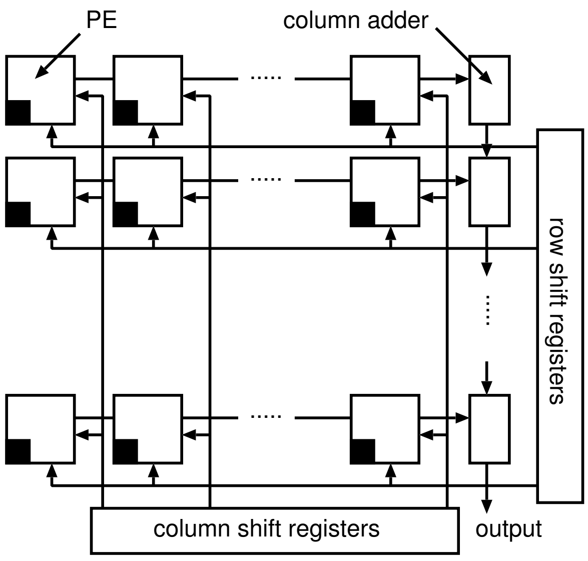


Figure 3.17: IMAP-VISION2

cache, the processor must send invalidate request signal to other processors' caches, and wait until all acknowledgment signals are received. Both of these request and acknowledgment signals can be realized with one bit, and in terms of traffic pattern, it is a scatter and gather operation. Meantime, such operation is also called "snooping". After receiving all acknowledgment signals, the processor write data to its local cache.

There are some other extensions of MSI such as MESI (Modified-Exclusive-Shared-Invalidate), MOSI (Modified-Owned-Shared-Invalidate) and MOESI (Modified-Owned-Exclusive-Shared-Invalidate) protocols, but all of these protocols send and receive very small messages among the processors. However, the amount of signals in the traffic increase by the order of N^2 when the number of cache on a processor is N. Consequently, NUCA is required to deal with efficient scatter/gather operation of very short messages.

3.4.2 Application-specific traffic

Initially, applications are likely to treat multi-core and many-core chips simply as conventional symmetric multiprocessors (SMPs). However, chip multiprocessors (CMPs) offer unique capabilities that are fundamentally different from SMPs, and which present significant new properties.

- The inter-core bandwidth on a CMP can be many times greater than is typical for an SMP, to the point where it should cease to be a performance bottleneck.
- Inter-core latencies are far less than are typical for an SMP system, by at least an order of magnitude.
- CMPs could offer new lightweight coherency and synchronization primitives that only operate between cores on the same chip. The semantics of these fences are very different from what

we are used to on SMPs, and will operate with much lower latency.

3.4.2.1 Characterization of parallel application kernels

A the level of the physical hardware interconnect, multi-cores have initially employed buses or crossbar switches between cores and cache banks. However, such solution are not scalable to 1000s of cores. Scalable on-chip communication networks will borrow ideas from large-scale packet-switched networks.

Department of Electrical Engineering and Computer Science (EECS) in University of California at Berkeley claimed in 2006 that 13 types of applications kernels provide insight into communication topology and resource requirement for many parallel applications that run on future many-core processors [69]. Based on their studies on communication requirements on existing massively-parallel computations, they announce the following viewpoints for many-core interconnection network.

- Requirements for collective communication and point-to-point are different in following aspects;
  - Collective communication - requires global communication, tended to involve very small messages that are latency bound.
  - Point-to-point communication - provides high-bandwidth communication, tended to involve large messages that are bandwidth bound.

  As the number of cores increases, importance of such fine-grained, collective synchronization messages will likely increase. Since latency improves much more slowly than bandwidth, the EECS working group suggests separation of latency-oriented network dedicated for collectives.

- Most point-to-point communications are stable and sparse, and only use a fraction of available communication paths through a fully-connected network switch. A non-blocking crossbar will likely be over-designed for most application requirements. For applications that do not exhibit special communication pattern, a lower-degree interconnect topology provide better efficiency and space and power.

- Although point-to-point communication patterns are likely be sparse, certain fixed-topology interconnection does not trivially provide the best space and power solution. Careful job placements or a reconfigurable interconnect fabric to match application's communication topology will become more important to match the static topology of interconnect fabric.

One can also use less complex circuit switches to provision dedicated wires that enable the interconnect to adopt to communication pattern of the application at runtime. A hybrid design that combine packet switches with an optical circuit switch has been proposed as a possible solution.

However, hybrid switch designs that incorporate electrical circuit switches to adapt the communication topology may be able to meet all of the need of future parallel applications. A hybrid circuit-switched approach can result in much simpler and area-efficient on-chip interconnects for many-core processors by eliminating unused circuit paths and switching capability through custom runtime reconfiguration of an interconnect topology.

3.4.3 Analysis of interconnection network using an actual processor

For understanding the design and performance of on-chip networks implemented within the context of a commercial multi-core chip, There have been a few researches which report the impact of networks using real machines and practical application. The analytical model of GPU includes the impact of network communication [70] in order to estimate the performance of the target application program.

Ainsworth evaluated EIB network (the shared bus of Cell/B.E. processor) by latency characterization and throughput characterization [71]. This work proposed 5-tuple latency characterization model which can be used to identify the bottleneck of end-to-end control of EIB. The 5-tuple delay model comprises the following five cost parameters:

- Send occupancy: number of cycles required for each processing element (PE) to initialize a packet transfer
- Send latency: number of cycles a packet waits to be injected in a network
- Network hop latency: number of cycles needed to transport a message to adjacent node
- Receive latency: required number of cycles which a node waits until received data is ready for use
- Receiver occupancy: required number of cycles which a node spends occupied in making the remote data ready for use

The end-to-end latency of EIB can be precisely modeled using the above five metrics and following equation:

$$Latency = Sending\ Overhead + Time\ of\ Flight + Transmission\ Time + Receiver\ Overhead \tag{3.1}$$

Finally, the best and worst-case latency was studied by assuming non-conflicting traffic pattern and conflicting pattern using the model. The work concludes that although the data bus provides enough bandwidth, overhead due to command bus severely limits the use of bandwidth.

In our study in Chapter 5, we take a similar approach; we analyze a concrete value of sending and receiving overhead for commercial SIMD many-core processor, study its effect by evaluating best- and worst-case latency, and estimate the number of processors required to cover up the effect of such overheads.

3.4.4 Analytical traffic modeling and characterization

3.4.4.1 Statistical traffic model

There has been research into statistical traffic models to help refine the design of NoCs. There is a proposal of three-tuple traffic model which is derived from network traces gathered from full-system simulations of 30 applications on three different chip architectures: Trips, Raw, and conventional directory-based cache-coherent shared-memory CMP [72]. Their model captures the spatio-temporal characteristics NoC traffic by three parameters: H, p and ω, each representing temporal burstiness of traffic at each node, abstracting parameter of hop distance of traffic and distribution, and distribution of the ratio of total network traffic that each router injects into the traffic, respectively.

3.4.4.2 Analysis based on network calculus

Generally, NoC simulation using traffic patterns is slow for large systems, and different design parameters need to be manually modified. On the other hand, analytical model allows a fast evaluation of performance metrics of large systems in the early design process. Bakhouya examined the feasibility of incorporating network calculus model to estimate and evaluate performance metrics of two-dimensional mesh on-chip interconnects using a given traffic pattern [73].

Network calculus is a mathematical modeling framework which is based on queuing theory, and was originally proposed for Internet quality of service. It is used in many other fields such as embedded systems such as network processors, switched Ethernet networks. It allows designers to specify a system as a mathematical model and evaluate main performance bottlenecks such as end-to-end delay, thus it is effective for worst-case analysis.

In Bakhouya's work, knowing the arrival and service curves, main performance metrics such as end-to-end delay and buffer size requirements of NoCs can be evaluated. Communication between two network components are characterized as *flows*, and are represented by sequence of hops, from a source to a destination. The system is represented by acyclic digraph called a *Core Graph*, comprising PE and a communication flow edge expressed by output arrival curve. Having the core graph, input and output arrival curves can be expressed, and delay bound and buffer requirements can be formulated.

3.4.5 Summary for analytical studies of interconnection network

There are a large number of analytical studies of NoCs [10,12,13,74], and the bandwidth and latency of NoCs have been analyzed with the following techniques:

1. theoretical analysis [75] [76];
2. probabilistic cycle-accurate simulation [77];
3. execution driven simulation;

Table 3.1: Interconnect architectures of representative many-core processors

System	Topology	Switching	Routing
TILE64 [54]	8 2-D mesh	wormhole	XY DOR
80-Tile [13,56]	8×10 2-D mesh	wormhole	source routing DOR
48-core SCC [57]	6×4 2-D mesh	virtual cut-through	XY, YX DOR, Odd-even turn
Trips (operand network) [52,59]	5×5 2-D mesh	wormhole	YX DOR
Raw (dynamic network) [10,60]	4×4 2-D mesh	wormhole	XY DOR
aSOC [61]	2-D mesh	circuit switch	shortest path
Xpipe [62,63]	2-D mesh & irregular	wormhole	street-sign
Cell/B.E. (EIB) [81]	4 rings	circuit switch	shortest path
Larrabee [66]	Bi-directional ring	circuit switch	shortest path
IMAP-VISION 3rd generation [68]	2-D mesh with global bus	circuit switch	unknown

4. full system simulation [78] [79]; and
5. execution on a real chip [80].

It is difficult to apply theoretical analyzed results in order to make precise model of large complicated network systems under real parallel application traffics, although it includes packet contentions at a switch under synthesis traffic. Although the probabilistic simulation has been often used for analysis of routing algorithms and topologies, traffic pattern is not usually based on real applications. Because execution driven simulation could take a long time to simulate a minutely modeled host with operating system especially for large systems, performance evaluation and analysis with monitoring tools have been done. Real chip analysis gives the most accurate and practical results, but the target is fixed.

3.5 Chapter summary

Table 3.1 summarizes the network topologies, switching techniques, and routing schemes of representative NoC systems in state-of-the-art CPUs, tile architectures and accelerators. Following observation summarizes the architectural trends.

- Wormhole switching is the most common switching technique used in NoCs. It can be implemented with small buffers so as to store at least a header flit in each hop. The buffer size of routers is an important consideration especially in NoCs, since buffers consume a substantial area in on-chip routers and the whole chip area is shared with routers and processing cores that play a key role for applications; thus, wormhole switching is preferred for on-chip routers.

- Grid-based (2-D mesh) topology is widely used for processor with large number of cores, and rings network for processors that have smaller number of cores.
- As for routing methodology, dimension-order routing (DOR) is widely used in NoCs that have grid-based topology, and circuit switching for rings. Especially the routing function of the former algorithm can be implemented with a small combination logic, instead of routing tables that require registers for storing routing information, and can distribute non-local traffics effectively.

The conceptual fundamentals of these techniques are universal. However, although commercial and prototype NoCs tend to be lightweight networks, we must not forget that the trend may changes in the future. Also, according to Section 3.4, following proposals have been made based on observation of traffic characteristics:

- As the number of cores increases, importance of such fine-grained, collective coherence and synchronization messages will likely increase. Message size for sending and receiving request and acknowledgment signals of these protocol is very small, and in terms of traffic pattern, it is likely a scatter and gather operation.
- Most point-to-point communications are stable and sparse, and only use a fraction of available communication paths through a fully-connected network switch. A non-blocking crossbar will likely be over-designed for most application requirements.
- Separation of latency-oriented network dedicated for collectives and bandwidth-oriented network are suggested, and hybrid network is one viable solution.
- Performance characterization of interconnection network, such as impact of the packet contention, hops, and transmission overhead, is rarely illustrated using a real chips, and

Based on these observations, in this thesis, we focus on the following two topics:

- Flexible packet routing mechanism for efficiently delivering short messages in grid-based CMPs with small hardware size.
- Performance characterization of interconnection networks on SIMD accelerators based on parallel application execution.

The first topic propose a new adaptive routing mechanism called Semi-deflection in order to achieve higher throughput for increasing collective messages, and compare it with existing deterministic and adaptive routing. The second topic aims to make latency and throughput characterization of actual processors in order to contribute to identification of bottleneck for end-to-end control in many-core accelerators.

Chapter 4

Routing methodology for chip multiprocessors

Performance, silicon area and power consumption of an on-chip interconnects that form the backbone of Chip Multiprocessors (CMPs) are directly influenced by its routing algorithm and router architecture that supports it.

In general, the most flexible routing algorithm is non-minimal fully-adaptive routing, because it allows each router to maximize the number of alternative paths, as well as it is advantageous in terms of fault tolerance [82]. Non-minimal fully-adaptive routing supports dynamic selection of various paths among multiple routers. The challenge of such algorithm is to guarantee deadlock-freedom. Fully-adaptive routing algorithms usually introduce virtual channels to realize deadlock freedom of packet transfers. For example, a minimal fully-adaptive routing called Duato's protocol uses two virtual channels for mesh topologies, and three for torus topologies. However, virtual channels require an addition of channel buffers for each router ports, and they are sometimes used for providing different quality of services (QoS) classes. In this work, we propose a non-minimal fully-adaptive routing which does not require virtual channels.

In Section 4.1, we propose a novel deadlock-free, non-minimal fully-adaptive routing called Semi-deflection routing that meets the requirements above. Section 4.2 shows throughput and area evaluation results.

4.1 Semi-deflection routing

Details of Semi-deflection routing is shown in this section. Based on the background described in Chapters 2 and 3, Semi-deflection routing assumes the following bases:

- Packet is a single-flit structure.
- NoC topology is 2-D mesh.
- Router is single-cycle pipelined, and does not require virtual channels
- Routing mechanism is non-minimal and fully-adaptive with deadlock and livelock avoidance.

Semi-deflection routing avoids deadlocks by using the approach of Turn-model, and it requires to update output selection functions and arbitration mechanism of routers as described below.

4.1.1 Turn model

Turn model is a family of adaptive routing which supports deadlock-freedom by removing all cyclic channel dependencies in a network topology [6]. North-last model and West-first model described in Section 2.3.2.2 are representative examples.

4.1.2 Deadlock removal

This subsection describes the mechanism of Semi-deflection routing. It allows prohibited turns of turn model under certain conditions, while it guarantees deadlock-free. We focus on the fact that deadlocks do not occur as long as single-flit packets are transferred in a non-blocking manner on every prohibited turn, because each packet can independently detour blocked packets.

First, we define "non-blocking packet transfer" as follows.

Definition 1 *(Non-blocking packet transfer) A packet transfer is called "non-blocking" when a packet at a certain input port is allocated to an output port via router's crossbar switch, given priority by the arbiter, and transferred to the selected output port without being blocked by other packets.*

Even when a prohibited turn is taken, deadlock does not occur as long as packets are guaranteed to be transferred to adjacent routers in a non-blocking manner. Thus, fully deadlock-free routing is satisfied by always transferring packets which intend to take prohibited turns in a non-blocking manner.

Figure 4.1 shows pairs of input and output ports where non-blocking transfer needs to be guaranteed in case of North-last turn model. The non-blocking between these ports can be attained because each packet can independently deroute itself when single-flit packet mechanism is applied. In precise, we give a limitation when packets cannot take prohibited turns as follows. Assume that an incoming packet is expected to take a path along a prohibited turn, but its output channel is locked by another packet. In this case, the packet cannot wait only for that output port to become vacant. In other words, the packet must be transferred to another output channel if it is vacant even when that direction is not towards to the destination. This is a similar idea to deflection routing [14] which requires all packets to be moving around the network constantly at a router at every clock cycle.

For example, let's say that the gray-colored node in Figure 4.1 receives a packet from the south input port, and applied turn model is North-last model. Since selection of west, east or south output port makes prohibited turn, the packet must choose north output port if the rest of the ports are locked (occupied) by others. Only in case when a north output port is locked, a packet which came

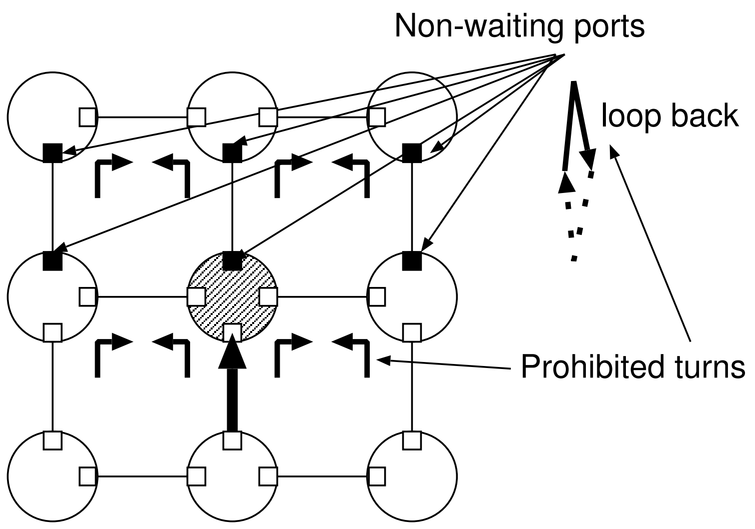


Figure 4.1: Prohibited turns and non-waiting ports

from a south input port waits for certain output port to become vacant. This is because unprohibited output ports behave as escape paths which prevents cyclic channel dependency. Idea of escape path was originally suggested in adaptive routing algorithms such as Duato's protocol [8] where they are implemented by using virtual channels.

Black-colored input ports in Figure 4.1 indicate that any available output port selection becomes a packet transfer along the prohibited turns. Thus, packets from these input ports must be transferred somewhere in a non-blocking manner.

Next, we define a term "Non-waiting port" as follows.

Definition 2 *(Non-waiting port) An input port and its adjacent input port connected with a link are both defined as "non-waiting ports" when either of the followings is satisfied: (a) when any available output port selection becomes a packet transfer along the prohibited turns, or (b) a port is paired with port defined in (a).*

Theorem 1 *There is one non-waiting port per router at most when network topology is 2-D mesh ($N \times N$), and applied turn model is North-last model. If the network size N is four or larger, it is also satisfied for West-first model.*

Proof *First, we consider the case of North-last routing. Packets which enter a router from the south input port are the only possible to take a prohibited turn (Figure 4.2 (a)). Consequently, these south input ports have only possibilities to be non-waiting ports according to Definition 2 (a). The non-waiting ports will be located in the first and second upper row routers in 2-D mesh topology, because for the rest of the routers, it is obvious that packets from south input ports may go straight upward to north output ports, so these routers do not have non-waiting ports. In the first row routers, only*

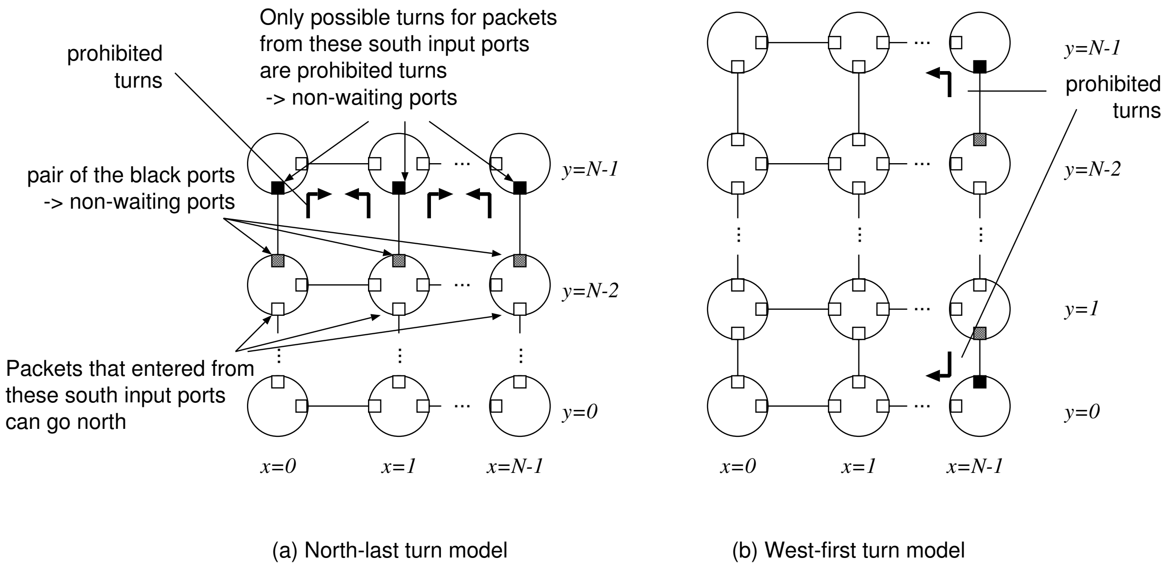

prohibited turns
Only possible turns for packets from these south input ports are prohibited turns -> non-waiting ports
pair of the black ports -> non-waiting ports
Packets that entered from these south input ports can go north
y=N-1
y=N-2
y=0
x=0
x=1
x=N-1
(a) North-last turn model
y=N-1
prohibited turns
y=N-2
y=1
y=0
x=0
x=1
x=N-1
(b) West-first turn model

Figure 4.2: Non-waiting ports for North-last and West-first turn models

south input ports satisfy Definition 2 (a), so Theorem 1 is satisfied. On the other hand, for Definition 2 (b), since there is only one input port that can be paired with (a), Theorem 1 is obviously satisfied.

For West-first routing, packets that go through routers at coordinate $x = N - 1$ can only take prohibited turn (Figure 4.2 (b)). Here, there is an output port which does not make prohibited turn either in x or y direction in case of $y \neq 0, y \neq N - 1$. Condition of Definition 2 (a) is satisfied either when $x = N - 1, y = 0$ or $x = N - 1, y = N - 1$ are satisfied. In the former case, condition is satisfied for a north input port, and a south input port for the latter case. On the other hand, for Definition 2 (b), since there is only one input port that can be paired with (a), satisfaction of Theorem 1 is obvious.

Definition 3 *(Loop-back transfer) When a packet transfers directly from a router's port to adjacent router's port which the packet was transferred from, it is called "loop-back transfer".*

Theorem 2 *When there is only one non-waiting port, a packet does not stay at non-waiting port after a certain period of time when following conditions are satisfied: (a) loop-back transfer is permitted, and (b) highest arbitration priority is given to the non-waiting port.*

Proof *When buffer of forwarding input port is vacant, a packet at a non-waiting port is transferred to one of vacant ports. According to Definition 2, a non-waiting port always exists on a router which a packet is transferred back to in a loop-back manner. If any input port of forwarding router is unavailable, a packet is transferred in a loop-back manner. Even when a packet exists in forwarding* *non-waiting input port, packets are simply exchanged between pairing non-waiting ports when highest arbitration priority is given to these ports. Since these packet transfer and exchange are done within a certain period of time, Theorem 2 is satisfied.*

Non-waiting ports in Figure 4.1 are marked with black color or diagonal lines. Livelock removal of non-waiting ports are described in the following section.

4.1.3 Router structure

This section describes the modification of router's output selection function and arbiters, which are necessary to support Semi-deflection routing mechanism.

4.1.3.1 Output selection function

Adaptive routing is characterized by determining adaptive routing algorithm and output selection function(OSF). Adaptive algorithm decides a set of available output ports, and OSF prioritizes output ports for assigning to packets.

Definition 4 *(Output Selection Function(OSF) of Semi-deflection routing) OSF of Semi-deflection routing prioritizes each port based on the following order:*

1. *Output towards the destination*
2. *Output which goes away from the destination, but does not loop back*
3. *Output which loops back*

Livelock between non-waiting ports is prevented by giving loop-back selection low priority.

4.1.3.2 Arbitration

An arbiter is in charge of allotting input packets to output ports. Priority of arbitration is described as follows. If none of the rules match, the order is based on the time stamp when packets entered the router.

Definition 5 *(Arbitration order of Semi-deflection routing)*

1. *Packets that were injected from non-waiting input port*
2. *Packets whose highest prioritized output port by the OSF makes prohibited turn*
3. *Other packets*

Livelock is prevented by arbitrating (1) with the highest priority. Also, non-blocking loop-back transfer is realized when other output ports are busy. By arbitrating (2) with high priority, non-blocking transfer to the prohibited direction is satisfied when the output port is available. If it is blocked, other vacant ports are allotted in prior to other input packets.

Finally, other packets which are not injected from non-waiting port nor wishes to take prohibited turn are arbitrated. If only available output port would keep packets away from destination nodes, packets can wait at the router input buffer until other ports to become vacant. This suppresses unprofitable increase of hop counts.

4.1.4 Semi-deflection routing mechanism

Based on above studies, routing which adopts the following rules is defined as Semi-deflection routing mechanism.

Definition 6 *(Semi-deflection routing mechanism)*

- *Prohibited turns and non-waiting ports are determined based on a turn model in the given network.*
- *Each router transfers packets based on priorities given by OSFs and arbiters in the previous subsections.*

Because turn model can be applied to arbitrary topology [83], application of Semi-deflection routing is not limited to a 2-D mesh topology.

Theorem 3 *Semi-deflection routing is deadlock-free*

Proof *(i) If a packet does not take a prohibited turn, it does not deadlock because no cyclic-dependency occurs. (ii) In case of taking prohibited turn, a packet can be transferred if buffers among forwarding input ports are vacant. (iii) Even when packets exist in all buffer of forwarding input ports, no deadlock occurs unless a packet transfer takes prohibited turn. Thus, buffers of forwarding input port will be vacant according to (i), and packet transfer will be possible.*

According to the above, only possible condition for causing a deadlock is when all packet transfer to forwarding input ports take prohibited turns, and packets also exist in their buffers. However, such input ports fulfilling the above conditions are non-waiting ports according to Definition 2, and a packet does not stay after a certain period of time according to Theorem 1 and Definition 5. Consequently, Semi-deflection routing is deadlock-free.

Table 4.1: Evaluation environments

RTL description language	Verilog-HDL
RTL verification tool	NC-Verilog 8.1
Synthesis tool	Synopsis Design Compiler 2007.12-SP3
Process library	Nangate 45nm library
Place and Route	SoC Encounter
Network simulation	irr_sim [84]
Application	NAS Parallel Benchmark [85]

Theorem 4 *Semi-deflection routing is livelock-free*

Proof *According to Definition 4 and Definition 5, an output port toward the destination is given higher priority. Also, Semi-deflection routing is a fully-adaptive non-minimal routing which assumes a single-flit packet structure. The above two conditions satisfy conditions for a livelock-freedom of a chaos router [16]*[1]*. Consequently, Semi-deflection routing is livelock-free.*

4.2 Evaluation

This section shows throughput evaluation results of Semi-deflection routing for 8x8 and 16x16 mesh topologies. Since adaptive routing allows various path selections, network performance is influenced by throttle mechanism of the router, which limits injection of new packets from host cores. Here we study the relationship between throughput and injection limitation.

4.2.1 Simulation conditions

The evaluation results are based on lightweight wormhole-based single-cycle on-chip router. Evaluation environments are shown in Table4.1. In addition, packet structure is single-flit, and router has 2-flit input channel buffer, 1-flit output channel buffer. Packet format assumes TRIPS OCN network, which is 140-bits wide [12]. It has 6-bit information for destination address, 3-bit of packet length, 3-bit for sequential identification value, and the rest 128-bit are payloads. The router architecture is single-pipelined, and does not have virtual channels.

The throughput was evaluated with 64 and 256 nodes in a 2-D mesh topology using irr_sim [84], a C++ flit-level network simulator. Here, latency is number of clock cycles between source core injects a packet in the network and destination core receives it. Accepted traffic is the average number of flits each core receives at a clock cycle in average. Maximum throughput is defined as maximum value of accepted traffic when latency is 1000 clock cycles or lower.

For evaluation, we used the following traffic patterns in addition to uniform traffic where destination is randomly selected.

[1]The livelock freedom arguments of the chaos router is given in Appendix A.

Table 4.2: Number of Non-waiting ports

Topology	# of Non-waiting ports	# of ports
k-ary mesh	$2k$	$4k(k-1)$

Table 4.3: Hardware structure of designed router

RTL file (module name)	description
inputc.v	Input channel (includes one “fifo” and one “rtcomp” modules)
fifo.v	Input buffer
rtcomp.v	Routing computation module of an input channel
cb.v	Crossbar switch (includes five “mux” and five “muxcont” modules)
mux.v	Multiplexer inside a crossbar
muxcont.v	Arbitration mechanism of a crossbar
outputc.v	Output channel
router.v	Top module including all the above components (consists of five “inputc”, five “outputc”, and “cb” modules)

- *matrix transpose*
  When array size is k, node (x, y) *transmits data to node* $(k-y-1, k-x-1)$*. Nodes on the diagonal axis (*$x+y=k-1$*) transmits data to node* $(k-x-1, k-y-1)$.

- *bit reversal traffic*
  When index values of source nodes are given as $(a_0, a_1, ..., a_{n-1})$*, each sends data to nodes* $(a_{n-1}, ..., a_1, a_0)$.

4.2.2 Number of non-waiting ports

Table 4.2 shows the number of non-waiting ports for each topology. The table only shows the number of switch-to-switch ports, because switch-to-host ports do not require non-blocking transfer.

As Table 4.2 shows, number of non-waiting port becomes relatively small as dimension number increases, and also as number of routers per dimension increases.

4.2.3 Hardware amount

RTL (Register Transfer Level) description of a router was written in Verilog-HDL, and the components of the router is given in Table 4.3. Table 4.4 shows the breakdown and total hardware amount of a router for 8×8 2-D mesh topology. The area is shown in a unit of “kilogates”, which is the

Table 4.4: Area of a router for 8 × 8 mesh network.

Component (module name)	Ecube	North-last	Semi-deflection	Number of modules included in a router
	[kilogates]			
fifo	2490			(1 for each inputc)
rtcomp	63	104	102	(1 for each inputc)
inputc	2868	2956	2957	5
outputc	203			5
cb	4667			1
total	21027	20994	21360	-

number of NAND gates divided by 1000. As Table 4.4 shows, The difference of router area is due to the size of "rtcomp" (routing computation) modules, comprising output selection function and arbitration mechanism. Because complexity of routing computation is higher for adaptive routers, their areas are slightly larger, but the differences are marginal. The area increase of Semi-deflection router is as small as 1.58% and 1.74% compared to dimensional order (Ecube) and North-last routers, respectively.

4.2.4 Throughput

4.2.4.1 Throughput using throttle mechanism

As shown in Figures 4.3 and 4.4, throughput of Semi-deflection routing decreases when traffic load exceeds certain value in any traffic pattern. Similar phenomena can be seen in fully-adaptive wormhole routing. In order to maintain network performance, we need injection limitation mechanism to keep traffic load constant.

Figures 4.3 and 4.4 show that network load value to maximize throughput depends on traffic patterns. Thus, better performance can be obtained by each local node independently controlling throttle mechanism. Several throttle mechanisms are proposed for wormhole routers [86] [87], which snoop the usage of virtual channels. However, since Semi-deflection routing does not assume the use of virtual channels, we need similar methodology that limits injection of packets.

First, local host of each router monitors the number of busy input ports and the request queue length of local host. Then we evaluate these values for each network load. Average value of busy input ports are shown in Figures 4.5 and 4.6. We find from the graphs that it is reasonable to limit packet injection when busy ports are between 2 and 3. In this evaluation, we set limitation value as 3 ports for 4×4 and 8×8 mesh, and 2 ports of 16×16 mesh topology.

It should be noticed that implementation of the throttle mechanism is simple, since each input port originally has busy signal output that indicates its utility, and no additional router hardware is required.

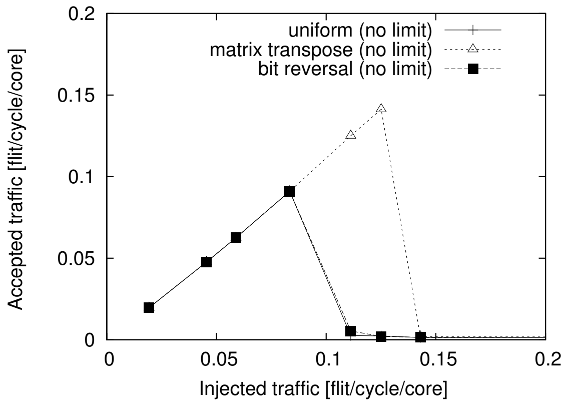


Figure 4.3: Throughput and traffic load without throttle mechanism (8x8 mesh)

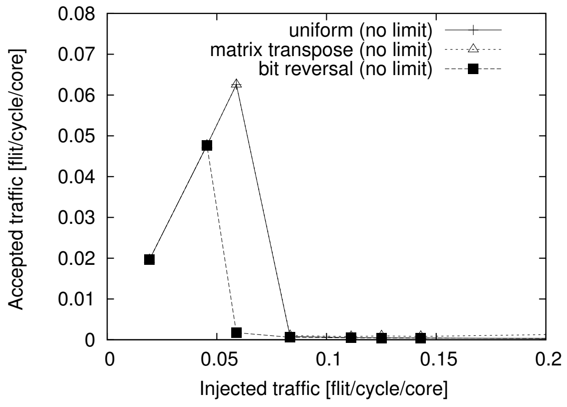


Figure 4.4: Throughput and traffic load without throttle mechanism (16x16 mesh)

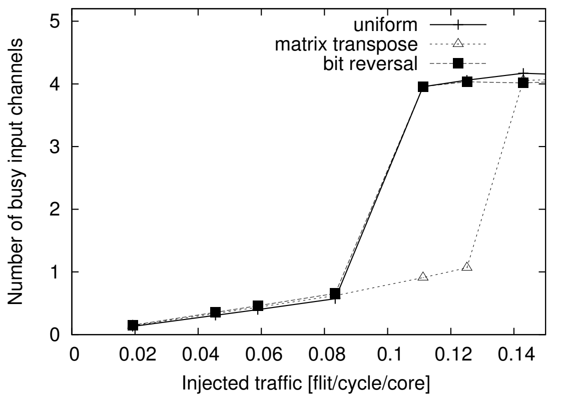


Figure 4.5: Number of used input port and traffic load without throttle mechanism (8x8 mesh)

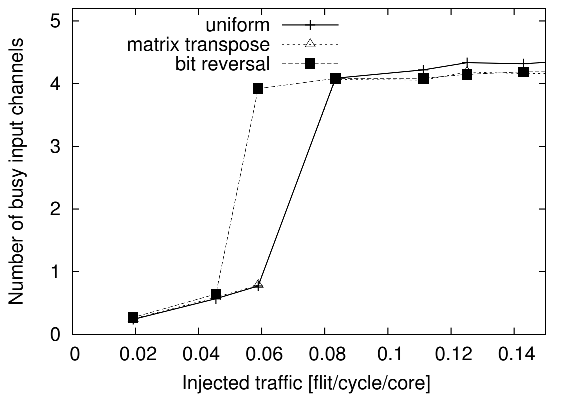


Figure 4.6: Number of used input port and traffic load without throttle mechanism (16x16 mesh)

4.2.4.2 Throughput of Semi-deflection routing and existing routing methodologies

Figure 4.7 gives an explanation to interpret throughput evaluation graphs shown through Figures 4.8 to 4.16. In these graphs, x-axes denotes the throughput or accepted traffic, whose units are average number of flits that could be injected in the network per cycle from each host core ([flit/cycle/core]). The y-axes represents the latency, that is the average number of clock cycles required for packets to be delivered to the destination. Obviously, smaller the y-axes values illustrated in such figures are, packets are delivered with lower latency, which is better. However, as traffic loads (x-axes values) increase, it would take longer time for packets to be delivered to the destination due to increase of waiting cycles a packet spends at one router until an input port of a next-hop router become vacant, and in case of adaptive routing, due to deflections, The y-axes values diverge as traffic load increase, and the maximum x values denote the maximum throughput. For example, if the performance of routing mechanisms (a) and (b) are depicted as Figure 4.7, (b) provides higher throughput because maximum x-axis value is larger.

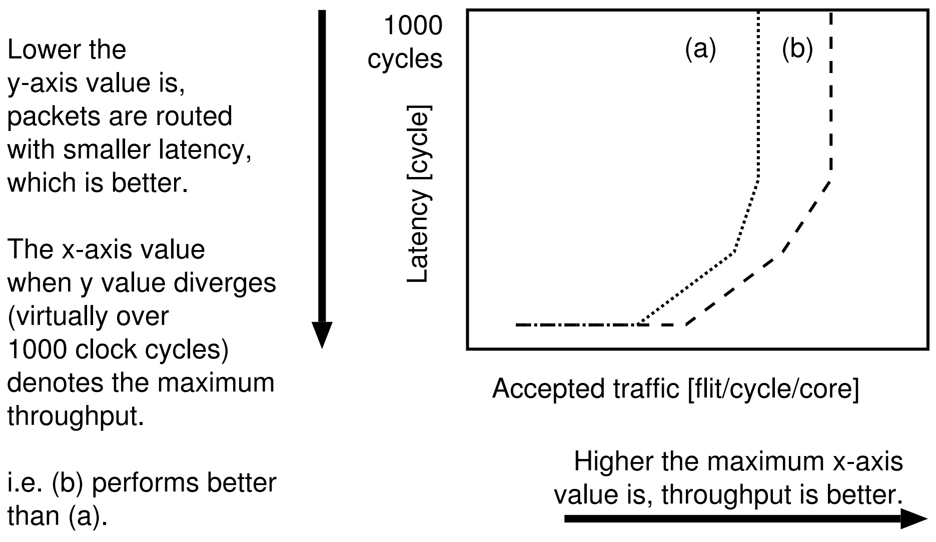

Lower the y-axis value is, packets are routed with smaller latency, which is better.
The x-axis value when y value diverges (virtually over 1000 clock cycles) denotes the maximum throughput.
i.e. (b) performs better than (a).
1000 cycles
Latency [cycle]
(a)
(b)
Accepted traffic [flit/cycle/core]
Higher the maximum x-axis value is, throughput is better.

Figure 4.7: Interpretation of throughput evaluation figures

Throughput of Semi-deflection routing is shown in Figure 4.8 to Figure 4.16. Each throughput was compared with Ecube routing and North-last adaptive routing. All routing assumes a single-flit packet structure, and routed with a single-cycle router.

As a result, throughput of Semi-deflection routing broadly outperforms dimensional order routing and North-last routing Figures 4.8 and 4.9 indicates that Semi-deflection routing does not provide impressive superiority for non-local traffic. Only exception that throughput become worse is 16×16 mesh in uniform traffic, as 4.10 indicates Contrary, through Figures 4.11 and 4.16, we can find that Semi-deflection routing has primacy over existing routing methodologies. Especially, Figure 4.15 indicates that Semi-deflection routing can provide maximum of 3.17 times higher throughput when traffic pattern and topology is bit reversal and 8×8 mesh, respectively.

4.2.5 Average hop count

Table 4.6 and 4.7 show average hop count of each traffic pattern by applying Semi-deflection routing to 8×8 and 16×16 mesh topology. Here, traffic load was set to the following.

- Low traffic: when packets are injected per 50 clock cycles to all routers
- Moderate traffic: when R is round-off value of average hop count in case of low traffic, packets are injected per R clock cycles to all routers
- Congested traffic: when packets were injected per one clock cycle to all routers

As Tables 4.5, 4.6 and 4.7 indicate, Semi-deflection routing effectively suppresses the increase of hop counts. Semi-deflection and North-last routing show contrasting characteristics as traffic load increases. The case of semi-deflection routing is intuitive; when packets have lower chances to take minimal path, they take detour route and average hop count increases. Contrary, hop count

Table 4.5: Average hop count (4x4 mesh)

Routing	Traffic pattern	Workload level low	moderate	congested
Ecube	Uniform	2.66	2.66	2.67
	Mtx. transpose	3.50	3.50	3.41
	Bit reversal	3.50	3.48	3.43
North -Last	Uniform	2.66	2.66	2.67
	Mtx. transpose	3.50	3.50	3.41
	Bit reversal	3.50	3.48	3.43
Semi -Deflec -tion	Uniform	2.69	2.77	2.84
	Mtx. transpose	3.62	3.85	3.88
	Bit reversal	3.62	3.95	3.85

decreases for North-last routing. This is due to difference in packet delivery capacity of both routing mechanisms. When traffic becomes closer to the maximum capacity of North-last routing, packet movement slows down and eventually terminates. Thus, hop count indicates the average value when traffic was below saturation. Since Semi-deflection routing provides escape paths, packet movement does not slow down, and this leads to the higher throughput as shown in the previous section.

4.2.5.1 Comparison of Semi-deflection routing and wormhole routing with variable-length packets

Let us discuss throughput for transferring variable-length packets with Semi-deflection routing and adaptive wormhole routing. We considered a case when message sizes were 4-flits and 8-flits.

- To route with Semi-deflection routing, a message was split into multiple single-flit packets, and routed with a router with 2-flit input buffers
- To route with wormhole routing, a multiple-flit message was treated as a single packet, and routed with a wormhole router

Wormhole router has 4-flit or 8-flit input buffers at each input port for 4- and 8-flit packets, respectively. Using a flit-level simulator synthetic traffic patterns stated previously. The results are shown in Figures 4.17 to 4.19. The unit of x-axis is set to [Byte/cycle/nodes]. Packet structure was also modified; for single-flit packet, 6-bit were given for destination address, 3-bit for packet length information, 3-bit for packet identification number, and the remaining were treated as payloads. By contrast, Packet structure for wormhole routing were set to the following: 2-bit were given as flit-type identification (header, body or tail), and header packet holds information of destination address and packet length.

Table 4.6: Average hop count (8x8 mesh)

Routing	Traffic pattern	Workload level low	moderate	congested
Ecube	Uniform	5.34	5.36	5.27
	Mtx. transpose	6.00	5.42	5.05
	Bit reversal	6.25	5.78	5.77
North -Last	Uniform	5.34	5.36	5.27
	Mtx. transpose	6.00	5.42	5.05
	Bit reversal	6.25	5.78	5.77
Semi -Deflec -tion	Uniform	5.37	5.51	5.69
	Mtx. transpose	6.41	6.53	7.29
	Bit reversal	6.34	6.35	7.24

As the result, maximum throughput of Semi-deflection for uniform traffic is approximately 75% of ecube and North-last routing in case of transferring 8-flit packets. Semi-deflection routing achieved 84% and 93% maximum throughput for matrix transpose and bit reversal traffic patterns with low latency.

4.2.5.2 Throughput evaluation using parallel applications

Finally, we evaluated throughput using CG (Conjugate Grid) and MG (MultiGrid kernel) from NAS Parallel Benchmark (NPB) traces [85] running on an eight-core CMP as shown in Figure 4.20. We executed NPB on GEMS [79] and Simics [78] simulator in order to capture the traffic trace that records the source and destination pairs of all packets. Using these traffic traces, we obtained throughput as shown in Figures 4.21 and 4.22.

CG benchmark is a type of sparse linear algebra, a vector computation with many gather and scatter operations. Data sets include many zero values. Data is usually stored in compressed matrices to reduce the storage and bandwidth requirements to access all of the nonzero values. One example is blocked compressed sparse row. Because of the compressed formats, data is generally accessed with indexed loads and stores.

On the other hand, MG benchmark is a type of structured grids. Represented by a regular grid, points on grid are updated together. The traffic has high spacial locality, and updates may be in place or between two versions of the grid. The grid may be subdivided into another grids in areas of interest, and the transition between granularity may happen dynamically.

According to Figures 4.21 and 4.22, Semi-deflection outperforms in CG benchmark with many gather/scatter operations. Consequently, we can expect better throughput with broadcast operations using Semi-deflection routing. Meanwhile, throughput performance is nearly equal with existing

Table 4.7: Average hop count (16x16 mesh)

	Traffic	Workload level		
Routing	pattern	low	moderate	congested
	Uniform	10.67	10.66	10.17
Ecube	Mtx. transpose	11.62	9.39	6.91
	Bit reversal	11.62	9.99	10.22
	Uniform	10.67	10.66	10.17
North	Mtx. transpose	11.62	9.39	6.91
-Last	Bit reversal	11.62	9.99	10.22
Semi	Uniform	10.69	12.01	11.45
-Deflec	Mtx. transpose	11.73	12.24	16.19
-tion	Bit reversal	11.67	11.77	13.28

routing algorithm for MG benchmark. This result supports our previous evaluation result that Semi-deflection routing is suitable for local traffic.

4.3 Chapter summary

In this chapter, we proposed Semi-deflection routing, a non-minimal fully-adaptive routing and makes the best use of a single-flit packet structure in order to provide higher flexibility for NoC on CMPs. The proposed routing strategy successfully distributed the congestion of synthetic local traffic, and the throughput improvement was 3.17 times in maximum in case of 8×8 2-D mesh topology compared to existing deterministic and adaptive routing. Also, it provided identical or better result with actual parallel benchmark traces which include cache coherency signals. Semi-deflection routing is also capable of transferring variable-length packets, and hardware increase was also marginal compared to single-cycle single-flit router with existing routing mechanisms. Further discussion will be given in Section 6.1.

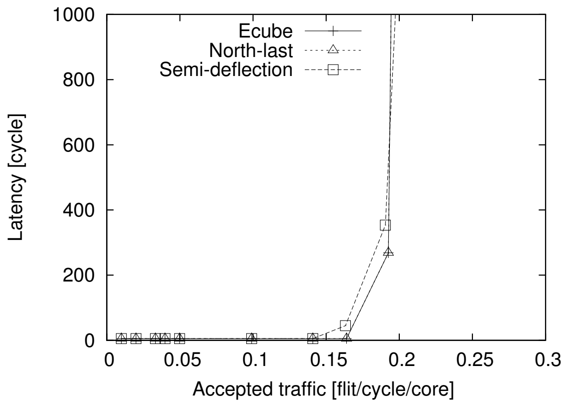


Figure 4.8: Uniform traffic (4x4 mesh)

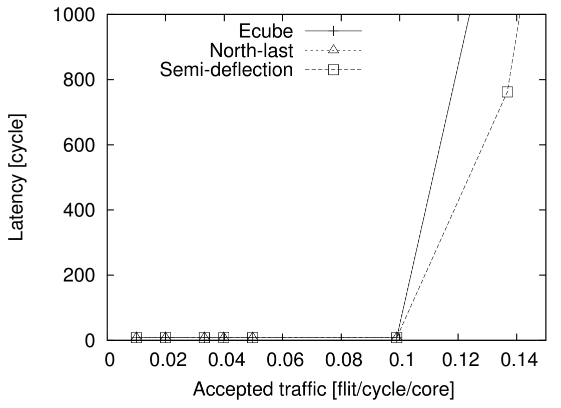


Figure 4.9: Uniform traffic (8x8 mesh)

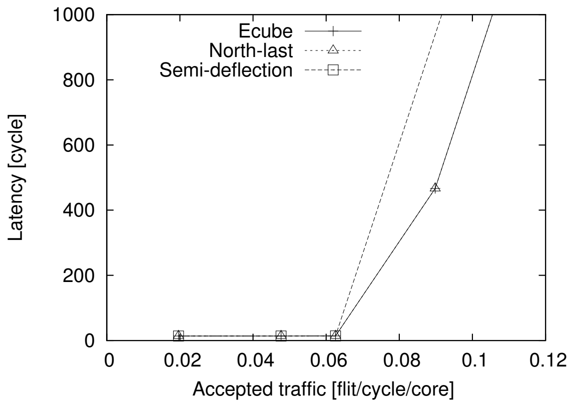


Figure 4.10: Uniform traffic (16x16 mesh)

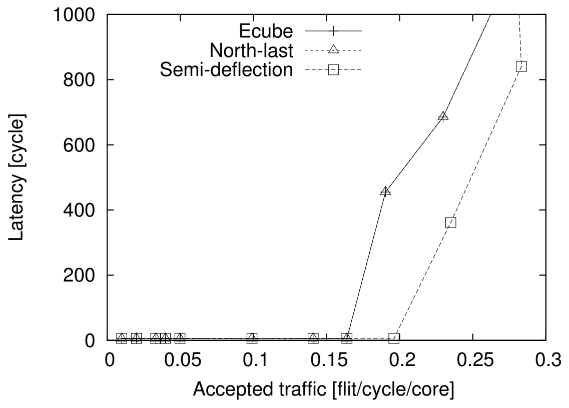


Figure 4.11: Matrix transpose (4x4 mesh)

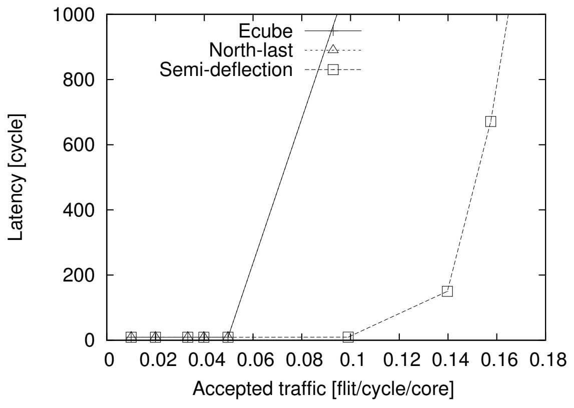


Figure 4.12: Matrix transpose (8x8 mesh)

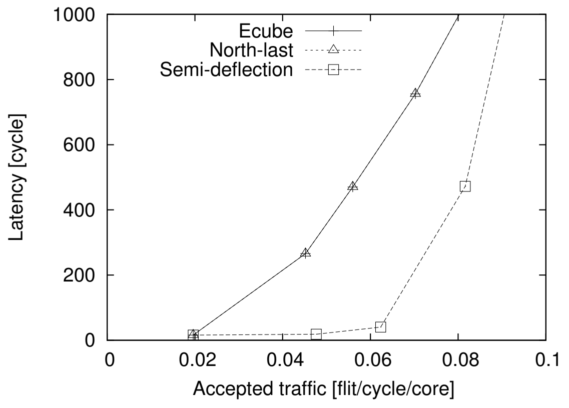


Figure 4.13: Matrix transpose (16x16 mesh)

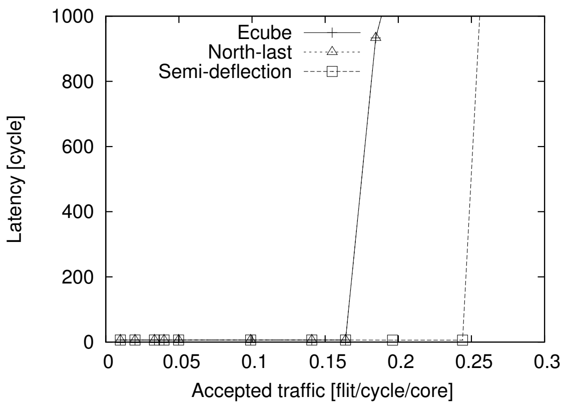


Figure 4.14: Bit reversal (4x4 mesh)

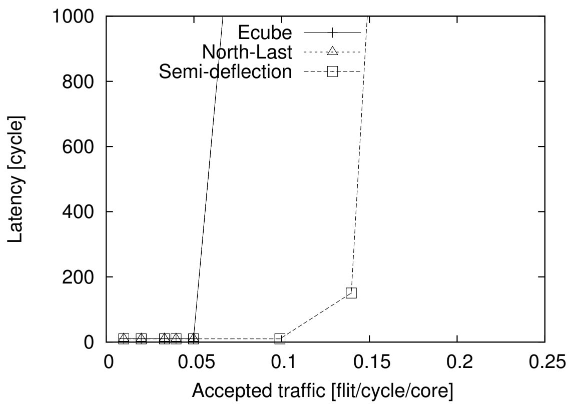


Figure 4.15: Bit reversal (8x8 mesh)

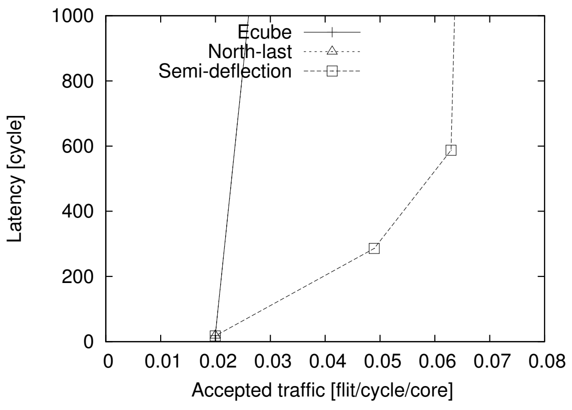


Figure 4.16: Bit reversal (16x16 mesh)

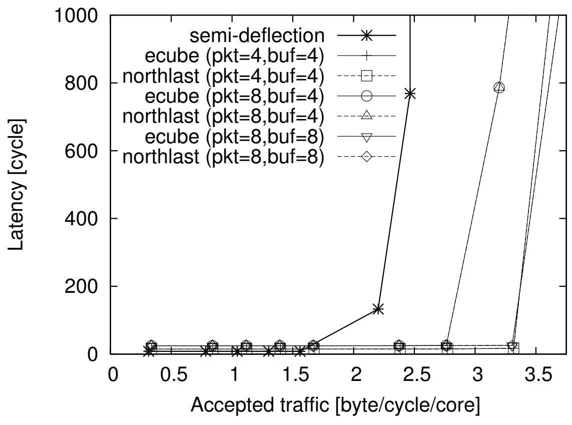


Figure 4.17: Uniform traffic (versus 4-flit and 8-flit WH)

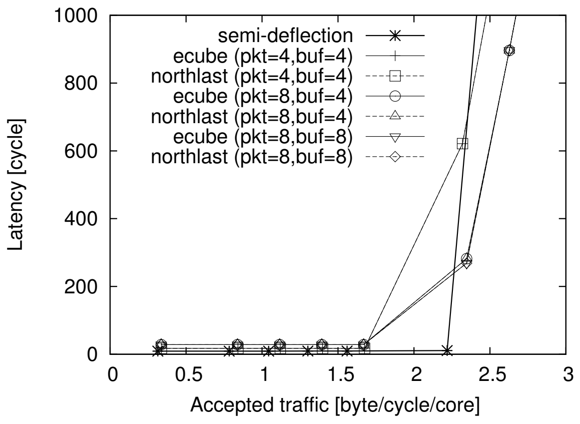


Figure 4.18: Matrix transpose (versus 4-flit and 8-flit WH)

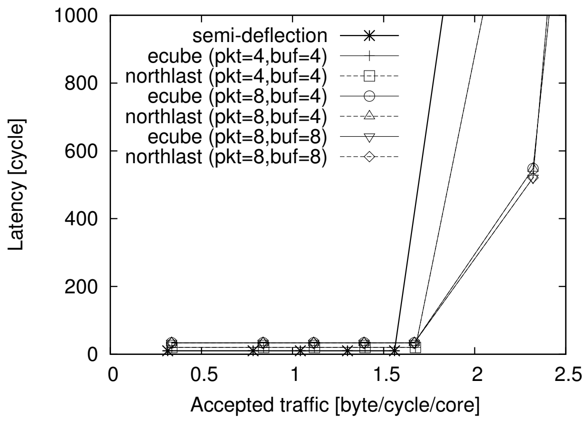


Figure 4.19: Bit reversal (versus 4-flit and 8-flit WH)

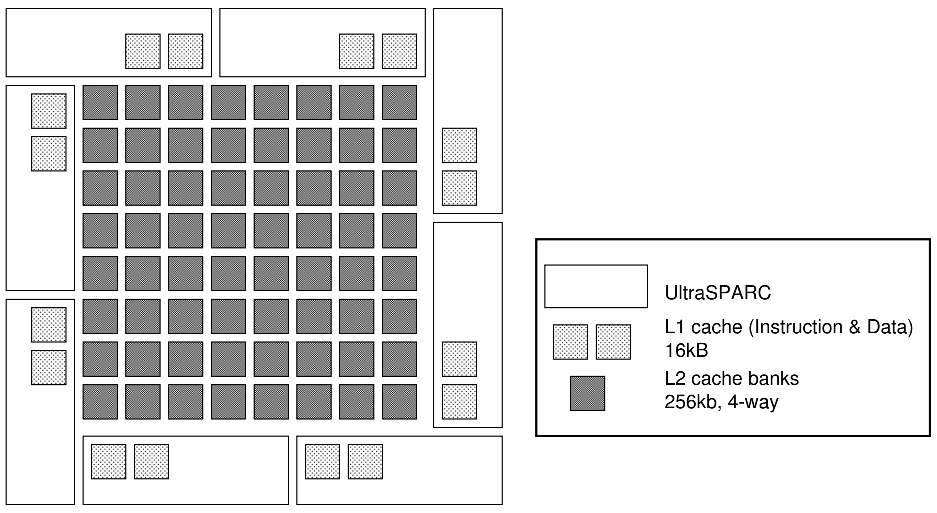

UltraSPARC
L1 cache (Instruction & Data) 16kB
L2 cache banks 256kb, 4-way

Figure 4.20: Target 8-core CMP architecture for throughput evaluation

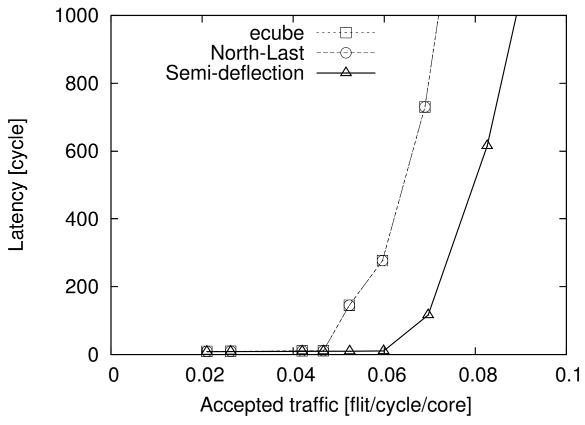


Figure 4.21: NPB CG (Class=W)

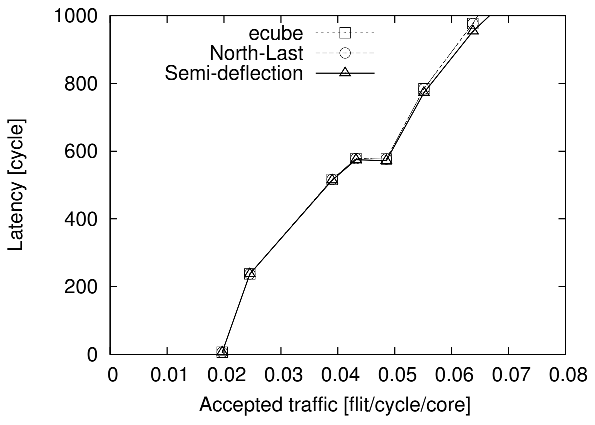


Figure 4.22: NPB MG (Class=W)

Chapter 5

Network analysis of a many-core SIMD accelerator

In this chapter, we will study network performance of one-dimensional many-core SIMD processors in order to understand the behavior of NoCs in an existing modern many-core SIMD system. Here we aim to precisely models the end-to-end NoC latency that includes sending and receiving overhead at each core by using a real SIMD machine. By using the model, we will show a large impact of the overhead on real applications which runs on a simple SIMD architecture whose NoC structure is shared bus and one-dimensional array. As we stated in the introduction, communication latency between components such as PEs and shared memories has large influence on execution time of programs, and accurate end-to-end latency estimation is crucial to maximize network performance, and make the best use of SIMD processors. We believe that accuracy of existing simulation studies and performance models for various topologies would increase by making their overheads in consideration, and this work can make the preface of it by showing one example of actual overhead values.

First, we introduce the target architecture in Section 5.1. Then Section 5.2 makes explatory evaluation of actual network performance. Based on this preliminary evaluation, Section 5.3 analyzes and proposes a network performance models, and the models are verified in Section 5.4. Section 5.5 summarizes this chapter.

5.1 Many-core SIMD processor with one-dimensional array topology

5.1.1 Target architecture of the study

A typical one-dimensional SIMD architecture is illustrated in Figure 5.1, a target architecture for introducing a communication latency model in this work. Multiple PEs are connected with an instruction bus, and a memory access bus which also connects an external memory interface that communicates with external large-scale shared memory. For higher computation flexibility, each PE may have direct access to that of adjacent PEs. Also, each PE has a small local memory, which can also

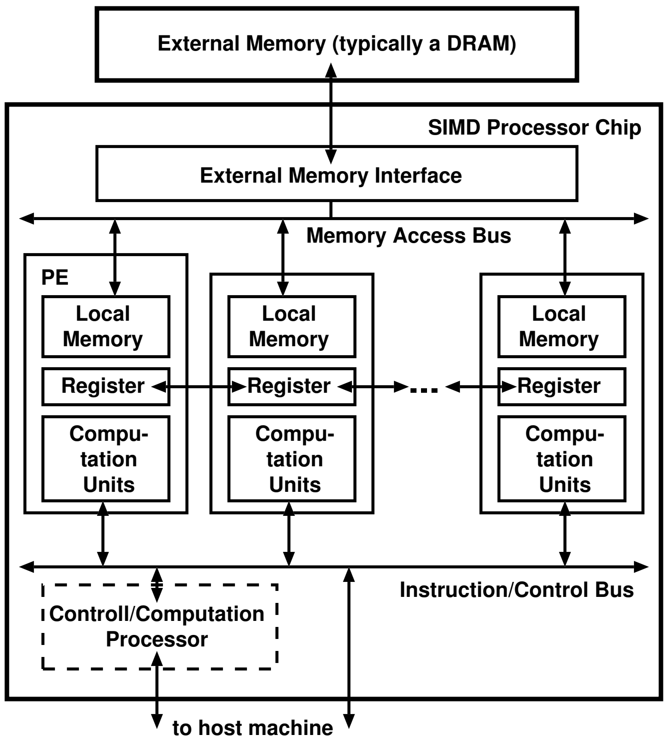


Figure 5.1: Block diagram of one-dimensional SIMD processor

communicate with external shared memory via memory access bus. The SIMD processor usually operates as an accelerator which computes computation-intensive processes offloaded from a host processor. Thus, it does not run operating systems, and is fully or partially controlled by the host processor. The processor may have control/computation processing element, which is in charge of computing sequential portion of a program and synchronization of data transfers between external memory and local memories of each PE. Instructions can also be issued from the control processing element or given from the host processor.

IMAP-VISION chip manufactured by NEC [67] (described in Section 3.3.3) and CSX600 by ClearSpeed fall into this category of architecture. In this work, we aim to model the network performance of the architecture by analyzing an actual CSX600 SIMD processor.

5.1.2 ClearSpeed's CSX600

5.1.2.1 Overview of CSX600

ClearSpeed's CSX600 is a simple SIMD accelerator with 96 PEs. It targets a variety of applications such as scientific calculations and DSPs. Since it provides standard libraries for typical applications such as BLAS, LAPACK, and AMBER with double precision, it is well known as a cutting-edge accelerator for high performance computing. It has been widely used as an accelerator for small scale personal supercomputing system to a very large-scale cluster TSUBAME [88].

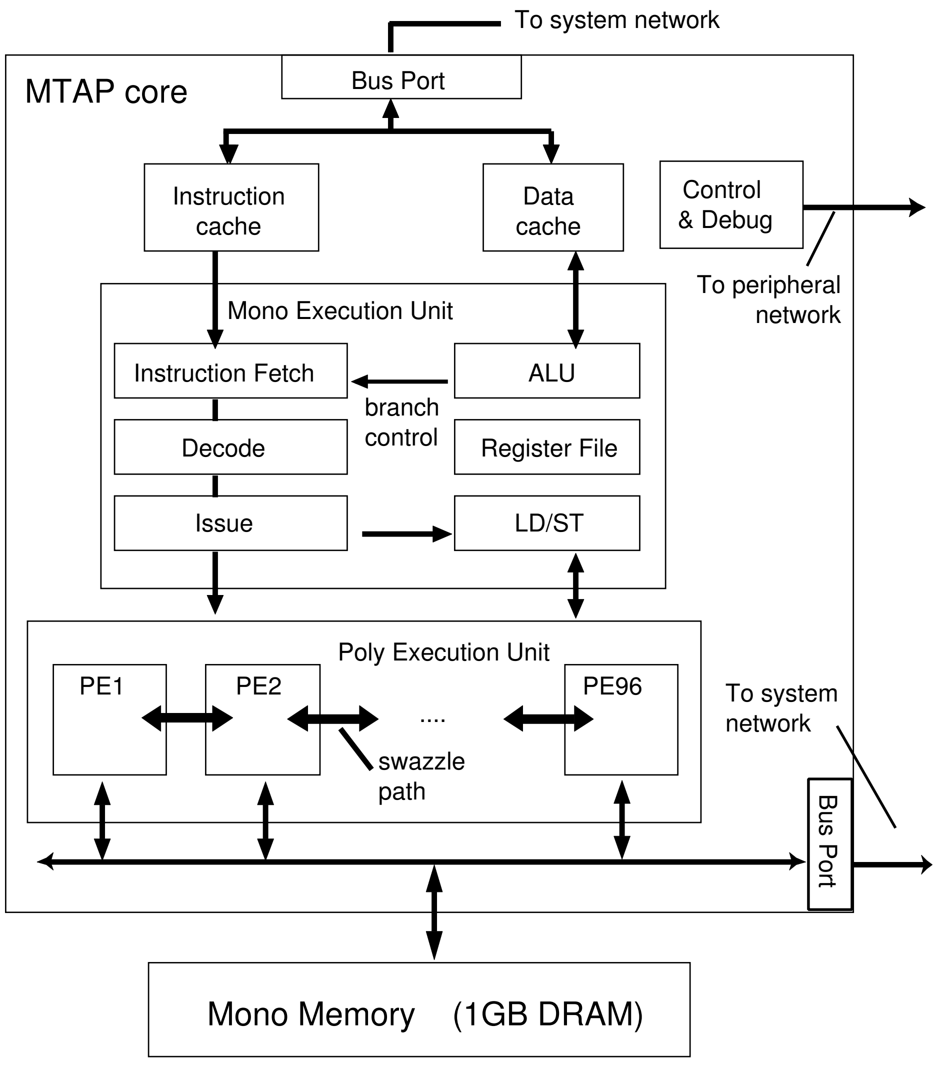


Figure 5.2: MTAP architecture of CSX600

Main unit of CSX600 is called Multi-Threaded Array Processor (MTAP), as shown in Figure 5.2. It comprises a Poly execution unit, Mono execution unit, instruction and data caches, and I/O ports. Poly execution unit has 96 PEs connected with a single dimensional array interconnection. On the other hand, Mono execution unit fetches and caches instructions from external memory, executes scalar instruction, provides flow control, and schedules multiple thread execution.

Figure 5.3 shows the structure of each PE with a single/double precision floating point unit(FPU), an integer ALU, a 128-byte register file, a 16-bit MAC with 64-bit accumulator, a 6 kilobyte SRAM local memory and a Programmed I/O port including 64-bit I/O buffers. Each PE is also connected with a 64-bit wide “Swazzle” interconnect. It provides high speed communication with adjacent PEs, because it is directly connected to a register file of each PE.

Besides MTAP, CSX600 comprises an internal SRAM, an external DRAM interface, and high-speed I/O ports that connect each PE. In particular, the external DRAM is called a “Mono” memory,

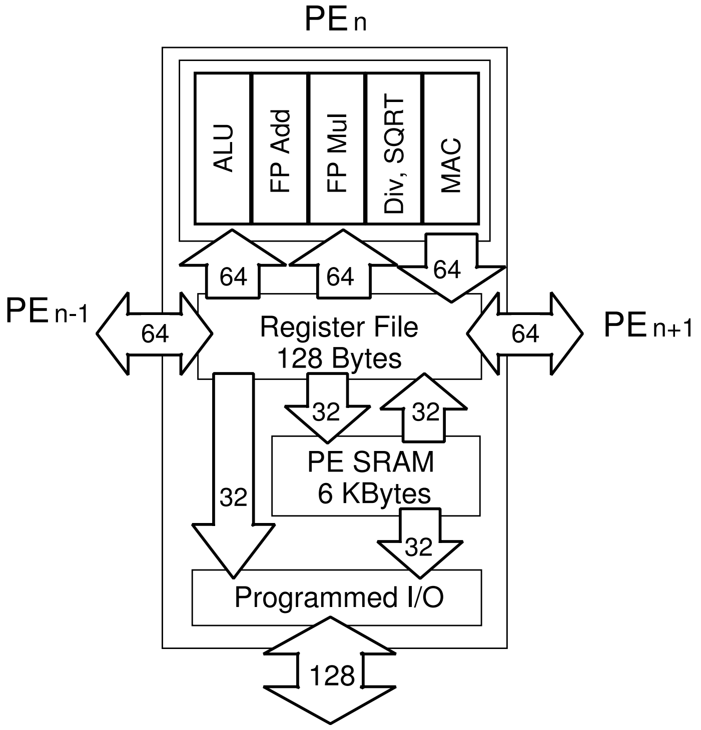


Figure 5.3: Structure of PEs (numbers inside arrows indicate the width of data paths)

and the internal SRAMs in each PEs are called “Poly” memories. The whole interconnect of these subsystems is called “ClearConnect”, but in this paper, the term “ClearConnect” specifies an interconnect between Mono and the Poly memories. Further specification of the memory architecture and ClearConnect bus are given in Appendix B

MTAP is capable of concurrently executing eight prioritized threads. This feature enables background transfer of data between Mono and Poly memories while computation is being conducted. An inter-PE transfer using Swazzle interconnect is regarded as computation, and cannot be overlapped with other computation. On the contrary, it can be concurrently conducted with data transfer between Mono and Poly memories.

Performance of ClearSpeed’s CSX600 is shown in Table 5.1 Its effective computation performance is 33GFLOPS which was measured by DGEMM [89], a benchmark that calculates matrix product of double-precision floating-point elements.

5.1.2.2 Data transfer procedures of ClearConnect

PEs can directly refer data in a shared memory. However, for an effective SIMD computation, a set of data such as array data needs to be transferred to each PE in advance. For these reasons, following I/O instructions are defined:

- `memcpym2p`: data transfer from Mono to Poly memory

Table 5.1: Specifications of CSX600

Effective performance (DGEMM)	33 GFLOPS
Maximum operating frequency	210 MHz
Average power consumption	10W
Internal memory bandwidth (Swazzle)	96 GByte/s
External memory bandwidth (ClearConnect)	3.2 GByte/s

- `memcpyp2m`: data transfer from Poly to Mono memory

In this paper, these procedures will be called “Mono-to-Poly” and “Poly-to-Mono” transfers. In both instructions, transfer data size can be set in unit of bytes. Although PEs that do not meet the conditions will be invalidated, the number of transfer clock cycles depend only on number of PEs that were activated at runtime. In Mono-to-Poly transfers, each PE can specify different address in Mono memory space, and the same holds for Poly-to-Mono transfer.

The instructions introduced above is called “synchronous” type, and they cannot be overlapped with computation. In addition to synchronous type, “asynchronous” communication instructions, `async_memcpym2p` and `async_memcpyp2m`, are supported to perform computation and communication simultaneously. Synchronization is conducted by programmers by semaphores.

Data transfer between Mono and Poly memories is processed as follows.

1. Initial process

   (1-1) PEs copy control information and data into their own I/O buffer

   (1-2) PEs set parameters such as data size

2. PEs serially transfer or receive data

3. Finishing process

   (3-1) PEs move data from I/O buffer to Poly memory in case of Mono to Poly transfer

   (3-2) PEs complete acknowledgment.

5.2 Exploratory evaluation of Swazzle interconnect and ClearConnect bus

In this section, we show the network performances of Swazzle interconnect and ClearConnect bus.

5.2.1 Evaluation endpoints and environments

For evaluating network performance of two interconnects in CSX600, we measured communication latency for transferring data by developing a measuring program where PEs insert packets at the highest injection rate. We used ClearSpeed's Software Development Kit version 2.51. The SDK includes a compiler, runtime, and a visual profiler.

Internal cycle counter of CSX600 was used to measure communication latency, which is the number of processor clock cycles for transferring certain amount of data. In case of CSX600, one clock cycle is an inverse of 210MHz, which is approximately 4.76 ns. Based on the obtained communication latency, we calculated the throughput by using the following equation.

$$Throughput[\text{GByte/s}] = 0.210[\text{GHz}] \times P \times M[\text{byte}]/Cycle \tag{5.1}$$

where M is data size, P is number of PEs, and $Cycle$ is communication latency cycles. Value "0.210" corresponds to the operational frequency of CSX600. All the throughput graphs shown in this section are obtained by the above equation. Based on the network performance study given in this section, we will study execution time breakouts of the performance measurement program by using the visual profiler, and formulate communication latency. The details will be described in Section 5.3.

5.2.2 Swazzle interconnect

Swazzle is a 64-bit wide interconnect between register files of adjacent PEs, and is capable of shifting or exchanging 2, 4, 8, and 16 bytes data. The official announcement shows that the peak throughput (bandwidth) of Swazzle interconnect is 96 GByte/s, as shown in Table 5.1. Since Swazzle interconnect is a linear topology, bandwidth of each link is calculated as 1 GByte/s.

5.2.2.1 Neighboring traffic pattern

We firstly measured throughput when each PE communicates with the neighboring PE. The SIMD processing sometimes generates a large number of the neighboring communications. Indeed, the instruction set on CSX600 includes operations that exchange or shift data between the neighboring PEs.

Figure 5.4 shows the throughput of Swazzle for sending 2, 4, 8, and 16-byte data. The figure indicates that 16-byte transfer uses the link bandwidth efficiently, and Swazzle interconnect treats the small-sized data transfer well. Also, total throughput of Swazzle interconnect is 107.38 GByte/s, and we found that it is possible to accomplish throughput higher than 96 GByte/s taken from specifications of CSX600 in actual parallel programing.

5.2.2.2 Overlapped traffic pattern

To encourage flexibility of Swazzle interconnect, ClearSpeed's instruction set provides two types of particular instructions which send data to nonadjacent PEs. The former instruction supports data

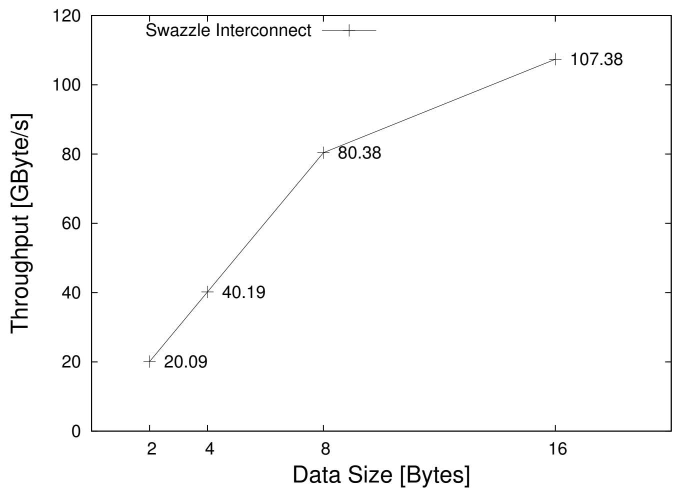


Figure 5.4: Total throughput of Swazzle interconnect

Table 5.2: Throughput of Swazzle Interconnect

Distance	1 hop	2 hops	4 hops
Total on-chip throughput (GByte/s)	107.4	64.5	35.8

transfer from PE_n to $PE_{n\pm2}$, and the latter would send data to $PE_{n\pm4}$. These instructions are exclusively supported for high speed transfer of 16-byte data packet.

To understand the impact of these instructions on performance, we used a synthetic traffic where each PE sends packet to alternate PEs at the highest injection rate for the measurement, as shown in Figure 5.5.

Table 5.2 shows the result of throughput in all traffic patterns. Obviously, each data packet is sent by shifting it to adjacent PEs, but the data are soon overwritten by the next shift. All the traffic patterns except the neighboring communication introduce per-hop latency, including link delay and overhead of register-to-register transfer. Thus, as the number of packet hops increased, the throughput was reduced.

5.2.3 Network performance of ClearConnect

In this section, we will measure performance of ClearConnect on CSX600, and we illustrate several conditions to accomplish high throughput between Mono and Poly memories, or to possibly regulate performance.

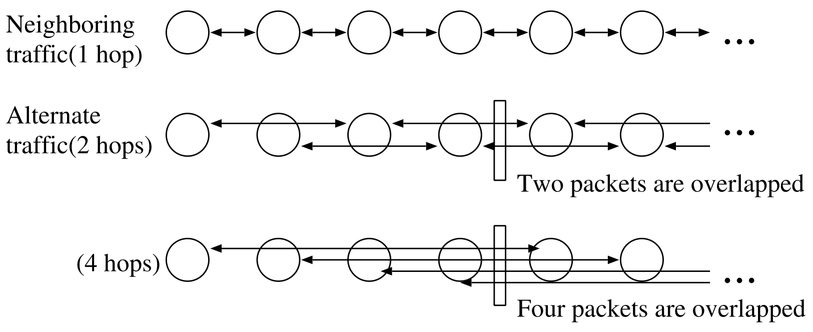

Neighboring traffic(1 hop)
Alternate traffic(2 hops)
Two packets are overlapped
(4 hops)
Four packets are overlapped

Figure 5.5: Traffic Patterns

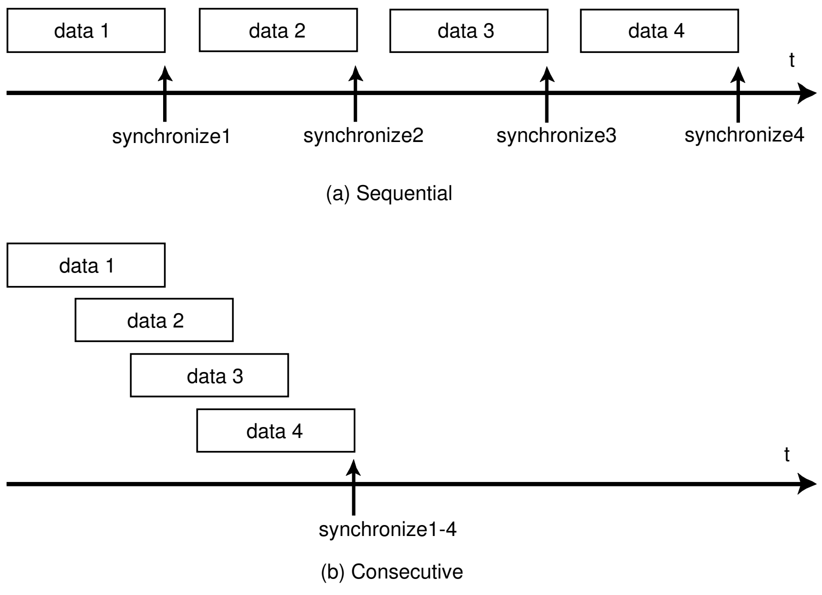


Figure 5.6: Timing of synchronization

5.2.3.1 Measurement conditions

To investigate the relationship between number of activated PEs and throughput, we implemented a test program that performs asynchronous memory copy, and measured throughput between Mono memory and Poly memories with following parameters:

- Number of PEs: 16, 32, or 96
  CSX600 allows users to preliminarily modify the number of PEs that involves in program execution by editing a configuration file of CSX600 (We will refer to such PEs as term "activated PEs" in this paper.). Number of activated PEs is unmodifiable once CSX600 begins to execute a target program.
- Packet size: Between 4 and 4,096 bytes
  We modified the ratio of stack and heap region of Poly memory to 1:2 during dynamic com-

pilation, and allocated 4,096 bytes to heap region so that PEs can receive large data packets (Normally, the ratio is 1:1 for total of 6-kilobyte memory.).

- Data alignments: Aligned, 4 bytes misaligned, 8 bytes misaligned, or 12 bytes misaligned
  Ideally, alignment needs to be 16 bytes because the bus is 16 bytes wide, and there is a time penalty when this is not satisfied.

- Timing of synchronization: After every 16 data transfers
  Synchronization can be performed in a manner shown in Figure 5.6(a) and (b). In Figure 5.6(a), data are synchronized after each communication instruction, while in Figure 5.6(b), communication instructions are issued contiguously until they are synchronized at once. As the result of exploratory evaluation, the throughput saturates when data is synchronized at every 16 data transfers, so in this evaluation, Figure 5.6(b) manner is adopted; data is synchronized after 16 communication instructions were issued.

5.2.3.2 Network performance measurement of aligned data transfer

Figures 5.7 and 5.8 show total throughput of Mono-to-Poly and Poly-to-Mono transfers, respectively. In each graph, we illustrated throughput of the following cases separately.

- data size is the multiple of 16 bytes
- remainder of data size divided by 16 is 12 bytes

Especially, the latter condition makes least use of ClearConnect bandwidth, due to reasons explained in Subsection 5.3.3.1. We can find from these figures that ClearConnect can handle arbitrary size of transfer data at a sacrifice of throughput.

5.2.3.3 Network performance measurement of misaligned data transfer

Figures 5.9 and 5.10 show throughput when the address of header data in the Mono memory is shifted by 4 bytes, 8, or 12 bytes, which causes memory access misalignment. The figures show that ClearConnect can handle misaligned access, but the misaligned case degrades performance by 17.4% and 17.9% on average compared with that of the optimized-aligned data with Mono-to-Poly and Poly-to-Mono transfer, respectively.

5.3 Analysis and performance modeling of Swazzle interconnect and ClearConnect bus

In this section, we will analyze performance factors of ClearConnect and Swazzle interconnect using profile data of CSX600, and formulate a communication latency model.

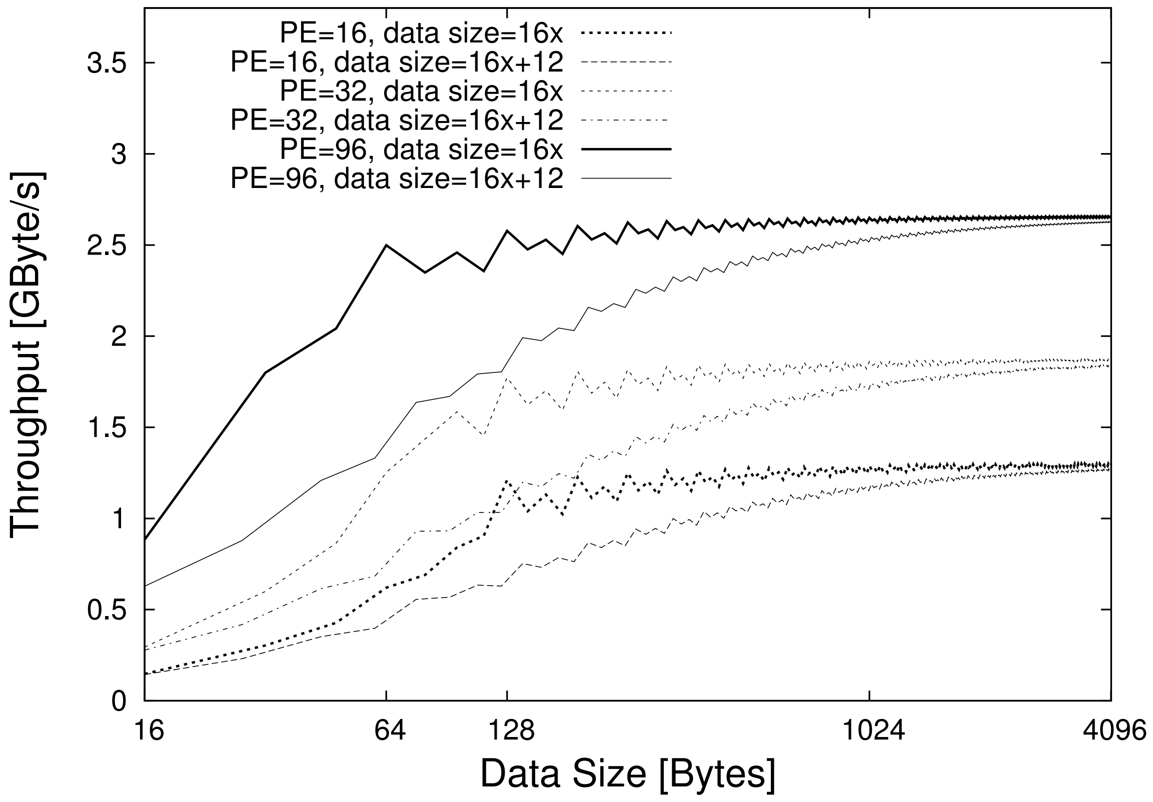


Figure 5.7: Mono-to-Poly throughput

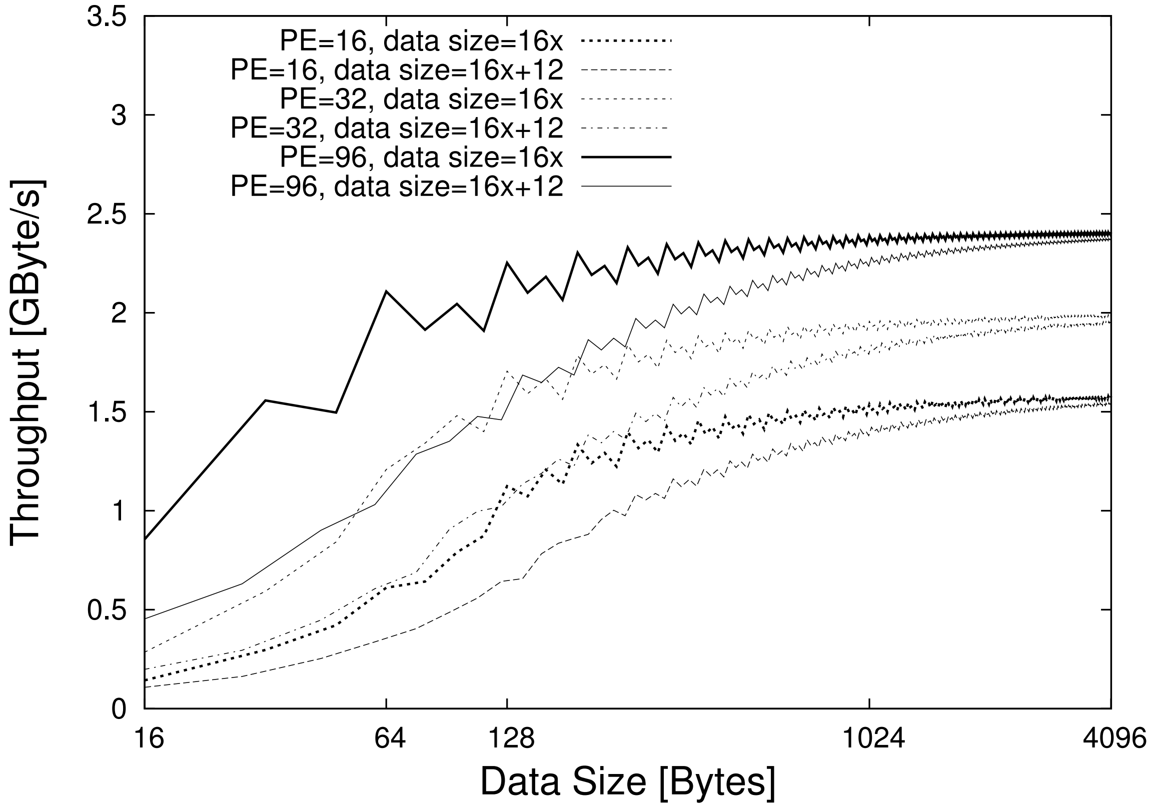


Figure 5.8: Poly-to-Mono throughput

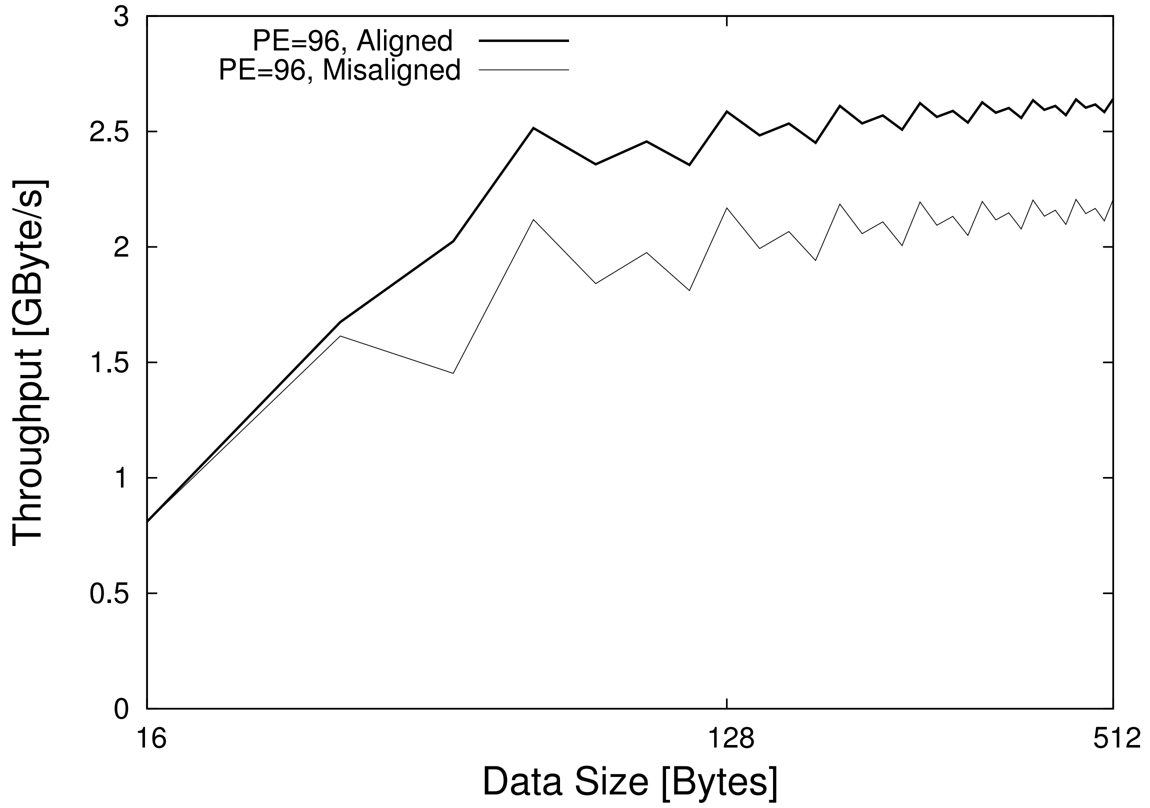


Figure 5.9: Mono-to-Poly throughput with misalignment

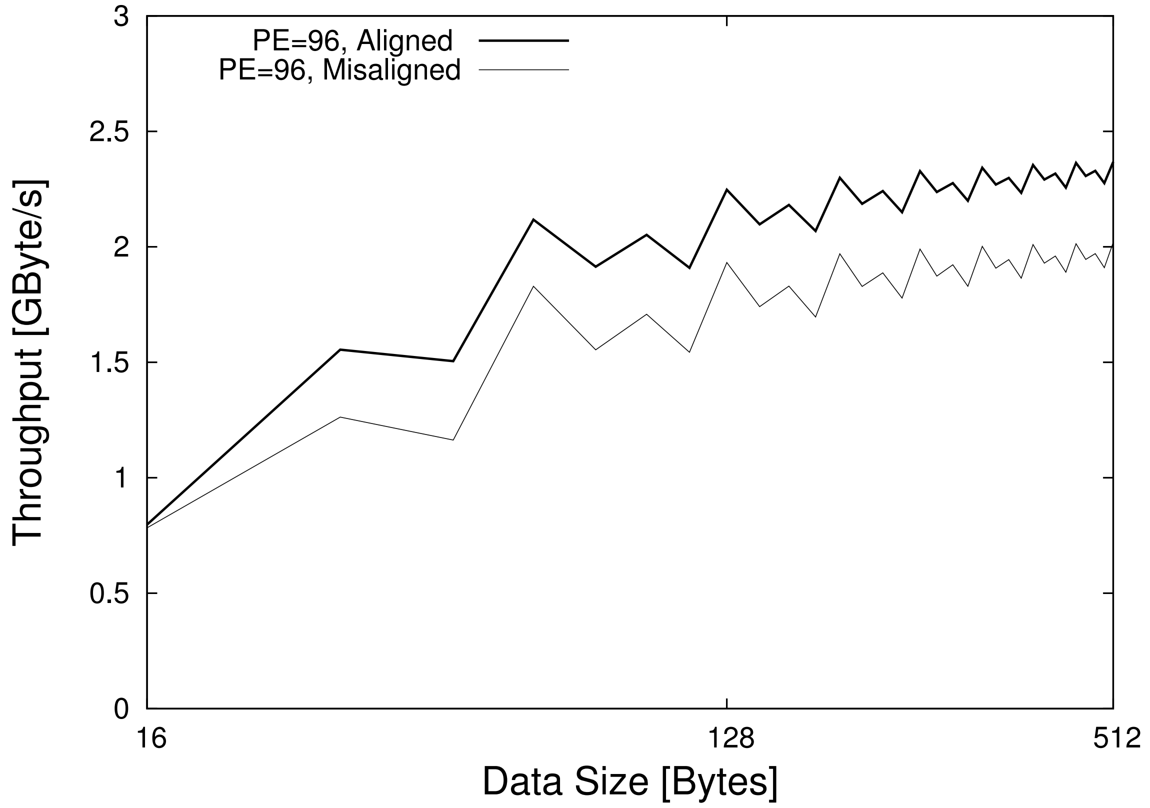


Figure 5.10: Poly-to-Mono throughput with misalignment

Table 5.3: Data size and number of cycles for Swazzle interconnect

Traffic Type	Hops	Data size	Cycles
		2	2
Neighboring		4	2
(Shift/Swap)	1	8	2
		16	3
Alternate	2	16	5
(Shift only)	4	16	9

5.3.1 Analysis and modeling policy

Firstly we studied the breakouts of communication latency, and distinguished actual data transfer time and communication overhead by using ClearSpeed's visual profiler. We analyzed how data transfer time or overheads are affected by the number of PEs, direction of data transfer (Mono-to-Poly or Poly-to- Mono), data size and misalignment. Secondly, based on these observations, we formulate its analytical model. Finally, we used the model to obtain communication latency for various size of transfer data, calculated throughput using Equation (5.1), and compared with the measured throughput in order to confirm the accuracy of the model. It should be noted that a statistical study of overhead and latency values based on all data packets transmitted to every nodes is beyond this work.

5.3.2 Analysis of Swazzle interconnect

Values shown in Table 5.3 are average number of cycles required for shifting or exchanging 2, 4, 8, and 16-byte data to PE's own or adjacent register, which were obtained as the result of profile analysis. As PE registers are byte addressed, best performance can be obtained as long as data are accessed in the unit of bytes. Thus, for Swazzle interconnect, values shown in Table 5.3 simply determine the internal network throughput which can be calculated using Equation 5.1. $P = 96$ is assumed in this case.

The synchronous operations of PEs enable the packet to be sent with non-blocking once the data transfer starts, since the overhead of packet switching is regarded as the register read/write operations. It is similar to the circuit switching whose setup overhead is usually high, but Swazzle interconnect supports the low setup-overhead by exploiting the simplicity of the one-dimensional array topology, which is the lowest in terms of degree.

Consequently, this simplicity allows us to make the precise analytical model to estimate the delay and throughput of Swazzle interconnect. Indeed, the performance gap between the measurements in Figure 5.4 and estimated values using Equation (5.1) were up to 0.3%.

5.3.3 Analysis of ClearConnect

5.3.3.1 Transfer data format

First, we studied the profile of data transfer time between 4 to 4,096 bytes data transfer. As the result of analyzing the case of 4,096-byte transfer, we found that the data is broken down to 64 data blocks whose size is 64 bytes. In other words, the maximum data block size is 64 bytes, and bandwidth is effectively used when total data size is the multiple of 64 bytes. When data size is not the multiple of 64 bytes, we found that data are broken down to several data sub-blocks according to Equation (5.2).

$$\begin{aligned} B_{64} &= \lfloor D[\text{bytes}]/64 \rfloor \\ B_{32} &= \lfloor (D - 64 \times B_{64})/32 \rfloor \\ B_{16} &= \lceil (D - 64 \times B_{64} - 32 \times B_{32})/16 \rceil \end{aligned} \tag{5.2}$$

Here, B_{64}, B_{32} and B_{16} represent quantity of 64-byte block, 32-byte block and 16-byte block, respectively. D indicates total transfer data size. For example, when D is 190 bytes, quantity of each block would be $B_{64} = 2$, $B_{32} = 1$ and $B_{16} = 2$, coming to five blocks in total. On the other hand, 192-byte data packets would be divided into three 64-byte blocks. This provides an explanation why network performance was jagged in Figures 5.7 and 5.8.

5.3.3.2 Analytical latency model

Based on the profile analysis, end-to-end latency model of ClearConnect is introduced here. To concrete the delay estimation, we extracted profile data from data transfer programs run on CSX600 for sending 128 packets. On the other hand, data transfer phases have been already given in Section 5.1.2.2, that is, it is consisting of initial process, serial data transfer process, and final processes. In addition, through profile analysis, we found that some dead cycles can occur for some cases, which are referred as "additional overhead" here. Based on the observations and assumptions above, the end-to-end latency can be expressed as follows.

$$\begin{aligned} Latency &= Sending\ Overhead \\ &+ Transport\ Latency \\ &+ Receiving\ Overhead \\ &+ Additional\ Overhead \end{aligned} \tag{5.3}$$

Next, since it is given that packets are sent serially in data transfer process, we could make an assumption that sending and receiving overheads are independent from the number of PEs, and only transport latency is effected by it. This assumption and observation of communication latency breakouts yielded the following equation, which determines end-to-end communication latency.

$$Cycle_{(D,P,M)} = S_M + T_M(P-1) + R_M + LS_M + D \tag{5.4}$$

Table 5.4: Data transfer latency between Mono and Poly memories

Direction		Overhead	Under 16byte	32byte	64byte
Mono to Poly	Default values	Sending	14	14	14
		Transport/PE	1	2	4
		Receiving	28	29	33
		Load/Store	58	58	58
	Additional overhead	Tail packet	2	2	2
		Misalignment	130	130	130
Poly to Mono	Default values	Sending	20	23	26
		Transport/PE	2	3	5
		Receiving	42	42	43
		Load/Store	37	37	37
	Additional overhead	Head packet	0	0	46
		Tail packet	2	2	2
		Misalignment	93	93	93

Unit: cycles

$Cycle_{(Dir,P,M)}$ is the number of cycle required for transferring the split M-byte data. P is the number of PEs involved in data transfer, and Dir determines whether data transfer is Mono-to-Poly or Poly-to-Mono S_M, T_M, R_M and LS_M are sending overheads, transport latency for sending data to each PE, receiving overheads and load/store overheads when the data size is M byte, respectively. D is the constant value caused by data misalignment or when the packet is the header or tailor.

Next, we analyzed concrete overhead and latency values from profile data. As mentioned in Section 5.3.3.1, data is split into 64-byte, 32-byte and 16-byte data blocks. Based on this knowledge, we obtained the average number of each overhead and transport latencies for each data block sizes, and regarded these values as S_M, T_M, R_M, LS_M and D to be assigned to Equation (5.4). The values are shown in Table 5.4.

As Table 5.4 shows, the latency that depends on the number of activated PEs is a few cycles, whereas the other latency is the order of hundred cycles. Moreover, the additional overhead caused by misalignment is also the order of hundred cycles. Thus, large proportion of communication latency is dominated by constant latency regardless of the number of PEs.

It should be noted that Poly-to-Mono transfer includes additional overhead incurred by the head packet, which is due to an initial process for sending the first 64-byte block. Afterward, trailing 64-byte blocks are sent in a burst manner without any sending overheads. Meanwhile, trailing packets of Mono-to-Poly transfer are not sent in a burst manner. It is assumed that this is due to performance

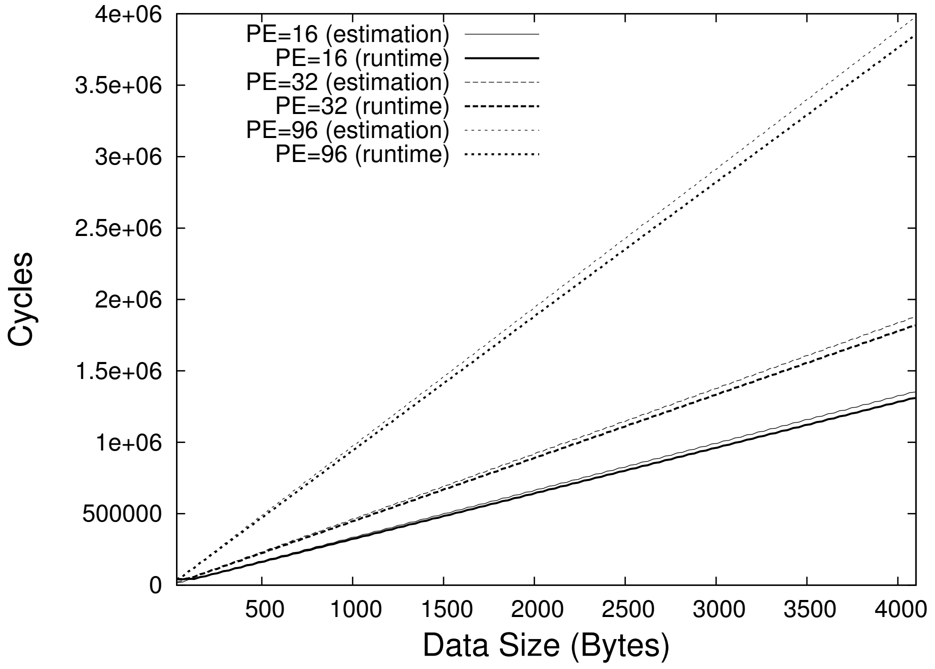


Figure 5.11: Measured execution cycle and estimated cycles for Mono-to-Poly transfer

gap of burst accesses between a DRAM (Mono memory) and an SRAM (Poly memory). In general, SRAM outperforms in treating burst read/write accesses to both sequential and random addresses compared to DRAM, which we guess is a cause of head packet overhead for Mono-to-Poly transfer. Consequently, link utilization rate of ClearConnect becomes higher for Poly-to-Mono transfer, as Figures 5.7 and 5.8 indicate.

5.3.3.3 Throughput estimation

Throughput of ClearConnect can be estimated by using Equation 5.1 and values shown in Table 5.4. Figures 5.11 and 5.12 show measured execution cycles and estimated cycles for transferring 128 data packets whose data size is shown in x-axis. The measurement was done for P =16, 32, and 96, and transferred data size were between M = 32 and 4,096 at 32-byte intervals.

As the result of comparing actual and estimated execution cycle in case of P = 96, we found that our latency model estimated execution cycles with 3.27% and 3.39% of error on average for Mono-to-Poly and Poly-to-Mono transfer, respectively. By analysis using the graphic profiler, we found that error was mainly caused by instruction cache miss-hit.

Through the analysis and modeling, we found that the number of used PEs, the size of transferred data, and alignment of Mono memory are the main factors to make the best use of bandwidth of ClearConnect and Swazzle interconnect in CSX600.

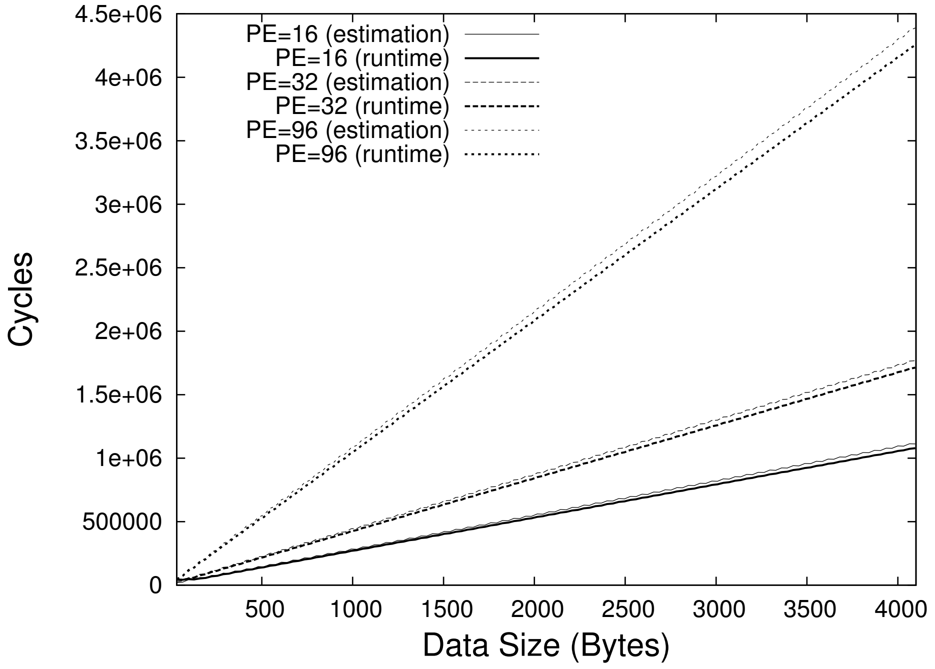


Figure 5.12: Measured execution cycle and estimated cycles for Poly-to-Mono transfer

5.4 Model verification using parallel benchmarks

This section illustrates the estimated best-case and worst-case latency of ClearConnect in the traffic patterns of communication-constraint parallel applications based on latency model given as Equation (5.4). Validity of the model was confirmed by actually executing all communication instruction in the programs, and checked whether the execution time fits in between best and worst-case latencies.

In general, the worst-case latency is difficult to be estimated in most NoCs. However, as we will show in Section 6.1.2.2, bandwidth of ClearConnect is not saturated even when all PEs communicate with the shared memory with the highest access rate. This is because the performance bottleneck is due to sending and receiving overhead rather than the sequential access in ClearConnect, and this allows us to theoretically compute the worst-case latency. Assume that the number of PEs that were activated is set to P, and maximum transfer unit is set to M bytes in this section.

5.4.1 Overview and implementation of benchmarks

From the results of the analysis in the previous section, we estimated data transfer time based on extracted traffic patterns of communication-constrained parallel applications. Since each PE only has 6 kilobytes of local memory, communication latency is usually crucial to achieve high performance in parallel applications using CSX600.

We selected two communication-bound benchmarks suitable for SIMD execution.

- Himeno Benchmark [90]

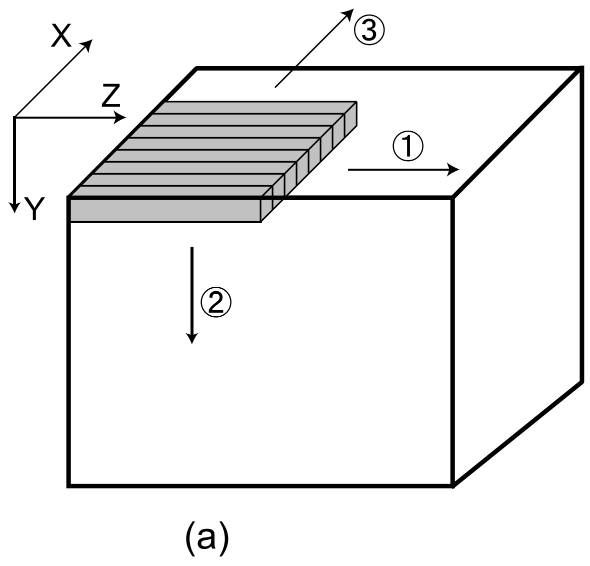


Figure 5.13: Data partitioning and transfer sequence

- "MG" of NAS Parallel Benchmark [91]:

Himeno Benchmark and MG of NPB both deal with three-dimensional matrix. Since memory capacity of Poly memory is small, large-scale data need to be stored into Mono memory, and transferred to each PE when it is needed.

It is easy to parallelize and apply data transfer instructions when problem size is multiple of number of PEs, and when data size of partitioned data is small enough to fit into Poly memory. However, when neither of the above is satisfied, it is important to decide data partitioning and data transfer procedure policies. In this work, we applied the following sequence, as shown in Figure 5.13.

1. Divide each row of $x - z$ plane in parallel with z-axis, and allocate each data row to each PEs
2. When size of a data row is larger than Poly memory capacity, the residual is fetched in the next round of computation.
3. Data fetch, computation and write proceed in y-axis direction.
4. When total number of data row is larger than the number of PEs, go back to 1.

Adopting the policies above, we implemented the following two benchmarks.

5.4.1.1 Himeno benchmark

Himeno Benchmark is an application widely used to evaluate performance of high-performance computing systems. It measures computation speed of major processes in Jacobi iterations for solving

Table 5.5: Problem size of target benchmarks

Benchmark	Class	Matrix Size	Precision
Himeno	Size L	$512 \times 256 \times 256$	Single
NPB-MG	Class A	$256 \times 256 \times 256$	Double

Poisson Equations. It differs from other parallel benchmarks by requiring high-speed memory access and inter-node transfer in order to achieve good results.

Major process involves iterative floating-point operations of a three-dimensional array p. For this reason, its program code has four loops in a nested structure. After a set of operations is completed, all values in the array are updated before going on to the next iteration.

In terms of parallelizing Himeno benchmark, data needs to be referred from hexagonally-adjacent nodes in 3-D array for 19 times within each innermost loop, and this is the main reason for requiring a large memory bandwidth. In this work, process of the outermost loop (the x-axis) of the three-dimensional array was allocated to each PE. Within each innermost loop, a set of $j-1, j, j+1$-th row data in $y-z$ plane were fetched from Mono memory to each PE, and were written back after computation. Swazzle path was used in order to refer adjacent $y-z$ plane data, but this referral was overlapped with Mono-to-Poly/Poly-to-Mono data transfer.

5.4.1.2 MG benchmark

MG from NAS parallel benchmark solves a 3D Poisson partial differential equation using Multigrid method. Multigrid method in general is a methodology for analytically solving differential equations by discretizing a target problem space using various size of grids. Memory access develops a certain pattern, but since there are variation in transfer data size, the benchmark is suitable for testing the accuracy of our model.

Major process involves a projection from fine grid to the coarse, approximation using the coarsest grid, and a prolongation to the fine grid. After each prolongation, residual computation and smoother are applied [92]. Each of the above process has three or four loops in a nested structure, so the implementation technique is quite similar to that of Himeno benchmark, except that transfer data size varies according to the grid size.

5.4.2 Amount of data transfer

For an impartial performance measurement, problem size needs to be fixed for each computing core for both benchmarks. Table 5.5 shows problem sizes that were used in this measurement. Size L was selected for Himeno Benchmark, which deals with $512 \times 256 \times 256$ single precision matrices. Class A was selected for MG benchmark, whose grid data is represented as $256 \times 256 \times 256$ double precision matrices.

Table 5.6: MG (Class A): Number of Data Transfer (Mono-to-Poly)

Buffer	Active	Actual transfer size (bytes)								
Size	PEs	512	272	256	144	128	80	48	32	16
512 bytes	96	199216	1488	9560	888	0	456	240	56	59872
	64	325776	1488	14728	888	0	456	240	56	97552
	32	574144	2512	24288	888	0	456	240	56	169712
	16	1069968	4000	43144	1536	0	456	240	56	312856
256 bytes	96	-	-	399920	752	19256	456	240	56	61496
	64	-	-	653040	752	29592	456	240	56	99176
	32	-	-	1150800	752	48712	456	240	56	172360
	16	-	-	2143936	1264	86560	456	240	56	317128

Table 5.7: MG (Class A): Number of Data Transfer (Poly-to-Mono)

Buffer	Active	Actual transfer size (bytes)							
Size	PEs	512	272	256	144	80	48	32	16
512 bytes	96	67672	664	0	408	216	120	32	20184
	64	109008	664	0	408	216	120	32	32520
	32	188968	1056	0	408	216	120	32	55776
	16	348232	1720	0	672	216	120	32	101632
256 bytes	96	-	-	136072	344	216	120	32	20912
	64	-	-	218744	344	216	120	32	33248
	32	-	-	379056	344	216	120	32	56896
	16	-	-	698312	544	216	120	32	103480

Tables 5.6, 5.7, and 5.8 break down the details about the number of data transfer executed in each benchmark. Here, *P* was set to 16, 32, 64, and 96, and *M* was set to 256 and 512 bytes.

5.4.3 Estimated and measured latencies

Figures 5.14 and 5.15 show the estimated and measured total time spent for data transfers in MG and Himeno benchmarks. Smaller values in each figure are the estimated best-case latencies, when there is no misalignment of data in Mono memory. Larger values in each figures are worst-case latencies for data transfer when there is misalignment of data. The results show that misalignment of data gives distinct influence on performance.

Shaded bars in Figures 5.14 and 5.15 indicate actual measured values of all data transfer instructions. As columns "Actual Transfer Size" in Tables 5.6, 5.7 and 5.8 indicate, data transfer sizes are

Table 5.8: Himeno Benchmark (Size L): Number of Data Transfer

Buffer Size	Active PEs	Actual Transfer Size (bytes) m2p: 512 bytes	p2m: 504 bytes
512	96	9108	3036
	64	12144	4048
	32	24288	8096
	16	48576	16192
		m2p: 256 bytes	p2m: 248 bytes
256	96	18216	6072
	64	24288	8096
	32	48576	16192
	16	97152	32384

m2p: Mono-to-Poly transfer, p2m: Poly-to-Mono transfer

not necessarily the multiple of 64 bytes. Thus, beginning addresses of some data blocks are not properly aligned. In other words, data transfer instructions in these programs do not necessarily satisfy the best-case conditions. Consequently, measured values should be in between the estimated best- and worst-case latencies.

As the figures indicate, plots of almost all measured values were in between the estimated best- and worst-case values. Actual measured time of MG benchmark were near worst-case values, apparently due to numerous 16-byte packet transfers and misalignment. On the other hand, as Figure 5.15 shows, measured time of Himeno Benchmark were slightly larger than best-case values, supporting the fact that majority of data transfer were aligned because beginning addresses of each data block were multiple of 512 or 256 bytes.

Overall, validity of communication latency model was supported for the two benchmarks used in this evaluation. This latency model provides target execution time to implement program with many communication instructions on CSX600.

According to this study, one approach for tuning applications on CSX600 is as follows. First, programmers should break down the number of data transfers and their data size as shown in Tables 5.6, 5.7, and 5.8. Once breakouts are obtained, they can estimate best- and worst-case latencies and optimize communication instructions to investigate whether execution time matches with the estimation. Finally, computation-intensive codes can be written so that they overlap with data transfer. Also, programmers can select optimal number of active PEs to make the best balance of computation and communication time by assigning various number of PEs to P in Equation (5.4). Because increasing the number of active PEs does prolong communication latency, using all PEs on the pro-

Table 5.9: MG(Class A): Breakdown of Data Transfer (Mono to Poly, Active number of PEs = 96, M = 512)

Used	Actual transfer size							
PEs	512	272	256	144	80	48	32	16
96	120896		4128					33304
66	54816		1040					16664
34	23504	1024	4392					9904
18		464		648				
10				240	328			
6					128	168		
4						72	56	

cessor is not necessarily the best choice for communication-intensive programs, whose typical case is when problem size is not the multiple of P.

5.4.3.1 Idea for improving data transmission mode

With current architecture of CSX600, ClearConnect provides global network to all 96 PEs, and they can receive both broadcast and individually addressed data from Mono memory. However, since PEs determine whether they receive data or not according to the results of local conditional branching, data may be sent to PEs in vain. As Equation (5.4) indicates, communication latency is dominantly influenced by the number of PEs activated at the runtime of the program. For example, even when only one PE receives data from Mono memory, $4\times\{(96-1)-1\} = 376$ cycles are uselessly consumed according to Equation (5.4).

These dead clock cycles are often consumed in real applications. Table 5.9 shows the breakdown of Mono-to-Poly data transfer for MG benchmark when P and M were set to 96 and 512, respectively. Row indicated as “Used PEs” shows the number of PEs that were the targets of data transfer. This shows that considerable proportion of data are transferred to selected PEs, which corresponds to approximately 16% of transfer clock cycles being consumed in vain. This can be improved if Mono processor can collect status of each PE and send data to them exclusively.

5.5 Chapter summary

To understand the behavior of NoCs in an existing modern many-core SIMD system, in this chapter, we have studied network performance of one-dimensional many-core SIMD processors which can also communicate with an external shared memory. Based on throughput evaluation, profiling of communication latency and observation of data packets, we introduced an analytical model for estimating communication latency. Using the model, we estimated communication latency and network throughput with different number of PEs and data size. To concrete the evaluation, we studied performance of NoCs called ClearConnect and Swazzle of ClearSpeed's CSX600 [17], which is a typical SIMD processor that consists of 96 Processing Elements (PEs) connected in a form of one-dimensional array topology.

We presented throughput measurement results under several conditions, and have shown that communication performance is influenced by (1) the number of used PEs, (2) the size of transferred data, and (3) data alignment of a shared memory. Based on these observations, we modeled communication latency for the following contributions: (a) programmers can estimate requisite minimum communication time in order to optimize their application programs, (b) programmers can estimate communication time and network throughput with different number of PEs so that they can decide suitable number of PEs to balance communication and computation time, and (3) existing NoC performance models can be extended by incorporating the overhead we have shown in this chapter.

Indeed, we illustrated the breakdown of the data transfers whose overhead can be estimated in the applications in Section 5.4, and these results enable to identify the ratio of the communication overhead in the applications. This study of NoCs with one-dimensional array and shared bus can contribute to enhance accuracy of existing general NoC performance models with various interconnect structures, by considering the sending and receiving overhead at each core. We believe that modeling, breakdown, and evaluations of actual NoC between components such as many simple processing cores and memories offer useful hints to efficiently update the structure of NoCs, such as router, topology, or routing for future many-core era.

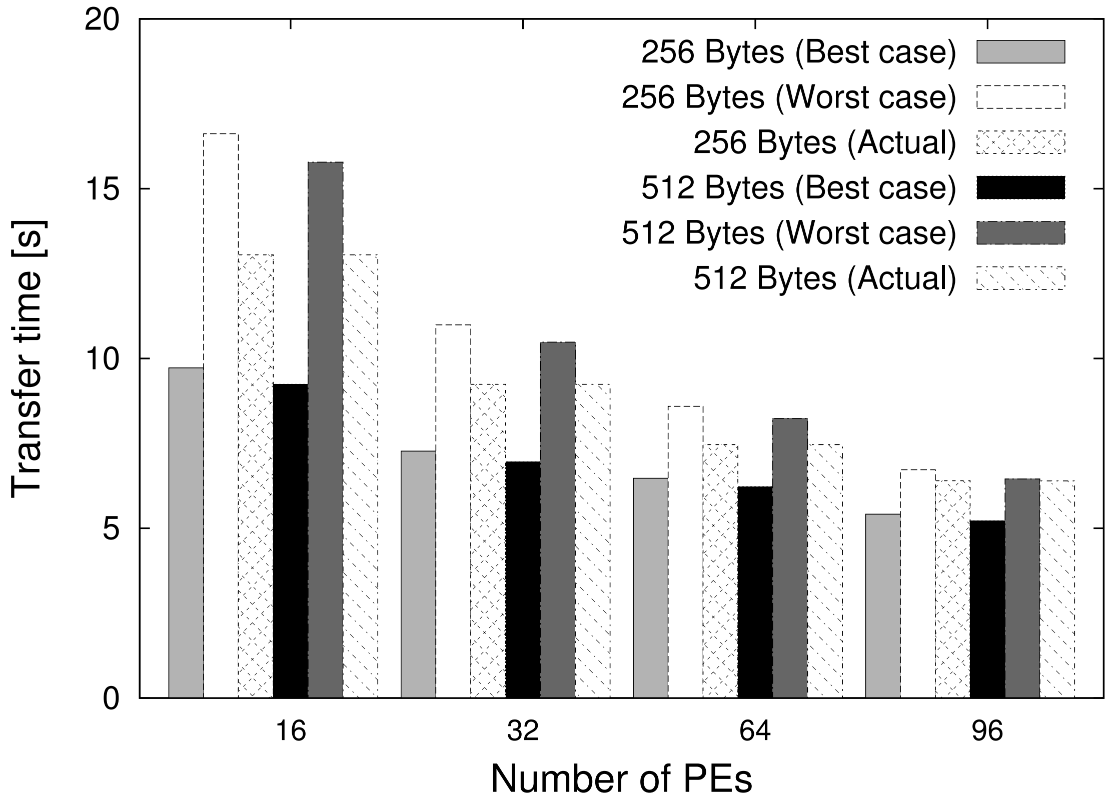


Figure 5.14: Data Transfer Time in MG Benchmark Class=A

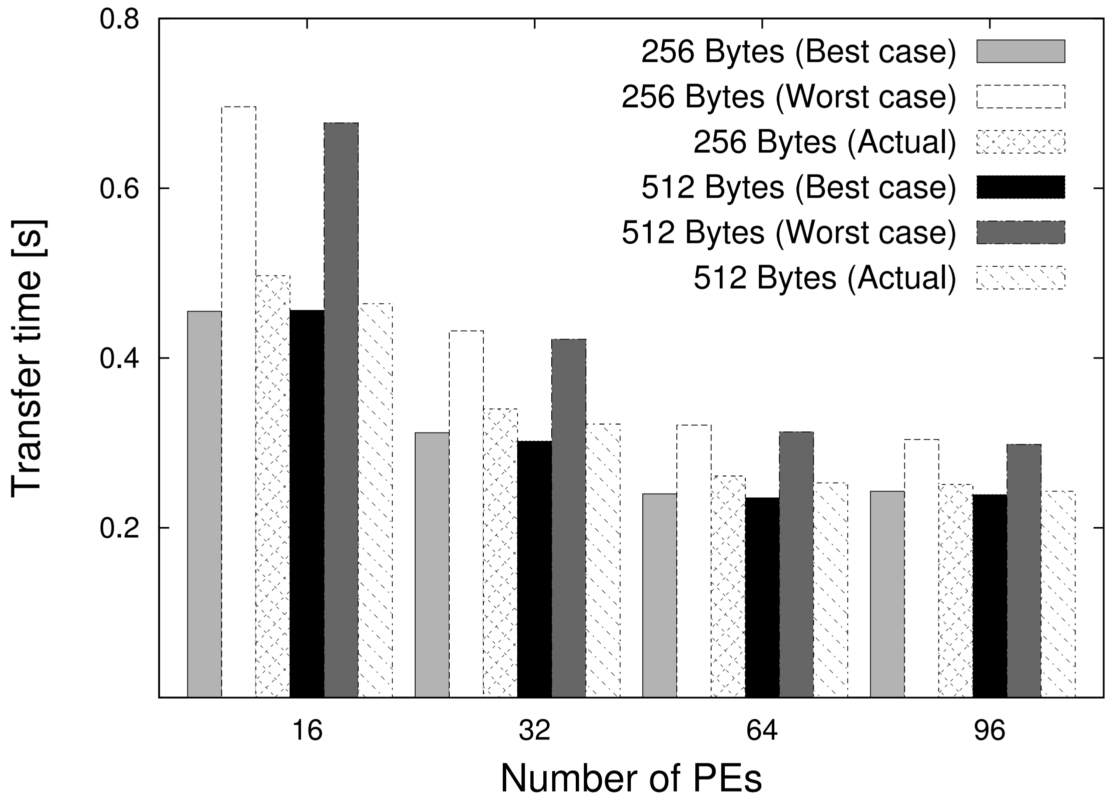


Figure 5.15: Data Transfer Time of Himeno Benchmark Size=L

Chapter 6

Conclusions

In order to expand a design space for many-core interconnection architectures, we proposed a packet routing scheme for chip multiprocessors, and also studied end-to-end latency of bus and one-dimensional ring architecture for SIMD many-core processor. We make a discussion on interconnection structure based on the analysis and proposed technique based on the evaluation result in Section 6.1. Finally, concluding remarks are derived in Section 6.2.

6.1 Discussion

6.1.1 Routing schemes for chip multiprocessors

In Chapter 4, we proposed a non-minimal fully-adaptive routing strategy called *Semi-deflection* in order to provide higher flexibility for NoC on CMPs by adopting single-flit packet structure and single-cycle router architecture. The proposed routing strategy successfully distributed the congestion of synthetic non-uniform traffic, and improved throughput by 3.17 times compared with existing deterministic routing and turn-model based adaptive routing.

Also, we assumed a case of adopting Semi-deflection routing on an actual eight-core CMP, and evaluated its throughput using actual benchmark traces which include cache coherency signals and other messages for running an operating system. As the result, Semi-deflection routing outperformed existing routing methodology in CG benchmark with numerous gather/scatter operations among cores, which support our view that Semi-deflection routing is suitable for local traffic, and also for cache coherency protocol with many-to-less operations or gather/scatter operations. The above results provides an insight that the routing methodology may provide high performance on cache-centric architectures [93], which generates frequent coherent transfers and traffic is non-uniform.

Semi-deflection routing is also capable of transferring variable-length packets. However, it is incapable of outperforming wormhole routing; maximum throughput of semi-deflection for non-local traffic is approximately 75% of wormhole existing routing methodologies with multiple-flit structure, and 80 to 93% for local traffic. However, because Semi-deflection assumes a single-flit packet structure, router area is small since the routing can be done with routers with small input buffers in their input channels. Compared with existing deterministic and adaptive single-flit routers, area increase was marginal. Thus, Semi-deflection routing is suitable for combining with existing routers with by separated network, just like Trips OCN and OPN, or TILE64's five networks with different roles. As a future work, we are planning to apply the routing scheme to two-dimensional torus, as well as evaluating throughput using actual cache coherent transfer traffic patterns.

6.1.2 Bus and one-dimensional ring architecture for SIMD processors

The analytical model proposed in Chapter 5 does not only give target execution time of parallel programs, but also provides some insights to NoC architecture of SIMD processors.

6.1.2.1 Comparison to existing NoC simulation study

This study precisely models end-to-end NoC latency that includes the sending and receiving overhead at each core, and it also illustrates that their overhead dominates latency and throughput of the NoCs whose structure is shared bus and one-dimensional array in simple SIMD architecture.

By using the existing theoretical analysis, probabilistic cycle-accurate simulation, or full system simulation, each component of the NoC, such as router, topology, or routing algorithm can be evaluated. However, in case of simple SIMD interconnects, performance bottleneck is the sending and receiving overhead at each core. Thus, even when the one-dimensional array of the SIMD interconnects is replaced with 2-D mesh, latency and throughput is not so much improved, or may become even worse. For example, we can compare communication latencies for transferring 64-byte data to 96 PEs via one-dimensional array topology, and 128-byte data to 48 PEs. In latter case, we assume that the other 48 PEs are only connected with Swazzle interconnect in a form of 2-D mesh, and they receive 64-byte data via Swazzle. According to Equation (5.2) and (5.4), a 128-byte data are broken down into two 64-byte packets, so large communication overhead would be doubled compared to the former case, and we can find that communication latency of the latter case is longer.

In the SIMD interconnects used in this study, topologies are one-dimensional array for neighboring communications with contention freedom, and shared bus with sequential access. Since existing simulation study of NoCs usually employs 2-D mesh, torus or fat tree, they are valuable for evaluating the future SIMD interconnects with a rich variety of components with the end-to-end communication latency model and results of this study.

6.1.2.2 Number of PEs required to achieve peak throughput and to alleviate effect of misalignment

According to the model, in this subsection, we will show that even with 96 PEs of CSX600, cannot make the full use of total bandwidth of ClearConnect due to the sending and receiving overheads. In order to estimate the required number of PEs to maximize transfer performance, we measured the relationship between number of PEs and bus throughput.

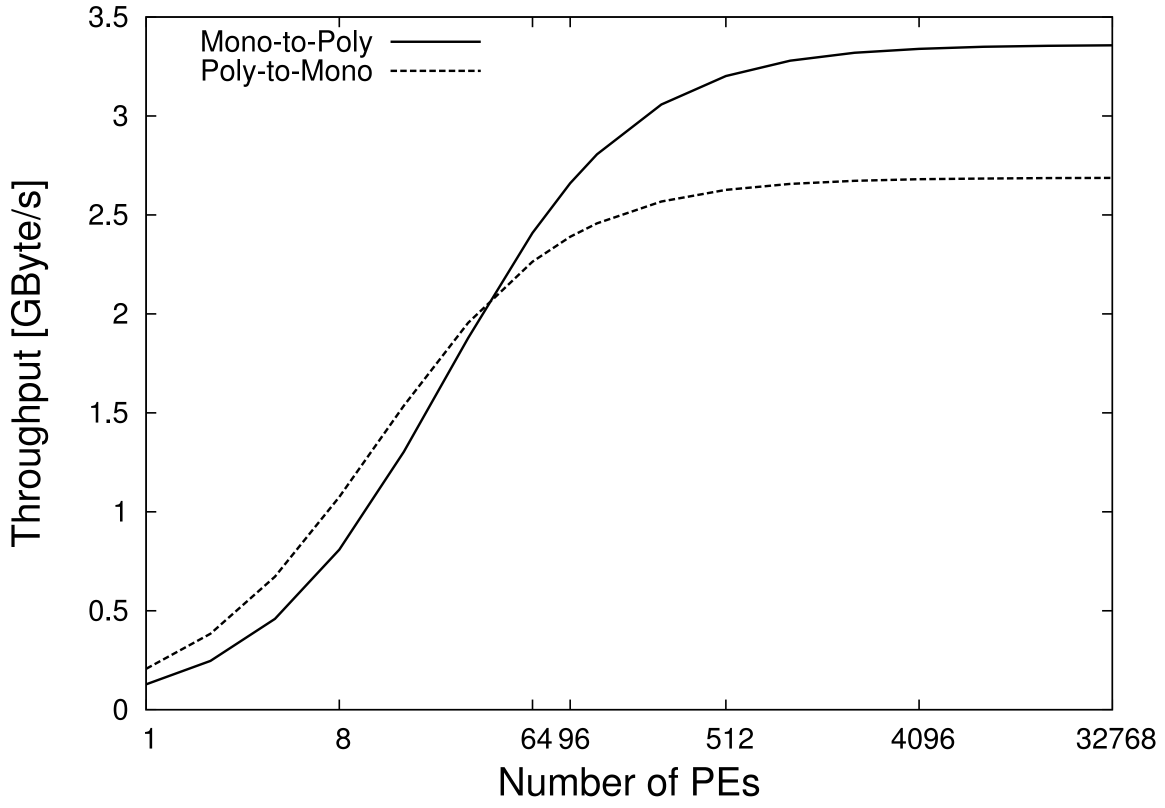


Figure 6.1: Throughput of ClearConnect with various number of PEs

Data size was set to 4 kilobytes due to the limitation of the capacity of Poly memory. We assume that its alignment was regular, and synchronization was conducted after 16 transfer instructions were issued. They are the conditions to utilize the bandwidth of ClearConnect most efficiently, as described in Section 5.2.3.

Estimated throughput is shown in Figure 6.1. According to the estimation, the throughput saturates at 3.34 Gbps and 2.85 Gbps for Mono-to-Poly and Poly-to-Mono, respectively. Thus, to accomplish higher usage of ClearConnect, larger number of PEs needs to be integrated on CSX600. In addition, bandwidth usage rate increases with larger number of PEs even when data alignment is not optimal. Compared to current architecture, influence of misalignment can be alleviated by 58% when 512 PEs were assumed to be mounted on CSX600.

Advantage of one-dimensional bus is its simplicity, but through this study we have found that sending and overhead dominates large proportion of end-to-end latency and total bus throughput is also effected. The above result suggests that increasing number of components connected to one-dimensional bus can derive its large bandwidth for many-core SIMD accelerators.

Recent major argue is that bus-based approaches cannot adequately handle the increasing communication demands with increasing number of cores, and indeed, this is true for MIMD-based processors. On the other hand, SIMD processors simply access bus components at the same time, so simple transfer mechanisms can satisfy SIMD application requirements. The main concern of bus-based architecture would be increase of wiring delay with technology advancement as described in Chapter 1, which would probably be a quite severe defect to maintain high frequency. Thus, to apply buses as its interconnect for future processor with new process technologies, it is predicted that such processor would be more throughput-oriented architecture with lower operating frequency.

6.2 Concluding Remarks

Interconnection networks which connect a number of cores are crucial components to design a chip for multi- and many-core processors as the number of cores increases. There are three mainstream designs many-core processor architectures: multi- to many-core general-purpose processors, many-core scientific accelerators for teraflops computation, and tile architectures. The first two design approaches in a bottom-up manner for designing versatile processor architectures by succeeding control mechanism of single-core architectures, or by optimizing for certain fields of applications with regards to analysis results of target algorithms and its traffic patterns. However, both many-core general-purpose processors and accelerators which adopt global buses are facing the increasing wiring delay problem due to process technology improvement. Thus, tile architectures that mainly adopt NoCs are being focused as solutions to achieve scalability with increasing number of cores From a viewpoint of interconnection network, support for various traffic requirements is necessary for versatility; it needs to be compatible of both volumic traffic using point-to-point links, and small latency-aware traffic with efficient broadcasting mechanism. It is a prime task to accumulate research outputs of interconnection network including buses and NoCs for future many-core era. Against such background, heterogeneous NUCA-based tile architecture with dedicated SIMD accelerator is expected as a new form of future many-core processor.

In this thesis, we first network traffic characteristics in recent many-core processors, including that of cache coherence protocols and parallel applications. Based on these observations, we focused on flexible packet routing mechanism for efficiently delivering short messages in grid-based CMPs with small hardware size, and also a performance characterization of interconnection networks on SIMD accelerators based on parallel application execution.

The first topic proposes a new routing mechanism called Semi-deflection routing for future tile-based chip multiprocessors with increasing amount of cache coherent transfers. We focused on enhancing performance for transferring increasing amount of short signals, and by packing them into a single-flit packet structure, we proposed a simple non-minimal fully-adaptive deadlock-free routing methodology without use of hardware-consuming virtual channels. We confirmed that the routing suppresses latency by maintaining maximum throughput.

Secondly, we explored characterizations and practical issues of interconnection in an existing many-core SIMD accelerators, and for this aim we focused on the network performance of a shared-memory one-dimensional many-core SIMD accelerator. We modeled end-to-end latency between components, and confirmed that actual communication time of parallel benchmarks fit in between the best- and the worst-case estimated values.

By using existing theoretical analysis, probabilistic cycle-accurate simulation, or full system simulation, each component of the NoC, such as router, topology, or routing algorithm, can be evaluated. However, in the case of the simple SIMD interconnects, the performance bottleneck is the sending and receiving overhead between each PEs and shared memory. Thus, even when the one-dimensional array of the SIMD interconnects is replaced with 2-D mesh, the latency and throughput is not so much improved as long as the network switch is not heavily saturated. In the SIMD interconnects used in this study, the topologies are one-dimensional array for neighboring communications with contention freedom, and shared bus with sequential access. Since existing simulation study of NoCs usually employs 2-D mesh, torus, or fat tree, they are valuable for evaluating the future SIMD interconnects with a rich variety of components with the model and results of this study.

Majority of current and near-future tile and accelerator architecture implementations apparently would not solely rely on buses nor NoCs, and would be composed of two or more separated networks: buses or dedicated networks for low-latency, short-message signaling, and NoCs for bandwidth bound transfers. Through this study, it is our view that both bottom-up traffic analysis of actual processors and top-down approaches of study-phase processors are essential for breaking out of the paradigm of many-core interconnection architectures. These explorations are very imaginative, and they would provide a large and untouched design space for the future processor interconnection networks.

Abbreviations and Acronyms

AHB	Advanced High-performance Bus
ALU	Arithmetic Logic Unit
AMBA	Advanced Microcontrolledr Bus Architecture
APB	Advanced Peripheral Bus
API	Application Programming Interface
ASB	Advanced System Bus
AVCI	Advanced Virtual Component Interface
CCB	ClearConnect Bus
CG	Conjugate Grid
CMOS	Complementary Metal Oxide Semiconductor
CMP	Chip Multiprocessor
CPU	Central Processing Unit
DCD	DMA Command Descriptor
DCR	Device Control Register
DDR2	Double Data Rate
DGEMM	Double-precision General Matrix Multiply
DMA	Direct Memory Access
DOR	Dimension Order Routing
DRAM	Dynamic Random Access Memory
EIB	Element Interconnect Bus
ESRAM	Embedded Synchronous Random Access Memory
FIFO	First-in First-out
GFLOPS	Giga Floating Operations Per Second
GPU	Graphic Processor
ILP	Instruction Set Architecture
ITRS	The International Technology Roadmap for Semiconductors
LMI	Local External Memory Interface
LRU	Least Recently Used
MAC	Multiply-Accumulate
MG	Multigrid

MIC	Memory Interface Controller
MIMD	Multiple Instruction Multiple Data
MMIO	Memory-Mapped Input/Output
MSI	Modified Shared Invalid
MTAP	Multi-Threaded Array Processor
MU	Memory Unit
NoC	Network-on-Chip
NPB	NAS Parallel Benchmark
NUCA	Non-Uniform Cache Architecture
OCN	On-Chip Network
OPB	On-chip Peripheral Bus
OPN	Operand Network
OSF	Output Selection Function
PE	Processor Element
PLB	Processor Local Bus
PPE	Power Processing Element
RAM	Random Access Memory
RC	Routing Computation
RTL	Register Transfer Level
SA	Switch Allocation
SAF	Store-and-Forward Switching
SAN	System Arena Networks
SIMD	Single Instruction Multiple Data
SoC	System-on-Chips
SPE	Synergistic Processing Elements
SPMD	Single Program Multiple Data
SRAM	Synchronous Random Access Memory
ST	Switch Traversal
TSC	Thread Sequence Controller
UART	Universal Asynchronous Receiver Transmitter
VA	Virtual Channel Allocation
VCI	Virtual Component Interface
VCT	Virtual-Cut Through
VLSI	Very Large Scale Integration
WH	Wormhole

Bibliography

[1] ITRS. *International Technology Roadmap for Semiconductors 2009 Update*, 2008. available at `http://www.itrs.net/`.

[2] NVIDIA. *NVIDIA Tesla C1060 Computing Processor*, 2008. available at `http://www.nvidia.com/object/product_tesla_c1060_us.html`.

[3] Junichiro Makino, Kei Hiraki, and Mary Inaba. GRAPE-DR: 2-Pflops massively-parallel computer with 512-core, 512-Gflops processor chips for scientific computing. *SC07*, 2007.

[4] Intel. *Microarchitecture Codename Sandy Bridge*, 2011. available at `http://www.intel.com/technology/architecture-silicon/2ndgen/index.htm?wapkw=\%28sandy+bridge\%29`.

[5] AMD. *AMD Fusion Family of APUs: Enabling a Superior, Immersive PC Experience*, Mar. 2010.

[6] Christopher J. Glass and Lionel M. Ni. The Turn Model for Adaptive Routing. In *Proceedings of the International Symposium on Computer Architecture (ISCA'92)*, pages 278–287, May 1992.

[7] William J. Dally and Hiromichi Aoki. Deadlock-Free Adaptive Routing in Multicomputer Networks Using Virtual Channels. *IEEE Transactions on Parallel and Distributed Systems*, 4(4):466–475, 1993.

[8] José Duato. A Necessary And Sufficient Condition For Deadlock-Free Adaptive Routing In Wormhole Networks. *IEEE Transaction on Parallel and Distributed Systems*, 6(10):1055–1067, 1995.

[9] William James Dally and Charles L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Transactions on Computers*, 36(5):547–553, May 1987.

[10] Michael Bedford Taylor, Walter Lee, Jason E. Miller, David Wentzlaff, Ian Bratt, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jason Sungtae Kim, James Psota, Arvind Saraf, Nathan Shnidman, Volker Strumpen, Matthew Frank, Saman P. Amarasinghe, and Anant Agarwal. Evaluation of the Raw Microprocessor: An Exposed-Wire-Delay Architecture for ILP and Streams. In *Proceedings of the International Symposium on Computer Architecture (ISCA'04)* , pages 2–13, June 2004.

[11] Michael Kistler, Michael Perrone, and Fabrizio Petrini. Cell Multiprocessor Communication Network: Built for Speed. *IEEE Micro*, 26(3):10–23, May 2006.

[12] Paul Gratz, Changkkyu Kim, Karthikeyan Sankaralingam, Heather Hanson, Premkishore Shivakumar, Stephen W. Keckler, and Doug Burger. On-Chip Interconnection Networks of the TRIPS Chip. *IEEE Micro*, 27:41–50, 2007.

[13] Sriram Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Priya Iyer, Arvind Singh, Tiju Jacob, Shailendra Jain, Sriram Venkataraman, Yatin Hoskote, and Nitin Borkar. An 80-Tile 1.28TFLOPS Network-on-Chip in 65nm CMOS. In *Proceedings of the International Solid-State Circuits Conference (ISSCC'07)* , Feb. 2007.

[14] Erland Nilsson. Design and Implementation of a Hot-Potato Switch in Network On Chip. Master's thesis, Laboratory of Electronics and Computer Systems, Royal Institute of Technology (KTH), June 2002.

[15] Thomas Moscibroda and Onur Mutlu. A Case for Bufferless Routing in On-Chip Networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA'09)*, June 2009.

[16] Smaragda Konstantinidou and Lawrence Snyder. The Chaos Router. *IEEE Transactions on Computers*, 43(12):1386–1397, 1994.

[17] ClearSpeed Technology Limited. http://www.clearspeed.com, 2010.

[18] Lionel M. Ni. Issues in Designing Truly Scalable Interconnection Networks. In *Proceedings of the International Conference on Parallel Processing (ICPP'96)*, pages 74–83, 1996.

[19] David Flynn. AMBA Enabling Reusable On-Chip Designs. *IEEE Micro*, 17(4):20–27, July 1997.

[20] IBM Corporation. The CoreConnect^{TM} Bus Architecture, 1999. available at `http://www.chips.ibm.com/products/coreconnect/`.

[21] Sonics Inc. Sonics3220^{TM} SMART Interconnect IP^{TM}, 2002. available at `http://www.sonicsinc.com/`.

[22] Kenichiro Anjo, Atsushi Okamura, and Masato Motomura. Wrapper-based Bus Implementation Techniques for Performance Improvement and Cost Reduction. *IEEE Journal of Solid-State Circuits*, 39(5):804–817, May 2004.

[23] ARM Ltd. AMBA AXI Protocol Specification, 2003. available at `http://www.arm.com/products/solutions/AMBAHomePage.html`.

[24] William J. Dally and Brian Towles. Route Packets, Not Wires: On-Chip Interconnection Networks. In *Proceedings of the Design Automation Conference (DAC'01)*, pages 684–689, June 2001.

[25] Luca Benini and Giovanni De Micheli. Networks on Chips: A New SoC Paradigm. *IEEE Computer*, 35(1):70–78, January 2002.

[26] Luca Benini and Giovanni De Micheli. *Networks on Chips: Technology And Tools*. Morgan Kaufmann, 2006.

[27] José Duato, Sudhakar Yalamanchili, and Lionel M. Ni. *Interconnection Networks: An Engineering Approach*. Morgan Kaufmann, 2002.

[28] William James Dally and Brian Towles. *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[29] Milica Mitic and Mile Stojcev. A Survey of Three System-on-Chip Buses: AMBA, CoreConnect and Wishbone. In *Proceedings of XLI International Scientific Conference on Information, Communication and Energy Systems and Technologies (ICEST 2006)*, pages 282–286, 2006.

[30] ARM. CoreLink System IP & Design Tools for AMBA, 2010. available at `http://www.arm.com/products/system-ip/amba/`.

[31] IBM Microelectronics. CoreConnect bus architecture, 1999. available at `https://www-01.ibm.com/chips/techlib/techlib.nsf/techdocs/852569B20050FF77852569910050C0FB/\$file/crcon_wp.pdf`.

[32] Kanishka Lahiri, Anand Raghunathan, and Ganesh Lakshminarayana. LOTTERYBUS: A New High-Performance Communication Architecture for System-on-Chip Designs. In *Proceedings of Design Automation Conference*, pages 15–20, 2001.

[33] Naresh R. Shanbhag. Reliable and Efficient System-on-Chip Design. *IEEE Computer*, 37(3):42–50, Mar. 2004.

[34] Giovanni Strano and Salvatore Tiralongo and Carlo Pistritto. OCP STBUS Plug-in Methodology. In *Proceedings of the International Embedded Solutions Event*, 2004.

[35] Chris. Rowen. *Engineering the Complex SoC: Facts, Flexible Design with Configurable Processors*. Prentice Hall, 2004.

[36] Charles E. Leiserson. Fat-Trees: Universal Networks for Hardware-Efficient Supercomputing. *IEEE Transactions on Computers*, 34(10):892–901, October 1985.

[37] André DeHon. Unifying Mesh- and Tree-Based Programmable Interconnect. *IEEE Transactions on Very Large Scale Integration Systems*, 12(10):1051–1065, October 2004.

[38] Charles E. Leiserson, Zahi S. Abuhamdeh, David C. Douglas, Carl R. Feynman, Mahesh N. Ganmukhi, Jeffrey V. Hill, W. Daniel Hillis, Bradley C. Kuszmaul, Margaret A. St. Pierre, David S. Wells, Monica C. Wong-Chan, Shaw-Wen Yang, and Robert Zak. The Network Architecture of the Connection Machine CM-5. *Journal of Parallel and Distributed Computing*, 33(2):145–158, March 1996.

[39] Hussein G. Badr and Sunil Podar. An Optimal Shortest-Path Routing Policy for Network Computers with Regular Mesh-Connected Topologies. *IEEE Transactions on Computers*, 38(10):1362–1371, October 1989.

[40] Jie Wu. An Optimal Routing Policy for Mesh-Connected Topologies. *Proceedings of International Conference on Parallel Processing*, 1:267–270, 1996.

[41] Jie Wu. Maximum-shortest-path (MSP): An Optimal Routing Policy for Mesh-Connected Multicomputers. *IEEE Transaction on Reliability*, 48(3):247–255, 1999.

[42] Ge-Ming Chiu. The Odd-Even Turn Model for Adaptive Routing. *IEEE Transaction on Parallel and Distributed Systems*, 11(7):729–738, 2000.

[43] F. Silla and J. Duato. High-Performance Routing in Networks of Workstations with Irregular Topology. *IEEE Transactions on parallel and distributed systems*, 11(7):699–719, 2000.

[44] Robert Mullins, Andrew West, and Simon Moore. Low-Latency Virtual-Channel Routers for On-Chip Networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA'04)*, pages 188–197, June 2004.

[45] Robert Mullins, Andrew West, and Simon Moore. The Design and Implementation of a Low-Latency On-Chip Network. In *Proceedings of the Asia and South Pacific Design Automation Conference (ASP-DAC'06)*, pages 164–169, January 2006.

[46] Jongman Kim, Chrysostomos Nicopoulos, Dongkook Park, Vijaykrishnan Narayanan, Mazin S. Yousif, and Chita R. Das. A Gracefully Degrading and Energy-Efficient Modular Router Architecture for On-Chip Networks. In *Proceedings of the International Symposium on Computer Architecture (ISCA'06)*, pages 14–15, June 2006.

[47] William James Dally. Virtual-Channel Flow Control. *IEEE Transactions on Parallel and Distributed Systems*, 3(2):194–205, 1992.

[48] Li-Shiuan Peh and William J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'01)*, pages 255–266, January 2001.

[49] Michihiro Koibuchi, Hiroki Matsutani, Hideharu Amano, and Timothy Mark Pinkston. A Lightweight Fault-tolerant Mechanism for Network-on-Chip. In *Proceedings of the International Symposium on Networks-on-Chip (NOCS'08)* , April 2008. (to appear).

[50] Hiroki Matsutani, Michihiro Koibuchi, Hideharu Amano, and Tsutomu Yoshinaga. Prediction Router: Yet Another Low Latency On-Chip Router Architecture. In *Proceedings of the International Symposium on High-Performance Computer Architecture (HPCA'09)* , pages 367–378, February 2009.

[51] Yuto Hirata, Hiroki Matsutani, Michihiro Koibuchi, and Hideharu Amano. A Variable-pipeline On-chip Router Optimized to Traffic Pattern. In *Proceedings of the Third International Workshop on Network on Chip Architectures*, pages 57–62, 2010.

[52] Doug Burger, Stephen W. Keckler, Kathryn S. McKinley, Mike Dahlin, Lizy K. John, Calvin Lin, Charles R. Moore, James Burrill, Robert G. McDonald, William Yoder, and the TRIPS Team. Scaling to the End of Silicon with EDGE Architectures. *IEEE Computer*, 37(7):44–55, July 2004.

[53] Kevin J. Barker, Kei Davis, Adolfy Hoisie, Darren J. Kerbyson, Mike Lang, Scott Pakin, and Jose Carlos Sancho. A Performance Evaluation of the Nehalem Quad-Core Processor for Scientific Computing. *Parallel Processing Letters*, 18(4):453–469, 2008.

[54] David Wentzlaff, Bruce Edwards, John Iii, and John F. Brown. On-chip Interconnection Architecture of the TILE Processor. *IEEE Micro*, pages 15–31, 2007.

[55] Tilera Corporation. Processors, 2010. available at `http://www.tilera.com/products/processors`.

[56] Sriram Vangal, Jason Howard, Gregory Ruhl, Saurabh Dighe, Howard Wilson, James Tschanz, David Finan, Arvind Singh, Tiju Jacob, Shailendra Jain, Vasantha Erraguntla, Clark Roberts, Yatin Hoskote, Nitin Borkar, and Shekhar Borkar. An 80-Tile Sub-100-W TeraFLOPS Processor in 65nm CMOS. *IEEE Journal of Solid-State Circuits*, 43(1):29–41, Jan. 2008.

[57] Jason Howard, Saurabh Dighe, Sriram R. Vangal, Gregory Ruhl, Nitin Borkar, Shailendra Jain, Vasantha Erraguntla, Michael Konow, Michael Riepen, Matthias Gries, Guido Droege, Tor Lund-Larsen, Sebastian Steibl, Shekhar Borkar, Vivek K. De, and Rob Van Der Wijngaart. A 48-Core IA-32 Processor in 45nm CMOS Using On-Die Message-Passing and DVFS for Performance and Power Scaling. *IEEE Journal of Solid-State Circuits*, pages 173–183, Jan. 2011.

[58] Timothy G. Mattson, Rob F. Van Der Wijngaart, Michael Riepen, Thomas Lehnig, Paul Brett, Werner Haas, Patrick Kennedy, Jason Howard, Sriram Vangal, Nitin Borkar, Greg Ruhl, and Saurabh Dighe. The 48-Core SCC Processor: The Programmer's View. In *Proceedings of IEEE Supercomputing (SC10)*, Nov. 2010.

[59] Karthikeyan Sankaralingam, Ramadass Nagarajan, Haiming Liu, Changkyu Kim, Jaehyuk Huh, Doug Burger, Stephen W. Keckler, and Charles R. Moore. Exploiting ILP, TLP and DLP with the Polymorphous TRIPS Architecture. In *Proceedings of the International Symposium on Computer Architecture (ISCA'03)*, pages 422–433, June 2003.

[60] Michael Bedford Taylor, Jason Sungtae Kim, Jason E. Miller, David Wentzlaff, Fae Ghodrat, Ben Greenwald, Henry Hoffmann, Paul Johnson, Jae-Wook Lee, Walter Lee, Albert Ma, Arvind Saraf, Mark Seneski, Nathan Shnidman, Volker Strumpen, Matthew Frank, Saman P. Amarasinghe, and Anant Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. *IEEE Micro*, 22(2):25–35, April 2002.

[61] Jian Liang, Andrew Laffely, Sriram Srinivasan, and Russell Tessier. An Architecture and Compiler for Scalable On-Chip Communication. *IEEE Transactions on Very Large Scale Integration Systems*, 12(7):711–726, July 2004.

[62] Matteo Dall'Osso, Gianluca Biccari, Luca Giovannini, Davide Bertozzi, and Luca Benini. xpipes: a Latency Insensitive Parameterized Network-on-chip Architecture For Multi-Processor SoCs. In *Proceedings of the International Conference on Computer Design (ICCD'03)*, pages 536–539, October 2003.

[63] Davide Bertozzi and Luca Benini. Xpipes: A Network-on-Chip Architecture for Gigascale Systems-on-Chip. *IEEE Circuits and Systems Magazine*, 4(2):18–31, 2004.

[64] Sonics Inc. Open Core Protocol Specification 1.0, 2000. available at `http://www.socworks.com/`.

[65] STI. *Cell Broadband Engine Documentation*, August 2005. available at `http://www.ibm.com/developerworks/power/cell/,http://cell.scei.co.jp`.

[66] Larry Seiler, Robert Cavin, Roger Espasa, Ed Grochowski, Toni Juan, Pat Hanrahan, Doug Carmean, Eric Sprangle, Tom Forsyth, Michael Abrash, Pradeep Dubey, Stephen Junkins, Adam Lake, and Jeremy Sugerman. Larrabee: A Many-Core x86 Architecture for Visual Computing. *ACM Transactions on Graphics*, 27(3):1, Aug, 2008.

[67] Yoshihiro Fujita, Nobuyuki Yamashita, and Shin'ichiro Okazaki. IMAP-VISION: An SIMD Processor with High-Speed On-chip Memory and Large Capacity External Memory. In *IAPR Workship on Machine Vision Applications*, pages 170–173, November 1996.

[68] Takashi Komuro, Shingo Kagami, and Masatoshi Ishikawa. A Dynamically Reconfigurable SIMD Processor for a Vision Chip. *IEEE Journal of Solid-State Circuits*, 39(1):265–268, Jan. 2004.

[69] Krste Asanovic, Ras Bodik, Bryan Christopher Catanzaro, Joseph James Gebis, Parry Husbands, Kurt Keutzer, David A. Patterson, William Lester Plishker, John Shalf, Samuel Webb Williams, and Katherine A. Yelick. The Landscape of Parallel Computing Research: A View from Berkeley. Technical Report UCB/EECS-2006-183, EECS Department, University of California, Berkeley, Dec 2006.

[70] Sunpyo Hong and Hyesoon Kim. An Analytical Model for a GPU Architecture with Memory-level and Thread-level Parallelism Awareness. *SIGARCH Comput. Archit. News*, 37(3):152–163, 2009.

[71] Thomas William Ainsworth and Timothy Mark Pinkston. Characterizing the Cell EIB On-Chip Network. *IEEE Micro*, 27(5):6–14, September 2007.

[72] Vassos Soteriou, Hangsheng Wang, and Li-Shiuan Peh. A Statistical Traffic Model for On-Chip Interconnection Networks. In *Proceedings of 14th IEEE International Symposium on Modeling, Analysis, and Simulation (MASCOTS '06)*, pages 104–116, 2006.

[73] Mohamed Bakhouya, Suboh A. Suboh, Jaafar Gaber, and Tarek A. El-Ghazawi. Analytical Modeling and Evaluation of On-Chip Interconnects Using Network Calculus. In *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS2009)*, 2009.

[74] David Arditti Ilitzky, Jeffrey D. Hoffman, Anthony Chun, and Brando Perez Esparza. Architecture of the Scalable Communications Core's Network on Chip. *IEEE Micro*, 27(5):62–74, September 2007.

[75] Mohammad Baharloo, Reza Hajisheykhi, Mohammad Arjomand, and Amir-Hossein Jahangir. An Analytical Performance Evaluation for WSNs Using Loop-Free Bellman Ford Protocol. In *Proceedings of AINA*, pages 568–571, 2009.

[76] Mohammad Arjomand and Hamid Sarbazi-Azad. A Comprehensive Power-performance Model for NoCs with Multi-flit Channel Buffers. In *Proceedings of ICS*, pages 470–478, 2009.

[77] Nilanjan Banerjee, Praveen Vellanki, and Karam S. Chatha. A Power and Performance Model for Network-on-Chip Architectures. In *Proceedings of Design Automation and Test in Europe Conference (DATE'04)*, pages 1250–1255, February 2004.

[78] Peter S. Magnusson, Magnus Christensson, Jesper Eskilson, Daniel Forsgren, Gustav Hallberg, Johan Hogberg, Fredrik Larsson, Andreas Moestedt, and Bengt Werner. Simics: A Full System Simulation Platform. *IEEE Computer*, 35(2):50–58, February 2002.

[79] Milo M. K. Martin, Daniel J. Sorin, Bradford M. Beckmann, Mic hael R. Marty, Min Xu, Alaa R. Alameldeen, Kevin E. Moore, Mark D. H ill, and David A. Wood. Multifacet General Execution-driven Multiprocessor Simulator (GEMS) Toolset. *ACM SIGARCH Computer Architecture News (CAN'05)*, 33(4):92–99, November 2005.

[80] Umit Y. Ogras, Radu Marculescu, and Hyung Gyu Lee nad Naehyuck Chang. Communication Architecture Optimization: Making the Shortest Path Shorter in Regular Networks-on-Chip. In *Proceedings of DATE*, pages 712–717, 2006.

[81] Dac Pham, Hans-Werner Anderson, Erwin Behnen, Mark Bolliger, Sanjay Gupta, Peter Hofstee, Paul Harvey, Charles Johns, Jim Kahle, Atsushi Kameyama, John Keaty, Bob Le, Sang Lee, Tuyen Nguyen, John Petrovick, Mydung Pham, Juergen Pille, Stephen Posluszny, Mack Riley, Joseph Verock, James Warnock, Steve Weitzel, and Dieter Wendel. The Design and Implementation of a First-Generation CELL Processor. In *Proceedings of the International Solid-State Circuits Conference (ISSCC'05)*, February 2005.

[82] Adán Kohler and Martin Radetzki. Fault-Tolerant Architecture and Deflection Routing for Degradable NoC Switches. In *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS 2009)*, 2009.

[83] Akiya Jouraku, Michihiro Koibuchi, and Hideharu Amano. An Effective Design of Deadlock-Free Routing Algorithms Based on 2-D Turn Model for Irregular Networks. *IEEE Transactions on Parallel Distributed Systems*, 18(3):320–333, 2007.

[84] Hiroki Matsutani, Michihiro Koibuchi, Daihan Wang, and Hideharu Amano. Adding Slow-Silent Virtual Channels for Low-Power On-Chip Networks. In *Proceedings of the International Symposium on Networks-on-Chip (NOCS'08)* , pages 23–32, April 2008.

[85] William Saphir, Rob Van Der Wijngaart, Alex Woo, and Maurice Yarrow. New Implementations and Results for the NAS Parallel Benchmarks 2. In *8th SIAM Conference on Parallel Processing for Scientific Computing*, March 1997.

[86] P. Lopez, J.M. Martinez, and J. Duato. DRIL: Dynamically Reduced Message Injection Limitation Mechanism for Wormhole Networks. In *Proceedings of the International Conference on Parallel Processing*, pages 535–542, Aug 1998.

[87] E. Baydal, P. Lopez, and J. Duato. A Simple and Efficient Mechanism to Prevent Saturation in Wormhole Networks. *Parallel and Distributed Processing Symposium, International*, 0:617, 2000.

[88] GSIC Web, 2010. http://www.gsic.titech.ac.jp/index.html.en/.

[89] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Iain Duff. A Set of Level 3 Basic Linear Algebra Subprograms. *ACM Transaction on Mathmatical Software (TOMS)*, 16(1):1–17, 1990.

[90] ACCC. RIKEN. *Himeno Benchmark*, 2010. available at `http://accc.riken.jp/HPC_e/HimenoBMT_e.html`.

[91] David Bailey, Tim Harris, William Saphir, Rob Van Der Wijngaart, Alex Woo, and Maurice Yarrow. The NAS Parallel Benchmarks 2.0. *NAS Technical Reports NAS-95-020*, December 1995.

[92] Haoqiang Jin, Michael Frumkin, and Jerry Yan. The OpenMP Implementation of NAS Parallel Benchmarks and Its Performance. Technical report, 1999.

[93] Ran Manevich, Isask Walter, Israel Cidon, Avinoam Kolodny, and Technion Israel. Best of Both Worlds : A Bus Enhanced NoC (BENoC). In *Proceedings of the 3rd ACM/IEEE International Symposium on Networks-on-Chip (NOCS2009)*, pages 1–10, 2009.

Publications

Related Papers

Journal Papers

[1] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi, Kenichi Miura and Hideharu Amano, An Analytical Network Performance Model for SIMD Processor CSX600 Interconnects, Journal of Systems Architecture (JSA), Vol.57, No.1, pp.146–159, Jan 2011.

[2] Yuri Nishikawa, Michihiro Koibuchi, Hiroki Matsutani and Hideharu Amano, Semi-deflection Routing: A Non-minimal Fully-adaptive Routing for Virtual Cut-through Switching Network, International Journal of Computer and Network Security (IJCNS), Vol.10, No.2, pp.52–58, Oct 2010.

[3] Yuri Nishikawa, Michihiro Koibuchi, Hiroki Matsutani and Hideharu Amano, A Non-minimal Fully Adaptive Routing Using Single-Flit Single-Cycle Routers for NoCs, IPSJ Transactions on Advanced Computing Systems, Vol.3, No.SIG 3 (ACS31), pp.88–99, Sep 2010. (In Japanese)

International Conference Papers

[4] Yuri Nishikawa, Michihiro Koibuchi, Hiroki Matsutani and Hideharu Amano, A Deadlock-Free Non-minimal Fully Adaptive Routing Using Virtual Cut-Through Switching, *Proc. of IEEE Fifth International Conference on Networking, Architecture and Storage (NAS2010)*, pp. 431–438, Jul 2010.

[5] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi, Akihiro Shitara, Kenichi Miura and Hideharu Amano, Performance Analysis of ClearSpeed's CSX600 Interconnects, *Proc. of IEEE International Symposium on Parallel and Distributed Processing with Applications (ISPA 2009)*, pp.203–210, Sep 2009.

[6] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi, Kenichi Miura and Hideharu Amano, Performance Improvement Methodology for ClearSpeed's CSX600, *Proc. of the International Conference on Parallel Processing (ICPP'07)*, CD-ROM, Sep 2007.

Domestic Conference Papers and Technical Reports

[7] Yuri Nishikawa, Michihiro Koibuchi, Hiroki Matsutani and Hideharu Amano, A Non-minimal Fully Adaptive Routing Using a Single-Flit Packet Structure, *IEICE Technical Reports (CPSY)*, Vol. 109, No. 394, pp.53–58, Jan 2010. **SLDM Student Presentation Award** (In Japanese)

[8] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi, Akihiro Shitara, Kenichi Miura and Hideharu Amano, Evaluation of Communication Performance on ClearSpeed's SIMD Processor, *IEICE Technical Reports (SWoPP09)*, Vol. 109, No. 168, pp.91–96, Aug 2009. (In Japanese)

[9] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi, Kenichi Miura and Hideharu Amano, A Study of Time Prediction Method for Running Parallel Applications on ClearSpeed's SIMD-Based Multi-Core Processor, *IPSJ SIG Technical Reports 2007-ARC-174 (SWoPP07)*, pp.43–48, Aug 2007. (In Japanese)

[10] Yuri Nishikawa, Michihiro Koibuchi, Masato Yoshimi, Kenichi Miura and Hideharu Amano A Proposal of Performance Improvement Based on a Parallel Benchmark Evaluation on a ClearSpeed Coprocessor, *IPSJ SIG Technical Reports 2007-ARC-172 (HOKKE07)*, pp.257–262, Mar 2007. (In Japanese)

Other Papers

Journal Papers

[11] Masato Yoshimi, Yuri Nishikawa, Hideharu Amano, Mitsunori Miki, Tomonori Hiroyasu and Oskar Mencer, Performance Evaluation of One-dimensional FPGA-cluster CUBE for Stream Applications, IPSJ Transactions on Advanced Computing Systems, Vol.3, No.SIG 3 (ACS31), pp.209–220, Sep 2010. (In Japanese)

[12] Hideki Yamada, Naoki Iwanaga, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Hideharu Amano, Akira Funahashi, Noriko Hiroi and Kiyoshi Oguri Automatic Pipeline Construction Focused on Similarity of Rate Law Functions for an FPGA-based Biochemical Simulator, IPSJ Transaction on System LSI Design Methodology, Vol.3, pp. 244-256, 2010. (In Japanese)

[13] Masato Yoshimi, Yuri Nishikawa, Yasunori Osana, Akira Funahashi, Noriko Hiroi Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, Design and Evaluation of an FPGA-based Stochastic Biochemical Simulator for High-throughput Execution, IPSJ Transactions on Advanced Computing Systems, Vol.1, No. 3 (ACS24), pp.120–135, Dec 2008. (In Japanese)

[14] Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano,

A Biochemical Model Compiler for Simulations on an FPGA, The IEICE Transactions on Information and Systems, Vol. 91, Nov. 9, pp.2205–2216, Sep 2008. (In Japanese)

[15] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Akira Funahashi, Noriko Hiroi Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, FPGA-based Stochastic Biochemical Simulator, IPSJ Transactions on Advanced Computing Systems, Vol. 48, No. SIG3 (ACS17), pp.45–58, Feb 2007. (In Japanese)

[16] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, ReCSiP: An FPGA-based General-purpose Biochemical Simulator (translation of OsanaIEICE06), The IEICE Transactions on Information and Systems (Part II: Electronics), Vol. 90, No. 7, pp.1–10, 2007.

[17] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, ReCSiP: An FPGA-Based General PurPose Biochemical Simulator, The IEICE Transactions on Information and Systems, Vol. 89, No. 6, pp.1163–1172, Jun 2006. (In Japanese)

International Conference Papers

[18] Masahiro Yamada, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, A Proposal of Thread Virtualization Environment for Cell Broadband Engine, *In Proc. of Parallel and Distributed Computing and Systems*, Nov. 2010.

[19] Masato Yoshimi, Yuri Nishikawa, Mitsunori Miki, Tomoyuki Hiroyasu, Hideharu Amano and Mencer, Oskar, A Performance Evaluation of CUBE: One-Dimensional 512 FPGA Cluster *In Proc. of Reconfigurable Computing: Architectures, Tools and Applications*, pp. 372–381, 2010.

[20] Akihiro Shitara, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, Preliminary Evaluation of Batch-Learning Self-Organizing Map Algorithm on a Graphic Processor, *In Proc. of Parallel and Distributed Computing Networks*, Feb. 2010.

[21] Tomonori Ooya, Hideki Yamada, Tomoya Ishimori, Yuichiro Shibata, Yasunori Osana, Kiyoshi Oguri, Masato Yoshimi, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi and Hideharu Amano, Configuring area and performance: Empirical Evaluation on an FPGA-based Biochemical Simulator, *In Proc. of International Conference on Field Programmable Logic and Applications (FPL2009)*, pp. 679–682, 2009.

[22] Tomoya Ishimori, Hideki Yamada, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Hideharu Amano, Akira Funahashi, Noriko Hiroi and Kiyoshi Oguri, Pipeline Scheduling with Input Port Constraints for an FPGA-Based Biochemical Simulator, *In Proc. of ARC '09: Proceedings of the 5th International Workshop on Reconfigurable*

Computing: Architectures, Tools and Applications, Vol. 108, No. 48, pp.368–373, 2009.

[23] Akihiro Shitara, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, Implementation and Evaluation of Self-Organizing Map Algorithm on a Graphic Processor, *In Proc. of Parallel and Distributed Computing and Systems*, Nov. 2009.

[24] Hideki Yamada, Yasunori Osana, Tomoya Ishimori, Tomonori Ooya, Masato Yoshimi, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Hideharu Amano, Yuichiro Shibata and Kiyoshi Oguri, A Modular Approach to Heterogeneous Biochemical Model Simulation on an FPGA, *In Proc. of the 2009 International Conference on Reconfigurable Computing and FPGAs (RECONFIG09)*, pp. 125–130, 2009.

[25] Masato Yoshimi, Yuri Nishikawa, Yasunori Osana, Akira Funahashi, Yuichiro Shibata, Hideki Yamada, Noriko Hiroi, Hiroaki Kitano and Hideharu Amano, Practical Implementation of a Network-based Stochastic Biochemical Simulation System on an FPGA, *In Proc. of International Conference on Field Programmable Logic and Applications (FPL2008)*, pp. 663–666, 2008.

[26] Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, A Framework for Implementing a Network-Based Stochastic Biochemical Simulator on an FPGA, *In Proc. of International Conference on Field-Programmable Technology (ICFPT2007)*, pp. 193–200, 2007.

[27] Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Yuichiro Shibata, Naoki Iwanaga, Hideki Yamada, Hiroaki Kitano, Akira Funahashi, Noriko Hiroi and Hideharu Amano, FPGA Implementation of a Data-Driven Stochastic Biochemical Simulator with the Next Reaction Method, *In Proc. of International Conference on Field Programmable Logic and Applications (FPL2007)*, pp. 254–259, 2007.

[28] Hideki Yamada, Naoki Iwanaga, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Hideharu Amano, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Kiyoshi Oguri, A Combining Technique of Rate Law Functions for a Cost-Effective Reconfigurable Biological Simulator, *In Proc. of International Conference on Field Programmable Logic and Applications (FPL2007)*, pp. 808–811, 2007.

[29] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, Hardware Design of a Stochastic Biochemical Simulator, *In Proc. of Winter Simulation Conference 2006*, 2006.

[30] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano, Hideharu Amano, A Hardware Accelerator for Biochemical Simulations, *In Proc. of Winter Simulation Conference*

2006, 2006.

[31] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, An FPGA Implementation of High Throughput Stochastic Simulator for Large-Scale Biochemical Systems, *In Proc. of International Conference on Field Programmable Logic and Applications (FPL2006)*, pp.1–6, 2006.

[32] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, Performance Evaluation of an FPGA-Based Biochemical Simulator ReCSiP, *In Proc. of International Conference on Field Programmable Logic and Applications (FPL2006)*, 2006.

[33] Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, An Acceleration of a Biochemical Simulator on Programmable Hardware, *In Proc. of The Seventh International Conference on Systems Biology(ICSB2006)*, 2006.

[34] Yuri Nishikawa, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, An Evaluation Investigation of Dual-Thread Numerical Integration Mechanism for FPGA-based Biochemical Simulator ReCSiP, *In Proc. of CoolChips IX*, 2006.

[35] Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, A System Design of Accelerating Biochemical Simulations on Programmable Hardware, *In Proc. of the 10th Forum on Software Platforms for Systems Biology*, 2005.

Domestic Conference Papers and Technical Reports

[36] Toshiaki Kamata, Akihiro Shitara, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, Implementation and Evaluation of Program Development Middleware for Cell Broadband Engine Clusters, *IEICE Technical Reports (CPSY)* Vol. 110, No. 361 pp.7–12, Jan 2011. (In Japanese)

[37] Akihiro Shitara, Toshiaki Kamata, Masahiro Yamada, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, The Implementation of Development-support Middleware on Multiple-node Environment of OpenCL Accelerator, *IPSJ SIG Technical Reports 2010-ARC-192 (HOKKE20 10)*, No. 22, Dec 2010. (In Japanese)

[38] Toshiaki Kamata, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, A Proposal of Offload Structure for Cell Broadband Engine, *IPSJ SIG Technical Reports 2010-ARC-190 (SWoPP 2010)*, No. 21, Aug 2010. (In Japanese)

[39] Tetsuya Nakahama, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, The Proposal of Inter Node Communication in Heterogeneous Interconnect Cluster, *IPSJ SIG Technical Reports 2010-ARC-190 (SWoPP2010)*, No. 16, Aug 2010. (In Japanese)

[40] Masahiro Yamada, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, The Implementation of Cache Mechanism on the Thread Virtualization Environment for cell cluster, *IPSJ SIG Technical Reports 2010-ARC-190 (SWoPP2010)*, No. 14, Aug 2010. (In Japanese)

[41] Nobuhiro Onishi, Toshiaki Kamata, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, Implementation and Evaluation of Photon Mapping on Cell Broadband Engine *IEICE Technical Reports (CPSY)* Vol. 110, No. 167, pp.19–24, Aug 2010. (In Japanese)

[42] Masahiro Yamada, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, Proposal of Thread Virtualization Environment on Cell Broadband Engine, *IEICE Technical Reports (CPSY)* Vol. 110, No. 2, pp.27–32, Apr 2010. (In Japanese)

[43] Masato Yoshimi, Mitsunori Miki, Yuri Nishikawa, Akihiro Shitara, Hideharu Amano and Oskar Mencer, Performance Evaluation of Levenshtein-Distance Computation on One-Dimensional FPGA Array Cube, *IEICE Technical Report (CPSY)*, Vol. 109, No. 198, pp. 13-18, Sep 2009. (In Japanese)

[44] Tomonori Ooya, Hideki Yamada, Tomoya Ishimori, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yuri Nishikawa, Hideharu Amano, Akira Funahashi, Noriko Hiroi and Kiyoshi Oguri, Performance Evaluation of an Auto-generation Aalgorithm of Hardware Modules for an FPGA-based General-purpose Biochemical Simulator, *IEICE technical report*, Vol. 109, No. 26, pp.37–42, May 2009. (In Japanese)

[45] Akihiro Shitara, Yuri Nishikawa, Masato Yoshimi and Hideharu Amano, Implementation and Evaluation of Self-Organizing Map Algorithm on a Graphic Processor, *IPSJ SIG Technical Reports 2009-ARC-182 (HOKKE2009)*, No. 14, pp. 31-36, Feb 2009. (In Japanese)

[46] Masato Yoshimi, Yuri Nishikawa, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Kitano Hiroaki and Hideharu Amano, Proposal of FPGA-based Calculation System Exploiting Thread Level Parallelism for Scientific Applications, *IPSJ SIG Technical Reports 2009-ARC-180 (DesignGaia2008)*, No. 101, pp.63–66, Oct 2008. (In Japanese)

[47] Hideki Yamada, Tomoya Ishimori, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yuri Nishikawa, Hideharu Amano, Akira Funahashi, Noriko Hiroi and Kiyoshi Oguri, An Automatic Combine Algorithm of Arithmetic Pipelines for an FPGA-based Biochemical Simulator Focused on Similarities of Rate Law Functions, *IEICE technical report (CPSY)*, Vol.108, No.220, pp.21–26, Sep 2008. (In Japanese)

[48] Tomoya Ishimori, Hideki Yamada, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Hideharu Amano, Akira Funahashi, Noriko Hiroi and Kiyoshi

Oguri, Pipeline Scheduling with Input Port Constraints for an FPGA-based Biochemical Simulator, *IEICE technical report*, Vol.108, No.48, pp.113–118, May 2008. (In Japanese)

[49] Masato Yoshimi, Yuri Nishikawa, Toshinori Kojima, Yasunori Osana, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Hideki Yamada, Hiroaki Kitano and Hideharu Amano, Evaluation of a Data-Driven Architecture for a Stochastic Biochemical Simulator on an FPGA, *IEICE technical report*, Vol. 107, No.342, pp.43–48, Nov 2007. (In Japanese)

[50] Hideki Yamada, Naoki Iwanaga, Yuichiro Shibata, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Hideharu Amano, Akira Funahashi, Noriko Hiroi, Hiroaki Kitano and Kiyoshi Oguri, Automatic Combining of Rate Law Functions for an FPGA-based Biochemical Simulator ReCSiP, *IEICE technical report*, Vol.107, No.41, pp.13–18 May 2007. (In Japanese)

[51] Toshinori Kojima, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, Implementation and Evaluation of General-Purpose Host Interface on ReCSiP Board, *IEICE technical report*, Vol.106, No.49, pp.55–60, May 2006. (In Japanese)

[52] Yuri Nishikawa, Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, A Performance Improvement Strategy for Numerical Integration on an FPGA-Based Biochemical Simulator ReCSiP, *IEICE technical report*, Vol.105, No.516, pp.53–58, Jan 2006. (In Japanese)

[53] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, Hardware-resource Utilization Analysis on an FPGA-Based Biochemical Simulator ReCSiP, *IEICE technical report*, Vol.105, No.518, pp.59–64, Jan 2006. (In Japanese)

[54] Masato Yoshimi, Yasunori Osana, Yow Iwaoka, Yuri Nishikawa, Toshinori Kojima, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, A Stochastic Biochemical Simulator with a Data-transfer Network on an FPGA, *IEICE technical report*, Vol.105, No.518, pp.47–52, Jan 2006. (In Japanese)

[55] Yow Iwaoka, Yasunori Osana, Masato Yoshimi, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, Building of the SBML System for an FPGA-based Biochemical Simulator, *IEICE technical report*, Vol.105, No.287, pp.61–66, Sep 2005. (In Japanese)

[56] Yasunori Osana, Masato Yoshimi, Yow Iwaoka, Toshinori Kojima, Yuri Nishikawa, Akira Funahashi, Noriko Hiroi, Yuichiro Shibata, Naoki Iwanaga, Hiroaki Kitano and Hideharu Amano, The Control Mechanism of an FPGA-Based Biochemical Simulator ReCSiP, *IEICE technical report*, Vol.105, No.287, pp.55–60, Sep 2005. (In Japanese)

Appendix A

Livelock arguments of the Chaos router

Livelock freedom of the chaos router [16] is given. In non-minimal routers, as messages are allowed to move further from their destinations, some mechanism is required in order to guarantee that mesages will eventually make progress towared their destinations. Mechanism to deterministically guarantee message delivery is quite expensive. The following show that by choosing randomly a message to deroute, the chaos router guarantees message delivery with high probability. Lemma 1 formalizes the key idea of that no message is forever derouted with certainty, or in other words, every message has a nonzero chance of avoiding derouting.

Lemma 1 $^{\exists}\epsilon > 0$, *such that for every message M entering the queue of some node P , M will be routed with probability $p \geq \epsilon$ and derouted with probability $q = 1 - p$.*

Proof *For every message M that enters the queue of some node P, M is guaranteed to leave the queue in a finite amount of time [16]. Thus, there is a maximum amount of time $T = \Sigma_{i=0}^{m}$ that message M can stay in the queue of some node P after which M will certainly be routed. During this time T, node P makes a finite number of routing decisions. Thus message M can be subjected to at most a finite number, r, of derouting decisions. At each derouting decision, a message is randomly selected from the queue with uniform probability. The probability that message M can escape derouting at each derouting decision is $(c - 1)/c$. Thus, the probability that message M will be routed is at least $\epsilon = (\frac{c-1}{c})^r$.*

Using Lemma 1, it can be said that the probability of long paths in the network diminishes as their length increases. The path of a message in the network is defined as a sequence of *moves*. At each move, message M either moves closer to its destination with probability $p \geq \epsilon$ or further from its destination with probability $q = 1 - p$.

Clearly, a message cannot move further than $\log N$ from its destination. Now here we define a *game* as a sequence of $\log N$ moves. Message M starts game i at distance a_i and finishes at distance a_{i+1}. Let l_i denote the event that M was *not* delivered during game i and w_i the event that M was delivered during game i. Let $Q(i)$ be the probability that message M has not been delivered after i games. Then

$$Q(i) = P(l_i l_{i-1} ... l_1) = P(l_i | l_{i-1} l_1) \cdot P(l_{i-1} ... l_1)$$

For simplicity, let us substitute F_k for $l_k ... l_2 l_1$, $1 \leq k < i$ and let us define $P(l_1 | F_0) \equiv P(l_1)$ $P(w_1 | F_0) = P(w_1)$. Then

$$Q(i) = P(l_i | F_{i-1}) \cdot P(l_{i-1} | F_{i-2}) \cdots P(l_1) \tag{A.1}$$

Clearly, $P(l_j | F_{j-1}) = 1 - P(w_j | F_{j-1})$, $1 \leq j \leq i$. In the following we will estimate $P(w_j | F_{j-1})$. Let $s_{j,k}$ denote the event that message M starts game j at Hamming distance k from its destination. Events $s_{j,k}$ are mutually exclusive and one of them necessarily happens. Thus

$$\begin{aligned}
P(w_j | F_{j-1}) &= P(w_j S_{j,1} \cup ... \cup w_j S_{j,\log N} | F_{j-1}) \\
\Leftrightarrow P(w_j | F_{j-1}) &= \sum_{k=1}^{\log N} P(w_j S_{j,k} | F_{j-1}) \\
\Leftrightarrow P(w_j | F_{j-1}) &= \sum_{k=1}^{\log N} P(w_j | S_{j,k} F_{j-1}) \cdot P(S_{j,k} | F_{j-1}).
\end{aligned}$$

But $P(w_j | S_{j,k} F_{j-1}) \geq \epsilon^{\log N}$, thus

$$P(w_j | F_{j-1}) \geq \epsilon^{\log N} \sum_{k=1}^{\log N} P(S_{j,k} | F_{j-1}).$$

Since

$$\begin{aligned}
\sum_{k=1}^{\log N} P(S_{j,k} | F_{j-1} &= 1 \\
\Leftrightarrow P(w_j | F_{j-1}) &\geq \epsilon^{\log N} \\
\Leftrightarrow P(l_j | F_{j-1}) &\leq 1 - \epsilon^{\log N}
\end{aligned} \tag{A.2}$$

Finally Equations A.1 and A.2 derive $Q(i) \leq (1 - \epsilon^{\log N})^i$.

Thus the probability that M will *not* have been delivered after i games, where $i \to \infty$ is

$$\lim_{i \to \infty} Q(i) = (1 - \epsilon^{\log N})^i = 0$$

The probability $P(i)$ that M will be delivered after i games, where $i \to \infty$ is

$$\lim_{i \to \infty} P(i) = 1.$$

Appendix B

Specifications of ClearSpeed's CSX600 memory architecture and ClearConnect bus

B.1 Memory architecture

Figure B.1 illustrates the memory architecture of CSX600. The CSX600 processor supports up to 4 Gbytes of external Dynamic RAM (DRAM) accessed through Direct Memory Access (DMA) controller. The DMA can transfer data to and from the external memory and any other device on the ClearConnect bus. External memory is connected to the CSX600 processor through a 64-bit DDR2 DRAM interface. The processor supports 64-bit addressing, and address space is mapped into 48-bit physical address space. On-chip SRAM is included which operates as instruction and data cache.

The memory system consists of An external DRAM, A Memory Unit (MU) including a DMA unit and a DDR2 controller, Processing Elements (PE) memory comprised of a register file and a Static RAM (SRAM), and a thread sequence controller comprised of an instruction cache (iCache), a data cache (dCache) and a register file.

The Memory unit (MU) provides CSX600 processor with a large external DRAM. This minimizes latency between MTAP core and DRAM, and allows the memory capacity and bandwidth to scale with compute power.

ClearConnect Bus (CCB) memory requests are serviced through the CCB data channel and an Advanced Virtual Component Interconnect (AVCI) target port. The DMA unit produces memory requests both internally to a local external memory interface and externally to the CCB data channel through AVCI initiator interface. The MU incorporates the following features.

- One source or destination of DMA transfer always uses the LMI, and the other is any target on the CCB.
- Global target addressing uses the same 64-bit address scheme with MTAP.
- Coherency of DMA operation memory is supported by a local hardware semaphore.
- Overall throughput approaches peak memory bandwidth under best case conditions, which are:

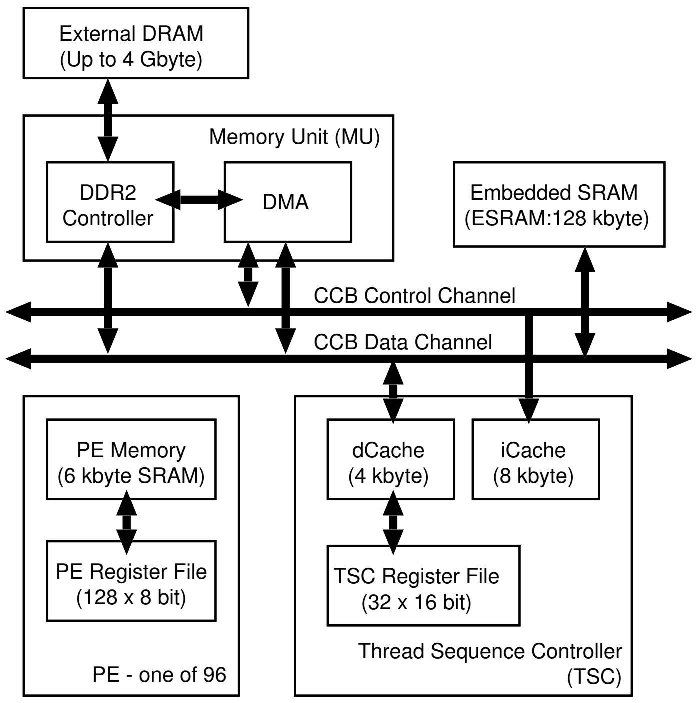


Figure B.1: Memory unit block diagram of CSX600

- 2.6 GBytes/s at 166MHz
- 3.2 GBytes/s at 200MHz

The DMA performs autonomous transfer of data either into or out of the LMI. The DMA issues read instructions on the CCB to a specified start address, with a specified addressing mode. Once enabled, the DMA unit waits for a dedicated local semaphore. A processor that wishes to initiate a DMA operation firts writes a DMA command descriptor (DCD) to some memory accessible over the CCB. Then it issues a write to the hardware semaphore for the DMA channel. This contains the 64-bit global logical address of the DCD for a DMA transfer. The DMA channel then reads the DCD from global memory. Also, two DMA transfer modes are supported:

- Chained: points to a subsequent DCD to be executed in the same DMA sequence
- Not chained: points to a global semaphore to be signalled when current DMA transfer is complete.

With above-mentioned mechanism, one or more DMA transfers may execute autonomously once their DCDs have been set up in the memory. When the DMA channel has completed its operation, it signals a global semaphore.

Embedded SRAM (ESRAM) is an array of static RAM that can be freely accessed by using read and write transactions on the CCB. It supports fully pipelined operation, one data word per clock cycle, for contiguous read or write operations. For mixed read and writes, the memory supports reads and writes on consecutive clock cycles with no dead clock cycles. The ESRAM is 128 Kbytes, organized as 8 kwords × 16 bytes wide.

The Thread Sequence Controller (TSC) is an embedded processor which can fetch instructions from external memory for local execution or from MTAP.

B.2 ClearConnect Bus (CCB)

The CCB is a transport level mechanism that enables fuctional block to exchange packets of variable length with other functional blocks in the local CSX600 processor and a host processor. Physically, CCB is a pipelined on-chip network that consists of a series of ClearConnect nodes which are connected to form two ClearConnect channels: the data channel and the control channel. A channel typically contains a pair of ClearConnect lanes, unidirectional transfer paths arranged in opposite directions to form a bidirectional channel. The resulting linear bus structure enables the interconnection of a set of functional blocks, either on a single CSX600 processor or spanning multiple devices.

The CCB posses no predefined definition of transaction types, but provides a raw transport mechanism on to which the Virtual Component Interface standard (VCI), used by the functional blocks, is mapped. Especially, CCB data channel and CCB control is compatible with Advanced VCI (AVCI) and Peripheral VCI (PVCI) described in Subsection 2.2.3.4.

Packets are passed over the CCBs with guaranteed delivery. In the event of channel congestion, the nodes are able to buffer packets, so a packet may be delayed in delivery but never lost.

The CCB provides packet exchange between functional blocks including the ClearConnect Bridge (CCBR) and the Host Debug Port (HDP). Thse packets represent bus transactions: memory read/write requests and responses. The CCB consists of two channels:

- A 128-bit data channel, which is used for memory read and write. This channel contains two subchannels:
  - VC0: the request channel
  - VC1: the response channel
- A 32-bit control channel, which is used for the control ports of the processor core and other functional blocks for read/write access to their registers. It is also used by the processor core for issuing interrupt request messages to a host processor.

Functional blocks send and receive signals that conform to the VCI standard. ClearConnect Interfaces convert the VCI “cells” (a unit of data transfer over the VCI) to and from the flow control digits (flits) used on the CCB.