APPLICATION CENTRIC NETWORKS-ON-CHIP DESIGN SOLUTIONS FOR FUTURE MULTICORE SYSTEMS

A Dissertation

by

RAHUL BOYAPATI

Submitted to the Office of Graduate and Professional Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Chair of Committee,	Eun Jung Kim
Committee Members,	Valerie Taylor
	Rabi Mahapatra
	Peng Li
Head of Department,	Dilma Da Silva

May 2017

Major Subject: Computer Engineering

ABSTRACT

With advances in technology, future multicore systems scaled to 100s and 1000s of cores/accelerators are being touted as an effective solution for extracting huge performance gains using parallel programming paradigms. However with the failure of Dennard Scaling all the components on the chip cannot be run simultaneously without breaking the power and thermal constraints leading to strict chip power envelops. The scaling up of the number of on chip components has also brought upon Networks-On-Chip (NoC) based interconnect designs like 2D mesh. The contribution of NoC to the total on chip power and overall performance has been increasing steadily and hence high performance power-efficient NoC designs are becoming crucial.

Future multicore paradigms can be broadly classified, based on the applications they are tailored to, into traditional Chip Multi processor(CMP) based application based systems, characterized by low core and NoC utilization, and emerging big data application based systems, characterized by large amounts of data movement necessitating high throughput requirements. To this order, we propose NoC design solutions for power-savings in future CMPs tailored to traditional applications and higher effective throughput gains in multicore systems tailored to bandwidth intensive applications. First, we propose Fly-over, a light-weight distributed mechanism for power-gating routers attached to switched off cores to reduce NoC power consumption in low load CMP environment. Secondly, we plan on utilizing a promising next generation memory technology, Spin-Transfer Torque Magnetic RAM(STT-MRAM), to achieve enhanced NoC performance to satisfy the high throughput demands in emerging bandwidth intensive applications, while reducing the power consumption simultaneously. Thirdly, we present a hardware data approximation framework for NoCs, APPROX-NoC, with an online data error control mechanism, which can leverage the approximate computing paradigm in the emerging data intensive big data applications to attain higher performance per watt.

DEDICATION

To my parents for their support.

ACKNOWLEDGMENTS

I would like to thank my advisor, Dr.Eun Jung Kim, for first motivating me to pursue a doctoral degree and then supporting me with her guidance and patience through the course of my PhD. I would like to thank Dr.Peng Li, Dr.Rabi Mahapatra and Dr.Valerie Taylor for serving as my committee members. I also thank Dr.Ki Hwan Yum for helping me with meticulous feedback on my research and paper writing.

I would like to thank all the members of Dr.Kim's research group who helped me with research and were there for me throughout this journey. I would especially like to thank Jiayi Huang who worked with me on the research detailed in chapters II and IV, and Hyunjun Jang who collaborated with me for research presented in chapter III.

CONTRIBUTORS AND FUNDING SOURCES

Contributors

The work needed for this research was completed by me with support from a dissertation committee consisting of Professors Dr.Eun Jung Kim(Advisor), Dr.Valerie Taylor and Dr.Rabi Mahapatra of the Department of Computer Science and Engineering and Professor Dr.Peng Li of the Department of Electrical Engineering.

Funding Sources

Graduate study was supported by a fellowship from Texas A&M University and a dissertation research fellowship from NSF Foundation.

NOMENCLATURE

NOC	Networks-on-Chip
FLOV	Fly-Over mechanism
STT-MRAM	Spin-Transfer Torque Magnetic Random Access Memory
ECC	Error Correcting Codes
VAXX	Value Approximation mechanism

TABLE OF CONTENTS

Page

ABSTRACT ii

DEDICATION iv

ACKNOWLEDGMENTS v

CONTRIBUTORS AND FUNDING SOURCES vi

NOMENCLATURE vii

TABLE OF CONTENTS viii

LIST OF FIGURES xi

LIST OF TABLES xiv

1. INTRODUCTION 1

2. DISTRIBUTED POWER GATING MECHANISM 7

2.1 Introduction 7
2.2 Related Work 8
2.3 FLOV Router Architecture 11
2.3.1 Baseline NoC Router Architecture 11
2.3.2 FLOV Router Architecture 11
2.4 Restricted FLOV and Generalized FLOV Handshake Protocols 13
2.4.1 Restricted FLOV 13
2.4.2 Generalized FLOV 19
2.5 Dynamic Routing Algorithm 21
2.5.1 FLOV NoC Architecture 21
2.5.2 Dynamic Routing Algorithm 22
2.5.3 Overhead Analysis 25
2.6 Experimental Evaluation 26
2.6.1 Experimental Methodology 26
2.6.2 Synthetic Workload Evaluation 27
2.6.2.1 Performance 30
2.6.2.2 Power Consumption 32

2.6.2.3 Real Workload Evaluation 34
2.6.3 Reconfiguration Overhead Analysis 34
2.7 Conclusions 35

3. POWER-EFFICIENT AND RELIABLE ON-CHIP INTERCONNECTS USING STT-MRAM ROUTERS 37

3.1 Introduction 37
3.2 Background and Design Challenges 40
3.2.1 STT-MRAM 41
3.2.2 Fine-tuning STT-MRAM for High Performance NoCs 41
3.2.2.1 Retention Time 41
3.2.2.2 Switching Current and Switching Time 42
3.2.2.3 Cell Area 43
3.2.2.4 Impact of Process Technology 44
3.2.3 STT-MRAM Design Challenges 44
3.2.3.1 Retention Failure and Error Protection 44
3.2.3.2 Determining Proper Retention and Switching Times . . 45
3.3 Motivation 47
3.3.1 STT-MRAM for NoC Routers 47
3.4 STT-MRAM Router Architecture 48
3.4.1 Baseline Router Architecture 48
3.4.2 STT-MRAM Router Design 49
3.4.2.1 Multibank STT-MRAM Buffer 49
3.4.2.2 Read/Write and Refresh Logic 53
3.4.2.3 Handling Uncorrectable Errors in Refresh Operations . 55
3.4.3 Nonvolatility-Relaxed STT-MRAM Buffer 56
3.4.3.1 Refresh with ECC Scheme 56
3.4.3.2 Dynamic Buffer Refresh Schemes 58
3.5 Evaluation 61
3.5.1 System Configuration 61
3.5.2 Performance and Power Analysis 65
3.5.3 Sensitivity Analysis 70
3.6 Discussion 71
3.6.1 Impact of Write Delays of STT-MRAM 71
3.6.2 Comparison with Other NoC Techniques 72
3.6.3 Impact of End-to-End and Per-Hop Error Protection 74
3.7 Related Work 74
3.8 Conclusions and Future Work 77

4. NOC DATA APPROXIMATION FRAMEWORK 78

4.1 Introduction 78

4.2 Related Work 80
4.2.1 Approximation 80
4.2.2 NoC data compression 81
4.3 Motivation and challenges 82
4.3.1 Motivation 82
4.3.1.1 Data movement is becoming the critical component in multicore systems 82
4.3.1.2 Frequently repeated patterns appear in applications . . . 82
4.3.1.3 Data accuracy is not required 83
4.3.1.4 Defining approximate data similarity is necessary 83
4.3.2 Challenges 84
4.3.2.1 Value approximation and compression are not cheap . . 84
4.3.2.2 Quality control is important 84
4.4 Approx-NoC Framework Architectural Overview 85
4.4.1 APPROX-NoC Framework 86
4.4.2 Approximate Value Compute Logic Design 88
4.5 Implementation of APPROX-NoC 90
4.5.1 Frequent-Pattern Mechanisms 90
4.5.1.1 FP-VAXX Implementation 91
4.5.2 Dictionary-Based Mechanisms 92
4.5.2.1 DI-VAXX Implementation 95
4.5.3 Latency Overhead 98
4.6 Evaluation 98
4.6.1 Methodology 98
4.6.1.1 Experimental Setup 98
4.6.1.2 Workloads 100
4.6.2 Performance Analysis 101
4.6.2.1 Impact on Performance 103
4.6.2.2 Throughput Analysis 107
4.6.3 Sensitivity Studies 107
4.6.3.1 Error Threshold 108
4.6.3.2 Approximable Packets Ratio 109
4.6.4 Full System Impact Analysis 109
4.6.4.1 Overall Application Output Error 109
4.6.4.2 Overall Application Performance 112
4.6.5 Power Consumption and Area Overhead 113
4.7 Conclusions 113

5. CONCLUSIONS 115

REFERENCES 117

LIST OF FIGURES

FIGURE Page

2.1 Baseline NoC Router Architecture. 12

2.2 FLOV Router Architecture. 13

2.3 Router Power State Transition Diagram. 15

2.4 An Example of rFLOV in Timeline from (a) to (f). 17

2.5 FLOV NoC Architecture. 21

2.6 Destination Partitioning in a 2D Mesh Network (a), Turns Allowed/Not Allowed in the Escape Sub-Network (b). 22

2.7 Routing Algorithm Examples: X indicates a power-gated router. 24

2.8 Average Latency, Dynamic and Total Power Comparison for Injection Rates of 0.02 (top row) and 0.08 (bottom row) flits/node/cycle with Uniform Random Traffic. 28

2.9 Average Latency, Dynamic and Total Power Comparison for Injection Rates of 0.02 (top row) and 0.08 (bottom row) flits/node/cycle with Tornado Traffic. 28

2.10 Packet Latency Breakdown (a,b), and Full system evaluations (c,d). 29

2.11 Static Power Comparison of FLOV with RP and Baseline. 33

2.12 Reconfiguration Overhead of RP and Comparison with gFLOV. 35

3.1 Per-Application Intra-Router Latency Distribution (*canneal* in PARSEC Benchmarks) 39

3.2 STT-MRAM Cell Structure 41

3.3 The Relationship between Switching Current and Switching Time for Different MTJ Retention Times 43

3.4 BCH ECC Decoder Block Diagram 44

3.5 Performance Comparison between SRAM and STT-MRAM based Routers under the Same Area Budget 47

3.6 Baseline Router Architecture 48

3.7 Multibank STT-MRAM Buffer 51

3.8 Dual-Bank STT-MRAM Buffer Example (Sequence of Operations: ① ~ ⑤) 52

3.9 A Baseline SRAM Input Buffer (a) and A Dual-Bank STT-MRAM Input Buffer (b) 54

3.10 A General Multibank STT-MRAM Buffer (k: Total Number of Flits Buffered, To Hide n-cycle Write Latencies, n-1 Latches and n Banks Are Needed.) 54

3.11 Circular Queue for Dual-Bank STT-MRAM Buffer (*Assuming all errors are correctable / Sequence: (a) ~(d)*) 55

3.12 Probability of the Number of Bits Flipped (*Note that the sum of error probabilities under a specific residence time is 100 %*) 59

3.13 An Example of a 2-bit Global Counter (GC) Refresh Logic (*Assuming refresh time is 80 cycles (40 ns in 2 GHz)*) 60

3.14 Performance Comparison with Different Synthetic Workloads 63

3.15 Performance Comparison with Different Topologies 64

3.16 Normalized Power Consumption - SRAM/Hybrid/STT-MRAM with Different Refresh Rates (*Low-ECC: Low Refresh Rate (80ns) / Opt-ECC: Optimal Refresh Rate (40ns)*, See Section 3.4.3.1 for details.) 66

3.17 PARSEC Benchmark Results 66

3.18 Sensitivity Analysis 69

3.19 Normalized STT-MRAM Density under the Same Per-Router Area Budget 71

3.20 Comparisons with BLESS and WPF (UR) 72

3.21 Comparisons between Different ECC Schemes (End-to-End vs. Per-Hop) 73

3.22 Normalized Number of Packets Retransmitted under Different ECC Schemes 75

4.1 APPROX-NoC Architectural Overview. 85

4.2 APPROX-NoC Operation Flowchart. 86

4.3 Compression and Decompression of a 6-Word Cache Block. 87

4.4 Approximate Value Compute Logic. 89

4.5 Frequent Pattern Compression. 91

4.6 FP-VAXX Microarchitecture. 92

4.7 The Encoder PMT at Node 3 and the Decoder PMT at Node 6. 93

4.8 DI-VAXX Microarchitecture. 96

4.9 Average Packet Latency Breakdown and Overall Approximation Quality. . 102

4.10 Fraction of Encoded words Breakdown to Exact Compression and Approximation (a) and Compression Ratio Improvement of VAXX (b). . . . 103

4.11 Reduction in Number of Injected Flits. 104

4.12 Throughput Analysis with Different Benchmark Data Traces Under Uniform Random (UR) and Transpose (TR) Traffic Patterns. 106

4.13 Error Threshold Sensitivity Analysis. 108

4.14 Approximable Packets Ratio Sensitivity Analysis. 108

4.15 Application Output Accuracy and Normalized Performance. 110

4.16 Approximate versus Precise Output of Bodytrack. 111

4.17 Dynamic Power Consumption Normalized to Baseline. 112

LIST OF TABLES

TABLE Page

2.1 Simulation Testbed Parameters 27

3.1 CMP System Configuration 61

3.2 SRAM and STT-MRAM Parameters with Different Retention Times *(The Hybrid Buffer scheme utilizes 10 ms.)* 62

4.1 APPROX-NoC Simulation Configuration. 99

1. INTRODUCTION

With advances in technology [1], future multi core systems scaled to 100s and 1000s of cores/accelerators, are touted as an efficient solution for extracting huge performance gains using parallel programming paradigms. However with the failure of Dennard Scaling [2], all the components on the chip cannot be run simultaneously without breaking the power and thermal constraints. Thus future Multicore systems will have to work under strict power envelops. The scaling up of on chip components has also brought upon Networks-On-Chip (NoC) based interconnect designs like 2D mesh. The contribution of NoC to the total on chip power and overall performance has been increasing steadily. Recent studies [3, 4, 5] have shown that NoCs consume about 10% to 36% of the total on-chip power budget. Therefore designing power-efficient NoCs which can deliver the performance required by future multicore systems is critical.

Emerging applications/workloads that are being executed on the future multicore systems can be classified broadly to traditional Chip MultiProcessor (CMP) applications and Big data applications. Traditional CMP applications are usually characterized by low core utilization [6, 7] and lower communication load on the NoC. Standard NoC designs are usually over provisioned with respect to such low communication loads and hence efficient designs that can save power are crucial for CMPs. But in parallel, with the advent of the big data era, a multitude of traditional applications and systems are unable to efficiently process massively large data sets. In sharp contrast to traditional CMP applications, these emerging memory intensive applications in the big data era place a significant amount of stress on the interconnection network for high memory throughput, triggering many designs that try to solve the memory bandwidth issue [8, 9, 10, 11]. Hence designing an interconnection network that can efficiently provide high throughput, has become critical to overall system performance for such applications.

We propose to tackle the issue of over-provisioning in traditional CMPs using a power-saving technique. Next, to get higher performance per watt in future multicore systems with bandwidth intensive workloads, we propose the following two techniques. We plan on utilizing a promising next generation memory technology, namely Spin-Transfer Torque Magnetic RAM(STT-MRAM), to achieve enhanced NoC performance, to satisfy the high throughput demands in emerging bandwidth intensive applications, while reducing the power consumption. In addition, we propose novel techniques that increase the effective throughput without requiring additional network resources by leveraging the approximate computing paradigm in big data applications.

First, we propose to investigate a **light-weight distributed power-gating mechanism** for NoCs to reduce the static power consumption. Static power consumption of the on-chip circuitry is increasing at an alarming rate with the scaling down of feature sizes and chip operating voltages towards near-threshold levels. Previous studies [12, 13, 14, 15, 16] have shown that the percentage of static power in the total NoC power consumption increases from 17.9% at 65nm, to 35.4% at 45nm, to 47.7% at 32nm and to 74% at 22nm. According to this trend, as we reach towards sub-10nm feature sizes, static power will become the major portion of the NoC power consumption.

Power-gating, cutting off supply current to idle chip components, is an effective circuit-level technique that can be used to mitigate the worsening impact of on-chip static power consumption. Due to low average core utilization in most modern workloads [6, 7], significant number of studies have proposed efficient mechanisms for power-gating cores with marginal impact on performance [17, 18, 19]. Some studies [20, 16] have proposed power-gating selected router components in a fine-grained fashion using topology reconfiguration. However limited research [21, 14, 22] has been done regarding mechanisms for power-gating routers, which will reduce NoC static power consumption.

Previous research has been proposed to power-gate routers, either by reacting to the network traffic [14] or based on the power state of the attached core [21]. Significant research at Operating System (OS) level has been proposed for achieving static power savings in CMPs by power-gating idle cores by consolidating the thread executions to fewer cores [17, 18, 19, 23]. Therefore, it is imperative to design router power-gating mechanisms that can work in synergy with OS level core power-gating mechanisms. Router Parking (RP) [21] power-gates routers whose attached cores are power-gated, but requires a centralized fabric manager for network reconfiguration, which creates a huge synchronization overhead, and the whole network has to stall until the reconfiguration is completed. RP also creates a single point of failure if the centralized fabric manager goes down.

We propose Fly-Over (FLOV), a light-weight distributed power-gating mechanism that eliminates the need for centralized control to power-gate routers. FLOV tries to power-gate routers as soon as the attached cores are powered down by the OS, in a distributed manner. Since such a distributed power-gating mechanism may create interconnect partitions without communication paths, FLOV links in power-gated routers are provided to enable incoming packets to travel straight through for network connectivity. Our full system evaluations show that FLOV reduces the total and static energy consumption by 18% and 22% respectively, on average across several benchmarks, compared to state-of-the-art NoC power-gating mechanism while keeping the performance degradation minimal.

Secondly, we plan to design better performance-per-watt NoCs for bandwidth intensive workloads by investigating **STT-MRAM based NoC router designs**. Buffers in NoC routers consume significant dynamic power [24], and this consumption increases rapidly as data flow rates increase. Furthermore, the area occupied by an NoC router is dominated by the buffers [25]. Consequently, designing an innovative buffer structure plays a crucial role in architecting high performance and low power on-chip interconnects.

Input buffers, in NoC routers, are commonly implemented with SRAM because it guarantees fast access speed for read and write operations. However, non-negligible area cost and leakage power consumption of SRAM gives lots of pressure on scalable NoC design. Spin-Transfer Torque Magnetic RAM (STT-MRAM) [26, 27] is a promising next generation memory technology that can replace conventional RAMs due to its near-zero leakage power and high density. Adopting STT-MRAM in NoC has significant merits since an on-chip router can provide larger input buffers under the same area budget compared with conventional SRAM routers. Thus, STT-MRAM input buffers contribute to improving throughput, which results in enhanced system performance with less power consumption, improving the performance per watt return. STT-MRAM is CMOS-compatible, and provides virtually infinite write endurance [28] compared with other memory technologies such as Phase Change Memory (PCM), Flash, and Memristor. This makes STT-MRAM a more viable solution as an on-chip memory that should tolerate frequent write accesses. Besides, STT-MRAM is immune to the radiation induced soft errors, thus providing robust cell storages, and can scale beyond 10 nm technology [29]. However, the weaknesses of STT-MRAM, long latency and high power consumption in write operations and thermal fluctuation-induced random bit flips, should be properly addressed because fast accesses to on-chip memories that guarantee data integrity must be assured for high performance and reliable NoCs.

In this work, we propose the first NoC router design that uses only STT-MRAM in buffers, while preserving data integrity. By eliminating SRAM, it offers much larger buffer space with less power consumption. To hide the multicycle write latencies of STT-MRAM, we propose a novel pipelined input buffer design, a multibank STT-MRAM buffer, which is a virtual channel (VC) with multiple banks where every incoming flit is delivered to each bank alternately via a simple latch inside a router. Through this, we can avoid performance degradation while consuming less area and power. To ensure data integrity under the limited retention time and random bit flips of STT-MRAM, we propose cost-efficient dynamic buffer refresh schemes, the processes in which cells' values are kept valid by triggering refreshes in a timely manner. Our evaluations show 20.7% throughput improvement and 17% total power saving compared to a conventional SRAM based router with the proposed STT-MRAM router scheme.

Thirdly, we observe that hardware data approximation techniques can be a potential solution to tackle the memory bandwidth issue in NoCs. This is abetted by the fact that Approximate Computing [30, 31, 32, 33] has emerged as an attractive alternate compute paradigm by trading off computation accuracy for benefits in both performance and energy efficiency. Approximate techniques rely on the ability of applications and systems to tolerate imprecision/loss of quality in the computation results. Various applications in machine learning, image/video processing and pattern recognition have already employed approximation to achieve better performance [34, 35, 36, 37, 38]. Hence we propose to leverage the inaccuracy allowed in applications to reduce the effective communication load in the NoC by transmitting approximate versions of data.

Previous research has proposed several approximation techniques for emerging data-intensive applications. Software approximation mechanisms [39, 40, 41] have attempted to reduce the computation overhead by approximately executing particular sections of application code. Hardware mechanisms, that either advocate approximate computation or storage, propose to tradeoff accuracy for high performance and energy efficiency. These hardware techniques can be broadly categorized into compute-based or memory-based approximation. Compute-based approximation techniques use inexact compute units [33, 42, 43] or neural network models [35, 44, 45, 46] for code acceleration. Memory-based techniques [47, 32, 48] exploit data similarity across memory hierarchies to achieve larger capacity and energy efficiency. Most of these techniques operate by requiring the programmer to annotate portions of an application that can be approximated and then the compiler can exploit the underlying hardware approximation techniques available. A significant portion of research on hardware approximation techniques has focused on either the computation units for accelerated inaccurate execution, or the storage hierarchy (cache/DRAM-based) for low overhead (area/power) memory. However, there has been no prior research on approximate communication techniques for the interconnection fabric of multicore systems.

In this work we propose APPROX-NoC, a data approximation framework for NoCs to alleviate the impact of heavy data communication stress by leveraging the error tolerance of applications. APPROX-NoC proposes to reduce the transmission of approximately similar data in the NoC by delivering approximated versions of precise data to improve the data locality for higher compression rate. The proposed framework operates by first utilizing an approximation engine, with a lightweight error control logic, to approximate the given data block to the nearest compressible reference data pattern. Then the encoder module of an underlying NoC compression technique [49, 50] is used to compress the data block. We propose a data-type aware value approximatiion technique (VAXX), with a light weight error margin compute logic, which can be used in the manner of plug and play module for any underlying NoC data compression mechanisms. *VAXX* approximates the value of a given data block to the closest compressible data pattern based on the data type,with fast quantitative error margin calculation. The error threshold to control the extent of data approximation allowed can be determined by the compiler or annotated by the programmer and can be dynamically adjusted at run time. Our evaluation results show that the best APPROX-NoC mechanism reduces the average packet latency up to 21.4% over state-of-the-art NoC data compression mechanism. In addition, our evaluation results with synthetic workloads show that the best APPROX-NoC mechanism improves throughput up to 60% compared to state-of-the-art compression mechanisms.

2. DISTRIBUTED POWER GATING MECHANISM

2.1 Introduction

The failure of Dennard Scaling [2], supply voltage not scaling down with the transistor size, means that all the components on the chip cannot be run simultaneously without breaking the power and thermal constraints. Thus future CMP designs will have to work under stricter power envelops. Recent studies [3, 4, 5] have shown that NoCs consume a significant portion, ranging from 10% to 36%, of the total on-chip power budget. Hence power-efficient NoC designs are of the highest priority for power-constrained future CMPs. We observe that static power is becoming a significant portion of the total NoC power consumption as technology shrinks and hence it is critical to control static power consumption to satisfy the power envelops of future multicore systems.

We propose Fly-Over (FLOV), a light-weight distributed power-gating mechanism that eliminates the need for centralized control to power-gate routers. FLOV tries to power-gate routers as soon as the attached cores are powered down by the OS, in a distributed manner. Since such a distributed power-gating mechanism may create interconnect partitions without communication paths, FLOV links in power-gated routers are provided to enable incoming packets to travel straight through for network connectivity.

Specifically, FLOV comprises FLOV router architecture, handshake protocols, and its partition-based dynamic routing algorithm. We design FLOV router architecture by modifying the baseline router architecture to provide FLOV links over power-gated routers. Based on this FLOV architecture, we first present a handshake protocol working under restricted conditions, called *restricted FLOV (rFLOV)*, where no consecutive routers in a row/column can be power-gated at the same time. Then another handshake protocol, called *generalized FLOV (gFLOV)*, is presented, where two or more consecutive routers in a row/column can be power-gated simultaneously. Clearly, rFLOV is simpler than gFLOV, but gFLOV can provide more power saving capability. Note that a power-gated router does not have routing functionality and incoming packets can only travel in the same direction. Thus, without prior knowledge about such power-gated routers in a packet's path, localized routing decisions cannot ensure the packet's delivery to the destination. Therefore, we propose a dynamic routing algorithm that ensures network routing functionality without the need for any global NoC information or needing to wakeup intermediate power-gated routers. The routing algorithm dynamically decides the output direction based on the destination and the power states of its neighboring routers.

We evaluate the FLOV scheme using BookSim [51], a cycle-accurate interconnect simulator, for detailed NoC evaluation and using gem5 [52] for full system evaluation, and compare against RP [21]. Our full system evaluations show that FLOV reduces the total and static energy consumption by 18% and 22% respectively, on average across several benchmarks, compared to state-of-the-art NoC power-gating mechanism while keeping the performance degradation minimal.

The rest of this chapter is organized as follows. The related work is briefly summarized in section 2.2. The baseline NoC router and FLOV router architectures are described in section 2.3, followed by two handshake protocols in section 2.4. In section 2.5, the dynamic routing algorithm is explained. Evaluation of the proposed design is presented in section 2.6 and, finally, we draw conclusions and mention future work in section 2.7.

2.2 Related Work

Recently significant research [17, 53] has been performed in applying power-gating techniques in NoCs for power savings. Kim et al. [54], Soteriou et al. [55], Matsutani et al. [20], Kim et al. [56] and Parikh et al. [16] propose fine grained power-gating of components inside the NoC router. But such approaches require significant additional power gating circuitry. Kim et al. [54] proposed a dynamic link shutdown (DLS) technique together with dynamic voltage scaling to save link energy. Soteriou et al. [55] proposed a power-aware network that reduces static power consumption by monitoring the link utilization and power-gating the underutilized links. Matsutani et al. [20] applied the power-gating technique to individually control the power supply of different components in an ultra fine-grained way. Kim et al. [56] proposed a buffer organization to adaptively adjust active buffer size with a power gating technique. Parikh et al. [16] came up with power-aware routing and topology reconfiguration to minimize detours while selected components in routers are power-gated. This feedback-based mechanism is slow, and reconfiguration takes place only on per epoch basis. Power-gating components inside a router in a fine-grained way fashion requires additional circuitry. These approaches work well to reduce the static power consumption, however, they only power-gate certain components of a router.

In [57], lookahead routing is utilized to wake up sleeping routers two hops in advance to hide the wakeup latency. However, as clock frequency increases, wake up latency cannot be totally hidden. Chen et al. [22] introduced a performance-aware, non-blocking In [22], Chen et al. introduced a performance-aware, non-blocking power-gating scheme that wakes up powered-off routers along the path of a packet in advance, thereby preventing the packet from suffering router wakeup latency. Catnap [58] proposed a mechanism where a light-weight subnetwork can be power-gated based on the priority and predicted traffic load. This work is orthogonal with FLOV, since FLOV can be applied on top of the powered-on subnetworks to achieve even more power savings.

Chen et al. [14] proposed a node-router decoupling (NoRD) approach to leverage the independence of power-gating a core and its attached router. They provide a decoupling bypass route that connects the ejection and injection channels to form a bypass link to the router. The decoupling bypass links ensure network connectivity even for the extreme cases of all routers being turned off by using an escape ring network. However, a bypass ring is not scalable to large network sizes. Another issue with NoRD is that a bypass can be constructed in a $(k \times k)$ mesh, if and only if k is even.

Samih et al. [21] proposed Router Parking (RP) to power-gate as many routers as possible when their attached cores are sleeping while maintaining network connectivity. RP dynamically parks (or power-gates) routers to maintain a balanced trade-off between power saving and performance. However, this scheme requires centralized control using a Fabric Manager (FM) and typically takes a long time to reconfigure the network that may suspend new injections into the network during this phase. On the other hand, FLOV is a distributed power-gating mechanism that avoids the need for centralized control and keeps the network functionalities while routers are being power-gated.

Zhan et al. [59] propose a mechanism that can activate powered down cores for performance gains while considering thermal aware floor planning and to this order they also explore topological/routing support. Some studies have proposed bypass style mechanisms for different purposes in NoCs [60, 61, 62, 63, 64]. Kumar et al. [60] proposed express virtual channels that virtually bypass intermediate routers for packet transmission to achieve high performance. In [61], dual functional physical channel buffers were proposed to bypass a router and keep packets in the links along the path. Long-range link [62, 63] and skip-link [64] were proposed to bypass routers for faster packet delivery. Unlike these studies, FLOV stands from a power saving perspective with performance-aware considerations. FLOV links in a router act as a simple connector between the upstream and downstream routers, thus making them logical neighbors for credit-based flow control. A flit entering a FLOV link already has a buffer slot allocated in the downstream router and does not take risk of creating protocol deadlocks.

2.3 FLOV Router Architecture

This section explains the baseline NoC router architecture, and proposes the FLOV router architecture.

2.3.1 Baseline NoC Router Architecture

The baseline microarchitecture is based on a state-of-the-art 3-stage virtual-channel router [65]. Figure 2.1 shows the main building blocks of the baseline router: input buffers, routing computation logic, VC allocator, switch allocator, and crossbar. The processing inside a router is pipelined into 3 stages: Routing Computation (RC), VC Allocation and speculative Switch Allocation (VASA), and finally Switch Traversal (ST). The output port to which a packet should traverse is computed in the RC stage based on the destination information in the head flit. In the VASA stage, an available VC in the next downstream router is assigned to this packet based on the credit information. At the same time, speculative arbitration between the inputs and outputs of the crossbar is processed in parallel. The flits with an assigned VC and the successfully granted switch will traverse the crossbar in the ST stage. Finally, Link Traversal (LT) is external to the router pipeline and is also assumed to take one clock cycle. Wormhole switching along with credit-based flow control is used in this study.

2.3.2 FLOV Router Architecture

As shown in Figure 2.2, the FLOV router architecture has multiplexers and demultiplexers added to input/output links, in addition to a latch in each direction. When a FLOV router is powered-on, it functions like a baseline 3-stage virtual-channel router [65], and the muxes/demuxes are set to 0 as well as the latches are power-gated. When the router is power-gated, all the components of the baseline router are power-gated and the muxes/demuxes are set to 1 to activate the FLOV links. For the routers placed on the

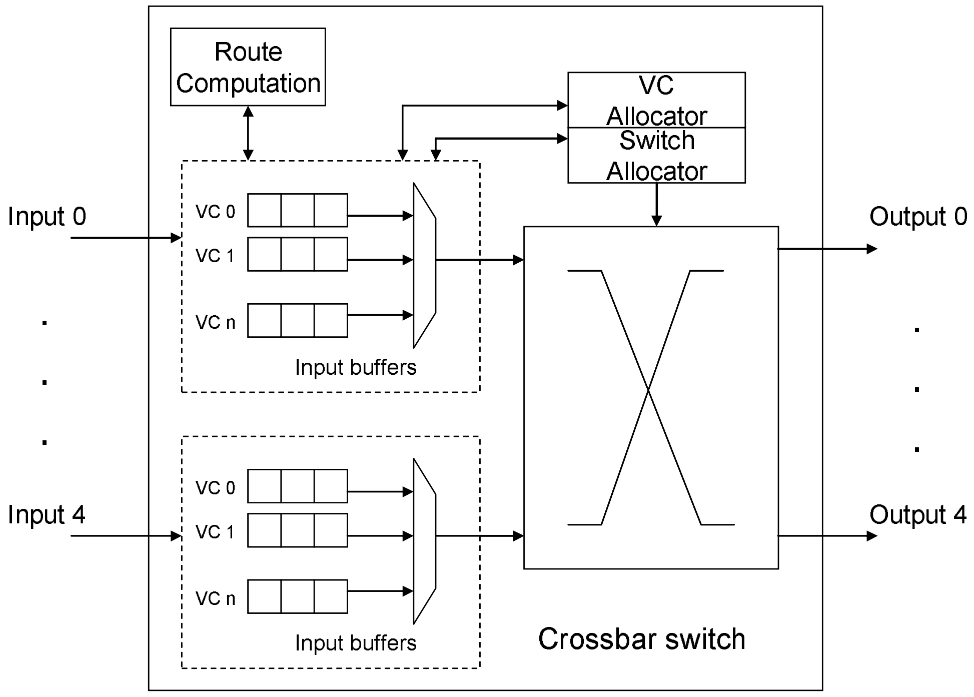


Figure 2.1: Baseline NoC Router Architecture.

edges of the 2D mesh, the FLOV links are activated only in the dimension (X or Y) where there are neighbors in both directions. The Routers on the four corners of the 2D mesh do not have any FLOV links, since they can be isolated once they are power-gated. The HandShake Control logic (HSC) block is introduced, connecting to all the neighboring routers, which implements the handshake protocol between adjacent routers required before power-gating a router. Two sets of Power State Registers (PSRs) hold the power states of the immediate neighboring routers and the nearest powered-on routers (logical neighbors) in each direction, respectively. PSRs for logical neighbors are only used in the complex gFLOV power-gating mechanism described in Section 2.4.2. The Credit Control Logic (CCL) is modified to interact with HSC so as to always hold the buffer availability (credit) information of the nearest powered-on downstream router.

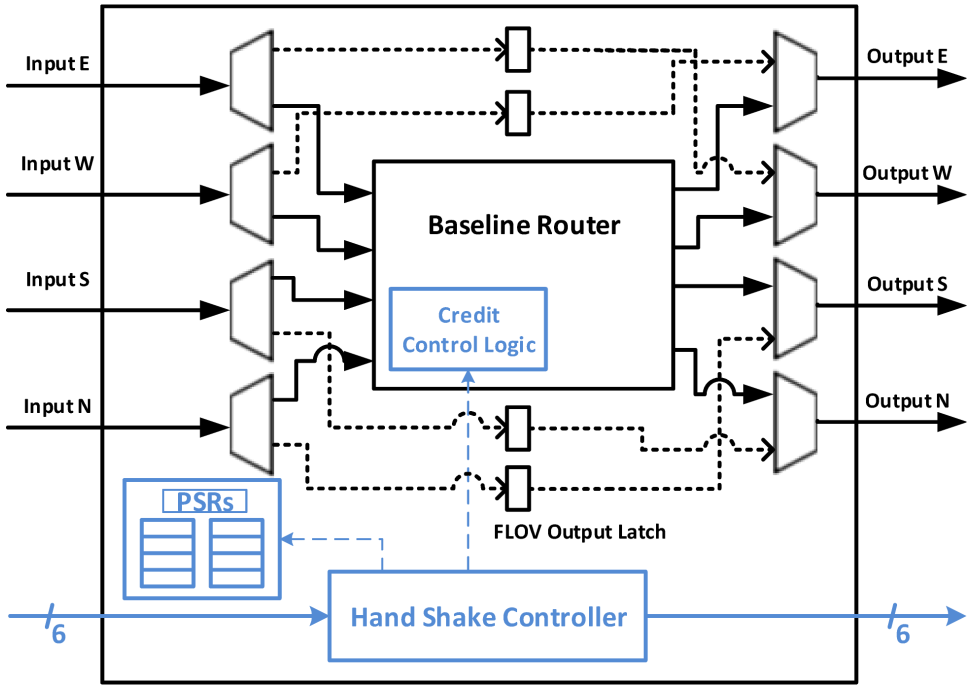


Figure 2.2: FLOV Router Architecture.

2.4 Restricted FLOV and Generalized FLOV Handshake Protocols

Using the FLOV router architecture in Section 2.3, two handshake protocols for FLOV routers are proposed: restricted FLOV (rFLOV) and generalized FLOV (gFLOV). rFLOV has a simpler protocol but its power saving is limited, while more complex gFLOV shows better power saving.

2.4.1 Restricted FLOV

In this scheme, when a core is powered down, its attached router waits for packets coming from the core or going to the core for a certain number of cycles. The state transition diagram in Figure 2.3 depicts the power states a router can be in. If there are no packets detected, the router sends a signal to its neighbors using out-of-band control lines to indicate that it is in the *Draining* state. During this state, its neighbors cannot initiate any new packet transmission to this router, while they are allowed to finish current packet deliveries.

In rFLOV, no two consecutive routers in a row/column are allowed to be powered down. Therefore, if a router in the *Draining* state receives the same signal from its neighboring router, only one of them with a smaller router id is allowed to proceed, while the other is back to normal (*Active* state). Hence, even though the attached core is powered-down, a router is not allowed to drain if one of its neighbors is in draining or sleeping.

A router in *Draining* checks its input buffers for any residing flits and continues to forward them to downstream routers normally. After emptying all its input buffers and receving drain_done signals from all its neighbors, the router power-gates itself by shutting down the baseline router portion (*Sleep* state). Meanwhile, all the muxes/demuxes are switched to 1, and the router sends a signal to all neighbors so that new packet transmission can be initiated and the neighbors can update their immediate neighbor PSRs.

Once the FLOV router is power-gated, a flit coming into the router is stored in the FLOV output latch without any routing/arbitration. In the next cycle, it is delivered to a designated VC in the downstream router since the VC was already calculated in the upstream router. From the downstream router, the packet delivery becomes normal. When an FLOV router is in the *Sleep* state, the credit counts of its downstream router are copied to the upstream router so that the upstream router can get the correct credit information of the downstream router.

A powered-down FLOV router wakes up when its core becomes active or its neighbor has a packet destined for its core (*Wakeup* state). When a currently sleeping FLOV router wakes up due to aforementioned conditions, it first signals its neighbors to stop new packet transmission. After finishing current packet deliveries and emptying its output latches, the FLOV router powers on the baseline router portion and switches the muxes/demuxes to 0. During *Wakeup*, the FLOV router still relays credit counts of its downstream router to its upstream router. However, once becoming *Active*, the router receives credit information from its downstream router, and its upstream router sets the corresponding credit to fully available.

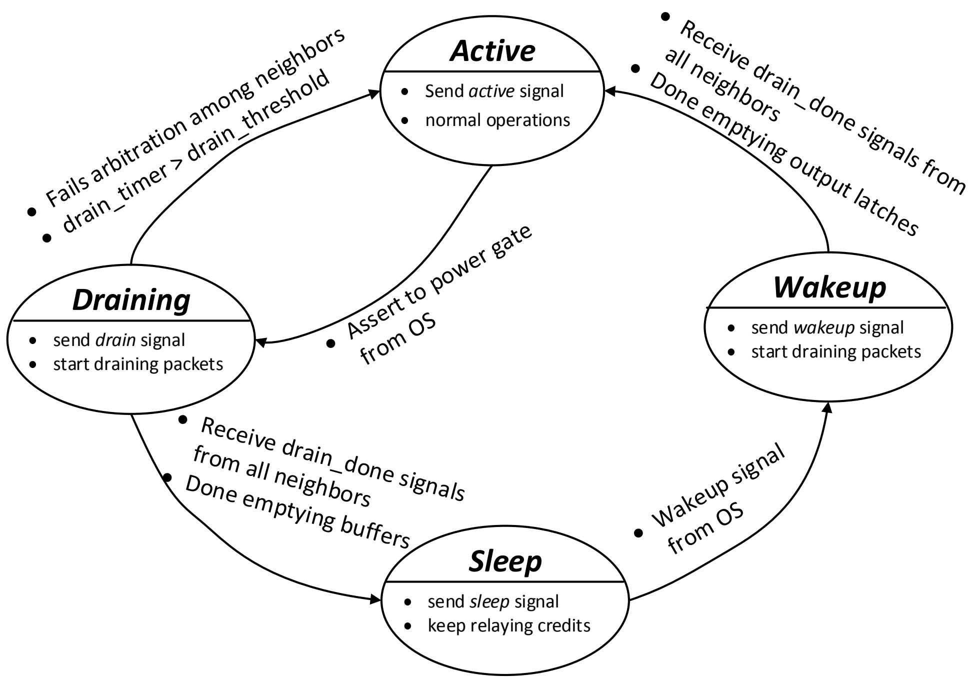


Figure 2.3: Router Power State Transition Diagram.

The state transition diagram in Figure 2.3 depicts the power states a router can be in. Each state is represented in a circle, where the operations performed by the router in that state are also shown. The conditions that trigger state transitions are depicted on the transition arrows. As explained above, the router goes into *Draining* from *Active* when it wants to be power-gated. The router immediately sends a drain signal to its neighbors and starts to drain packets in its input buffers. Routers that want to drain at the same time but fail to win arbitration with their neighbors come back into *Active*. Routers may be forced to go back to *Active* when the time spent in the *Draining* state exceeds a certain predetermined threshold (drain_threshold), which is set empirically. This is done to avoid protocol-level deadlocks when the router is trying to drain packets that depend on packets from the neighbor router making progress towards the current router. So when the drain time threshold is reached, the router goes back to *Active* and the packets can make forward progress.

Once all the router's neighbors finish any intermittent transmissions destined to it and the packet draining is finished, the router can go into *Sleep*. In the *Sleep* state the router sends a sleep signal to all its neighbors after turning off the baseline router operation and starting the FLOV operation. The router starts relaying credits between its powered-on neighbors.

The router goes to *Wakeup* from *Sleep* when its core is powered on. Then it sends wakeup signals to its neighbors and starts draining packets residing in its output latches. Once draining is done, the router goes into *Active*. After entering the *Active* state, the router sends an active signal and resumes normal router operations.

Figure 2.4 shows a working example of the rFLOV protocol. For simplicity, draining of the packets and credit control are shown only for one direction, but a router has to perform these actions for all its neighbors before state transitions.

- In Figure 2.4 (a), all three FLOV routers are *Active*. Router A holds the body (B1) and tail (T1) flits of packet 1 as well as the head flit (H2) of packet 2. Router B holds the head flit(H1) of packet 1 and Router C is empty. The PSR entries of the routers show the power states of the immediate neighbors in the East (Routers A and B) or West (Router C). The current credit status of VC1 of the downstream routers is also shown. The shaded portion indicates the power-gated components that are the output latches here.

- In Figure 2.4 (b), both Routers B and C send Drain signals to their neighbors to

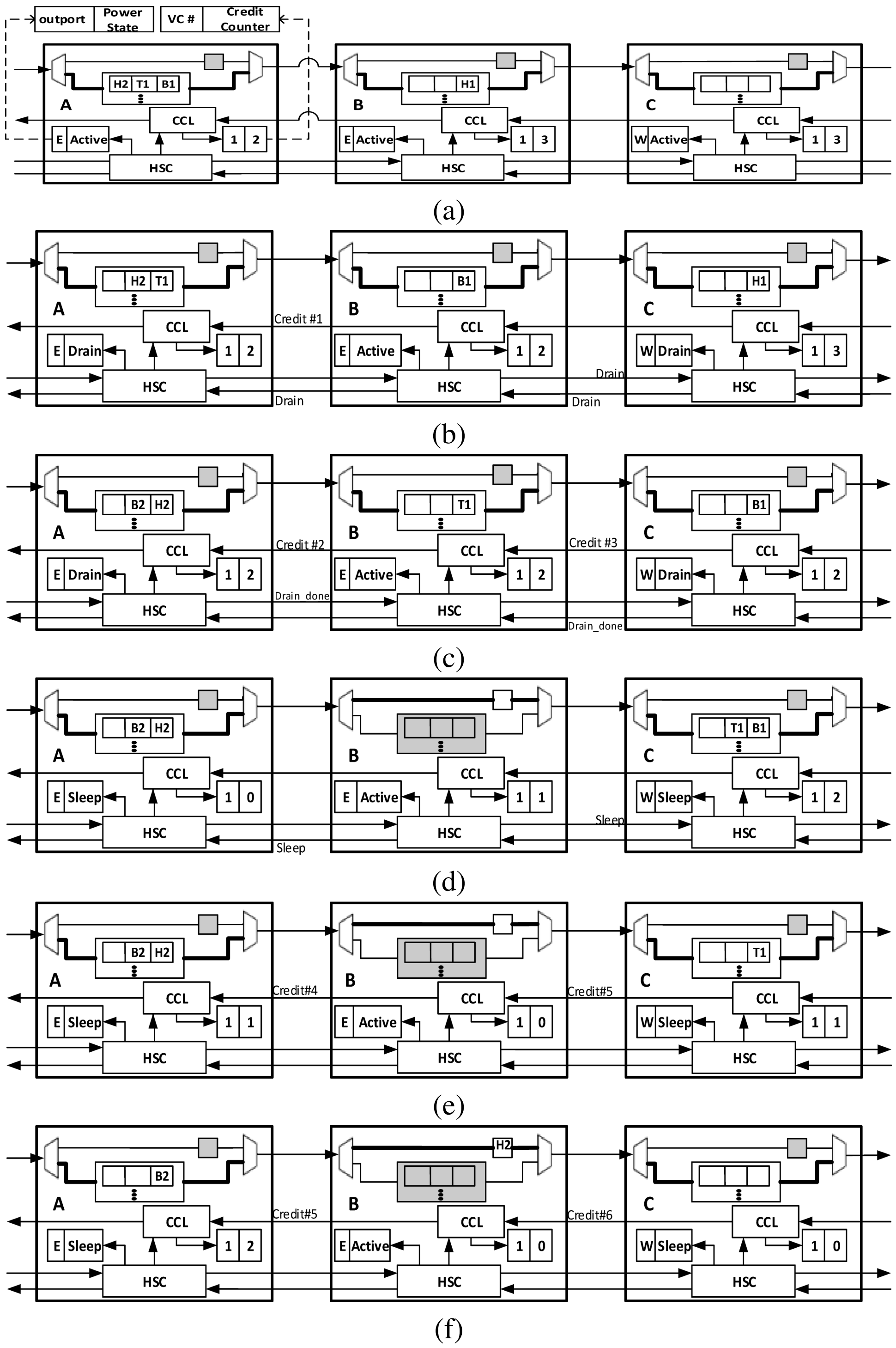


Figure 2.4: An Example of rFLOV in Timeline from (a) to (f).

indicate their willingness to go into the *Draining* state. Since Router B has the lower router id, it wins the arbitration and Router C has to go back into the *Active* state. The PSR entries in Routers A and C are updated to *Drain* due to Router B. Router A transfers flit B1 to Router B and B transfers flit H1 to Router C. The corresponding credit counters are updated as shown.

- In Figure 2.4 (c), Router A sends the drain_done signal to Router B indicating that it finished transmitting packet 1 to B. Similarly, Router C sends the drain_done signal to B. But since Router B has not finished draining its buffers yet, it has to wait before going into the *Sleep* state.

- Figure 2.4 (d) depicts the situation after Router B finishes draining packet 1 to Router C and goes into the *Sleep* state. The shaded VC buffer indicates that the baseline router has been power-gated and the FLOV links (output latches) have been activated. Router B sends the *Sleep* signal to its neigbhors so that they can update their corresponding PSR entries and also the credit counters are zeroed as shown in Router A. Note that even though Router A had a flit (H2) to send Router B, it has to wait until B finishes its power state transition.

- Figure 2.4 (e) shows the credit control and maintenance between Routers A and C while Router B is power-gated. After Router B goes into the *Sleep* state, Router A zeroes its credit counter entry and the credit information is copied from Router B to A (Credit #4). This is because Router C is now logically the downstream router of Router A, so A has to keep track of the buffer availability (credits) in C. Credit #5 carries the newly available credit in Router C to Router B.

- In Figure 2.4 (f), we can see how the Credit #5 is relayed by the power-gated Router B to Router A, which then updates its credit counters. This is how Router A can keep track of the credit status of Router C via the relaying scheme in Router B.

The wakeup procedure is similar with the draining procedure, since a waking up router sends wakeup signals to its neighbors and starts to drain packets from its output latches. The router also waits for all its neighbors to finish any intermittent transmissions destined to it and sends drain_done signals. The router then receives the credit information from the downstream router and sends a signal to notify the upstream router to make its corresponding credit counter to fully available. Once this happens, the router switches the muxes/demuxes and resumes baseline operations.

2.4.2 Generalized FLOV

Power saving is limited in rFLOV since, when a router goes to sleep, none of its neighbors are allowed to sleep regardless of the power states of their attached cores. In this section, we propose *generalized FLOV (gFLOV)* where two or more consecutive routers in a row/column can be power-gated simultaneously.

The main challenge of gFLOV in comparison with rFLOV is the added complexity of handshaking between routers so as to keep consistent PSRs and maintain the credit information of downstream routers. This is because, unlike in rFLOV, consecutive routers can be power-gated, the handshake signaling between two active routers (logical neighbors) may need to cross several power-gated routers. In rFLOV, there is no need for handshake relaying because the handshaking occurs always between two immediate (physical) neighbors, whereas when a router wants to drain/wake up in gFLOV, it has to handshake with the nearest powered-on router in each direction (if there is one), which is its logical neighbor. The power-gated routers in the middle should forward the handshake signals, in addition to updating their corresponding logical and physical neighbor routers' power states in the PSRs.

The credit control is similar with rFLOV, where the power-gated router is responsible for copying its credit counters to its upstream router. Since there might be multiple consecutive power-gated routers in the middle, the credit information is relayed across these sleeping routers until it reaches a powered-on upstream router. Like rFLOV, a router that wakes up will receive credit information from its downstream router and the upstream router sets its credits to full availability.

The handshake protocol of gFLOV requires some protocol level restrictions and additional functionalities, when compared with rFLOV, which are described as follows.

- In gFLOV, after a router finishes power-gating (goes into the *Sleep* state), it should send its corresponding logical downstream neighbor's power state in each direction to its upstream router, in addition to its current power state. This is because the logical downstream router of the power-gating router will now become the logical downstream router for its upstream router. This way the logical PSRs of all the routers are kept up-to-date.

- In gFLOV, no two logical neighbor routers in the same row/column are allowed to stay in *Draining-Draining* or *Draining-Wakeup* state combinations at the same time in order to avoid protocol deadlock. Since *Wakeup* is more crucial for performance, *Draining* has lower priority if one of the handshaking routers is trying to wake up and the other trying to drain. However, for simplicity of handshaking, if a power-gated router has a downstream router in the *Draining* state, it cannot wake up until the draining router changes its state. Similar with rFLOV, if the handshaking routers are trying to drain at the same time, only the one with a smaller router id can proceed.

- Two routers in the same row/column can wake up at the same time in gFLOV. Unlike the *Draining-Draining* combination, two waking up routers have no dependence on each other. Any of the handshaking *Wakeup* routers should relay the drain_done handshake signal to the other *Wakeup* router.

2.5 Dynamic Routing Algorithm

In this section the overall FLOV NoC architecture is introduced and the dynamic routing algorithm is proposed.

2.5.1 FLOV NoC Architecture

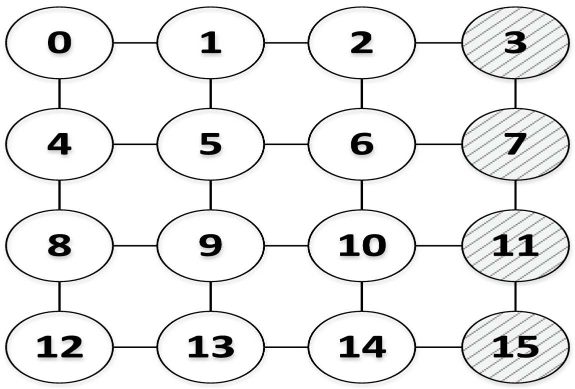


Figure 2.5: FLOV NoC Architecture.

Figure 2.5 shows a (4×4) 2D mesh network with the proposed FLOV routers. The pattern-shaded routers (3, 7, 11, and 15) are connected to memory controller (MC) nodes that should be never power-gated [1], where we use the baseline routers. All the other routers are FLOV routers that are connected to processing cores and can be power-gated if the cores are powered down. Maintaining connectivity in the network without any global information, which is critical to FLOV, is ensured by a combination of keeping all the routers in the last column powered-on and the proposed routing algorithm below. One VC of each powered-on router is reserved for deadlock recovery, called an escape VC.

[1]MC nodes can be located in other places. Depending on this MC placement, the routing algorithm may be slightly different.

2.5.2 Dynamic Routing Algorithm

The FLOV NoC baseline architecture is a two dimensional mesh topology with one column or row of routers (on the edge) which are always powered on. This is to facilitate connectivity across the topology using our routing algorithm which is explained below. One VC of each powered-on router is reserved for deadlock recovery, called an escape VC. The proposed routing algorithm consists of routing for packets in the regular VCs and routing for packets in the escape sub-network. A packet in a regular VC can be sent to an escape VC when required by the deadlock recovery mechanism. Note that routing computation is performed in powered-on routers, while power-gated routers only forward packets without changing the direction.

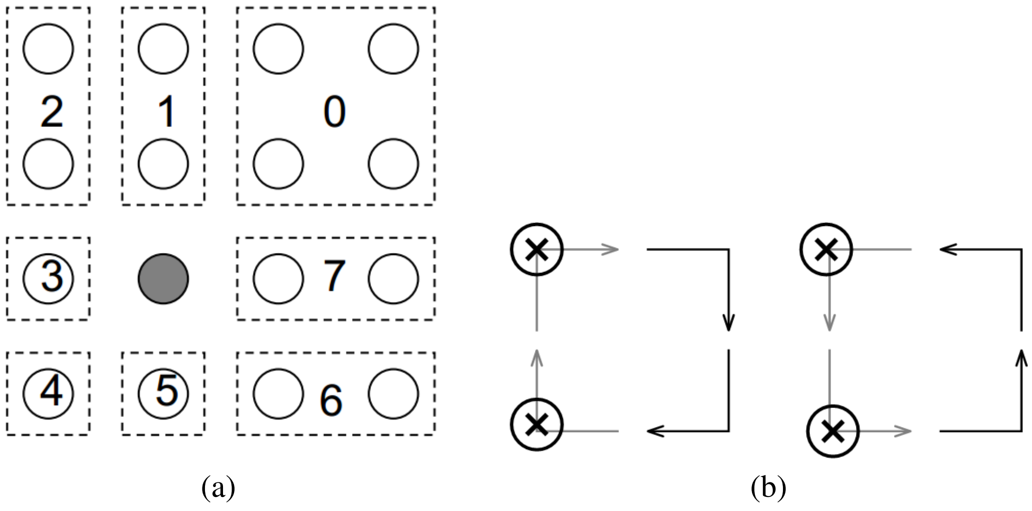


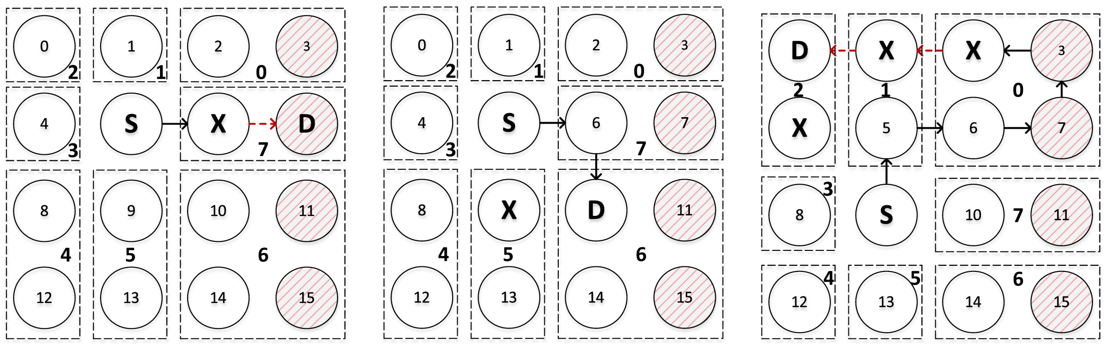
Figure 2.6: Destination Partitioning in a 2D Mesh Network (a), Turns Allowed/Not Allowed in the Escape Sub-Network (b).

We propose a partitioned-based dynamic routing algorithm based on YX routing for packets in regular VCs. Each router divides the network into partitions as shown in Figure 2.6 (a). The routing decision is made based on two variables, the partition which the destination falls into and the power states of neighboring routers. For packets with destinations in partitions 1, 3, 5, and 7, the router will send them directly to North(Y+), West(X-), South(Y-), and East(X+) downstream routers, respectively. This is because even in case of power-gated downstream routers, FLOV links will ensure the connectivity to the destinations.

For packets with destinations in partitions 0, 2, 4, and 6, the route will include a turn towards the destination. In the proposed dynamic routing algorithm, if the neighboring router in the Y direction is powered-on, the packet will be sent to this router using YX routing. If this neighboring router is power-gated, the router will check the state of the neighboring router in the X direction, and if this router is powered-on, the source router will send the packet to it.

In case both the routers in the X and Y directions are power-gated, a viable route to the destination cannot be guaranteed since the current router is not aware of the power states of the farther downstream routers. Then the packet will be forwarded to the neighbor, in the East direction, toward MC node routers using the FLOV link of the neighboring power-gated router. The packet is not sent to the router in the Y direction because, in the worst case, if all the downstream routers in the Y direction are powered off, the packet will not be able to make a turn and hence cannot be routed to the destination. In contrast, once the packets are directed to the East direction, we can guarantee that the packet will be able to make a turn toward the destination in the always powered-on MC node router of the corresponding row. Noted that a router cannot send a packet back to the direction from which it arrived so as to avoid livelock situations, where a packet keeps bouncing between two neighbors.

The proposed adaptive routing algorithm is not necessarily deadlock-free. We use Duato's algorithm and a timeout mechanism to ensure deadlock recovery in our scheme [66]. If a packet has been waiting in a buffer for a long time, it will exceed a certain threshold



(a) Example 1 (b) Example 2 (c) Example 3

Figure 2.7: Routing Algorithm Examples: X indicates a power-gated router.

and be directed to the escape VC in the downstream routers to reach the destination using the deadlock-free escape sub-network.

The routing algorithm in the escape sub-network is also based on the partitioning from Figure 2.6 (a). Packets with destinations in partitions 1, 3, 5, and 7, will be sent directly to North, West, South, and East, respectively. Packets whose destinations are in partitions 0, 2, 4, and 6, should be sent to East where the MC routers are located for the same reason mentioned above. Figure 2.6 (b) shows the turns that are allowed and not allowed in our escape routing algorithm which ensure deadlock freedom.

The proposed dynamic routing algorithm is explained in details using examples in Figure 2.7.

- In Figure 2.7 (a), the destination is in partition 7 of the source router's partitions, so even though the next router is power-gated, the packet is forwarded to the East using the FLOV link.

- In Figure 2.7 (b), the destination is in partition 6, so the routing algorithm first checks for Router 9's state. Since Router 9 is power-gated, the packet is sent to Router 6 that is powered-on, which will then in turn route the packet to the destination.

- In Figure 2.7 (c), the destination is in partition 2, so Router 5's state is checked. Since it is powered-on, the packet is forwarded to Router 5. Router 5 then executes the same logic and since Routers 1 and 4 are both power-gated, the packet has to be sent to Router 6 so that it can at least make a turn at the MC router. Router 6 computes that the destination is in partition 2 and checks Routers 2 and 5. Since Router 2 is power-gated and it cannot send the packet back to Router 5, the packet is forwarded to Router 7. Router 7 then routes the packet to Router 3 where it makes another turn toward the destination. If the packet wait time in any router exceeds the threshold, it is routed to the escape VC. Once the packet enters the escape VC, it has to remain in the escape sub-network until it reaches the destination.

2.5.3 Overhead Analysis

In this section we discuss the area and power overhead incurred by the proposed scheme. The modifications proposed to the router microarchitecture include 4 multiplexers and 4 demultiplexers in addition to the four output latches. The mux and demux selection signals are only toggled when the router powers on or off, so the logic needed for the select signals is minimal. Every router has two sets of PSRs, where each entry incurs a 2 bit overhead (for power state). Hence the total overhead for the PSRs accounts to 16 bits (2 sets of 4-entry registers). The credit control logic is modified to be connected to the HSC so that the credit counters can be reset or zeroed based on signals from the HSC. The additional overhead incurred due to this is mainly the connecting wires and minor modifications to the CCL logic for decoding the two HSC signals. The HSC requires 6-bit wires to connect the adjacent neighbor routers (4 bits for current and logical neighbor router power state change notifications, 1 bit for draining notification and 1 bit for physical neighbor assertion). This accounts to approximately 0.1% of baseline router area according to our modeling using DSENT [15]. The HSC also includes the power state transition

FSM implementation (4 states), which incurs minimal area overhead. The overall area overhead for the above components for a single router in 32nm technology is quantized at $2.8 \times 10^{-3}\ mm^2$ which is 3% of the baseline router area. The power consumption of the HSC is also minimal due to the handshaking occurring only after long intervals of time (reconfiguration times) as shown in Section 2.6. The power consumption overhead for the handshaking and the credit relaying is accounted for in the DSENT model [15] and is included in the power consumption evaluation results in the next section. None of the modifications incur significant critical path delay and do not impact the frequency of operation of the NoC. This is because the data path of a packet is only impacted by the demuxes and muxes, and they incur negligible delay, therefore not violating the clock cycle time. The modifications to the routing and CCL are minor and will not violate the critical router pipeline stage delay.

2.6 Experimental Evaluation

In this section we evaluate the FLOV mechanism by comparing static, dynamic and total power consumptions in addition to NoC latency with Router Parking [21] [2].

2.6.1 Experimental Methodology

We use Booksim [51] for synthetic workload experiments, and integrate it with gem5 [52] for full system simulation. DSENT [15] is used to estimate static and dynamic power consumptions of the interconnect components with a switching activity of 50% in 32nm technology. A 2GHz clock frequency is assumed for the routers and links. Table 2.1 summarizes the simulation configuration parameters. We use both synthetic and real workloads to evaluate the performance and power-savings of rFLOV and gFLOV against the Baseline interconnects with no router power-gating (Baseline) and Router Parking (RP). We use Uniform Random and Tornado traffic for synthetic workloads and nine benchmarks from

[2]We do not compare with NoRD due to different assumptions on power-gating criteria.

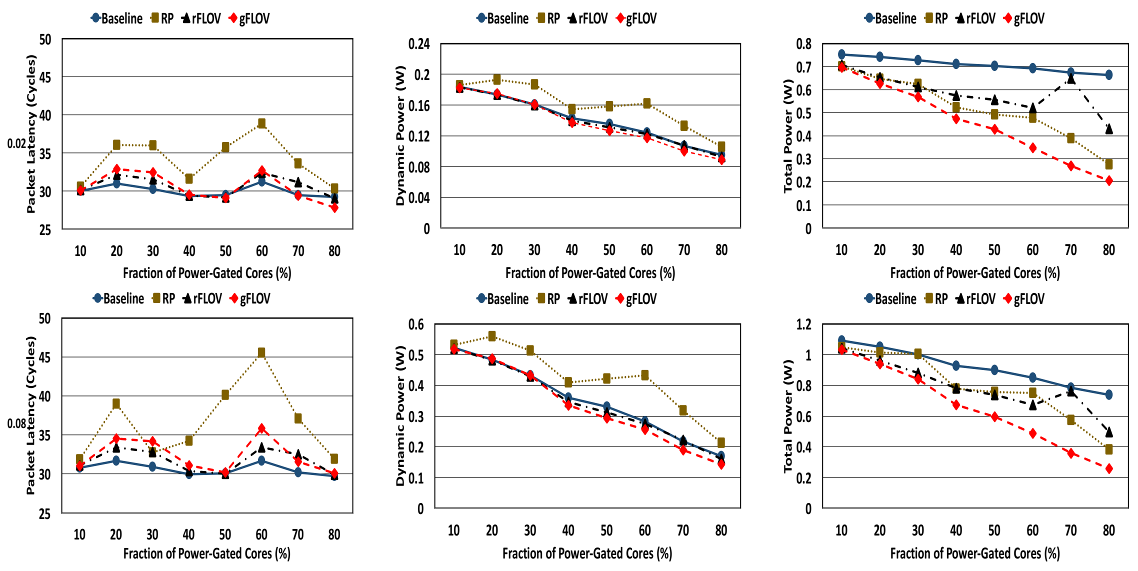
PARSEC benchmark suite [6] for our evaluation.

Table 2.1: Simulation Testbed Parameters

Network Topology	8×8 Mesh
Input Buffer Depth	6 flits
Router	3-stage (3 cycles) router
Virtual Channel	3 regular VCs and 1 escape VC per vent, 3 vnets
Packet Size	4 flits/packet for synthetic workload
Memory Hierarchy	32KB L1 I/D \$, 8MB L2 \$ MESI, 4 MCs at 4 corners
Technology	32nm
Clock Frequency	2GHz
Link	1mm, 1 cycle, 16B width
Power-Gating Parameters	Power-Gating overhead = 17.7pJ wakeup latency = 10 cycles
Baseline Routing	YX Routing

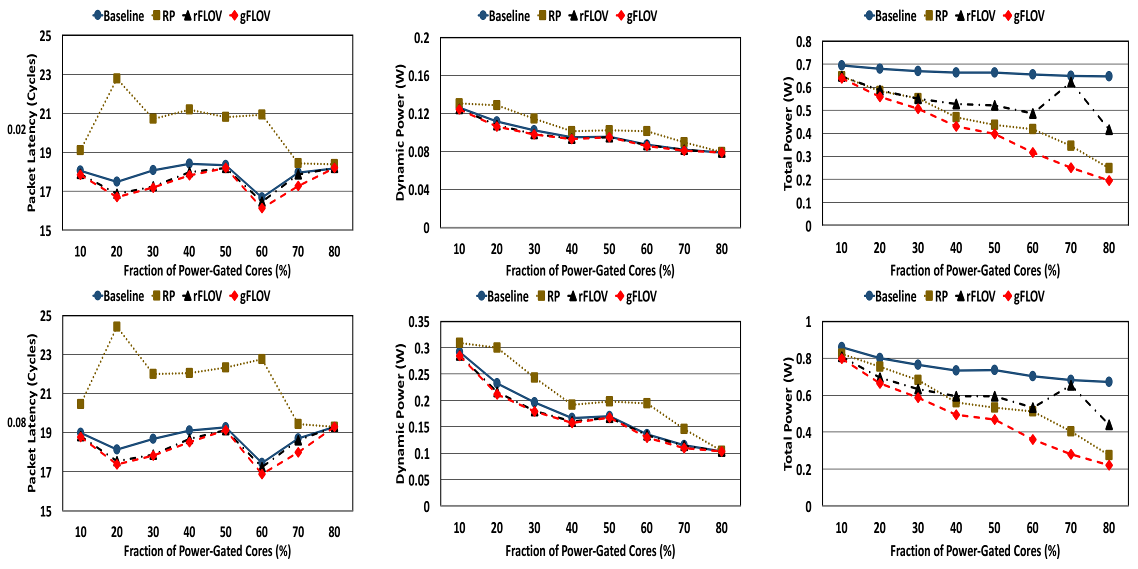
2.6.2 Synthetic Workload Evaluation

For synthetic workloads, we use first 10,000 cycles to warm up the simulation and run for 100,000 cycles in total. Figure 2.8 summarizes the simulation results using Uniform Random traffic. Similarly, Figure 2.9 shows the results for Tornado traffic. In the figures the top row is for the injection rate of 0.02 flits/cycle/router and the bottom row is for the injection rate of 0.08 flits/cycle/router. Each column shows average latency, dynamic, and total power consumptions for a given injection rate, respectively. Figures 2.10(a) and (b) break down average packet latencies of the different mechanisms into accumulated router latency (number of hops $\times$ router pipeline latency), link latency (total link traversals), serialization latency (number of flits per packet) contention latency, and FLOV latency (number of FLOV links traversed). The static power consumption analysis for Uniform



(a) Average Latency (b) Dynamic Power Consumption (c) Total Power Consumption

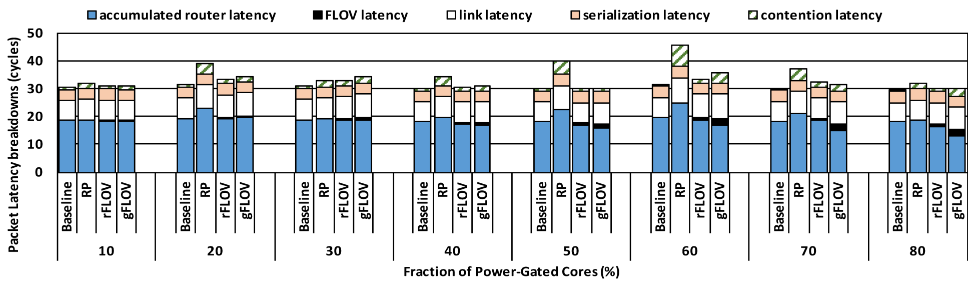
Figure 2.8: Average Latency, Dynamic and Total Power Comparison for Injection Rates of 0.02 (top row) and 0.08 (bottom row) flits/node/cycle with Uniform Random Traffic.



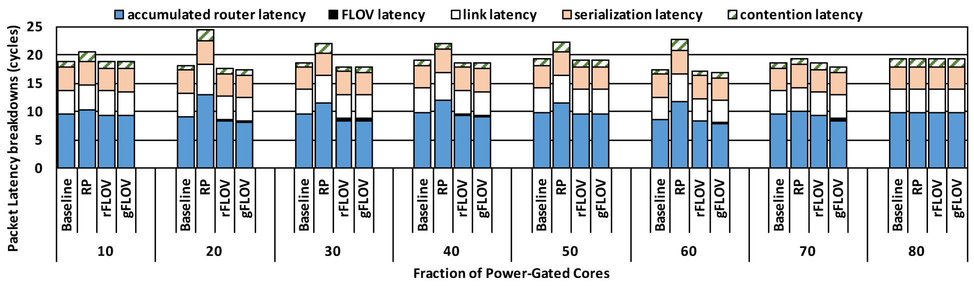
(a) Average Latency (b) Dynamic Power Consumption (c) Total Power Consumption

Figure 2.9: Average Latency, Dynamic and Total Power Comparison for Injection Rates of 0.02 (top row) and 0.08 (bottom row) flits/node/cycle with Tornado Traffic.

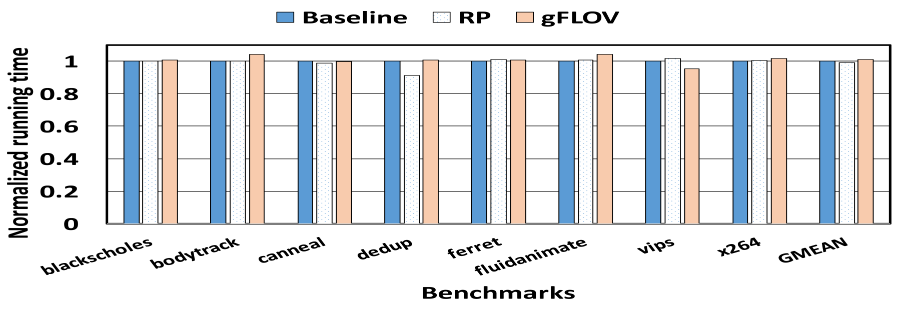
Random and Tornado traffic is shown in Figure 2.11.



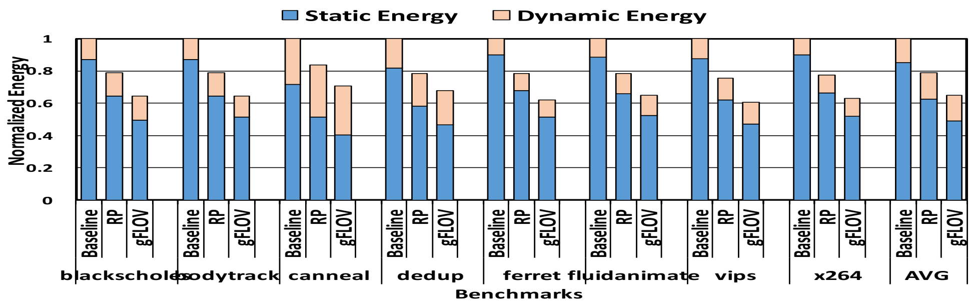
(a) Uniform Random Traffic Pattern (0.08 filts/cycle/node)



(b) Tornado Traffic Pattern (0.08 filts/cycle/node)



(c) Application execution time



(d) Full system energy consumption

Figure 2.10: Packet Latency Breakdown (a,b), and Full system evaluations (c,d).

2.6.2.1 Performance

Figure 2.8 (a) and Figure 2.9 (a) show average latency comparison of rFLOV and gFLOV with RP and Baseline. Both rFLOV and gFLOV perform better than RP across different traffic and injection rates. This is because, in RP, a packet will always need to route through powered-on routers and links connecting them, which may be non-minimal, thereby increasing the path length. In the FLOV mechanism, we take advantage of all the links, thus trying to route a packet through a minimal path using FLOV links. Even when minimal routing is not possible due to the proposed routing algorithm in Section 2.5, the average packet latency can be reduced since the FLOV links do not incur the 3-cycle baseline router per-hop latency, since the flit is only temporarily held in the FLOV latch for one cycle. This can be observed clearly in Figure 2.10(a) and (b), where the accumulated router latency for RP is larger than that of the FLOV mechanism, due to non-minimal detours. In Figure 2.10 (a), under Uniform Random traffic, the FLOV latency increases as more cores are power-gated for the FLOV mechanism, which shows the increased FLOV link utilization. For Tornado traffic in Figure 2.10 (b), the communication occurs between two power-on nodes in the same row/column, and the routers in the rightmost column are always active. Therefore, less number of FLOV links are used, which leads to reduced FLOV latency.

As the number of power-gated cores increases, rFLOV power-gates as many routers as possible under the aforementioned restrictions, and gFLOV power-gates all the routers attached to the power-gated cores, whereas RP makes a dynamic decision based on maintaining network connectivity. When the fraction of power-gated cores is low, rFLOV and gFLOV perform significantly better than RP in terms of average latency due to less detour and fast FLOV links. Also average latencies of rFLOV and gFLOV are similar due to the numbers of power-gated routers being similar at lower fractions of power-gated cores.

However, when the fraction of power-gated cores is high, rFLOV can only power-gate at most half the routers, while gFLOV can do more.

Figure 2.8 (a), at the fraction of 70% power-gated cores, shows a case where gFLOV slightly outperforms rFLOV. This is counterintuitive since lesser number of power-gated routers in rFLOV should generally incur more minimal routing paths and higher network performance. This is due to the reduced per hop latency of FLOV links showing more impact on average latency than minimal routing capability. Figure 2.10 (a) shows that the accumulated router latency for rFLOV is significantly larger compared to gFLOV at 70%, since gFLOV utilizes the FLOV links more. Figure 2.8 (a) shows that the performance of RP becomes closer to the FLOV mechanism as the fraction of power-gated cores becomes larger since the traffic injected into the network becomes very low due to lesser number of active cores. This can be also observed in Figure 2.10(a) and (b), where the contention latency and accumulated router latency for RP decrease as the fraction of power-gated cores goes from 60% to 80%.

Another observation is that as the injection rate increases from 0.02 to 0.08, the performance impact on RP is higher than on rFLOV and gFLOV. This is because certain routers, connecting different network partitions to ensure network connectivity, become network hotspots in RP. Such routers become congested especially at high injection rates, thus creating communication bottlenecks. The proposed dynamic routing algorithm in FLOV avoids such network hotspots.

In Figure 2.9 (a), rFLOV and gFLOV outperform Baseline with Tornado traffic. This is because in Tornado, a significant portion of the traffic injected from each router is destined to a router in the same row/column. Thus rFLOV and gFLOV can use FLOV links with minimal paths and avoid the 3-cycle router latency.

One interesting observation is that, under Uniform Random traffic with an injection rate of 0.08 flits/cycle/router in Figure 2.8 (a), RP shows similar latency as both rFLOV and gFLOV when 30% of cores are power-gated. This is due to the fact that RP dynamically turns on additional routers attached to power-gated cores to negate the impact of higher traffic in the network. This can also be observed from Figure 2.8 (c), where total power consumption is increased when the fraction of power-gated cores goes from 20% to 30%. From these results, it is clear that RP trades off static power savings for latency benefits. This is also shown in Figure 2.10 (a), where the router latency of RP significantly decreases as the fraction of power-gated cores goes from 20% to 30% due to RP powering on additional routers to reduce the non-minimal detour paths.

In Figure 2.10(a) and (b), both rFLOV and gFLOV have relatively higher contention latency at high fractions of power-gated cores. One reason is that packets have higher probability of being routed to the MC column for guaranteed paths to the destinations, which may create congestion in the MC column. Also, when packets are routed through consecutive FLOV links in a row/column, packet transmission may be delayed due to the round-trip latency of credit information. However, the higher utilization of FLOV links compensates for the contention latency, which can be explained by the router and FLOV latencies. Note that RP also tends to have higher contention latency compared to the FLOV mechanism because of the high probability of hot spot creation.

2.6.2.2 Power Consumption

Figures 2.8 (b), 2.8 (c), 2.9 (b), and 2.9 (c) show dynamic and total power consumptions of the FLOV mechanism compared with RP and Baseline for multiple injection rates. In Figures 2.8 (b) and 2.9 (b), for multiple injection rates the dynamic power consumptions of rFLOV and gFLOV are lower than RP, since in RP every hop in the rerouted packet traversal requires the total router pipeline execution, whereas in FLOV the intermediate power-gated routers use FLOV links that consume significantly lower power. RP also consumes more dynamic power than Baseline due to its non-minimal path rerouting of

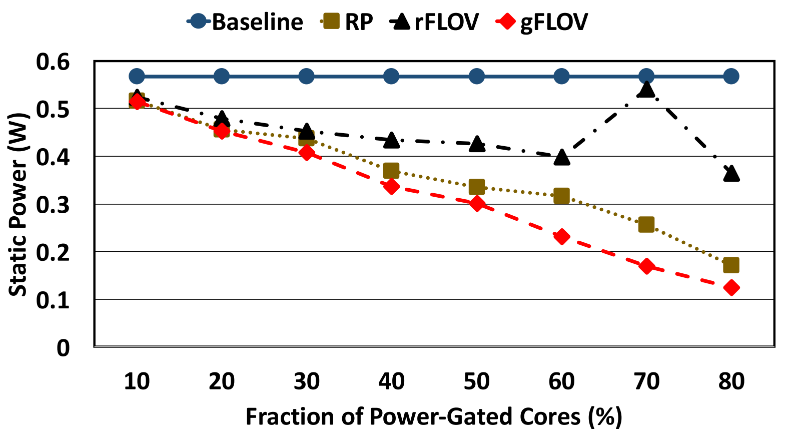


Figure 2.11: Static Power Comparison of FLOV with RP and Baseline.

packets as the number of power-gated cores increases. At higher fractions of power-gated cores, the FLOV mechanism consumes less dynamic power than Baseline due to avoiding the router pipeline execution. Figures 2.8 (c) and 2.9 (c) show total power consumptions of rFLOV and gFLOV compared with RP. It is clear that gFLOV unanimously has lower power consumption, since the dynamic and static power consumptions in gFLOV are lower than RP. Note that total power consumption of rFLOV is higher than RP at higher fractions of power-gated cores, mainly due to static power consumption explained below.

Figure 2.11 shows static power consumption comparison, which is injection rate and workload independent for rFLOV and gFLOV, since all routers attached to power-gated cores are power-gated in gFLOV, while rFLOV power-gates a limited number of routers to preserve the restriction. RP dynamically decides whether to conservatively or aggressively power-gate routers, using power saving versus latency tradeoff prediction based on the interconnect workload. To reduce redundancy of using the same results of the FLOV mechanism for multiple injection rates and workloads, we compare against the aggressive RP power-gating scheme that power-gates as many routers as possible, which will make the RP power results also workload independent. This allows a fair comparison with RP and lets us depict the static power evaluation in Figure 2.11.

In Figure 2.11 the static power consumption of gFLOV is lower than RP and the disparity increases as the number of power-gated cores increases. This is mainly due to the fact that gFLOV power-gates more routers than RP. rFLOV consumes more static power compared to RP, especially as the fraction of power-gated cores increases, since the number of routers that can be power-gated starts to saturate.

2.6.2.3 Real Workload Evaluation

To evaluate the behavior of the power-gating mechanisms under real workloads and show the impact on the full system environment, we run PARSEC 2.1 in gem5 [52] integrated with Booksim. The system parameters are described in Table 2.1. Figures 2.10 (c) and (d) show the execution time and the energy consumption, with the power-gating mechanisms compared to Baseline and RP. They show that FLOV achieves 43% reduction in static energy consumption on average compared to Baseline with only a 1% degradation in performance. This is due to the best-effort shortest-path routing and the low-latency FLOV link compensate for the detouring and round-trip credit loop latency. Interestingly, FLOV and RP have better performance than Baseline in *vips* and *dedup*, respectively. It is the effect of a good match of the traffic pattern and routing algorithm. Compared to RP, FLOV reduces static energy by 22% as a net effect from the distributed power-gating control and the dynamic routing algorithm.

2.6.3 Reconfiguration Overhead Analysis

In this section we analyze the impact of the network reconfiguration on packet latency in RP by comparing with gFLOV. Figure 2.12 shows average packet latency of gFLOV and RP across the timeline of execution using Uniform Random traffic with an injection rate of 0.02 flits/cycle/node when 10% of the cores are power-gated. In RP, whenever the configuration of power-gated cores changes (at 50,000 and 60,000 cycles), the network has to

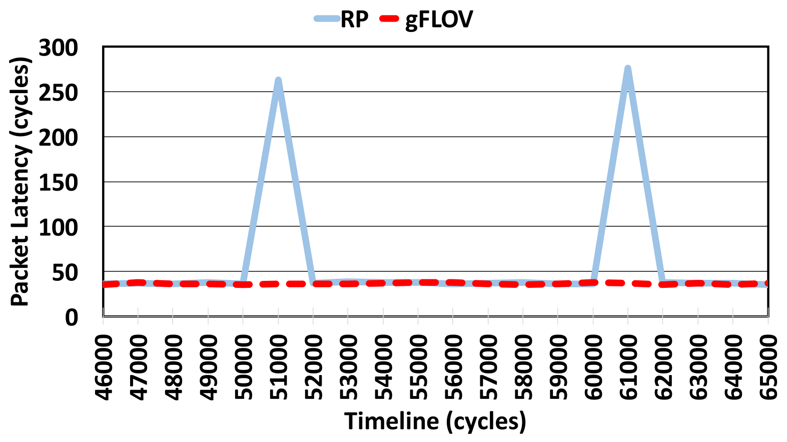


Figure 2.12: Reconfiguration Overhead of RP and Comparison with gFLOV.

be reconfigured by the FM and then the corresponding routing tables have to be distributed to the routers that will be active in the next epoch (Phase I of reconfiguration protocol in RP). While this reconfiguration is performed, the network has to stall and no new injections are allowed except reconfiguration packets, which incurs additional queuing delays in packet latency. Our evaluations show that the reconfiguration time in RP Phase I is more than 700 cycles. The performance overhead due to this is shown in Figure 2.12, where we can clearly observe that the newly injected packets during this time experience significant queuing delays in RP. In gFLOV, there is no such network reconfiguration overhead since the routers are power-gated in a distributed manner. So new packet transmissions can be initiated while some routers either power-gate or wake up independently.

2.7 Conclusions

In this work, we proposed Fly-Over (FLOV), a light-weight distributed router power-gating mechanism for NoCs. After constructing the FLOV router enabling FLOV links by modifying the baseline router microarchitecture, we presented two different handshake protocols for FLOV routers, called rFLOV and gFLOV, and explained the dynamic routing algorithm in details. FLOV power-gates routers attached to powered-down cores without global network information, but still ensures network connectivity.

Performance evaluations using synthetic and real workloads show that FLOV not only achieves better NoC power savings due to power-gating more routers but avoids aggregated traffic rerouting in the network unlike Router Parking. Also, average latency is reduced compared with Router Parking. We show that FLOV reduces the total and static energy consumption by 18% and 22% respectively, on average across several benchmarks, compared to state-of-the-art NoC power-gating mechanism while keeping the performance degradation within 1%.

We plan to extend our mechanism to aggressively power-gate routers, to achieve more power savings in domains such as CMPs with shared last level caches (LLC) and General-Purpose Graphics Processing Units (GPGPUs). The FLOV router can be enhanced to include injection/ejection capabilities so as to facilitate network traffic based fine-grained power-gating like NoRD [14]. We also plan to combine FLOV with lookahead routing [67] so that more aggressive 1- or 2-stage routers can be used for our study.

3. POWER-EFFICIENT AND RELIABLE ON-CHIP INTERCONNECTS USING STT-MRAM ROUTERS

3.1 Introduction

NoCs should be carefully designed due to the inherent constraints of the restricted power and area budgets in a chip. NoCs consume up to 28% of the chip power, and among the different components comprising on-chip interconnects, buffers are the largest leakage power consumers in NoC routers, consuming about 68% of the total router leakage power [68]. Buffers also consume significant dynamic power [24], and this consumption increases rapidly as data flow rates increase. Therefore, designing an innovative buffer structure plays a crucial role in architecting high performance and low power on-chip interconnects.

Spin-Transfer Torque Magnetic RAM (STT-MRAM) [26, 27] is a promising next generation memory technology that can replace conventional RAMs due to its near-zero leakage power and high density. However, the weaknesses of STT-MRAM, long latency and high power consumption in write operations and thermal fluctuation-induced random bit flips, should be properly addressed because fast accesses to on-chip memories that guarantee data integrity must be assured for high performance and reliable NoCs.

For addressing the write speed and energy limitations of STT-MRAM, several studies have been performed in designing caches and NoC routers. An adaptive block placement and migration policy for hybrid STT-RAM and SRAM last level caches has been proposed in [69]. A region-based hybrid cache [70] with small fast SRAM and large slow MRAM mitigates performance degradation and energy overheads. For NoC routers, an SRAM/STT-MRAM hybrid buffer [71] shows substantial throughput improvements across various workloads. However, the inevitable use of SRAM to hide the multicycle writes of STT-MRAM sacrifices area, and wastes significant dynamic power in migrating data between the disparate memories. The leakage power overhead due to SRAM also increases as network scale grows and technology scales down.

Thermal stability is another key issue of STT-MRAM, determining how much stability STT-MRAM can provide against thermal fluctuation, thus directly impacting data integrity [72]. Even under a high thermal stability, we cannot totally avoid the occurrence of bit flips because of the stochastic nature of STT-MRAM [73]. Therefore to ensure data integrity we need to provide proper measures for detecting and correcting such transient errors in STT-MRAM. Prior approaches have evaluated the impacts of thermal fluctuation on STT-MRAM reliability, and proposed schemes ensuring data integrity for caches and off-chip storages [72, 74]. Their schemes, however, cannot be directly applicable to NoCs since they are designed for memories having longer data residence time and larger capacities compared to those of latency-sensitive, area- and power-limited buffers in NoCs.

In this work, we propose the first NoC router design that uses only STT-MRAM in buffers, while preserving data integrity. By eliminating SRAM, it offers much larger buffer space with less power consumptions. To hide the multicycle write latencies of STT-MRAM, we propose a novel pipelined input buffer design, a multibank STT-MRAM buffer, which is a virtual channel (VC) with multiple banks where every incoming flit is delivered to each bank alternately via a simple latch inside a router. Through this, we can avoid performance degradation while consuming less area and power.

We use the write latency reduction technique [27], which sacrifices the *data retention time* of an Magnetic Tunnel Junction (MTJ), a bit storage of STT-MRAM. This can be possible due to the short intra-router latency[1] of a flit in on-chip routers. In our simulation, the average intra-router latency in PARSEC benchmarks in an (8x8) mesh network is less

[1]An intra-router latency is the time interval between the arrival of a flit at an input buffer and the departure from a router through a crossbar.

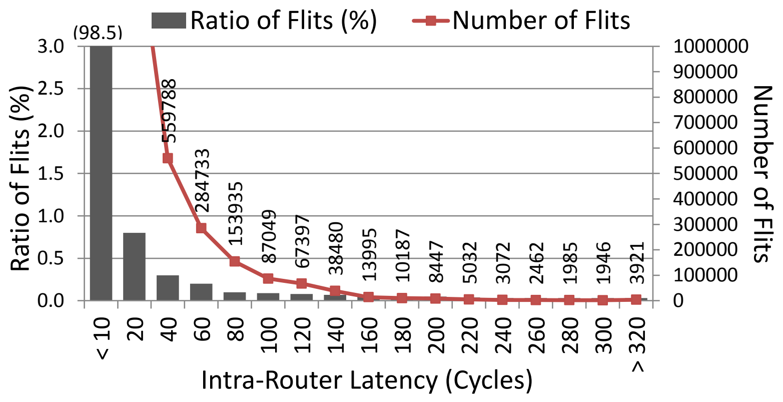


Figure 3.1: Per-Application Intra-Router Latency Distribution (*canneal* in PARSEC Benchmarks)

than 3 cycles[2]. However, for applications that exhibit bursty communication and heavy loads, we observe that flits are staying in STT-MRAM buffers longer than a given retention time, increasing the possibilities of flit losses due to the expired retention period.

This is because some flits have fairly high intra-router latencies while most of the flits are clustered around low intra-router latencies less than 10 cycles as shown in Figure 3.1. These lost flits incur noticeable performance losses especially when the flits are parts of control packets carrying critical cache coherence information. On average, 78.7% of traffic is such single-flit control packets in PARSEC benchmarks [75]. Therefore, to ensure data integrity under the limited retention time and random bit flips of STT-MRAM, we propose cost-efficient dynamic buffer refresh schemes, the processes in which cells' values are kept valid by triggering refreshes in a timely manner. Note that the refreshes are performed in tandem with Error Correcting Codes (ECC) that are extensively used in memories and storage devices to tolerate both transient and static errors [76]. ECCs detect and correct data corruption, thus mitigating the impacts of random bit flips on NoCs.

The main contributions of this paper are as follows:

[2]See Section 3.5 for detailed system configuration.

- We present a detailed analysis on design tradeoffs of an MTJ especially in terms of write performance, write power, and retention time, which are suitable for performance- and power-efficient NoCs.

- We propose a novel multibank input buffer design, which is implemented entirely with STT-MRAM and delivers optimal power saving and performance improvement.

- We suggest a cost-efficient buffer refresh scheme combined with ECC: a global counter refresh scheme, which periodically checks and restores data integrity in buffers, maintaining the validity of flits.

- We achieve 20.7% throughput improvement and 17% total power saving compared to a conventional SRAM based router with the proposed STT-MRAM router scheme.

The remainder of this chapter is organized as follows. The STT-MRAM characteristics and the corresponding design tradeoffs are described in Section 3.2, followed by motivation in Section 3.3, and STT-MRAM router architecture in Section 3.4. Section 3.5 presents performance and power analysis with experimental results, and related work in Section 3.7. Finally, Section 3.8 summarizes our work and makes conclusions.

3.2 Background and Design Challenges

In this section, we review key features of STT-MRAM and analyze design tradeoffs of an MTJ cell in terms of switching time (the time taken for completing a write operation in an MTJ cell, namely write latency), switching current (the power required to change an MTJ cell value, namely write power), and data retention time. We also elaborate design challenges that need to be addressed for ensuring high performance and data integrity in STT-MRAM based NoCs.

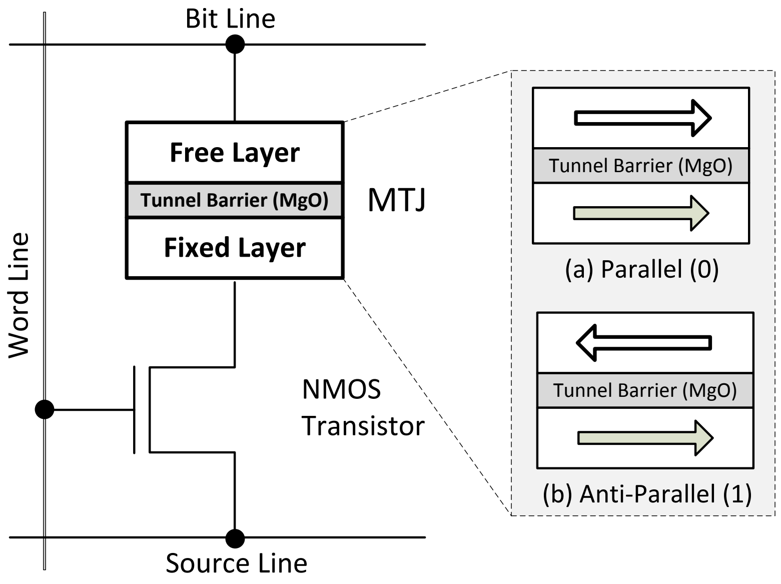


Figure 3.2: STT-MRAM Cell Structure

3.2.1 STT-MRAM

STT-MRAM is a next generation memory technology that exploits magnetoresistance for storing data. In STT-MRAM, each data bit is stored in an MTJ, a fundamental building block. An MTJ consists of three layers: two ferromagnetic layers and a Magnesium Oxide (MgO) tunnel barrier layer in the middle as shown in Figure 3.2. Depending on the current that propagates through the fixed layer, the spin polarity of the free layer changes to either parallel (*zero*) or anti-parallel (*one*) to that of the fixed layer. A single MTJ module is coupled with an NMOS transistor to form a basic memory cell of STT-MRAM, called a 1T-1MTJ.

3.2.2 Fine-tuning STT-MRAM for High Performance NoCs

3.2.2.1 Retention Time

The nonvolatility of an MTJ is quantitatively measured by the *data retention time*, which is the maximum time duration for which stored data can remain in an MTJ [77, 26].

The *data retention time*, T_{ret}, of an MTJ is defined as follows [78].

$$T_{ret} = 1ns \cdot e^{\Delta} \tag{3.1}$$

Δ is the *thermal stability factor* of an MTJ, and it is proportional to the saturation magnetization (M_s), the in-plane anisotropy field (H_k), and the volume of the free layer (V) in an MTJ as follows [79]. T denotes the working temperature.

$$\Delta \propto \frac{M_s H_k V}{T} \tag{3.2}$$

We decrease the *thermal stability factor* by reducing the MTJ area while adjusting M_s and the thickness of the free layer, as mentioned in [80], leading to reduced retention-time STT-MRAM [26].

3.2.2.2 Switching Current and Switching Time

In a *precessional switching mode* [81] where an MTJ switching time (T_s) is short (< 3 ns), the required *current density*, $J_c(T_s)$, is determined as follows.

$$J_c(T_s) \propto J_{c0} + \frac{C}{T_s}, \tag{3.3}$$

where J_{c0} is a switching threshold current density that also depends on M_s and H_k like the *thermal stability factor* (Δ). C is a constant affected by the initial angle between the magnetization vector of the free layer and the easy axis [27]. Reducing the retention time causes the *thermal stability factor* to decrease, which reduces M_s and H_k, and eventually decreases J_{c0}. Therefore, with smaller J_{c0}, we can achieve a shorter switching time with the reduced *current density*, $J_c(T_s)$.

Figure 3.3 depicts the inverse relationship between the switching current ($J_c(T_s)$) and

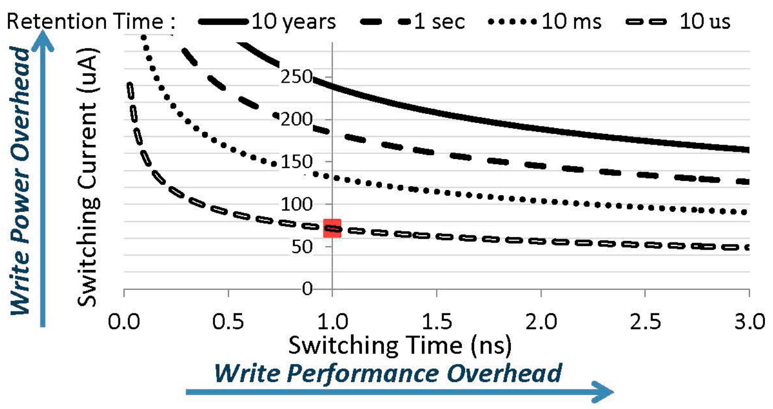


Figure 3.3: The Relationship between Switching Current and Switching Time for Different MTJ Retention Times

the switching time (T_s) under four different MTJ retention times ranging from 10 years to 10 μs. The retention time curves in Figure 3.3 are plotted based on previous studies [77, 26, 27], where the retention time is reduced up to tens of ms level, and for our STT-MRAM buffer design, we further reduce the retention time to 10 μs (proven to be feasible in [82]) based on MTJ device equations [77] and simulation with the PTM model [83] under 32 nm technology. As we further reduce the retention time, the required MTJ switching time and switching current get decreased accordingly, leading to better write performance and less write power overhead. When fixing the switching time at 1.0 ns, for instance, we can reduce the write current by 45.2% by relaxing the retention time from 10 ms to 10 μs. Based on this analysis, we integrate the buffer-level SRAM and STT-MRAM models in NVsim [84] and simulation results are shown in Table 3.2.

3.2.2.3 Cell Area

As an area model of STT-MRAM, we refer to ITRS projections [85] as well as the model used in [28], where a 1T-1MTJ size is 30 F^2. When we assume that an SRAM cell size is approximately 146 F^2 under 32 nm technology, one SRAM cell can be substituted by at least four STT-MRAM cells under the same area budget. An STT-MRAM cell area

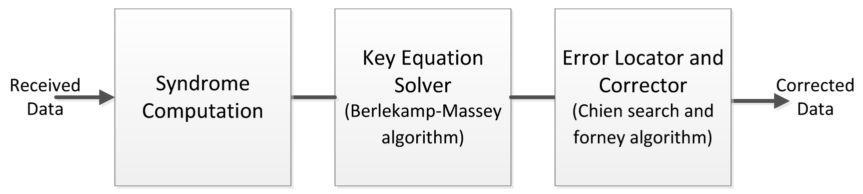

Received Data
Syndrome Computation
Key Equation Solver
(Berlekamp-Massey algorithm)
Error Locator and Corrector
(Chien search and forney algorithm)
Corrected Data

Figure 3.4: BCH ECC Decoder Block Diagram

is mostly determined by the NMOS transistor size since the MTJ cell is much smaller than the transistor.

3.2.2.4 Impact of Process Technology

Applying different process technologies can affect the overall STT-MRAM power-performance cost. As process technologies scale down, the future STT-MRAM is predicted to achieve a significantly smaller cell size, faster read/write with lower power consumption while maintaining the non-volatility property [29, 86]. For advanced technologies such as 22 nm, NVsim circuit-level simulation shows that the cell area is decreased by 48.4$\%$, the read/write dynamic power by 13$\%$, and the leakage power by 41.4$\%$ compared to those of 32 nm. The write delay can also be decreased further due to the smaller cell size and less current required for bit-flips. These trends indicate that STT-MRAM will become a more viable option for cost-efficient NoC routers.

3.2.3 STT-MRAM Design Challenges

3.2.3.1 Retention Failure and Error Protection

As we relax the nonvolatility of STT-MRAM and as STT-MRAM scales, the thermal stability factor (Δ) scales down linearly, thus increasing the probability of *retention failure* (random bit flips during the given retention time) accordingly. As technology scales, the retention failure is also expected to be dominant in STT-MRAM [73]. The retention failure rate ($P_{retFail}$) shown in Equation 3.4 [72] is exponentially dependent on the thermal stability factor (Δ) and is also increasing proportional to the data residence time (t_r) (the duration for which a flit stays inside a buffer).

$$P_{retFail} = 1 - e^{\frac{-t_r}{\tau_0} e^{-\Delta}} \tag{3.4}$$

where τ_0 is the attempt period representing how frequently reversal attempts occur, and the longer t_r is, the more likely errors are. Note that although the retention failure rate can be reduced by increasing Δ, the increased Δ inevitably increases both MTJ cell area [72] and performance/power overheads in STT-MRAM write operations. Such a stochastic retention failure in STT-MRAM can flip bits with no warning, if no proper detection/correction measures are employed. Thus, to ensure data integrity in buffers, we propose a dynamic buffer refresh scheme through which flits are periodically refreshed in tandem with ECC detecting and correcting errors occurred during the retention time (Section 3.4.3 details the proposed error protection scheme). For data protection, the binary Bose-Chaudhuri-Hocquenghem (BCH), a class of ECCs constructed with finite fields, is suited for NoCs because of its fast bit-parallel decoder and multi-bit error correcting capability [76]. We also consider the overheads accompanied with BCH, negatively affecting NoC power and performance; that is, the corresponding latency, power and area overheads of BCH increase as we employ higher error correcting capabilities. Figure 3.4 shows BCH ECC decoder block diagram, where the first phase (*syndrome computation*) detects the error occurrence and the subsequent two phases (*key equation solver* and *error locator and corrector*) locate and correct errors detected.

3.2.3.2 Determining Proper Retention and Switching Times

Based on Figure 3.3, for power- and performance-efficient NoC routers, it is important to identify what the ideal/feasible retention time should be. This is because significant retention time reduction will make the STT-MRAM buffer highly volatile and increase the probability of retention failure, leading to performance degradation due to flits corrupted; while on the other hand, increasing the retention time will negatively affect write performance and energy. Considering these tradeoffs, to locate the sweetspot of the retention time for the STT-MRAM buffer, we measure the average intra-router latency of CMP applications because it is the main factor affecting flits' lifetime. Once flits stay in the STT-MRAM buffer longer than a given retention time, they get invalidated. We conduct experiments with PARSEC benchmarks, where all results are measured under the same area budget, 6 SRAM slots per VC, for input buffers. The average intra-router latency across PARSEC benchmarks is less than 3 cycles, and thus based on such a short residence time, it is reasonable to further reduce the retention time to $microseconds$, rather than $milliseconds$ which is widely used in designing caches with STT-MRAM [77, 26, 27], thus leading to the least write and power overheads among four different retention times in Figure 3.3.

Note that the random bit flip probability causing retention failure should also to be considered for proper estimation of flits' lifetime, which is detailed in Section 3.4.3.

In an ideal case, STT-MRAM write latency should be equal to that of SRAM, thus writing to STT-MRAM must be done in a single cycle, which corresponds to less than 0.5 ns in 2 GHz clock frequency. Such fast write times of less than 0.5 ns have proved possible [82, 86], but as shown in Figure 3.3, it requires rather strong currents[3], and is far from the optimal efficiency [87]. Even with the shortest retention time, therefore, we conclude that it is inevitable to have more than 1-cycle latency for a write operation in the STT-MRAM buffer.

Our proposed scheme exploits these observations to accelerate STT-MRAM write

[3]MTJ switching time(ns) is determined by the amount of supplied switching current(uA) which is not a discrete single value, but a continuous stream. Therefore, to get STT-MRAM writes done in a single cycle, the supplied current (uA) could be exponentially increased to keep the switching time (ns) stay within the range between 0.0 and 0.5. Thus, 1-cycle latency is not affordable for STT-MRAM buffers.

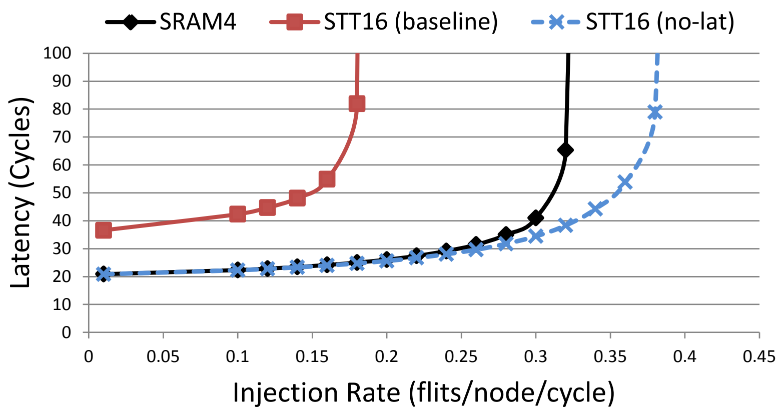


Figure 3.5: Performance Comparison between SRAM and STT-MRAM based Routers under the Same Area Budget

speeds with less power consumption. In Section 3.4.2, we propose router architectures that effectively hide the multicycle write latencies of STT-MRAM.

In summary, for power-performance co-optimized STT-MRAM buffer design, as detailed in Section 3.2.2, we reduce the retention time to 10 μs, and through this, 2-cycle write latency, corresponding to 1 ns in 2 GHz clock frequency, is achieved with 71.35 μA of switching current (See the point where 10 μs retention and 1.0 ns switching time intersect in Figure 3.3) with 30 F^2 of STT-MRAM cell size.

3.3 Motivation

In this section, we present key motivations that drive us to STT-MRAM based NoC routers for power and performance co-optimization.

3.3.1 STT-MRAM for NoC Routers

Figure 3.5 compares the performance of an NoC router equipped with SRAM, STT-MRAM, and ideal STT-MRAM buffers having no write delays. Under the same area budget, STT-MRAM provides 4 times more buffer space as described in Section 3.2.2. Due to the long write delay of STT-MRAM, *STT16 (baseline)*, the SRAM based router shows far

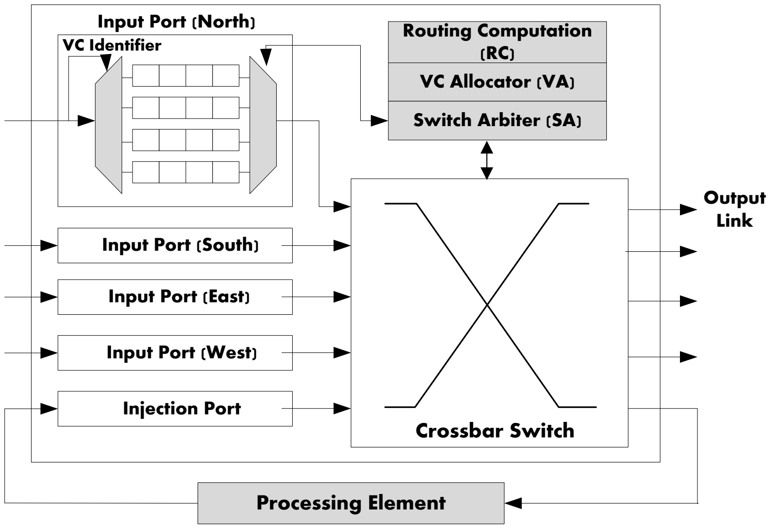


Figure 3.6: Baseline Router Architecture

better performance, but once we completely hide the write delay of STT-MRAM, *STT16 (no-lat)*, the overall throughput is increased by 20% compared with that of SRAM with no zero-load penalty. Also, STT-MRAM has near-zero leakage power, thus consuming much less total power compared with SRAM as described in Section 3.5.2. SRAM appears to be much more power hungry than STT-MRAM, and consequently gives STT-MRAM performance leeway in a power constrained NoCs. This motivates us to adopt STT-MRAM for NoC routers for better performance with less power consumption.

3.4 STT-MRAM Router Architecture

In this section, we describe a baseline router architecture with its buffer structure and present an STT-MRAM based router in detail. The key design goal of the proposed scheme is to enable flits to be written into buffers with no additional time delay.

3.4.1 Baseline Router Architecture

The baseline NoC router architecture is depicted in Figure 3.6, which is similar to that used by Kumar *et al.* [88] employing several features for latency reduction, including speculation [89] and lookahead routing. Each arriving flit goes through 2 pipeline stages in the router: routing computation (RC), VC allocation (VA), and switch arbitration (SA) during the first cycle, and switch traversal (ST) during the second cycle. Each router has multiple VCs per input port and uses flit-based wormhole switching. Credit-based VC flow control is adopted to provide the back-pressure from downstream to upstream routers. The necessity for ultra-low latency leads to a parallel FIFO buffer shown in Figure 3.9(a), where the parallel structure eliminates unnecessary intermediate processes making a flit traverse all buffer entries until it leaves the buffer. The read and write pointers in the parallel FIFO regulate the operations of the input and output MUXes, and the two pointers are controlled by a VC control logic.

3.4.2 STT-MRAM Router Design

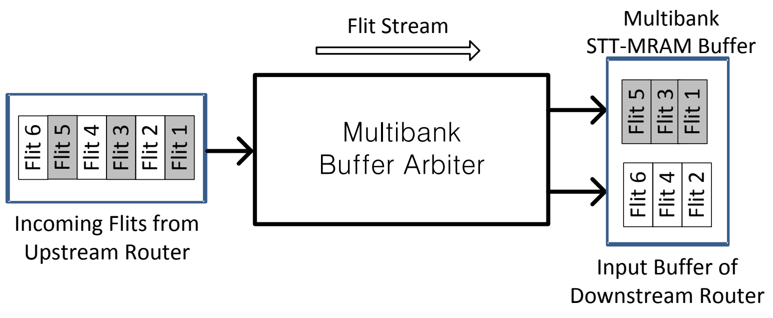
For conventional SRAM buffers, incoming flits are written to their designated buffers with no delay due to the short SRAM write latency. On the contrary, when we replace SRAM with STT-MRAM, only a single flit can be written to a buffer every n cycles, which causes subsequent incoming flits to be delayed. To guarantee seamless traversal of flits across the network, we propose a multibank STT-MRAM buffer that hides the long write latency inherent in STT-MRAM.

3.4.2.1 Multibank STT-MRAM Buffer

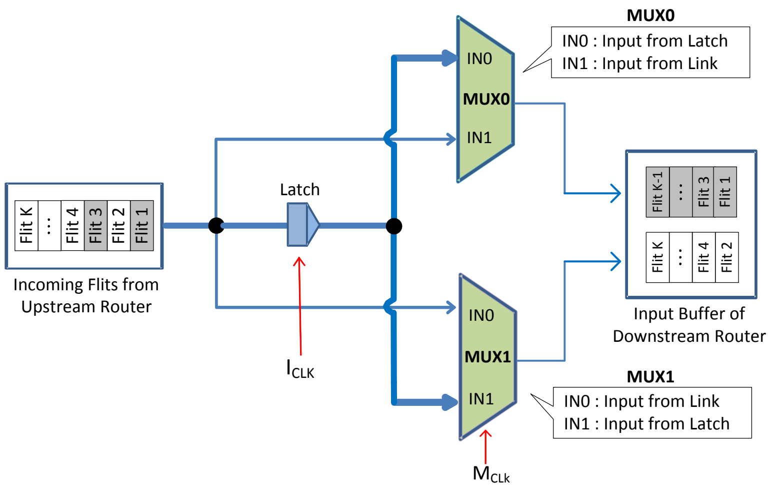
The multibank buffer scheme can be used to hide n-cycle write latency of STT-MRAM. For example, to hide 2-cycle write delay of STT-MRAM buffer, we divide each VC into two banks where every incoming flit is seamlessly pipelined to each bank alternately via a simple latch inside a router. Note that prior studies [68, 90] explore the latch-based NoC pipeline design, where latches along the link are utilized as temporary buffers that can hold and release data when necessary. The simple latch in this paper is controlled by a control block (as in the Channel buffer [90]) interfaced with the NoC clock, having the dual function of switching between storing and transmitting data. Let us refer to the two banks as **Odd** and **Even** banks, respectively, and incoming flits from upstream routers as Odd and Even flits as shown in Figure 3.7(a). Every odd numbered flit is sent to the Odd bank of a downstream router, and similarly, an even numbered flit to the Even bank through a Multibank Buffer Arbiter (MBA) that has one input port and two splitted output ports. The goal of this multibank buffer scheme is to enable the incoming consecutive flits to be written to different banks simultaneously to effectively hide the multicycle write latencies of STT-MRAM. To achieve this goal, two MUXes and one simple latch are used for the MBA as shown in Figure 3.7(b). Each MUX has two inputs: one input is connected to the communication link from the upstream router, and another to the simple latch inside the router. The simple latch is located at the front of the input buffer and functions as a temporary buffer. It holds an incoming flit for a cycle and dispatches the latched flit to its original target bank at the very next cycle. $Iclk$ and $Mclk$ are control signals originating from the control block in the input buffer, which represent the hold/release signals for the latch and the select signal for the MUXes, respectively. Note that the area overhead for this logic is negligible as compared to the buffer, and already added to the logic area controlling refresh/read/write pointers. An incoming Odd flit, for instance, is directly written to the Odd bank during the first cycle, and then during the next cycle, the latched flit is sent to the same Odd bank, thus completing its 2-cycle write process[4]. Similarly, a subsequent incoming Even flit follows the same process, but uses the other bank. Through this, without the need of any additional SRAM buffer as in the Hybrid buffer [71], we can seamlessly pipeline incoming consecutive flits to the input buffers of a downstream router.

Note that, in case of very light loads, an incoming flit might experience write delays in the STT-MRAM buffer, increasing zero-load latency, which results in degraded NoC

[4]Since it takes multicycles to write a single flit to a target buffer, there could be a potential glitch due to a momentary transient pulse (noise), or clock skew between communicating elements. These issues can be addressed by sizing the MUX, overlapping clock or duplicating input signal [91].


Flit Stream
Multibank STT-MRAM Buffer
Flit 6
Flit 5
Flit 4
Flit 3
Flit 2
Flit 1
Multibank Buffer Arbiter
Incoming Flits from Upstream Router
Input Buffer of Downstream Router

(a) Multibank Buffer Arbiter that Hides 2-Cycle Write Latency



(b) Multibank Buffer Arbiter Internal Structure

Figure 3.7: Multibank STT-MRAM Buffer

performance. To avoid this, we incorporate the buffer bypassing logic [24] widely used in NoCs for power-performance efficiency. Accordingly, when a flit arrives at an empty buffer, the flit heads straight to switch arbitration and gets sent directly to the crossbar switch, thus circumventing STT-MRAM input buffers. The latch inside a router serves as a bypass latch for the consecutive pipelining between the flit arrival and crossbar traversal. Therefore, the zero-load latency of the STT-MRAM router becomes comparable to that of the SRAM router.

In general, to hide n-cycle write latency, the STT-MRAM buffer scheme requires n MUXes for n splitted banks with $n-1$ latches inside a router as shown in Figure 3.10.

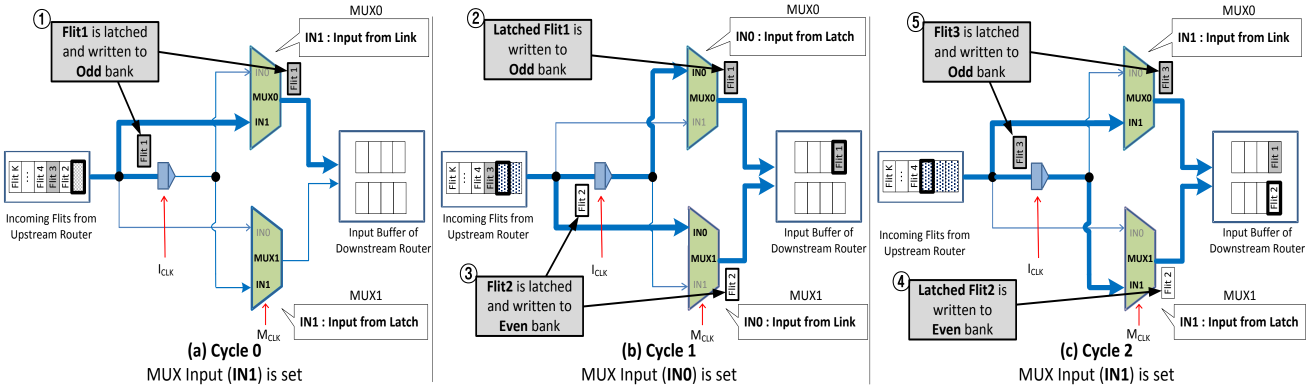

① Flit1 is latched and written to Odd bank
MUX0
IN1 : Input from Link
MUX1
IN1 : Input from Latch
Incoming Flits from Upstream Router
Input Buffer of Downstream Router
(a) Cycle 0
MUX Input (IN1) is set
② Latched Flit1 is written to Odd bank
IN0 : Input from Latch
③ Flit2 is latched and written to Even bank
IN0 : Input from Link
(b) Cycle 1
MUX Input (IN0) is set
⑤ Flit3 is latched and written to Odd bank
④ Latched Flit2 is written to Even bank
(c) Cycle 2
MUX Input (IN1) is set

Figure 3.8: Dual-Bank STT-MRAM Buffer Example (Sequence of Operations: ① ~ ⑤)

The increase of n can negatively affect the performance and area overheads of the STT-MRAM buffer. Note that n is the ratio of the STT-MRAM write latency to the clock cycle time of the NoC clock. As technology advances, we expect the write latency to be reduced as described in Section 3.2.2, while at the same time the NoC clock frequency increases. Therefore we do not expect n to increase drastically in the near future, hence keeping the proposed scheme feasible. In our analysis, when n stays within 5, we observe negligible performance degradation (less than 1%) with increased extra logic area. The detailed analysis of the impact of large n is discussed in Section 3.6.3. Figure 3.8 shows an example data flow for flits from an upstream router during 3 consecutive clock cycles. Initially, the control of both MUXes, denoted as MUX0 and MUX1, is assumed to be set to 0, and all VCs are empty. It is also assumed that the interconnect clock period is long enough to satisfy the setup and hold constraints of a simple CMOS MUX.

- Cycle 0: The input signal of both MUXes is set to 1 (IN1). This is the first write cycle for an incoming flit, $Flit1$. $Flit1$ is sent to the Odd bank input buffer of the downstream router through IN1 of MUX0, and at the same time, $Flit1$ is stored in the simple latch(①).

- Cycle 1: The input signal of both MUXes has changed to 0 (IN0). The output of MUX0 is $Flit1$ which was previously latched, and $Flit1$ is dispatched from the latch to its original target bank (Odd bank), and thus completing its second write cycle(②). Also, this is the first write cycle for a subsequent incoming flit, $Flit2$, to the Even bank input buffer. While $Flit2$ is transferred to the Even bank through IN0 of MUX1, it is simultaneously stored in the simple latch(③).
- Cycle 2: The input signal of both MUXes is switched back to 1 (IN1). Like the previous logic, this is the second write cycle of $Flit2$ from the latch to the Even bank(④), and the first write cycle for the incoming flit, $Flit3$(⑤).

Note that, at low loads, flits arrive at the input buffer intermittently. In this case, the arriving flit bypasses the input buffer, or unless the buffer is empty, the STT-MRAM buffer directs the flit to either Odd or Even bank based on a one-bit flag indicating the next bank. This ensures that incoming flits are placed in a VC without leaving unused buffer slots in banks. This also ensures sequential reads by maintaining the FIFO properties of input buffers.

3.4.2.2 Read/Write and Refresh Logic

Unlike the conventional SRAM input buffer that requires a read and a write pointer for read and write operations per VC (Figure 3.9(a)), the proposed multibank STT-MRAM buffer, assuming 2-cycle write latency, requires dual write pointers, *Wr_ptr (Odd)* and *Wr_ptr (Even)*, and a single read pointer, *Rd_ptr*, per VC as shown in Figure 3.9(b). The corresponding VC control logic generates proper read and write pointer values for handling flits in a timely manner. To be specific, initially, as shown in Figure 3.11(a), one of the write pointers, *Wr_ptr (Odd)*, points to the tail of the Odd bank, and *Wr_ptr (Even)* points to the tail of the Even bank, and the read pointer, *Rd_ptr*, points to the head of the buffer. For a write operation (Figure 3.11(b)), the incoming flit is written to the location

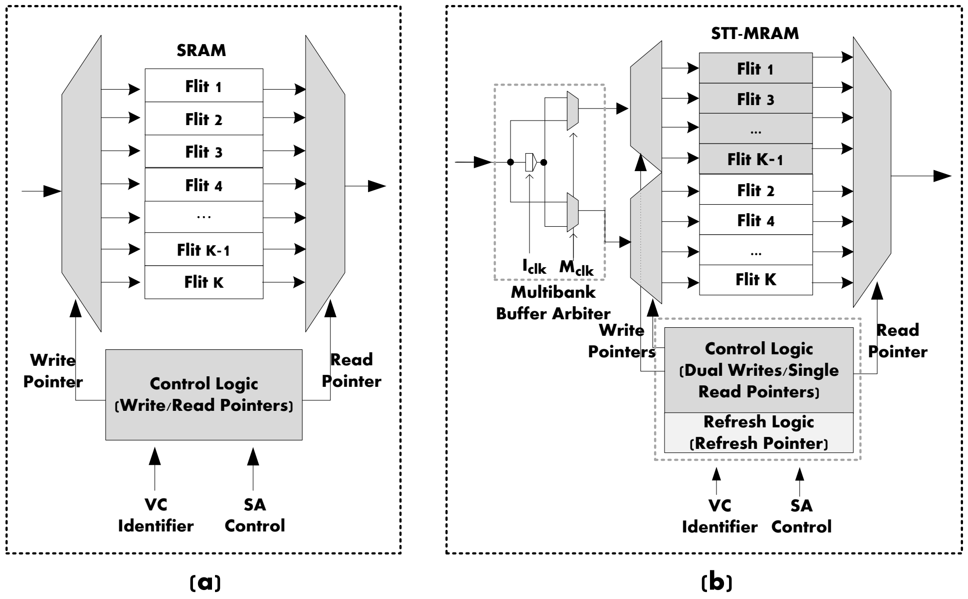


Figure 3.9: A Baseline SRAM Input Buffer (a) and A Dual-Bank STT-MRAM Input Buffer (b)

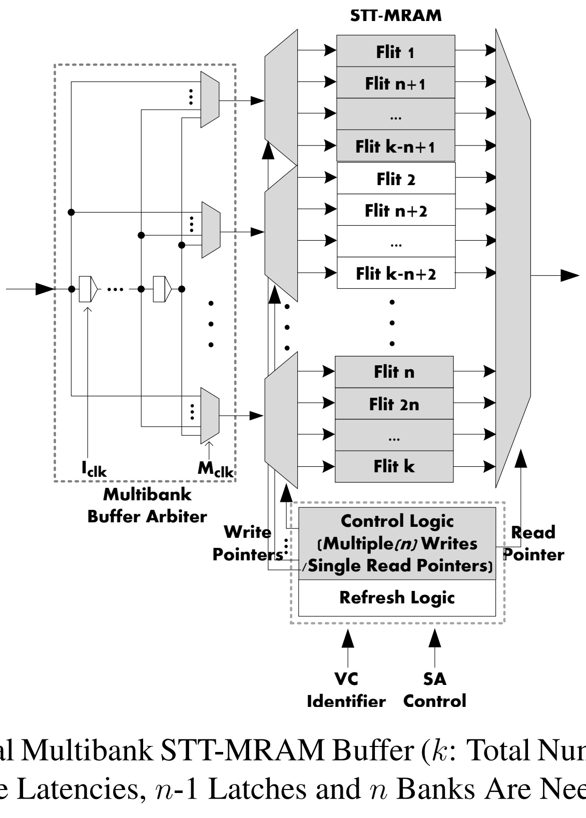


Figure 3.10: A General Multibank STT-MRAM Buffer (k: Total Number of Flits Buffered, To Hide n-cycle Write Latencies, n-1 Latches and n Banks Are Needed.)

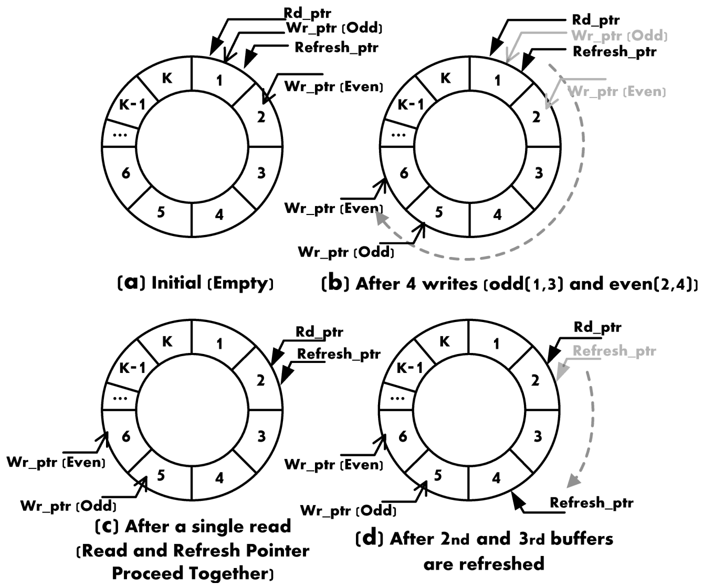

Rd_ptr
Wr_ptr (Odd)
Refresh_ptr
Wr_ptr (Even)
K
1
2
3
4
5
6
...
K-1
(a) Initial (Empty)
(b) After 4 writes (odd(1,3) and even(2,4))
(c) After a single read (Read and Refresh Pointer Proceed Together)
(d) After 2nd and 3rd buffers are refreshed

Figure 3.11: Circular Queue for Dual-Bank STT-MRAM Buffer (*Assuming all errors are correctable / Sequence: (a) ~(d)*)

pointed by the tail pointer in one of the banks. For a read operation (Figure 3.11(c)), the flit pointed by *Rd_ptr* is read out and dispatched to the crossbar. Note that STT-MRAM read latency is comparable to that of SRAM and thus no delay occurs for the read operation. The refresh pointer, *Refresh_ptr*, shown in Figure 3.11(d), moves according to the refresh logic which is described in Section 3.4.3.

3.4.2.3 Handling Uncorrectable Errors in Refresh Operations

In the read/write and refresh logics above, for simplicity, we assume all transient errors are *correctable* via ECC, thus read/write and refresh pointers keep proceeding without being interrupted by any *uncorrectable* errors. Actually, however, we need to consider such *uncorrectable* errors in the logics, because otherwise read operations might end up reading already-corrupted flits due to the uncorrected errors in flits. Such an issue could arise when read operations follow right after refreshes as in Figure 3.11(d). Once parts of control or data packets get corrupted beyond given ECC capability and thus *uncorrectable*, nodes need to nullify such packets and make room for newly incoming packets while retransmitting the corrupted packets for recovery. For single-flit control packets, it is relatively easy to handle the recovery because they always stay inside a single router. However, for data packets consisting of a single head/tail and multiple body flits, such recovery processes need to be handled carefully because data packets can span multiple nodes as network gets congested. In any case, we need to have each node aware of the data corruption so that the corrupted packets are to be properly handled inside each router. To implement refresh logics considering aforementioned cases, we add a *valid-bit* indicating the validity of each flit stored in a VC in a router. Specifically, when a flit is written into a buffer, the *valid-bit* is initially set to one (1), but once the flit becomes *uncorrectable* afterwards, the refresh pointer immediately set the *valid-bit* to zero (0), thus read pointer skips reading the corrupted flit. When there are parts of a corrupted packet, such as a head/body flit, already buffered in a neighboring router, a single-flit control packet, called *dummy tail*, is dispatched to the router, releasing VC reservation made by the corrupted packet, thus making the VC available for subsequent packets. Note that the negative impact of uncorrectable packets can be mitigated by adopting a proper refresh policy minimizing the number of uncorrectable packets, which is described in Section 3.4.3.1.

3.4.3 Nonvolatility-Relaxed STT-MRAM Buffer

In this section, we propose cost-efficient dynamic buffer refresh schemes in conjunction with ECC for error check and correction, which jointly ensures the validity of flits in buffers.

3.4.3.1 Refresh with ECC Scheme

A conventional DRAM-style refresh, which is triggered based on the retention time, is not enough for securing the reliability of STT-MRAM due to the *retention failure* detailed in Section 3.2.2. Thus, it is required to take counter measures integrated with proper architectural techniques such as ECC to ensure reliability in NoCs. Therefore we trigger refresh in tandem with ECC through which each flit stored in a buffer is read periodically and checked for errors. Once ECC detects *correctable* errors, the errors are corrected and the refresh operation writes the restored flit back into the buffer [5]. If the errors exceed a given ECC correction capability and thus *uncorrectable*, a *nack* signal is transmitted back to the source along the reverse direction to indicate the need for a retransmission. Note that we assume each source node has a network interface (NI) with an *ECC encoder* that appends parity bits to each generated flit. Thus we avoid redundant ECC encoding for incoming flits at each router. The parity bits are carried along with the flit and utilized at each hop to detect and correct erroneous bits through the *ECC decoder* (Figure 3.4) inside a router.

Refresh Periodicity and ECC Capability: Regarding such an ECC supported refresh operation, there are two key factors impacting power, performance, and area in NoCs: Refresh periodicity and ECC capability. First, refresh periodicity is important because excessive refreshes contribute to significant power consumption. Thus, it is necessary to set proper refresh periodicity to achieve a low power NoC while seamlessly checking and restoring data in buffers. Second, ECC capability also affects performance and area overheads in NoCs; that is, strong ECC takes a relatively longer time for multi-bit error detection and correction, and requires extra storages holding parity bits compared to those of simple ECC such as *single-error correction and double-error detection* (SECDED). The area of ECC decoder grows exponentially with the ECC error-correcting capability [76]. Basically, SECDED is sufficient to mask any single bit error, thus fitting in the 8-bit parity field[6] for a 128-bit flit [89], however, a strong ECC requires more parity bits, possibly increasing the total number of flits per packet due to the reduced space left for the payload

[5]For STT-MRAM, we assume it takes two cycles for a write operation. Such a write delay can be hidden by either ERB or IRB scheme detailed in Section 3.4.3.2.

[6]An SECDED can protect an n bit memory using $\log_2(n) + 1$ parity bits.

holding the actual data of a packet, thus contributing to degrade NoC performance.

Hitting the Sweetspot: To determine the sweetspot in both refresh periodicity and ECC capability that help achieve power- and performance-efficient NoCs, we consider both the average residence time of a flit in a buffer and the corresponding error probability for a flit across varying residence time. This is because the error probability due to MTJ free layer reversal (bit-flip) is linearly dependent on the average residence time (t_r) of the bit-cells as shown in Equation 3.4 in Section 3.2.2. First, in PARSEC benchmarks, most of the flits tend to stay short inside buffers; for example, 99.4 % flits stay within 40 ns (Figure 3.1). Second, for quantifying the error probability, we capture the average number of bits flipped for a 128-bit flit across PARSEC benchmarks using the probabilities derived from Equation 3.4 under varying residence time. And the result graph is shown in Figure 3.12, where flits are more likely to exhibit low error probability under a short residence time; for example, under a 40 ns residence time the probability of having more than 2-bit failure is less than 1 %. This empirical result is practically in line with the error probability (Equation 3.4) in that the shorter residence time leads to the less probability of a bit failure.

Based on our observations above, when the ECC supported refresh is triggered within the range of 40 ns, we can maintain the bit failure probability low and thus single-bit error correction via SECDED is sufficient to cover most of the bit failures without hurting the performance and power in NoCs, which is detailed in Section 3.5.

3.4.3.2 Dynamic Buffer Refresh Schemes

To mask errors in STT-MRAM, it is necessary to periodically *sweep* the memory by reading each location and correcting any single-bit error. Detecting and correcting errors soon after they occur reduces the possibility of the accumulation of errors in a flit. Thus, we employ a refresh scheme through which refresh operations are adaptively triggered

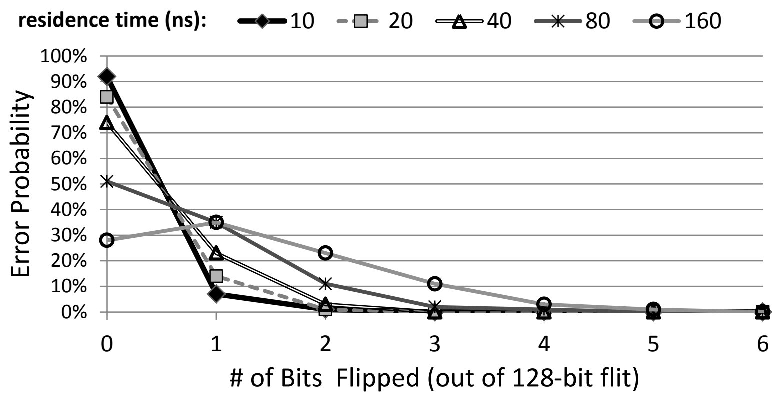


Figure 3.12: Probability of the Number of Bits Flipped (*Note that the sum of error probabilities under a specific residence time is 100 %*)

for flits which almost reach the end of a given refresh period. Basically, these refresh schemes require an n flit-deep refresh buffer per each physical channel (PC) to make up for n-cycle write latency. During a refresh operation, the target flit is read out from the input buffer into the refresh buffer, checked for correctness using ECC, and then written back to its original FIFO buffer. If a read request comes before refresh finishes, the flit is directly returned from the refresh buffer, thus the refresh buffer is also used as a read buffer compensating for n-cycle write latency of STT-MRAM, and the refreshes are seamlessly pipelined to allow for consecutive refreshes.

Global Counter (GC) Refresh Scheme: This scheme selectively triggers the refresh based on the estimated age of an individual flit per VC. To monitor the age of each flit, we add a refresh pointer, *Refresh-ptr*, shown in Figure 3.11, which is controlled by a VC refresh logic shown in Figure 3.9(b). Initially, the refresh scheduler makes the refresh pointer point to the flit queued in the head of a VC and moves it toward the tail of a VC one flit at a time whenever the pointed flit gets refreshed as shown in Figure 3.11(d). To decide the refresh timepoint, we adopt a per-router GC, which serves as a reference point for the refresh logic to determine if it needs to trigger refreshes. In this scheme, refresh

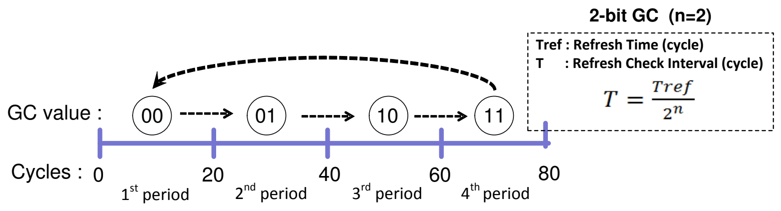


Figure 3.13: An Example of a 2-bit Global Counter (GC) Refresh Logic (*Assuming refresh time is 80 cycles (40 ns in 2 GHz)*)

time is divided into N periods, and each period is T cycles long. The per-router GC is used for the countdown to T cycles, and GC value indicates a specific period. At the end of every T cycles, the GC value is increased by 1, and loops over after the given refresh time. Figure 3.13 shows an example of 2-bit GC refresh scheme, where the GC value is updated at the end of every T cycles (T = 20), and when a flit arrives at a buffer, the value of the current GC (00, 01, 10, 11) is copied to the flit's Arrival Timestamp (AT). At the end of T cycles, the AT value of each flit is compared with GC value to see if the flit needs to be refreshed. When GC value is *01*, for example, all flits having AT equal to *10* get refreshed one by one per cycle. This is equivalent to refreshing flits that stay at the buffer for around 60 to 80 cycles. Note that AT value is assigned only when a flit arrives at a buffer, and unchanged until the flit gets dispatched. Also, as the interval of a period gets decreased (by assigning more bits to the GC), less refresh operations are performed since a refresh is triggered based on a more fine-grained clock counter, thus saving more dynamic power. A larger bit counter allows more time for a flit to stay in the buffer before applying any refresh. Our experimental results show that the GC suffices to detect the expiration time of the given refresh period without significantly affecting performance, which is described in Section 3.5.2.

Refresh Coverage: As a means of keeping the integrity of data, prior study [73] also suggests to periodically *sweep* the memory for error correction, but they refresh only data turned out to be corrupted while letting a majority of un-refreshed data keep on aging in place. This is because their target is a large scale memory (e.g. off-chip memory) or on-chip caches that can tolerate cache misses due to invalid data. However, such a selective refresh does not completely prevent the accumulation of errors because of the increasing probability of multi-bit error occurrence as detailed in Section 3.4.3.1 based on Figure 3.12. Thus, to maintain low error probability in NoCs, we propose to trigger refresh even for currently valid flits in a buffer, resetting the lifetime of flits, thus avoiding performance and power overheads originated from uncorrectable burst multi-bit errors that trigger multiple retransmissions for data recovery. Note that flits mostly stay short inside buffers, leaving buffers prior to refresh operations, thus refresh overheads are relatively low compared to those of caches having longer data residence time. The detailed power and performance impacts of refreshes are described in Section 3.5.

3.5 Evaluation

3.5.1 System Configuration

Table 3.1: CMP System Configuration

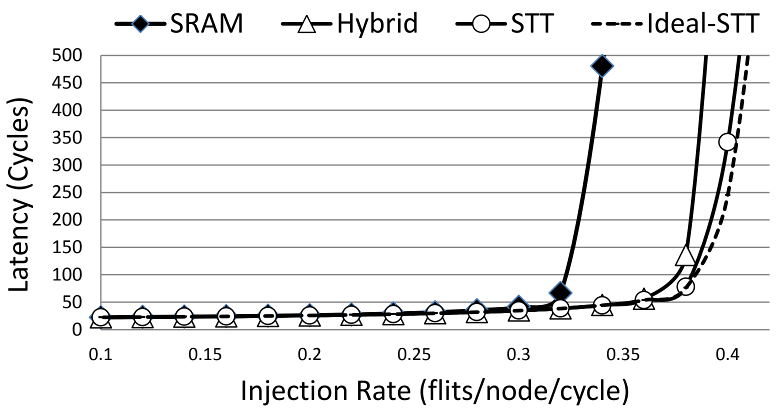
System Parameters	Details
Clock frequency	5 GHz / 2 GHz (Core / NoC)
# of processors	64, In-order, Alpha ISA
L1 I and D caches	Direct-mapped 32KB (L1I) 4-way 32KB (L1D), 3 cycles
L2 cache	8-way 16MB, 8 cycles 64 banks SNUCA, 256 KB/bank
Cache block size	64B
Coherence protocol	MESI
Memory latency	150 cycles
Flit size	16B
Packet size	1 Flit (Control), 5 Flits (Data)

Parameters	SRAM	STT-MRAM	
		10 *ms*	10 *µs*
Read Energy (*pJ*/flit)	5.25	3.8	2.7
Write Energy (*pJ*/flit)	5.25	40.0	13.7
Leakage Power (*mW*)	0.028	0.005	0.003

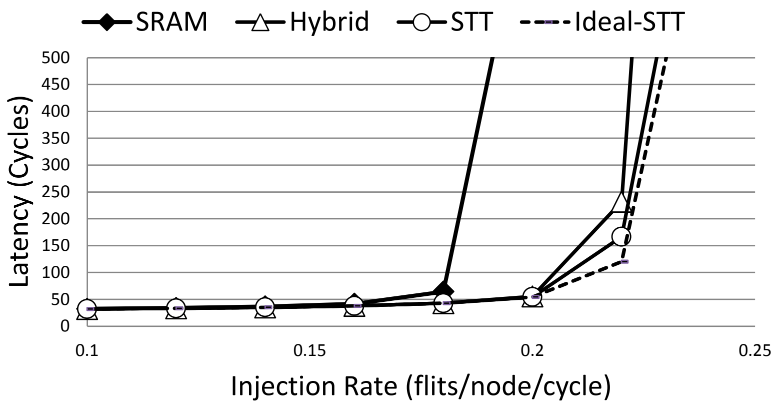
Table 3.2: SRAM and STT-MRAM Parameters with Different Retention Times *(The Hybrid Buffer scheme utilizes 10 ms.)*

A cycle-accurate NoC simulator is used to conduct the detailed evaluation of the proposed schemes. It implements the pipelined router architecture with VCs, a VC allocator (VA), a switch arbiter (SA) and a crossbar. The network is equipped with 2-stage speculative routers with lookahead routing. The router has a set of v VCs per input port. Each VC contains a k-flit buffer with 16B flit size. In our evaluation, we assume that v is 4, while k may vary with different buffer configurations. A dimension order routing, XY, is used with wormhole switching flow control in an (8x8) 2D-mesh. A variety of synthetic workloads are used to measure the effectiveness of the STT-MRAM buffer schemes: uniform random **(UR)**, bit complement **(BC)** and nearest neighbor **(NN)**. To evaluate the proposed schemes under realistic environments, we also run PARSEC parallel benchmarks via Netrace [92] incorporated into our NoC simulator. Table 3.1 specifies the detailed CMP configuration.

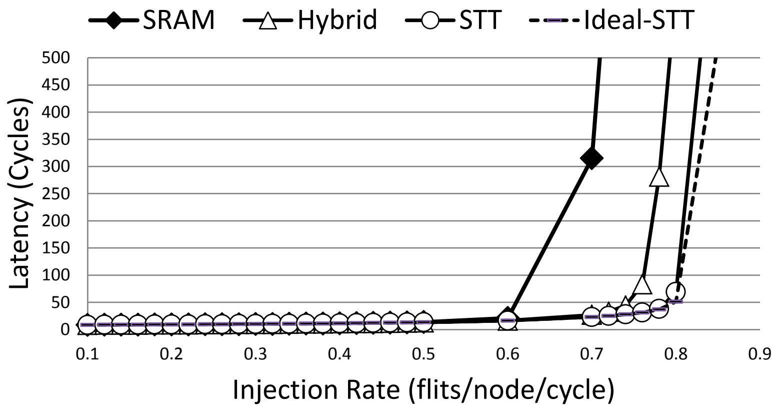
To estimate the power, area, and timing of SRAM/STT-MRAM routers operating with 1 V supply voltage in 2 GHz clock frequency, we modified an open source NoC router RTL model [93] and synthesized in Synopsys Design Compiler with a TSMC 45 nm technology library. SRAM/STT-MRAM parameter values in Table 3.2 are obtained through the STT-MRAM analyses described in Section 3.2.2 and based on relevant literatures [28, 71]. Note that the unit of the leakage power is mW per 1-flit buffer. Throughout this paper, the sizes of the SRAM and STT-MRAM buffers, defined by the number of flits per VC, are denoted by $SRAM\#$ and $STT\#$, respectively. As stated in Section 3.2, STT-MRAM basically provides 4 times more buffer capacity compared with SRAM under the same area budget



(a) UR

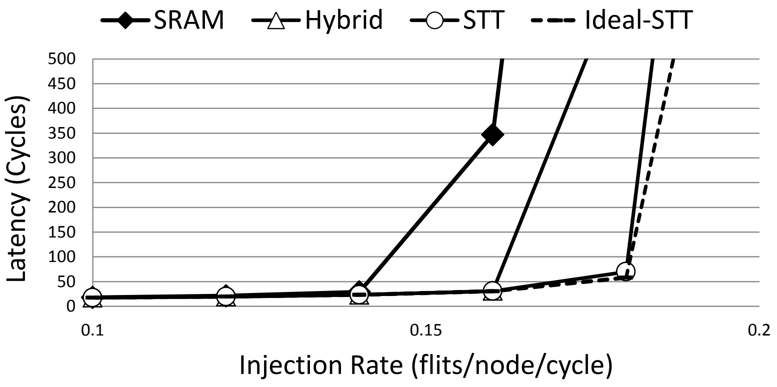


(b) BC

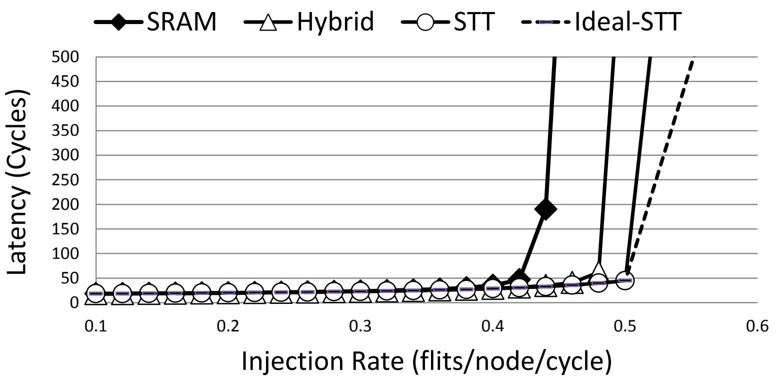


(c) NN

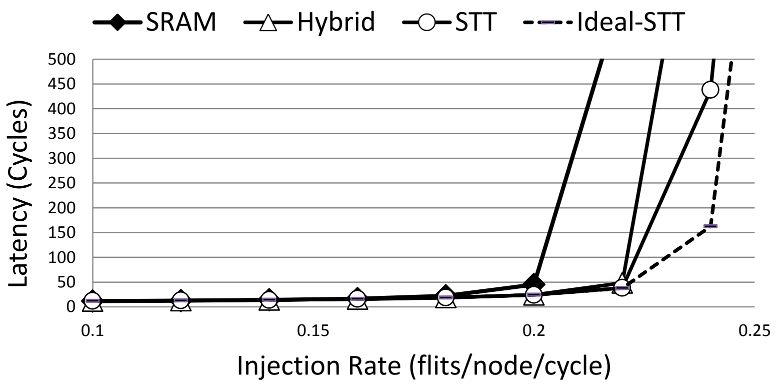
Figure 3.14: Performance Comparison with Different Synthetic Workloads



(a) CMesh



(b) 2D-Torus



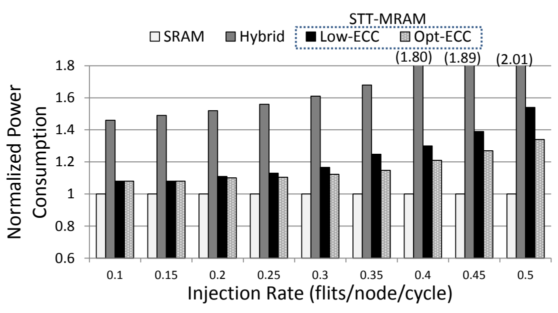
(c) Flattened Butterfly

Figure 3.15: Performance Comparison with Different Topologies

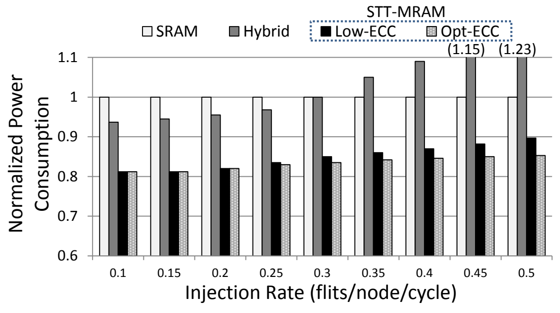
($SRAM4$ is equal to $STT16$). For the STT-MRAM buffer schemes, however, due to the extra area overheads incurred by additional circuitry for the MBA shown in Figure 3.7 and the ECC modules, 2.95% of buffer spaces get sacrificed under 2-cycle write latency. Thus, STT-MRAM can provide approximately 3.5 times more buffer capacities than the conventional SRAM buffer ($SRAM4$ is equal to $STT14$). The detailed area analysis is given in Section 3.6.3.

3.5.2 Performance and Power Analysis

Figure 3.14 shows performance results of four different buffer configurations: the SRAM buffer, the Hybrid buffer, the proposed STT-MRAM buffer, and an ideal STT-MRAM buffer having no write delays with significantly large buffer spaces under UR, BC, and NN traffic patterns. Note that the *Ideal-STT* is presented to show the theoretical upper bound of network throughput in NoCs, and for the SRAM and Hybrid buffers, we do not consider soft errors inherent in SRAM, thus performance and power graphs plotted here represent theoretically optimistic values for the SRAM and Hybrid designs. All results except the *Ideal-STT* are measured under the same area budget, $SRAM4$ per VC, for input buffers. The Hybrid buffer can accommodate 7 flits per VC, consisting of $SRAM3$ and $STT4$, which is an optimal hybrid design suggested in [71], while the STT-MRAM buffer accommodates 14 flits per VC. In all cases, on average, the STT-MRAM buffer shows better throughput by 19.3% for UR, 23.2% for BC, and 19.8% for NN compared with the SRAM buffer, and 5.0% compared with the Hybrid buffer across different traffic patterns. These results indicate that the potential performance degradation caused by the multicycle write latencies of STT-MRAM can be offset by the increased buffer size and the proposed multibank buffer scheme, thus resulting in significant performance improvement. Note that the throughput of the *Ideal-STT* is almost comparable with the STT-MRAM across different traffics. This is mainly because of Head-of-Line (HoL) blocking



(a) Dynamic Buffer Power Consumption



(b) Total Router Power Consumption

Figure 3.16: Normalized Power Consumption - SRAM/Hybrid/STT-MRAM with Different Refresh Rates (*Low-ECC: Low Refresh Rate (80ns) / Opt-ECC: Optimal Refresh Rate (40ns)*, See Section 3.4.3.1 for details.)

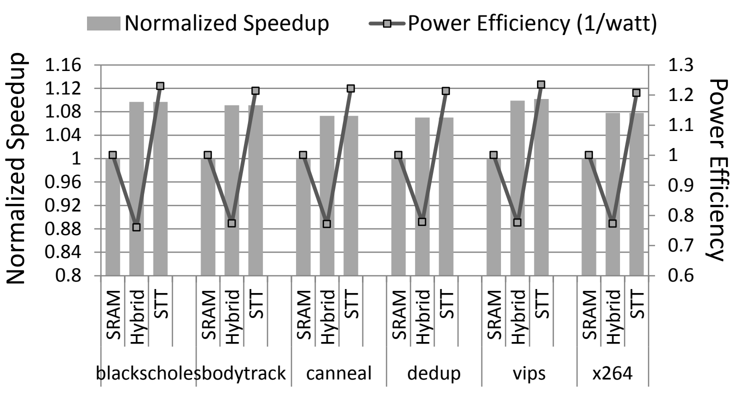


Figure 3.17: PARSEC Benchmark Results

caused by packet contention (Section 3.5.3 details this HoL effect on network throughput).

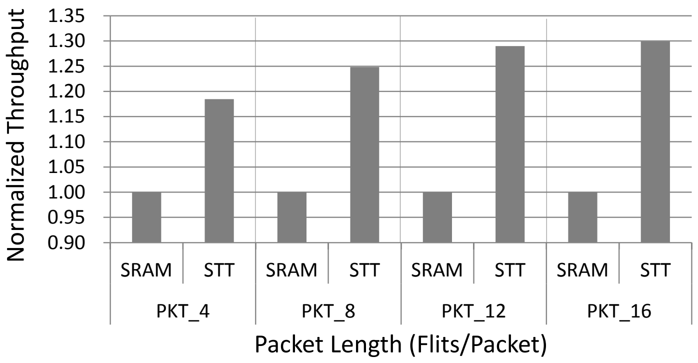
We also evaluate the STT-MRAM buffer under various topologies: Concentrated Mesh (CMesh), 2D-Torus, and Flattened Butterfly. Figure 3.15 shows that the STT-MRAM buffer helps increase throughput in CMesh, 2D-Torus, and Flattened Butterfly by 25.2%, 19.4%, and 9.5% compared with the SRAM buffer, and 5.2%, 8.9%, and 4.9% compared with the Hybrid buffer, respectively.

Power is one of the critical issues in designing NoC. We also measure the power consumption of the proposed multibank STT-MRAM buffer scheme against the SRAM and Hybrid buffers.

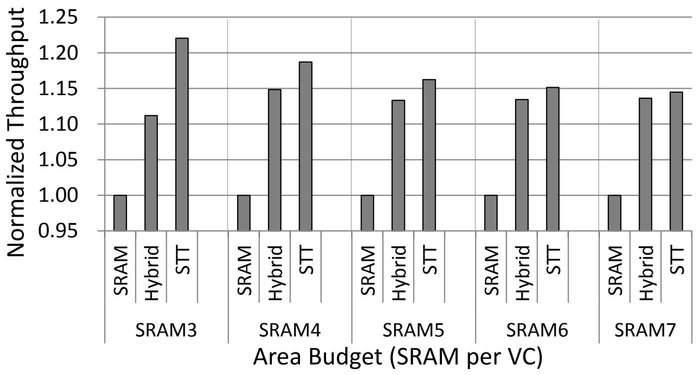
Figure 3.16(a) compares the dynamic power consumption of the SRAM, Hybrid, and STT-MRAM buffers with different packet injection rates under UR traffic. All results are normalized to that of the SRAM buffer. The first and second bars indicate the SRAM and Hybrid buffers and, in particular, the STT-MRAM buffer is evaluated based on different refresh rates (low / optimal refresh rate with ECC) to quantitatively measure their effectiveness in reducing overall power overheads, and find out the most power-efficient combination. Note that the refresh power overheads affect the dynamic power consumption in NoCs, and are increasing proportionally to the number of packet retransmissions and ECC refresh operations performed. Thus, to achieve a power-efficient NoC, it is necessary to employ a buffer refresh scheme that triggers less refreshes and packet retransmissions. In Figure 3.16(a), the baseline SRAM consumes the least normalized dynamic power because the Hybrid and STT-MRAM buffers require higher write energy compared to that of the SRAM (see Table 3.2). The Hybrid buffer consumes 1.7 times and 1.4 times more dynamic power, on average, compared with the SRAM and STT-MRAM buffers each. This is mostly due to the frequent migrations from SRAM to STT-MRAM inside the Hybrid buffer, and a higher write energy associated with a high retention STT-MRAM, 10 ms, in the Hybrid buffer, compared to that of the multibank STT-MRAM buffer. For the STT-MRAM buffer, *Opt-ECC* based refresh scheme consumes less dynamic power by 12.9% compared to *Low-ECC*. This is because *Opt-ECC* incurs less packet losses, thus consuming much less power in checking and correcting bits in STT-MRAM than its counterpart. And in a low network load, most of flits stay in the buffer only a short period of time, triggering less error correction logics in ECC, thus incurring less refresh power overheads. As injection rate increases, however, flits stay longer in buffers due to network congestion, increasing the possibility of flit losses as described in Section 3.4.3.1, thus consuming more energy for error correction via ECC.

Figure 3.16(b) compares the total power consumption of routers with different buffer schemes. The total router power includes dynamic and leakage power of all routers across the network. On average, there is 17% improvement in total router power going from baseline SRAM to STT-MRAM buffer design. With our proposed buffer refresh schemes, although there is an increase in the dynamic power, we consistently observe efficiency in total router power consumption of the proposed STT-MRAM buffer. This is attributed to the fact that the fraction of dynamic power to the total power is not significant compared to the very high leakage power [77, 70]. In this context, due to the power hungry nature of SRAM, the SRAM and Hybrid buffers consume significantly more power than the STT-MRAM buffer. The Hybrid scheme consumes 9.8% and 32.5% more total router power, on average, especially at high injection rates (> 0.3), compared to the SRAM and STT-MRAM schemes. This is because of the migration power overheads, high STT-MRAM write energy, and high SRAM leakage power in the Hybrid buffer.

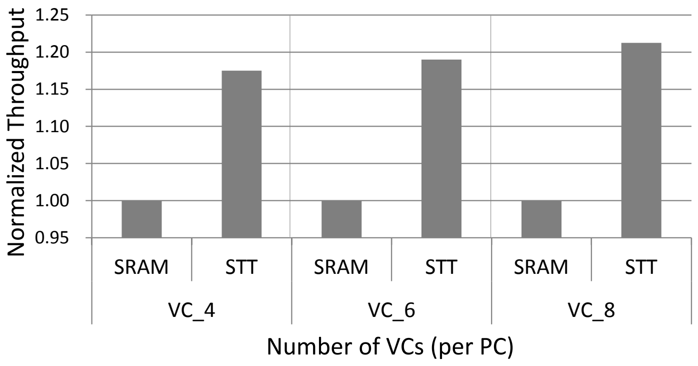
Figure 3.17 shows speedups and router power efficiency (the inverse of the total power consumption) relative to the SRAM baseline with PARSEC benchmarks. We assume $SRAM4$ per VC as an area budget, the same as a cache block size. The average network load in PARSEC benchmarks is low, but because they exhibit bursty communication and have *congestion periods* (the time period when the average ratio of buffer occupancy in injection ports is above a threshold, which is set to 75% in our study), our scheme contributes to improve NoC performance. In Figure 3.17, the relative performance improvement of the proposed scheme over the SRAM baseline is not comparable to those shown in Figure 3.14 (vips gets 11.0% improvement while blackscholes gets 9.3% and dedup gets 6.7%), and the STT-MRAM and Hybrid buffers show similar performance. However, when we analyze the performance during the congestion periods, the STT-MRAM buffer outperforms



(a) Different Packet Lengths



(b) Different Area Budgets



(c) Different Number of VCs

Figure 3.18: Sensitivity Analysis

the SRAM and Hybrid buffers by 15.0% and 9.1%, on average, respectively. Also, among the three different schemes, the STT-MRAM router is the most power-efficient, consuming 18.7% and 44.9% less power compared with the SRAM and Hybrid routers, respectively. *Blackscholes* consumes the least total power in the STT-MRAM router by 20.7% and 46.4% compared with the SRAM and Hybrid routers.

3.5.3 Sensitivity Analysis

We perform sensitivity analysis by varying packet lengths, area budgets, and number of VCs as shown in Figure 3.18 to examine their effects on NoC throughput. Figure 3.18(a) shows the normalized throughput improvement with different packet lengths: 4, 8, 12, and 16 flits per packet. All results are normalized to that of baseline SRAM buffer with 4 buffer slots per VC. It clearly shows that the STT-MRAM buffer works better as packet length increases. The longest packet consisting of 16 flits (*PKT_16*), shows the biggest performance improvement up to 30% in the STT-MRAM buffer over baseline SRAM. This is because when the buffer capacity is not big enough to accommodate a whole packet, the packets in transit tend to spread across multiple nodes, thus impeding subsequent packets from proceeding to their destination, which results in significant performance degradation. NoC throughput is also mutually affected by two important factors: buffer depth and number of VCs per PC. Figure 3.18(b) shows the impact of buffer depth on throughput improvement with five different area budgets: $SRAM3$, $SRAM4$, $SRAM5$, $SRAM6$, and $SRAM7$. The more we increase default area budget, the deeper buffer depth we can provide, thus improving network throughput. Across the given budgets, the STT-MRAM buffer shows the highest throughput improvement. Under the smallest budget ($SRAM3$), the STT-MRAM buffer enhances throughput by 22.7% and 10.7% over the SRAM and Hybrid buffers, respectively. However, deepening the buffer depth does not always yield tangible throughput improvement as shown in the largest budget ($SRAM7$). This is mainly

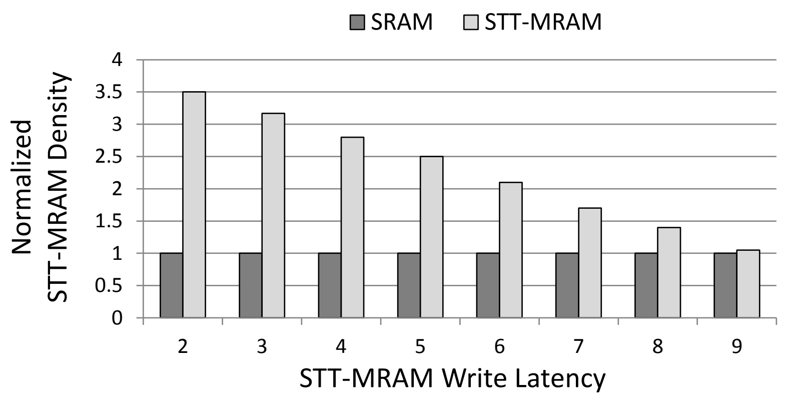


Figure 3.19: Normalized STT-MRAM Density under the Same Per-Router Area Budget

because HoL blocking occurs when many packets contend for router resources (limited number of VCs), but the increased buffer depth does not alleviate this problem. As shown in Figure 3.18(c), increasing the number of VCs per PC is an effective way of improving network throughput further because it allows more packets to share the same PC, thus reducing HoL blocking.

3.6 Discussion

In this section, we evaluate our STT-MRAM buffer scheme under different write latencies of STT-MRAM. We also compare the scheme with other on-chip network techniques, such as Bufferless Routing (BLESS) [94], and Whole Packet Forwarding (WPF) [75].

3.6.1 Impact of Write Delays of STT-MRAM

For our scheme, STT-MRAM write latency is an important factor affecting the overall NoC area and performance. Till now, in our experiments, we assume STT-MRAM has 2-cycle write delay with a density of 3.5 times SRAM, but as we increase the write latency further, the extra logics, such as MUXes and latches, hiding the multicycle writes need to be added. Due to such additional spaces taken up by the extra logics in the STT-MRAM buffer, STT-MRAM is given relatively less area in the given buffer space. Specifically,

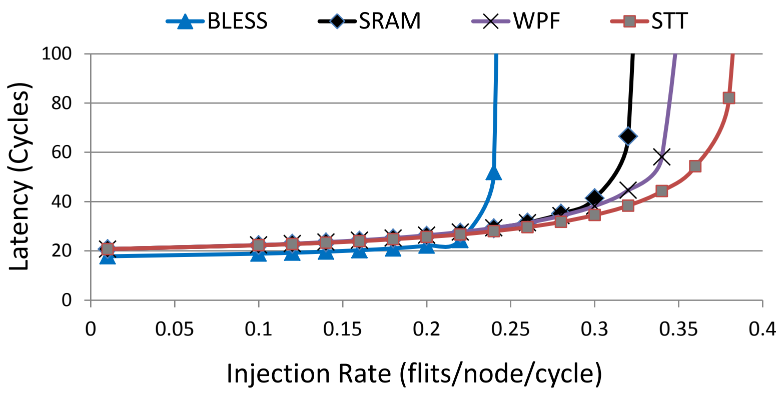


Figure 3.20: Comparisons with BLESS and WPF (UR)

when n (write delay) equals 2, initial single router area, its buffer area (per input port), and extra logic area (per buffer) are 106,709 μm^2, 14,762 μm^2 (A), and 689 μm^2 (B), respectively, where the effective buffer area is 14,073 μm^2 (A-B).

As we increase n, while keeping per router area budget intact, extra logic area increases by approximately 7.5% per additional write latency, thus leading to decreased effective buffer space per input port (Figure 3.19). Across the different write latencies of STT-MRAM (3, 4, and 5), we observe negligible performance degradation (less than 1%) under UR traffic. This is because STT-MRAM still provides enough buffer space to sustain the network bandwidth. However, the performance becomes equal to or less than that of SRAM baseline when the STT-MRAM buffer has similar capacities with the SRAM buffer due to the reduced density. In our configuration, this occurs when the STT-MRAM write latency is 9 or more cycles.

3.6.2 Comparison with Other NoC Techniques

There have been a few studies to improve performance with limited buffer resources in NoC design [94, 75]. We compare the power and performance benefits of our scheme with them. BLESS [94] reduces buffer power and area overheads by eliminating router buffers, and handles network contention by deflecting contending flits to any free output

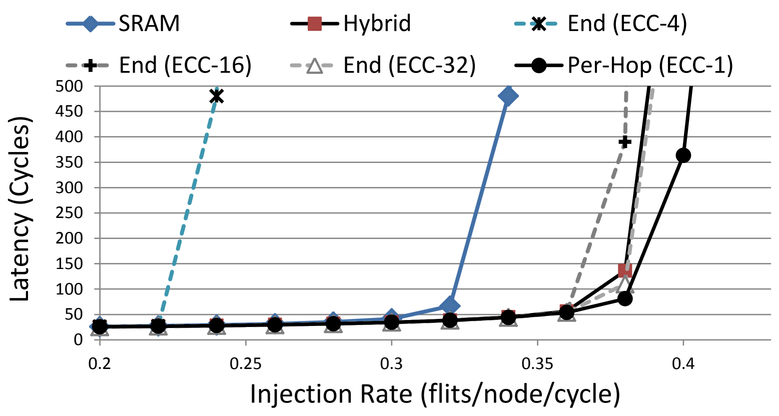


Figure 3.21: Comparisons between Different ECC Schemes (End-to-End vs. Per-Hop)

port. In our evaluation, the performance overheads of BLESS outweigh its gains due to the increased allocator complexities that avoid livelocks, and the extra packet overheads, where flits in a packet contains routing information to be independently routed to the destination. BLESS saves significant router area by eliminating buffer spaces, but the frequent deflections of BLESS at high loads consume significant dynamic power, and leads to early network saturation as shown in Figure 3.20. On the other hand, the STT-MRAM router provides higher throughput by 54.1% than that of BLESS, and is more power-efficient at flit injection rates greater than 22% compared to BLESS. WPF [75] proposes a bandwidth-efficient, fully adaptive routing scheme in VC-limited NoCs where short packets dominate. WPF makes packets traverse all minimal paths, thus enhancing routing flexibility, and provides deadlock avoidance techniques which allow non-empty VCs to be re-allocated, achieving high VC utilization. Under the same area budget, as shown in Figure 3.20, under UR traffic, the STT-MRAM router shows better performance by 8.5% than that of WPF due to the high density buffer properties of STT-MRAM.

3.6.3 Impact of End-to-End and Per-Hop Error Protection

Selecting the location at which error protection schemes should be implemented is another critical issue because it can affect overall NoC throughput and average packet latency due to transient errors in STT-MRAM. Till now, throughout this paper, we focus on per-hop error protection (as detailed in Section 3.4.3), where each node checks flits of an incoming packet and requests retransmission once flits are turned out to be *uncorrectable*. Alternatively, end-to-end data protection is also a viable alternative, where error protection is performed at the destination node and requests retransmission once a packet is in *uncorrectable* error. However, for end-to-end, as a packet travels long distances, the probability of the packet being received in error (beyond given ECC correction capability) increases accordingly, thus possibly resulting in degraded NoC throughput and average packet latency. Figure 3.21 empirically compares NoC performance under different ECC schemes (end-to-end and per-hop) with various ECC capabilities (ECC-1 to ECC-32), where ECC-n refers to ECC correction capability. Even under the highest ECC correction capability (ECC-32), end-to-end scheme shows less throughput compared to that of per-hop with simplest ECC (ECC-1). This is mainly because of the increased number of packets retransmitted in end-to-end scheme as shown in Figure 3.22. For end-to-end, employing strong ECC helps to reduce the number of retransmissions, however, strong ECC requires more parity bits, increasing the total number of flits per packet due to decreased payload space left, thus leading to degraded NoC performance. This clearly shows the benefits of our scheme (detailed in Section 3.4.3) over the counterpart (end-to-end).

3.7 Related Work

Guo *et al.* [28] detailed STT-MRAM based architectural techniques to offer power-efficient and scalable microprocessors. Goswami *et al.* [95] proposed STT-MRAM based GPGPU architectures and hybrid shared memory for power-performance optimizations.

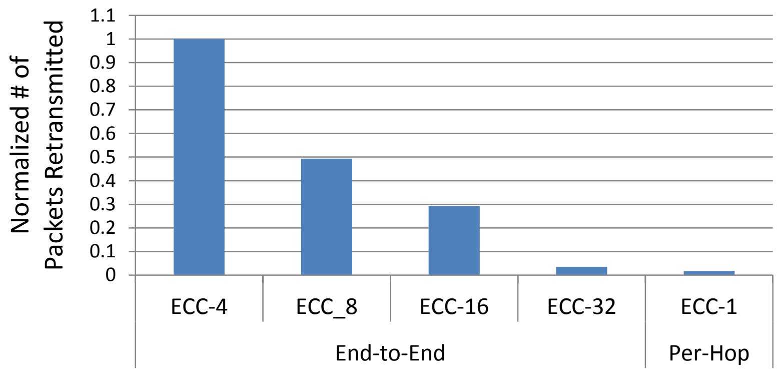


Figure 3.22: Normalized Number of Packets Retransmitted under Different ECC Schemes

In [77, 26, 27], the data retention time of STT-MRAM has been carefully adjusted to achieve better write performance and reduced write energy for caches in CMPs. However, the cache based schemes cannot be directly applicable to NoCs since they are designed for memories having longer data residence time and larger capacity compared to FIFO buffers.

PCM is another emerging memory that has high density, but its low endurance leads to wide adoption in off-chip memories rather than on-chip caches [96]. As an emerging on-chip memory, embedded DRAM (eDRAM) [97] is also gaining popularity due to its low power and high density advantages. However, eDRAM has higher leakage and refresh energy overheads compared with STT-MRAM. Chang *et al.* [97] tackled the refresh power overheads in eDRAM based on dynamic dead-line prediction for energy-efficient LLCs, but their refresh scheme cannot be directly applicable to NoC designs since refresh operations are bypassed for data that are unlikely to be reused, and the predictor works under a cache level with longer data residence times than FIFO buffers. In addition, eDRAM has scalability issues in sub-45 nm process technology [77] making it unsuitable for our NoC buffer design.As emerging memory technologies, PCM and eDRAM each has high density and low power features, but PCM has low endurance, thus leading to wide adoption in off-chip memories [96]. Also eDRAM has higher leakage and refresh power overheads compared to STT-MRAM [97]. Chang *et al.* [97] tackled such refresh power overheads using dynamic dead-line prediction, but the refresh scheme is not appropriate for NoC buffer designs since refresh operations bypass data that are unlikely to be reused, and the predictor works under a cache level. Kodi et al. [31] explored the use of repeaters along the inter-router links as potential buffer storages, and Michelogiannakis et al. [34] presented Elastic Buffers, an efficient flow-control scheme using storages already present in pipelined channels.

There are also prior studies exploring power-efficient architectural supports for NoCs. Power-gating is a circuit-level technique mitigating the static power consumption of NoCs by cutting off power temporarily. However, due to frequent state transitions and unavoidable wakeup time delays, as described in [98], power-gating rather consumes more power at high load, and has higher average packet latency at both low and high load compared to a non-power-gated NoC, and such overheads increase as network scale grows. Bufferless NoC eliminates buffers, thus the peak network throughput is reduced, and as stated in [99], it has higher packet latency overall, resulting in degraded performance. Also, although network power is often significantly reduced at low-to-medium load, bufferless NoC consumes more power as the network becomes congested compared to a buffered NoC mostly due to increased link power from frequent deflections. However, in this work we provide high bandwidth NoC with low power consumption even at high network load using the proposed STT-MRAM router. Similarly, Smullen *et al.* [26] and Sun *et al.* [27] reduced write latency and dynamic energy of STT-MRAM by decreasing the retention time for caches in CMPs. Mishra *et al.* [100] integrated STT-MRAM caches in a 3D CMP and hid long write latency by delaying cache accesses to busy STT-MRAM banks.

3.8 Conclusions and Future Work

In this work, we propose a novel pipelined input buffer design with STT-MRAM for NoC routers. To overcome the weakness of STT-MRAM, the long latency and high power consumption in write operations, we design a multibank STT-MRAM buffer which is a virtual channel with multiple banks. Through this, we avoid performance degradation while consuming less area and power. Also, we address the issue of random data corruption in STT-MRAM by proposing cost-efficient buffer refresh schemes combined with Error Correcting Codes (ECC). Our simulation results show significant performance improvement with less total router power consumption.

The multibank STT-MRAM buffer scheme can be used in multiple domains such as instruction queue, reorder buffer, and prefetch buffer. Since instructions are generally used up quickly, large instruction queues allow for better instruction level parallelism [101]. In this context, we will explore using our scheme as a worthy prospective in terms of power consumption and queue length. There are several studies dealing with the efficiency of reorder buffers [102, 103], and exploring the lifetimes of variables in programs [104], which show that a large portion of the variables whose values are held in the reorder buffers are short lived. This leads to a mixture of variables with irregular lifetimes. We plan to examine a hybrid SRAM/STT-MRAM buffer scheme which accounts for the variation in retention times required. We will explore utilizing STT-MRAM buffers as an attractive alternative to prefetch buffers [105] for saving power and providing larger buffer space since prefetched data do not need to be cached for a long time. We also plan on tackling the challenges of having data retention times tuned to the timeliness of prefetching so as to make our design feasible.

4. NOC DATA APPROXIMATION FRAMEWORK

4.1 Introduction

Approximate Computing [30, 31, 32, 33] has emerged as an attractive alternate compute paradigm by trading off computation accuracy for benefits in both performance and energy efficiency. Approximate techniques rely on the ability of applications and systems to tolerate imprecision/loss of quality in the computation results. Many emerging applications in machine learning, image/video processing and pattern recognition have already employed approximation to achieve better performance [34, 35, 36, 37, 38]. Networks-on-Chip (NoCs) have emerged as the most competent method to connect an ever increasing number of varied on-chip components including conventional cores, accelerators, caches and memory controllers. Communication-centric applications such as image/video processing and emerging memory intensive applications in the big data era place a significant amount of stress on the NoC for high memory throughput, triggering many designs that try to solve the memory bandwidth issue [8, 9, 10, 11]. Hence designing a high-performance NoC, which can efficiently provide high throughput, has become critical to overall system performance. Therefore, the need to explore hardware approximation techniques that can leverage the modern approximate computing paradigm for high throughput NoCs is imminent.

Approximation with error control is important for guaranteed output quality [34]. Previous research has either adopted training during compilation [35, 46] or error control at runtime [34, 32] to reduce the output noise. In NoCs, since the approximation lies on the critical path of data response, it is critical to facilitate low overhead control in the inaccuracy incurred.

In this work we propose APPROX-NoC, a data approximation framework for NoCs to alleviate the impact of heavy data communication stress by leveraging the error tolerance of applications. APPROX-NoC proposes to reduce the transmission of approximately similar data in the NoC by delivering approximated versions of precise data to improve the data locality for higher compression rate. The proposed framework operates by first utilizing an approximation engine, with a lightweight error control logic, to approximate the given data block to the nearest compressible reference data pattern. Then the encoder module of an underlying NoC compression technique [49, 50] is used to compress the data block. We propose a data-type aware value approximatiion technique (VAXX), with a light weight error margin compute logic, which can be used in the manner of plug and play module for any underlying NoC data compression mechanisms. *VAXX* approximates the value of a given data block to the closest compressible data pattern based on the data type,with fast quantitative error margin calculation. The error threshold to control the extent of data approximation allowed can be determined by the compiler or annotated by the programmer and can be dynamically adjusted at run time.

Tightly-coupling the approximation technique with the underlying compression is more economical in terms of area and power efficiency. To this order, we present two low overhead microarchitecture implementations of value approximation for both dynamic dictionary-based compression (DI-COMP), namely DI-VAXX, and static frequent pattern compression (FP-COMP), namely FP-VAXX. The proposed framework operates by compressing the data blocks being injected into the NoC using pattern matching, and decompressing them at the destinations. But unlike traditional NoC data compression techniques [49, 50], the pattern matching is not required to be exact and is controlled by an error threshold, i.e., a data value can be compressed as long as it differs from a reference pattern (frequent pattern/dictionary) within a specific error threshold.

The major contributions of the work are as follows:

- We exploit approximate data similarity in communication, which translates to high data compressibility to reduce traffic load in NoCs thereby improving performance.
- We design an approximation engine with a data-type aware value approximate technique (VAXX) and lightweight error control logic to cater to a wide range of applications.
- Low overhead microarchitectural implementations to materialize the value approximation technique for static and dynamic compression mechanisms are presented.
- Our evaluation results show that APPROX-NoC provides promising opportunities in big data application domain. With an data intensive graph processing benchmark, we achieve latency reduction of 36.7% compared to state-of-the-art compression mechanisms.

The rest of the chapter is organized as follows. Section 4.2 details the related work. In section 4.3 we motivate our work by presenting the motivation and challenges of approximation in NoCs. In section 4.4 we present the architectural overview of APPROX-NoC and the VAXX technique. Section 4.5 explains the microarchitectural implementation and functional principles of the VAXX techniques for dictionary-based and frequent pattern based compression mechanisms. Section 4.6 presents our experimental setup and evaluations. We conclude our work in Section 4.7.

4.2 Related Work

In this section we discuss the related work in hardware approximation techniques and NoC data compression.

4.2.1 Approximation

Significant research has been done regarding approximated computation and data storage in hardware for applications that allow inaccurate outputs. Sampson et al. [30, 47, 106] proposed code annotations and compiler framework for the programmers to define the data/computations in the application that can be approximated. They also propose hardware mechanisms like voltage scaling, reducing DRAM refresh rate and SRAM supply voltage, width reduction in floating point computations for energy savings. Esmaeilzadeh et al. [33] propose dual voltage operation where precise computations use high voltage mode and approximate operations use the low voltage mode. Previous research has also proposed energy efficient accelerators based on neural networks and analog circuits [35, 44, 107, 46]. Liu et al. [48] propose to reduce the refresh rate of DRAM memories which store data, that can be inaccurate, using application level input. Miguel et al. [32] propose, Doppelganger, a cache mechanism which eliminates the storage of cache blocks with data that is similar (need not be exact match). They keep the tags for all the cache blocks, but if two cache blocks are similar then only one is stored and both the tags point to this block. Our mechanism proposes to eliminate the transmission of similar cache blocks by encoding data to a similar data pattern, that is being tracked, at the source node (memory/cache) and hence can work in synergy with approximate storage mechanisms like Doppelganger cache.

4.2.2 NoC data compression

Previous research has explored data compression in NoCs. Das et al. [50] explored compression in caches and the NI of the routers while proposing techniques to amortize the decompression latency with communication latency. They observe that across wide range of workloads data compression leads to significant network power savings and performance benefits. Zhou et al. [108] proposed a data compression mechanism in packet-based NoC architectures by tracking frequently repeated values in the on-chip data traffic. Zhan et al. [109] introduced a base-delta compression technique in NoCs to exploit the small intra-variance in data communication. Jin et al. [49] proposed a data compression mechanism that learns frequent data patterns using a table-based mechanism and adaptively turns the compression on/off based on the efficacy of compression on the network performance. APPROX-NoC proposes to compress the data traffic by facilitating approximate matching with an online error control mechanism.

4.3 Motivation and challenges

In this section we first detail our motivation leading to the use of data approximation in NoCs and then present the challenges of implementing the proposed techniques.

4.3.1 Motivation

4.3.1.1 Data movement is becoming the critical component in multicore systems

The rapid explosion of computational units in comparison with memory bandwidth, and increasing data-movement-to-compute ratio of emerging data-intensive big data workloads has resulted in heavy NoC communication loads. In such scenarios, state-of-the-art NoC designs can rapidly become the communication bottleneck and struggle to deliver the traffic in an energy-efficient manner. Multiple potential solutions like near data processing [8], moving processing to memory plane to reduce the amount of data movement, are being proposed. But even these mechanisms still require significant data movement between processsing and memory planes or within the memory plane, between the different memory slices. Therefore mechanisms that can reduce the communication traffic load in state-of-the-art NoCs become critical to cater to emerging data-intensive applications.

4.3.1.2 Frequently repeated patterns appear in applications

Previous research [49, 50, 108] has proposed using data compression techniques in NoCs to facilitate low latencies even at saturation level of injection loads. It can be trivially deduced that if enough data repetition is present in applications, then data compression mechanisms will be an appropriate antidote to the communication bottleneck issue explained above.

4.3.1.3 Data accuracy is not required

Additionally, applications that allow for approximate outputs do not require exact data to be transmitted across the network for accurate computations. Previous research [30, 33] has proposed an EnerJ framework which can be used by programmers to annotate sections of the data in applications that can be stored approximately. Doppelganger [32] proposes an approximate cache architecture that leverages the similarity between different cache blocks to eliminate redundant data storage. These mechanisms prove that in addition to repetition of specific data patterns, sufficient amount of value similarities, with small variance, between data patterns exists in many applications. But the techniques mentioned above still incur the cost of bringing the data accurately to the cache before determining whether storage is required or not. Therefore, the data movement in the NoC can be further reduced by eliminating transmission of approximately similar cache blocks across the network by using network data approximation techniques.

4.3.1.4 Defining approximate data similarity is necessary

Data similarity is defined according to a predefined error threshold. For example, when 0% error is allowed then the two patterns must be an exact match to be considered similar, however with an error of $e\%$ allowed two patterns are considered similar if the difference between them is less than $e\%$. The value difference is defined as the variance in the value between the two patterns. For example, the 8 bit patterns 10101011 and 10100000 have a value difference of 11.

4.3.2 Challenges

4.3.2.1 Value approximation and compression are not cheap

Value approximation and data compression mechanisms are on the critical path of the data packet injection. The underlying compression techniques have considerable area and latency overheads. Dynamic compression requires storage for pattern tracking and data lookup while static compression incurs significant encoding and decoding logic. The approximation operation adds further latency overhead on to the compression mechanism's latency. Value range and error computation using complex multiplication is expensive and hence can eat up the benefits from flit reduction achieved through approximation and compression. Thus, low latency approximation and error compute logic design are required. Furthermore, the approximate cost should keep small while supporting both integer and floating-point data types. Although the approximation engine can be treated as a plugin module, it is economical to have tightly-coupled approximation and compression implementation alternatives for lower overheads in terms of area, latency and energy.

4.3.2.2 Quality control is important

Approximable applications still require some Quality of Service (QoS) guarantees in terms of the outputs produced or data being supplied. As mentioned in Rumba [34], it is also critical to differentiate overall quality control versus controlling errors in individual elements. Hence the proposed mechanism should be capable of controlling the data error rate individually in each cache block similar to Doppelganger [32] and also across the whole program execution. We assume that the programmer can determine the QoS needed and the compiler can translate this into error threshold allowed in different simultaneously available hardware techniques, i.e. if multiple hardware approximation techniques are concurrently available in the system the compiler/firmware can determine the error threshold each technique can incur. It should be noted that, this way, our mechanism can work in synergy with CPU/cache/storage approximation mechanisms to determine the error budget allowed in each scheme, respectively.

4.4 Approx-NoC Framework Architectural Overview

In this section, we first describe the baseline multicore system architecture and then detail the APPROX-NoC framework. The baseline system includes a collection of heterogeneous tiles connected via an NoC. Each tile may consist of core/accelerator units, FPGA/ASICs, private caches, a slice of the last level cache and/or an on-chip memory controller (MC) unit. The tiles are connected to routers of the NoC, in either a one-to-one or many-to-one (concentrated) fashion depending on the NoC design. Each router connects to the different components of a tile via Network Interface (NI) ports. The packetization/depacketization of injected communication and the flit fragmentation/assembly for flow control are performed in the NI. The NoC traffic consists of control packets for message passing/shared memory and data request/reply packets. The size of the packet varies depending on whether it is an address/control packet or a data packet.

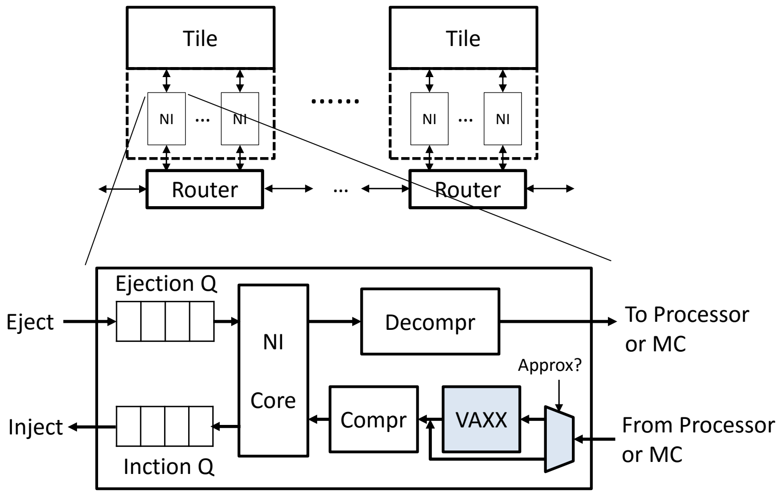


Figure 4.1: APPROX-NoC Architectural Overview.

4.4.1 APPROX-NoC Framework

Figure 4.1 shows the high level architectural depiction of the APPROX-NoC framework. Traditionally, when data to be transmitted enters the NI from the tile, it is packetized and fragmented into flits in preparation for transmission. The packet is then injected into the router via the NI port in a flit-by-flit fashion. When the packet reaches its destination, the flits are assembled to restore the packet. The APPROX-NoC framework consists of a value approximate module, namely VAXX, and an encoder/decoder pair for data compression in the NI. The encoder, of the underlying compression technique, tries to compress each word in the cache block to be transmitted and sends a small encoded index with meta data instead of the whole pattern, thereby reducing the size of the packet being injected into the network. Before compression, the VAXX module facilitates value approximation for the underlying compression scheme as detailed below, thereby improving the compression rate.

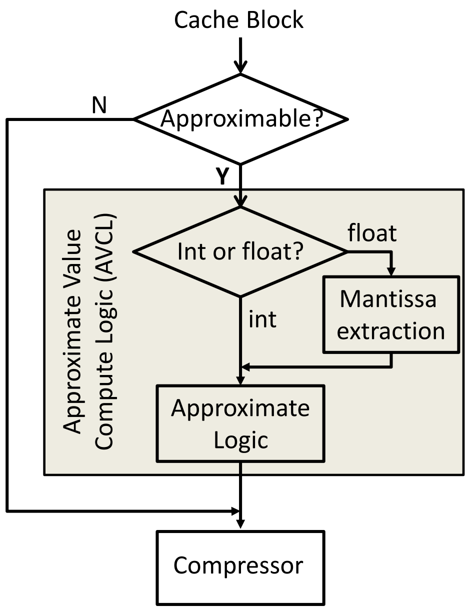


Figure 4.2: APPROX-NoC Operation Flowchart.

Figure 4.2 shows the flowchart describing the functioning of APPROX-NoC. For a cache block waiting to be injected into the network, metadata containing the approximable flag and data type are initially checked. If the cache block is not approximable, it bypasses the approximation (*VAXX*) engine and starts compression. In case of an approximable cache block, the data type is checked and the block is sent to the approximation logic if it is an integer. For floating-point data variables, we approximate only the mantissa fields and the approximation logic for integer values is reused to minimize the area and power overheads. The error range compute unit is also included in the approximation logic and the *VAXX* technique guarantees that the approximated data differs from the precise word within the preset error threshold. The approximated data blocks are then sent to encoder for compression operation.

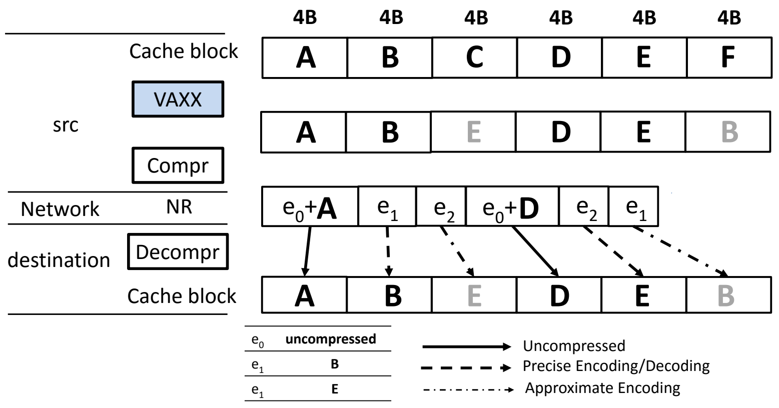


Figure 4.3: Compression and Decompression of a 6-Word Cache Block.

Figure 4.3 shows an APPROX-NoC working example by depicting the encoding of a cache block (24B with 6 x 4B words) at the source and its decoding at the destination. The encoder in this example has two recorded reference patterns B and E, which can be encoded, and the patterns C and F are determined to be approximately similar to E and B, respectively, using the VAXX technique. When the cache block is ready to be injected into the network, the encoder compresses the approximated block to an intermediate network representation (NR) by replacing the candidate data patterns with encoded code. The cache block, now in the NR form, is then packetized, fragmented into flits and injected into the attached router. Note that the patterns C and F are compressed approximately only if the compiler annotates the data to be safely approximable. When the packet reaches its destination, the decoder at the destination detects the reference pattern encoded code to decode the NR into the cache block which is an approximated version of the original cache block, with words C and F replaced by similar words E and B, respectively.

4.4.2 Approximate Value Compute Logic Design

We propose the VAXX value approximate technique to compute an approximate value for a given data block within a predetermined error threshold. In this work we focus on integer and floating-point value approximation. We approximate the cache block, to be transmitted, only when all the words in the block are approximable and this information is assumed to be carried with the access request for this block. The core of *VAXX* is implemented in the Approximate Value Compute Logic (AVCL), which consists of floating-point mantissa extraction, error range compute and approximate logic.

For a given value, the *VAXX* technique needs to compute the variance by which the approximate value can deviate from the provided precise value. For example, for a data pattern 1001(value = 9) and an error threshold of 20% the range of values 8(1000), 9(1001), 10(1010), 11(1011) can be potential matches, i.e., the data value patterns 8, 9, 10 or 11 can be approximately matched to the pattern 9(1001). We observe that in this example the 2 least significant bits are don't cares for the approximate matching, i.e., we can match the pattern "10xx" (approximate pattern) to any reference pattern to make the similarity decision. This computation can be performed using multiplication/division operations but

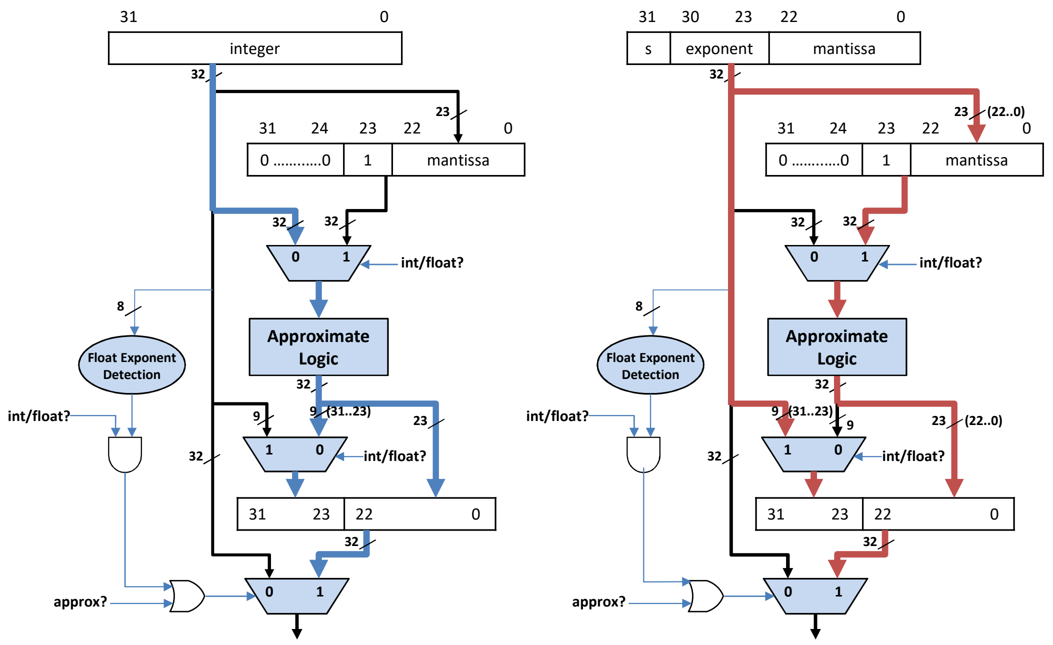


Figure 4.4: Approximate Value Compute Logic.

such a design is too expensive and also unscalable.

To calculate the error range, we first compute the number of bits to represent the largest error a value can tolerate given the predetermined threshold. We simplify the logic by precomputing the number of shift bits, 100/*e* where *e* is the error threshold (%), which are used to shift right the value to compute the error range (*error_range* = *given_value* × (*e*/100) => *given_value*/(100/*e*)). For example, for an error threshold of 25%, the number of shift bits is 4. Hence, when the data pattern value is 128, the *error_range* can be easily determined to be 32.

Floating-point value approximation is more complicated than integer due to the representation. A floating-point value is represented as: $(-1)^{sign} \times (1 + .mantissa) \times 2^{(exponent-bias)}$. We propose to approximate only the mantissa field of floating-point values. The mantissa part is extracted and transformed to scale to the size of an integer value, by padding the most significant bits with zeros. To transform and scale the value of a floating-point value, we extract the 23-bit mantissa part and concatenate it with a higher bit 1 to form the significant, where the exponent part is scaled out. This way both the integer and transformed floating-point variables can use the same approximate logic to maintain low overhead. Figure 4.4 shows the AVCL design in detail, where the datapaths taken by the integer and floating-point variables are represented separately for ease of understanding. The float exponent detection logic determines whether to bypass the approximation unit, for floating-point variables, whenever the exponent is 0 or all 1's, which represent special objects such as zero, denormalized numbers, infinity and NaN. For variables that are annotated to be non-approximable, the AVCL logic is bypassed.

The proposed APPROX-NoC framework can use the VAXX technique on top of any data compression mechanisms. But, trivially adding VAXX modules on top of NoC data compression can be expensive and unscalable due to the computation as well as latency overhead. Therefore it is critical to design microarchitectures that optimize the functionality of VAXX + compression as a whole in terms of area/latency/power. To this extent, in the next section, we showcase two microarchitectural implementation casestudies of the APPROX-NoC framework with two state-of-the-art NoC data compression mechanisms.

4.5 Implementation of APPROX-NoC

In this section, we first present the VAXX implementation for an underlying FP-COMP mechanism, namely FP-VAXX. Next we describe the implementation for a DI-COMP mechanism, namely DI-VAXX. Then we discuss about the latency overhead due to the approximation mechanisms.

4.5.1 Frequent-Pattern Mechanisms

First, we briefly describe the Frequent-Pattern Compression (FP-COMP) technique and then propose low cost microarchitectural implementation for FP-VAXX. Previous research [110] has proposed an FP-COMP mechanism for data compression and [50] has

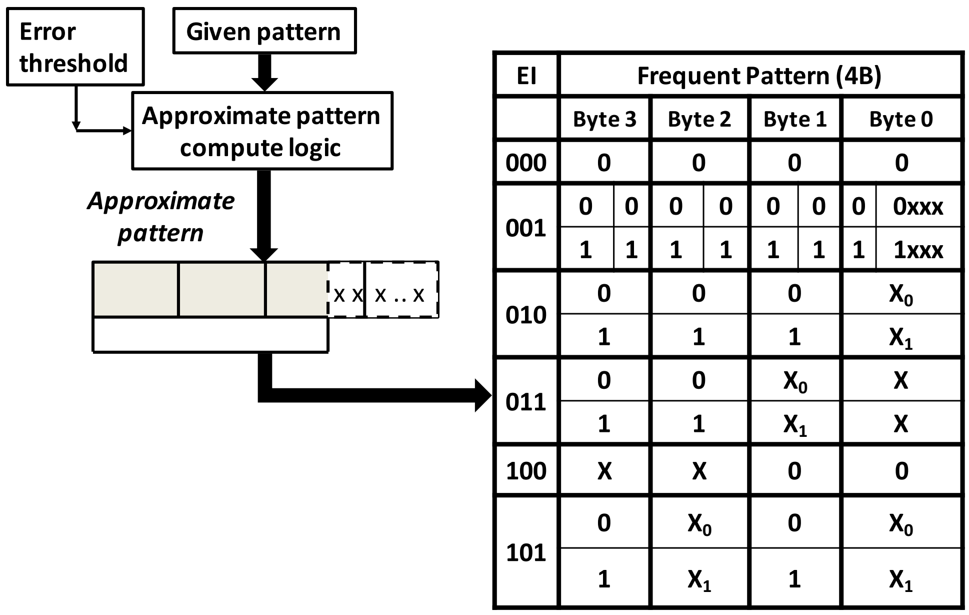
Index	Pattern encoded	Data Size
000	Zero run	3 bits
001	4-bit sign-extended	4 bits
010	One byte sign-extended	8 bits
011	Halfword sign-extended	16 bits
100	Halfword padded with a zero halfword	16 bits
101	Two halfwords, each a byte sign extended	16 bits
111	Uncompressed Word	32 bits

Figure 4.5: Frequent Pattern Compression.

extended it for NoCs with low overhead decompression which we adopt in this work. The mechanism compresses a static set of frequent patterns as shown in Figure 4.5, whereas DI-COMP mechanism detects recurring patterns during run time. The FP-COMP mechanism detects a match on one of the pattern types and sends adjunct data along with the encoded index. Therefore FP-COMP incurs additional decompression complexity due to variable length compression.

4.5.1.1 FP-VAXX Implementation

Figure 4.6 depicts the microarchitectural overview of the VAXX implementation for FP-COMP. For each data word, we first compute the approximate pattern, using the AVCL. Once the don't care bits of the word are determined, the rest of the data word (shaded portion in the figure) is matched with the corresponding portion of the frequent patterns in the Pattern Matching Table (PMT) to find a match and compress on a frequent pattern hit. We propose to utilize a content addressable memory based (CAM) based structure to implement the PMT structure for fast matching. By doing this only the bits, which can be approximated according to the value error threshold, are candidates for approximation and the rest of the pattern must be a complete match to a frequent pattern for compression. For data that is not annotated to be approximable, the AVCL is bypassed to enable exact



EI	Frequent Pattern (4B)			
	Byte 3	Byte 2	Byte 1	Byte 0
000	0	0	0	0
001	0 0	0 0	0 0	0 0xxx
	1 1	1 1	1 1	1 1xxx
010	0	0	0	X_0
	1	1	1	X_1
011	0	0	X_0	X
	1	1	X_1	X
100	X	X	0	0
101	0	X_0	0	X_0
	1	X_1	1	X_1

X_0: 0xxxxxxx X_1: 1xxxxxxx   CA: Compress Arbitration   EI: Encoded Index
X: xxxxxxxx   0: all 0's   1: all 1's

Figure 4.6: FP-VAXX Microarchitecture.

matching for given data words.

4.5.2 Dictionary-Based Mechanisms

Dictionary-based Compression (DI-COMP) keeps track of recurring data patterns dynamically and maintain an encoded-index consistency between senders and receivers so as to compress any occurrences of those data patterns in future communication between these senders and receivers. The most critical feature of a dictionary based compression scheme is the consistency in the encoded-index association between senders(encoders) and receivers(decoders). There should never be a case where a sender transmits an encoded-index which cannot be decoded (decompressed) by the receiver. We propose to utilize a distributed mechanism, similar to the one proposed in [49], and additional meta-data in the table entries, that work in tandem to ensure table consistency. Figures 4.7(a) and (b) shows the microarchitectural depiction of the encoder and decoder pattern matching

Data pattern	Frequency counter	Vector of indices							
		0	1	2	4	5	6	7	8
0000	--			11			00		
0101	--				00				
1011	--					00			
1111	--						01	11	

(a) Encoder PMT at Node 3.

Data pattern	Frequency counter	Index	Vector of valid bits							
			0	1	2	3	4	5	7	8
0000	--	00	0	0	1	1	0	0	0	1
1111	--	01	0	0	0	1	1	0	0	0
1100	--	10	0	0	0	0	0	0	1	0
1110	--	11	1	0	0	0	0	0	0	0

(b) Decoder PMT at Node 6.

Figure 4.7: The Encoder PMT at Node 3 and the Decoder PMT at Node 6.

tables(PMTs) with size of 4 entries respectively, in a 3x3 NoC. In the encoder pattern matching table(PMT) each entry contains a data pattern, frequency counter and a vector of encoded indices, each corresponding to one destination router (decoder), i.e. in a N node NoC each entry will have a vector of (N-1) encoded indices. For a data pattern in the encoder PMT, the vector of indices indicates whether this data pattern can be compressed for a particular destination in the network. In addition, the encoder PMT can have different encoded index values for different destinations, for the same data pattern, since each decoder performs detection in an independent fashion. The decoder PMT entries consist of the data pattern, frequency counter, encoded index and a vector of (N-1) valid bits, one for each of the N-1 encoders. The decoders detect recurrent data patterns and place them in decoder PMTs while sending an update notification to the encoder, with the new encoded index. The vector of valid bits indicates all the encoders that also have this data pattern in their PMTs and is used when replacements happen to invalidate the pattern at all encoders. While the mechanism explained above ensures the encoder and decoder PMT consistency, we discuss the PMT organization and auxiliary logic needed to provide low overhead matching operation for VAX and BAX in the following sections.

The operation of the table management mechanism is as follows:

- The receivers(decoders) detect recurrent data patterns and place them in their decoder PMTs, if it's not already present in the table.

- Then, the receiver sends an update notification to the sender of the corresponding new data pattern along with the encoded index and also sets the valid bit for the corresponding sender in the vector of valid bits in the decoder PMT. The encoder then updates its PMT with the data pattern, if its not already present in the table, and places the encoded index in the corresponding receiver's slot in the vector of encoded indices.

- From this point, the encoder can start compressing that particular data pattern. Whenever the data pattern appears in the communication traffic the encoder checks that there is a valid index set in the vector of indices(table entry) for the destination(for this data packet). This way the pattern is only compressed to an index if the receiver already has the pattern in its decoder PMT.

- The encoder might receive update notifications from multiple receivers for the same data pattern. If the data pattern is already present in the table when an update notification is received, the encoder just updates the receiver's slot in the vector of indices with the received index.

- Similarly, if a decoder detects a data pattern already present in the decoder PMT it checks to make sure that valid bit of the corresponding sender is set. If not set, then the decoder updates the valid bit and sends an update notification to the sender so that it can start compressing this particular data pattern.

- **Replacement:** Replacements in encoder or decoder PMTs lead to consistency issues since all the other nodes need to be notified of such changes.

- If a replacement needs to happen in an encoder PMT as a result of a new update, the frequency counter is used to select the least frequently accessed table entry as the candidate.

- If a decoder needs to replace a table entry, the valid bit vector of the candidate table entry is read and invalidate notifications are sent to all the senders(encoders) with a set valid bit. The encoders will then nullify the index for this decoder in the PMT entry's vector of indices, i.e, the encoder can still keep the data pattern if indices are set for other receivers. The decoder then replaces the candidate entry with the new pattern after it receives acknowledgements from all the valid encoders.

When a data pattern arrives at the decoder it checks its PMT and the TB in parallel. In case of a PMT hit, no operation is needed on the TB. In case of a PMT miss, the TB access result is checked. In case of TB miss, the new data pattern is placed in the TB and in case of a TB hit, the frequency counter is increased. If the frequency counter of a TB entry crosses a preset threshold, the data pattern is moved to the PMT. This mechanism reduces the number of decoder PMT replacements needed by ensuring that only frequently accessed patterns are placed in the PMT.

4.5.2.1 DI-VAXX Implementation

In order to optimize the microarchitectural cost of implementing VAXX matching with the DI-COMP mechanism we modify the operational flow of the approximation as described in Section 4.4. Instead of passing a given data block through the AVCL before

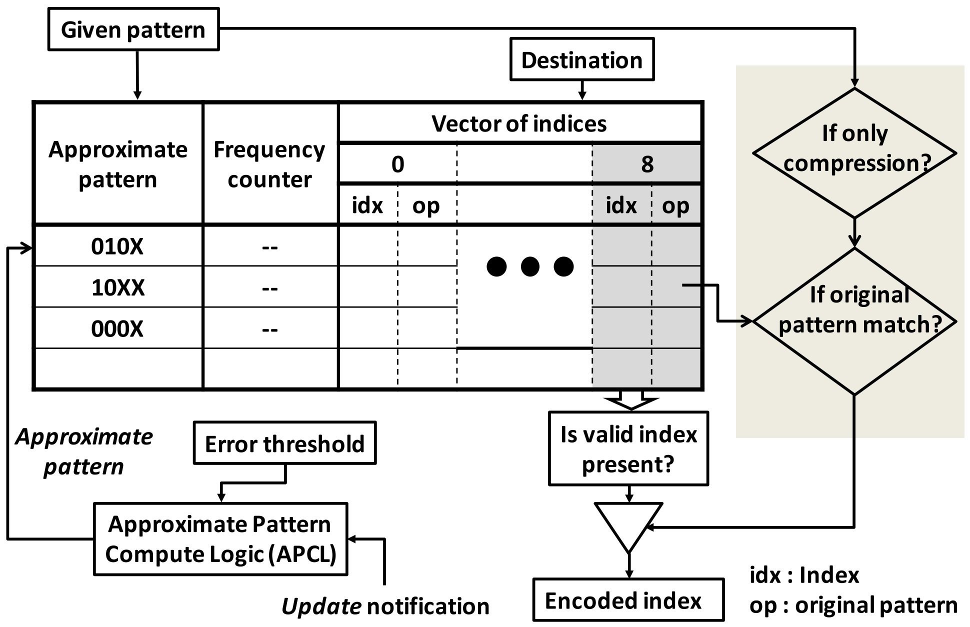


Figure 4.8: DI-VAXX Microarchitecture.

reaching the compression logic we integrate tightly the AVCL with the DI-COMP scheme. We propose to compute the approximate pattern for every reference pattern, at the time of the pattern being recorded, in the DI-COMP scheme and save the approximate versions of the reference patterns. This way any given pattern can be compared to a set of approximate patterns for fast matching and hence the AVCL is removed from the critical path of the packetization.

We propose to use a Ternary Content Addressable Memory (TCAM) structure to optimize the time required to perform value-based approximation. TCAMs function similar to a CAM, and in addition to 0 or 1, a third state of “x” (don’t care) is allowed, i.e., in a TCAM we can actually store 10xx for a pattern (1001) and the table entry will result in a match for the patterns 1000, 1001, 1010 and 1011. The decoders utilize a regular CAM structure to recover the original pattern (1001) based on the index. The microarchitecture of the TCAM-based encoder PMT microarchitecture is shown in Figure 4.8 and the operation is explained below:

- The receivers (decoders) detect frequent data patterns and send an update to the encoders to reflect in the PMT.
- When the encoder receives an update, instead of just storing the original pattern it computes the approximate pattern with don't care bits (e.g. 1001 –> 10xx) based on the error threshold, using the Approximate Pattern Compute Logic (APCL). Then the encoder records the approximate pattern in the TCAM and stores the index for the corresponding receiver. If a matching TCAM entry was already present the encoder just updates the index.
- When a data pattern arrives at the encoder, the TCAM is accessed and in case of a hit the encoded index is used for compression. This way the latency overhead on the critical path of compression is reduced.

For data packets that are not annotated for approximation this TCAM-based mechanism cannot provide compression since a TCAM match does not guarantee that the recovered pattern at the receiver is the same pattern the sender intended to transmit (e.g. 8 can match in TCAM and be recovered as 9). To facilitate exact matching along with approximate matching, we propose to add storage capability in the encoders for the original patterns in addition to the TCAM entry (approximate pattern). Figure 4.8 shows the encoder PMTs with the original pattern storage. Each TCAM entry can have multiple original patterns because different receivers (decoders) could have detected different patterns in the range of values. We propose to store multiple original patterns for each entry and this way when a data pattern which cannot be approximated arrives at the encoder, first the TCAM entry is matched and then an exact match on the corresponding original pattern (based on receiver) is checked before compressing it. The storage overhead can be optimized by storing only the bits of the original pattern that were made don't cares in the approximate pattern.

4.5.3 Latency Overhead

We assume a three cycle compression latency (two cycles matching + one cycle encoding) and two cycle decompression latency overhead for each cache block as mentioned in [50]. To ensure that the DI-VAXX and FP-VAXX matching can happen within the provisioned compression latency, based on the latency overhead evaluations we propose parallel hardware matching units. In case of DI-VAXX and FP-VAXX we have 8 parallel TCAM matching units since two matches per cycle in each unit is possible based on the model from [111] and in addition FP-VAXX requires 8 APCL units.

In addition, we propose to use two latency hiding optimizations to reduce the compression overhead. First, we propose to perform the virtual channel arbitration of the packet, using the header flit which is not compressed, in parallel with the compression. We amortize the compression overhead with the NI queueing time, i.e, if there are previous packets waiting in the queue, the compression overhead would not add to the critical path network latency of the packet.

4.6 Evaluation

In this section we first explain our experimental setup and then present the evaluation of the APPROX-NoC framework.

4.6.1 Methodology

4.6.1.1 Experimental Setup

We evaluate our APPROX-NoC framework using a cycle accurate, in house NoC simulator and a full system simulator, gem5 [52]. We implement the DI-VAXX and FP-VAXX mechanisms in addition to the DI-COMP and FP-COMP mechanisms [49, 50] in both the

Table 4.1: APPROX-NoC Simulation Configuration.

System parameters	32 Out-of-Order Cores at 2GHz 32KB L1I$and 64KB L1D$, 2-way 2MB L2$ and 16 directories Cache Coherence: MOESI_hammer
NoC parameters	4×4 2D concentrated-mesh 2GHz three stage router 4 Virtual channels(4-flit buffer) 64-bit flit size wormhole switching, XY routing
Error threshold	5%, 10%(default), 20%
Approximable data packet ratio	25%, 50% 75%(default)
Dictionary-based mechanisms	8 entry PMT

simulators. For detailed network impact evaluations we use the NoC simulator where we set the default error threshold as 10% and the percentage of approximable data packets is set to 75%. We later perform sensitivity studies to show the impact of varying these parameters. To evaluate the impact of our APPROX-NoC mechanism on the overall application output error, we utilize the Pin [112] tool for instrumentation. We hand-annotate the benchmarks mentioned below, in similar fashion to Doppelganger [32], to identify the data regions which can be approximated. The VAXX mechanism uses the knowledge of the data type (floating point or integer) of variables in each benchmark to determine the approximation operation. An important consideration while hand-annotating approximable data regions of benchmarks is the data type of the variables being determined to be approximable. We assume that the data type of the cache block being compressed is known to the

APPROX-NoC framework and we conservatively only compress cache blocks in which all the words have the same data type. This is because knowledge of the data type of each word would require significant metadata overhead. We use gem5 to evaluate the impact of our approximation mechanism on the overall system. The APPROX-NoC configuration and the NoC parameters used for our evaluation are listed in Table 4.1.

4.6.1.2 Workloads

We utilize benchmarks from the PARSEC [113], with simlarge, which have been previously utilized for evaluating approximation mechanisms [11]. In addition, we explore the approximation opportunities in big data analytics by modifying *SSCA2* [114], a data intensive graph benchmark, to evaluate betweenness centrality (BC) in real-world graphs [115]. BC is a popular graph analysis technique to identify important entities in large-scale networks. We approximate the floating-point pair-wise dependencies that is used for centrality calculation. Such applications in big data analytics can leverage approximation in data segments (e.g. weights in graphs) within a tolerable error margin since most algorithms approximate the result by only evaluating on a subset of the data with sampling. We run the benchmarks using gem5 [52] to evaluate the impact of our mechanisms on the system performance and to collect the communication traces for the region of interest, which are then fed into our NoC simulation environment and simulated for 100 million cycles for detailed NoC evaluations. To evaluate the throughput impact we utilize synthetic workloads. We collect the data injected at each node, from the gem5 benchmark traces and utilize the data traces to create data packets in the synthetic workloads. This way, the synthetic workloads can be used to vary the traffic pattern/injection rate but the data being communicated can be kept constant and correlated with data locality in the benchmarks.

4.6.2 Performance Analysis

In this section we present the NoC level performance evaluation of the APPROX-NoC framework using benchmarks from different application suites and synthetic workloads. We first, analyze the performance impact of APPROX-NoC on the average packet latency, compression ratio, then use synthetic workloads to evaluate the impact on network throughput.

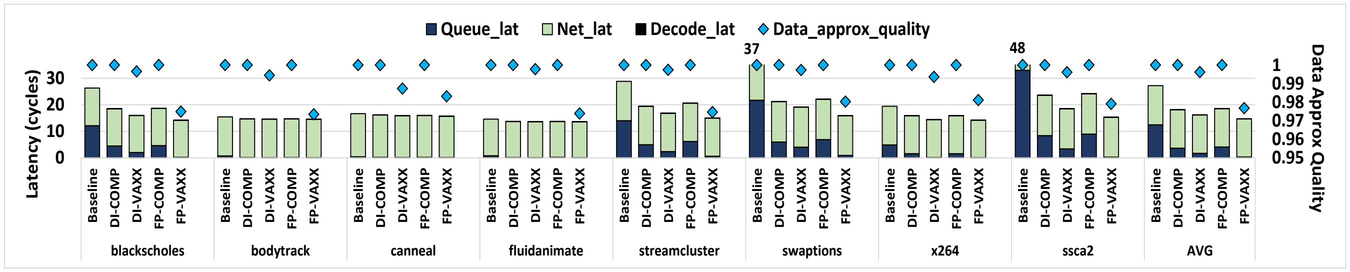
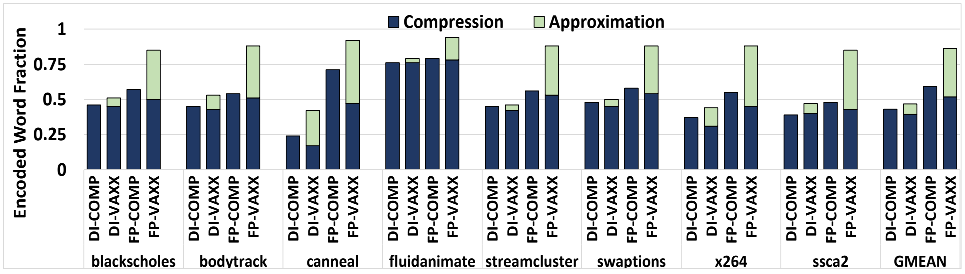
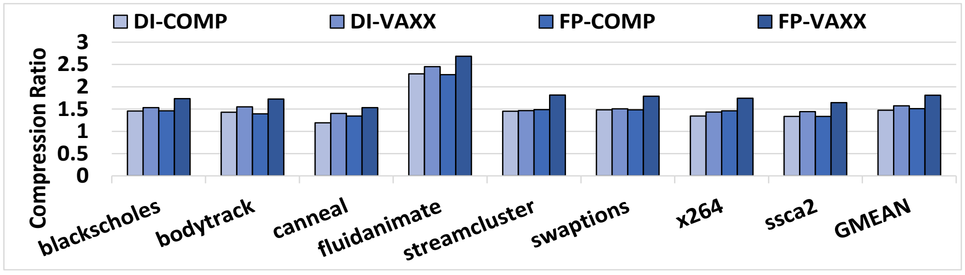


Figure 4.9: Average Packet Latency Breakdown and Overall Approximation Quality.



(a) Fraction of encoded words



(b) Compression ratio

Figure 4.10: Fraction of Encoded words Breakdown to Exact Compression and Approximation (a) and Compression Ratio Improvement of VAXX (b).

4.6.2.1 Impact on Performance

Average Packet Latency. The average packet latency comparison, in a 4x4 2D concentrated mesh NoC, for the two implementation of APPROX-NoC is shown in Figure 4.9. Across the benchmarks DI-VAXX reduces the average packet latency by 11% with respect to DI-COMP and 40.7% compared to Baseline. FP-VAXX achieves up to 21.4% and 46.5% latency reduction compared to FP-COMP and Baseline, respectively. This is mainly due to the fact that approximation allows for more reduction in the number of injected flits leading to performance benefits, especially when the network is congested during the bursty phases. The large packet latency reduction in *SSCA2* graph benchmark is owing to the data intensive nature of the application. With a large data set, the limited cache size cannot hold the whole working set of the benchmark, and hence its irregular data accesses incur large volume of data movement. We expect that data intensive applications, in big data era, that have a high ratio of data movement to computation traffic will benefit from APPROX-NoC.

Note that the queuing latency decreases significantly by introducing approximation since the single-flit control packets face lesser blocking delays caused by the long data packets. The decoding latency portion of the average packet latency is negligible because it is amortized over the large number of control packets, and also compensated by the reduced queueing latency. In addition, it is interesting that the VAXX techniques have larger impact on packet latency with the FP-VAXX mechanism compared to the DI-VAXX. This is because the DI-VAXX mechanism needs to learn the data locality at the beginning of each new communication phase by tracking and updating its locality tables, thereby loosing approximation opportunities. In contrast, the FP-VAXX can use the static patterns across the whole program execution. For some benchmarks (*bodytrack, canneal, fluidanimate*), *VAXX* only achieves moderate improvement. This is because packets in these benchmarks have low queueing and network latency and the flit reduction translating to lower serialization latency is offset by the approximation/compression/decompression overheads. In addition, the percentage of data packets injected is very minimal compared to control packets, and hence the reduction in data flits does not show a significant impact on overall packet latency. The low queuing latency also supports the argument of low data to control packet ratio.

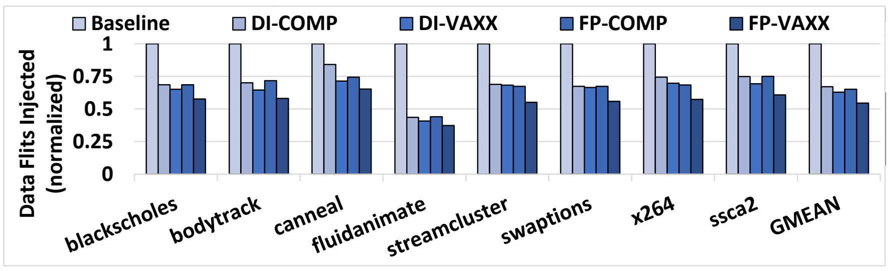
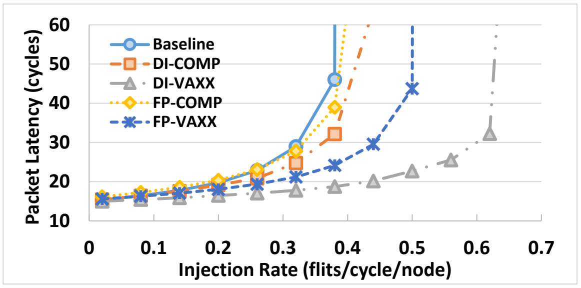


Figure 4.11: Reduction in Number of Injected Flits.

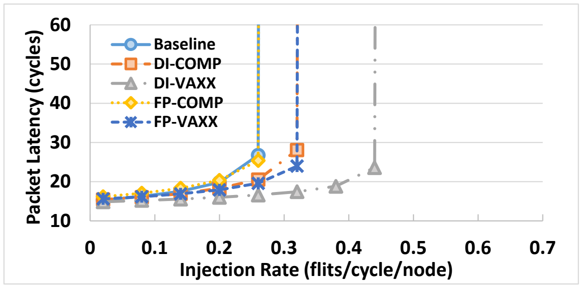
Approximation Effectiveness. The reduction in traffic load is shown by plotting the number of data flits injected under each APPROX-NoC mechanism in Figure 4.11. The DI-VAXX mechanism reduces the number of data flits injected by 3% and 38% compared to the DI-COMP and Baseline, respectively. Similarly, FP-VAXX reduces data flit volumes by 19% and 45% with respect to FP-COMP and Baseline, respectively. The moderate traffic reduction in *streamcluster* and *swaptions* benchmarks when juxtaposed with the large latency improvement seems to be counter intuitive. This can be explained by two reasons. Firstly, the value approximation enables injection acceleration for critical data, thereby translating to reduced queuing latency for the many short packets that are blocked. Therefore even though the reduction in injected flits is small the effective resulting latency reduction can be amplified. In addition, in dynamic compression, approximation may change the learnings of the DI-COMP mechanism, which might affect the compression chance of data that is required to be precise. Hence the overall flit reduction might be smaller due to changes in the operation of the DI-COMP learning. But overall we observe that the network traffic reduction translates to average packet latency improvement.

This is further supported by Figure 4.10 (a), which shows the breakdown of the fraction of encoded words to exact compression and approximated compression. We observe that the *VAXX* technique increases the encoded word fraction by up to 18% for DI-VAXX compared to DI-COMP and up to 37% for FP-VAXX over FP-COMP. Figure 4.10 (b) depicts the effectiveness of value approximation in improving the compression ratio. DI-VAXX and FP-VAXX enhance the compression ratio by up to 21% and 41% compared to the corresponding compression schemes, respectively. On average, the two VAXX implementation increase compression ratio by 10% and 30%. Figures 4.10 and 4.11 show that the reduction in number of injected flits does not scale proportionally to increase in compression rate due to approximation. This is because of internal fragmentation in the 8B flits where a large portion of the tail flit can be empty, since the NR might not be a multiple of 8B.

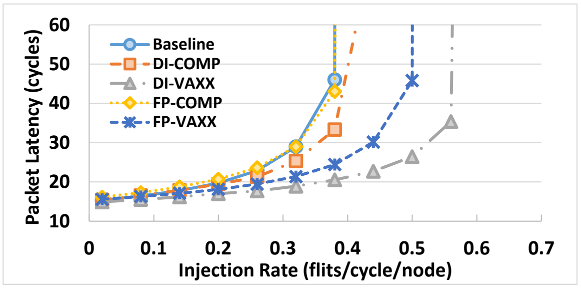
Data Value Quality. Figure 4.9 also depicts the data value quality for each benchmark, i.e., even though the error threshold is checked for approximating each word, the incurred error differs from word to word, so we compute the actual overall data error incurred across the benchmark execution and show the overall data value quality achieved. Across the benchmarks, though we allow for 10% error rate the effective data value quality is higher than 97%, which is due to a portion of the words being compressed without error and most of them matching with close proximity. Note that this is the quality of the integer and floating-point data values, and we analyze how this variance translate to overall application output error later.



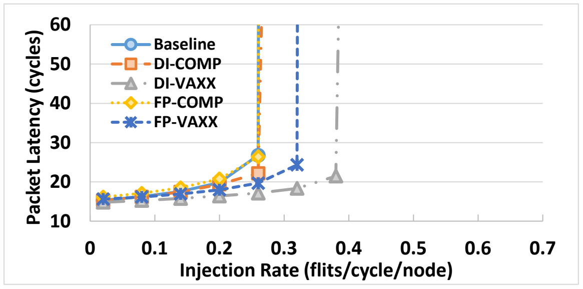
(a) blackscholes (UR)



(b) blackscholes (TR)



(c) streamcluster (UR)



(d) streamcluster (TR)

Figure 4.12: Throughput Analysis with Different Benchmark Data Traces Under Uniform Random (UR) and Transpose (TR) Traffic Patterns.

4.6.2.2 Throughput Analysis

We use synthetic workloads to analyze the impact of APPROX-NoC on the network throughput. Figure 4.12 plots the throughput of the APPROX-NoC mechanisms compared against the Baseline, DI-COMP and FP-COMP compression schemes. We plot for data traces from *blackscholes* and *streamcluster* benchmarks, and for the Uniform Random (UR), Transpose (TR) traffic patterns. The simulations are run for 1 million cycles and we assume a 25:75 data to control packet ratio to emphasize the significance of APPROX-NoC when large amount of data is communicated. When compared to the compression schemes, *VAXX* improves the throughput by up to 40% for UR and 69% for TR traffic patterns. This gain is achieved by reducing the effective injection load, due to approximating data. The huge increase in throughput compared to the latency benefits observed from benchmarks can be attributed to the larger ratio of data packets being injected. Another interesting observation is that the DI-VAXX perform better than the FP-VAXX. This is because of higher data value and temporal locality in the synthetic workloads at higher injection rates with larger data packet ratio. From our observations, the dynamic dictionary-based scheme tends to work well for applications with high data locality and intensive data movement due to its learning capability, while the static frequent pattern scheme tends to work well for applications with many frequent patterns and short communication phases without learning.

4.6.3 Sensitivity Studies

In this section we show the sensitivity of APPROX-NoC mechanisms to the error threshold and the percentage of approximable data packets.

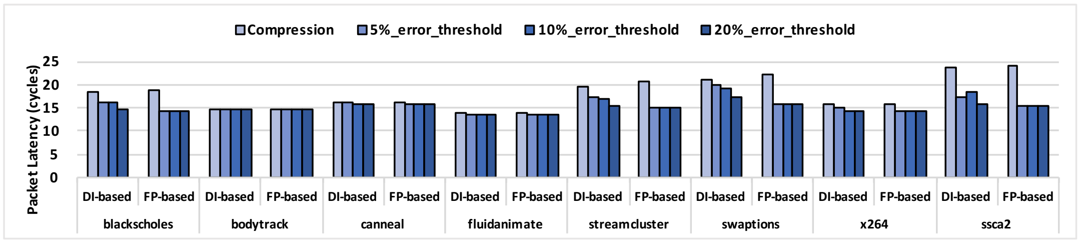


Figure 4.13: Error Threshold Sensitivity Analysis.

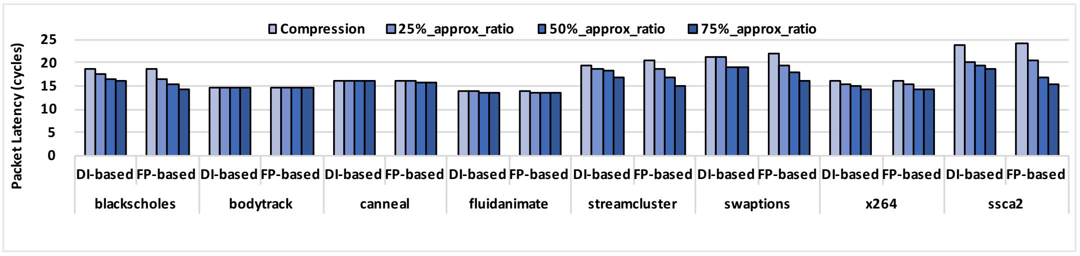


Figure 4.14: Approximable Packets Ratio Sensitivity Analysis.

4.6.3.1 Error Threshold

Figure 4.13 shows the average packet latency across the APPROX-NoC mechanisms for all the benchmarks by varying the error threshold. As the error threshold is increased from 5% to 10% (default) to 20% the impact of the APPROX-NoC mechanisms on packet latency amplifies due to the increased chance of approximate matching. One interesting observation is that FP-VAXX mechanism does not seem to have a significant impact on the packet latency even though a higher error threshold is allowed. The reason for that is our approximation technique can translate the approximate value into higher compression ratio even with small error threshold. It is well matched with the static frequent pattern compression. Despite the moderate latency improvement, we also observe that FP-VAXX incurs more overall error compared to DI-VAXX. This is because in the FP-VAXX mechanism, we always try to match with the highest priority frequent pattern in the PMT even though an exact match is available at lower priority. Hence some of the exact matches, when error threshold was lower, might be converted into approximate matches as the error threshold is increased. So these scenarios can lead to additional error incurred without latency benefits.

4.6.3.2 Approximable Packets Ratio

Figure 4.14 shows the average packet latency for the APPROX-NoC mechanisms across benchmarks as the percentage of packets approximable is varied. The packet latency benefits improve as the percentage of approximable packets increases due to the enhanced chances of approximate matching. This can be observed significantly in *SSCA2, swaptions, streamcluster* with both DI-VAXX and FP-VAXX, while the other benchmarks do not show compelling latency reduction as the percent of approximable packets is increased. The is due to the low queuing latencies in the NoC and small data-to-control packet ratio for these benchmarks leading to minimized impact of data flit reduction on the overall network latency.

4.6.4 Full System Impact Analysis

In this section we use Pin [112] and gem5 [52] based evaluations to analyze the impact of APPROX-NoC on the overall system. We present the overall application output errors and the overall runtime impact due to approximation on different benchmarks.

4.6.4.1 Overall Application Output Error

We analyze the impact of our mechanism on the overall application output quality in addition to the data quality using the Pin [112] instrumentation framework. We implement our approximate functionalities on top of a coherent cache simulator tool. We model a system with 16 cores and each core has a 64 KB two-way L1 private data cache of cache line size of 64 Bytes. We emulate packet response whenever a miss happens, that requires

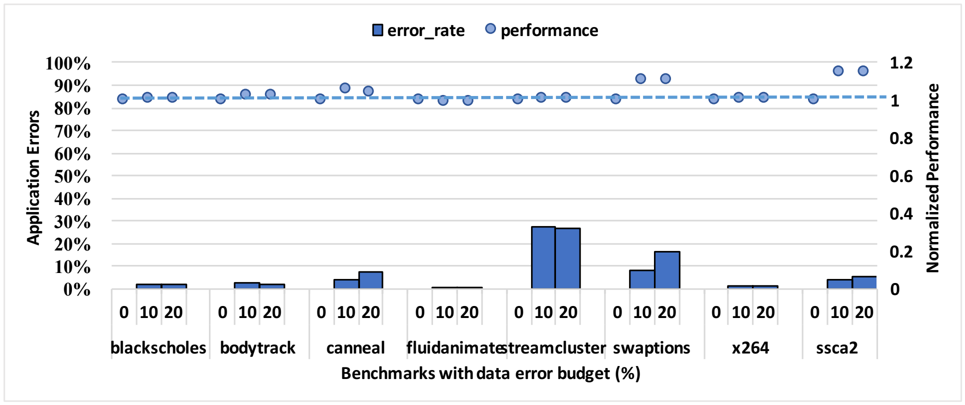


Figure 4.15: Application Output Accuracy and Normalized Performance.

a data response from another node.

To evaluate the applications' output quality, we extend application-specific accuracy metrics based on prior approximate computing research [39, 11, 32, 106]. In addition, we exploit value approximation opportunities in big data domain by studying a graph processing benchmark *SSCA2*. *SSCA2* calculates the betweenness centrality scores of the nodes in a small world network to identify the key entities. So we evaluate the pair-wise betweenness centrality difference between the approximate output and its precise counterpart for error calculation.

Applications' output accuracy for all benchmarks are shown in Figure 4.15. With the predetermined 10% data noise margin, all the benchmarks are well controlled within the error bound except for *streamcluster*. This because by approximating the coordinates, the cost between points and centers might deviate from the precise one and lead to mismatch of centers between the approximate version and precise version. As mentioned in previous work, through approximate space exploration or training during compilation we can improve the accuracy while maintain the performance benefit [39, 106].

In Figures 4.16, we show the application output of *bodytrack*'s approximated and precise pair. The two figures are very similar and the difference is hardly captured through

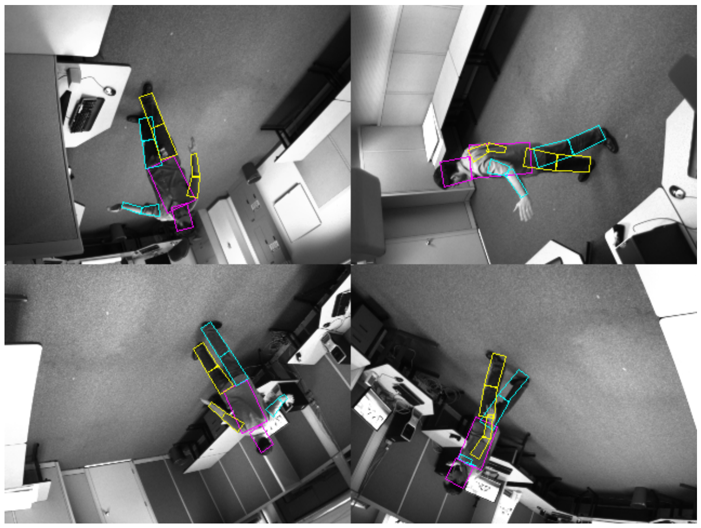
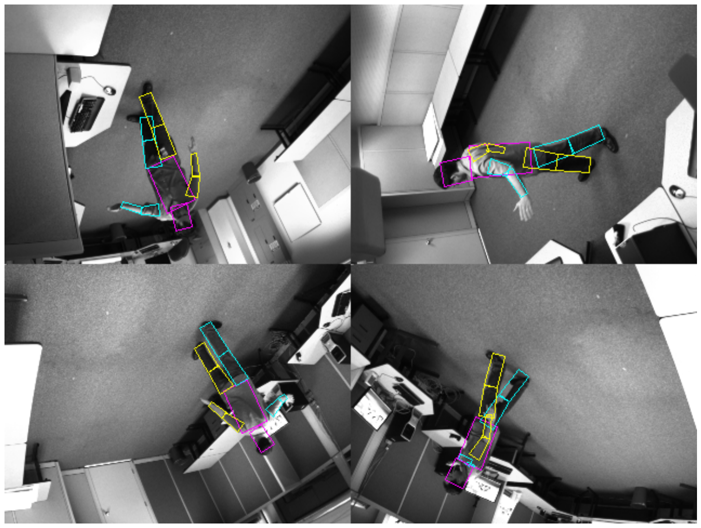

(a) Precise Output.

(b) Approximate Output.

Figure 4.16: Approximate versus Precise Output of Bodytrack.

human vision. In this experiment, we allow for 10% error threshold in the data and observe that the overall output vectors differ by 2.4%.

Figure 4.15 also shows the output accuracy with different error thresholds. Even with 20% error budget, the applications' output errors are close to 5% except for *streamcluster* and *swaptions*. With the bounded data error control, APPROX-NoC can achieve high throughput and low latency by exploiting approximate communications while maintaining acceptable output quality.

4.6.4.2 Overall Application Performance

Next we analyze the impact of APPROX-NoC on the overall system performance. We configure a 64-core CMP connected by an 8x8 mesh network, and run the benchmarks for 100 million instructions with medium input size. Figure 4.15 also shows the normalized performance for different benchmarks with the proposed approximation mechanism as the error threshold allowed is varied, normalized to 0% error threshold allowed. We observe that the performance is improved by upto 10% and 14% in *swaptions* and *SSCA2*, respectively, while we see moderate improvements on the rest of the benchmarks. This is because *swaptions* and *SSCA2* have higher degree of sharing in the approximable region of interest in the application code compared to the other benchmarks. Higher degree of sharing leads to a significant amount of similar approximable data being transferred across the NoC during the execution of these benchmarks, thereby improving the efficacy of our mechanism in impacting the overall performance.

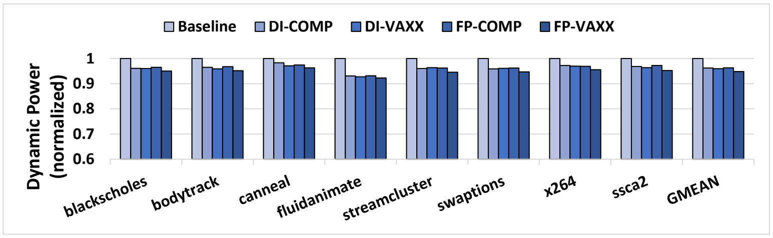


Figure 4.17: Dynamic Power Consumption Normalized to Baseline.

4.6.5 Power Consumption and Area Overhead

In this section, we evaluate the effect of APPROX-NoC on the network power consumption and area overhead, while taking into consideration the overhead of approximate matching and compression/decompression. The static power consumption does not vary across benchmarks and the static power overhead of all the APPROX-NoC mechanisms is minimal compared to the large baseline static power consumption. Hence to show the variation in power consumption between APPROX-NoC mechanisms and benchmarks, we depict dynamic power consumption in Figure 4.17. The best performing FP-VAXX mechanism reduces the dynamic power consumption on average by 5.4% compared to baseline and 1.3% compared to FP-COMP. Note that this can be primarily attributed to the reduction in the number of injected flits which compensates for the power overhead of VAXX techniques.

Based on the hardware requirements we evaluate the area overhead of the APPROX-NoC encoders using CACTI [116] and verilog based area analysis with 45nm technology. The DI-VAXX incurs 0.0037 mm^2 for each NI (router). Similarly, FP-VAXX require an overhead of 0.0029 mm^2. The decoder design does not change between the schemes and the overhead is as mentioned in [50].

4.7 Conclusions

In this work we propose APPROX-NoC, a hardware data approximation framework for high throughput NoCs in the memory intensive big data era. We present a value based approximate matching technique to use in a plug and play fashion with any underlying data compression mechanism. We also detail low cost microarchitectural implementations of the VAXX techbique with state-of-the-art dictionary-based and frequent pattern-based NoC data compression mechanisms. Our evaluation results show that the best APPROX-NoC mechanism reduces the average packet latency up to 21.4% over state-of-the-art NoC data compression mechanism. In addition, our evaluation results with synthetic workloads show that the best APPROX-NoC mechanism improves throughput up to 60% compared to state-of-the-art compression mechanisms. We observe that the FP-based mechanisms achieve higher approximation rate and hence performance benefits across the benchmarks, but the DI-based mechanisms outperform the FP mechanisms when there is significant data repetition. On average the application output quality is always above 99% across the benchmarks even though a 10% error threshold is allowed since a large portion of the words are within close proximity.

As future work, we intend to leverage this high approximation quality by using window based instead of word based error threshold, i.e., use cumulative error threshold over a set of data words defined by a window, so as to achieve more approximate matches. This can be applicable especially in cases of video/image applications where the error rate over a frame is more appropriate than a conservative per word error threshold.

5. CONCLUSIONS

In this research we propose NoC design solutions for power-savings in future CMPs tailored to traditional applications and higher effective throughput gains in multicore systems tailored to bandwidth intensive applications. First, we propose Fly-over, a light-weight distributed mechanism for power-gating routers attached to switched off cores to reduce NoC power consumption in low load CMP environment. After constructing the FLOV router enabling FLOV links by modifying the baseline router microarchitecture, we presented two different handshake protocols for FLOV routers, called rFLOV and gFLOV, and explained the dynamic routing algorithm in details. Performance evaluations using synthetic and real workloads show that FLOV not only achieves better NoC power savings due to power-gating more routers but avoids aggregated traffic rerouting in the network unlike previously proposed NoC power-gating mechanisms. We show that FLOV reduces the total and static energy consumption by 18% and 22% respectively, on average across several benchmarks, compared to state-of-the-art NoC power-gating mechanism, while keeping the performance degradation within 1%.

Secondly, we plan on utilizing a promising next generation memory technology, Spin-Transfer Torque Magnetic RAM(STT-MRAM), to achieve enhanced NoC performance to satisfy the high throughput demands in emerging bandwidth intensive applications, while reducing the power consumption simultaneously. In this work, we propose a novel pipelined input buffer design with STT-MRAM for NoC routers. To overcome the weakness of STT-MRAM, the long latency and high power consumption in write operations, we design a multibank STT-MRAM buffer which is a virtual channel with multiple banks. Through this, we avoid performance degradation while consuming less area and power. Also, we address the issue of random data corruption in STT-MRAM by proposing cost-efficient buffer refresh schemes combined with Error Correcting Codes (ECC). Our simulation results show significant performance improvement with less total router power consumption.

Thirdly, we present a hardware data approximation framework for NoCs, APPROX-NoC, with an online data error control mechanism, which can leverage the approximate computing paradigm in the emerging data intensive big data applications to attain higher performance per watt. We present a value based approximate matching technique to use in a plug and play fashion with any underlying data compression mechanism. We also detail low cost microarchitectural implementations of the VAXX techbique with state-of-the-art dictionary-based and frequent pattern-based NoC data compression mechanisms. Our evaluation results show that the best APPROX-NoC mechanism reduces the average packet latency up to 21.4% over state-of-the-art NoC data compression mechanism. In addition, our evaluation results with synthetic workloads show that the best APPROX-NoC mechanism improves throughput up to 60% compared to state-of-the-art compression mechanisms. We observe that the FP-based mechanisms achieve higher approximation rate and hence performance benefits across the benchmarks, but the DI-based mechanisms outperform the FP mechanisms when there is significant data repetition. On average the application output quality is always above 99% across the benchmarks even though a 10% error threshold is allowed since a large portion of the words are within close proximity.

REFERENCES

[1] G. Moore, "Cramming More Components onto Integrated Circuits," *Electronics*, vol. 38, no. 8, p. 56, 1965.

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

[3] M. B. Taylor, J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffman, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen, M. Frank, S. Amarasinghe, and A. Agarwal, "The Raw Microprocessor: A Computational Fabric for Software Circuits and General-Purpose Programs," *IEEE Micro*, vol. 22, no. 2, pp. 25–35, 2002.

[4] J. Howard, S. Dighe, S. R. Vangal, G. Ruhl, N. Borkar, S. Jain, V. Erraguntla, M. Konow, M. Riepen, M. Gries, G. Droege, T. Lund-Larsen, S. Steibl, S. Borkar, V. K. De, and R. 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*, vol. 46, no. 1, pp. 173–183, 2011.

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

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

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

[8] J. Ahn, S. Yoo, O. Mutlu, and K. Choi, "PIM-enabled instructions: a low-overhead, locality-aware processing-in-memory architecture.," in *Proceedings of the 42th Annual International Symposium on Computer Architecture (ISCA-43)*, pp. 336–348, 2015.

[9] J. Ahn, S. Hong, S. Yoo, O. Mutlu, and K. Choi, "A Scalable Processing-in-memory Accelerator for Parallel Graph Processing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA-43)*, pp. 105–117, 2015.

[10] S. Kumar, N. Vedula, A. Shriraman, and V. Srinivasan, "DASX: Hardware Accelerator for Software Data Structures," in *Proceedings of the 29th ACM on International Conference on Supercomputing (ICS 2015)*, pp. 361–372, 2015.

[11] J. S. Miguel, M. Badr, and N. E. Jerger, "Load Value Approximation," in *Proceedings of the 47th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-47)*, pp. 127–139, 2014.

[12] X. Chen and L.-S. Peh, "Leakage Power Modeling and Optimization in Interconnection Networks," in *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 90–95, ACM, 2003.

[13] A. Banerjee, R. Mullins, and S. Moore, "A Power and Energy Exploration of Network-on-Chip Architectures," in *International Symposium on Networks-on-Chip (NoCS)*, pp. 163–172, IEEE Computer Society, 2007.

[14] L. Chen and T. M. Pinkston, "NoRD: Node-Router Decoupling for Effective Power-Gating of On-Chip Routers," in *International Symposium on Microarchitecture (MICRO)*, pp. 270–281, IEEE Computer Society, 2012.

[15] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, "DSENT – A Tool Connecting Emerging Photonics with Electronics

for Opto-Electronic Networks-on-Chip Modeling," in *International Symposium on Networks on Chip (NoCS)*, pp. 201–210, IEEE, 2012.

[16] R. Parikh, R. Das, and V. Bertacco, "Power-Aware NoCs through Routing and Topology Reconfiguration," in *Design Automation Conference (DAC)*, pp. 1–6, IEEE, 2014.

[17] M. Annavaram, "A Case for Guarded Power Gating for Multi-Core Processors," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 291–300, IEEE, 2011.

[18] J. Lee and N. S. Kim, "Optimizing Throughput of Power- and Thermal-Constrained Multicore Processors Using DVFS and Per-Core Power-Gating," in *Design Automation Conference (DAC)*, pp. 47–50, IEEE, 2009.

[19] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan, and C. Kozyrakis, "Power Management of Datacenter Workloads Using Per-Core Power Gating," *Computer Architecture Letters*, vol. 8, no. 2, pp. 48–51, 2009.

[20] H. Matsutani, M. Koibuchi, D. Ikebuchi, K. Usami, H. Nakamura, and H. Amano, "Ultra Fine-Grained Run-Time Power Gating of On-Chip Routers for CMPs," in *International Symposium on Networks-on-Chip (NOCS)*, pp. 61–68, IEEE, 2010.

[21] A. Samih, R. Wang, A. Krishna, C. Maciocco, C. Tai, and Y. Solihin, "Energy-Efficient Interconnect via Router Parking," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 508–519, IEEE, 2013.

[22] L. Chen, D. Zhu, M. Pedram, and T. M. Pinkston, "Power Punch: Towards Non-Blocking Power-Gating of NoC Routers," in *International Symposium on High Performance Computer Architecture (HPCA)*, pp. 1–12, IEEE, 2015.

[23] A. Vega, A. Buyuktosunoglu, and P. Bose, "SMT-Centric Power-Aware Thread Placement in Chip Multiprocessors," in *Inernational Conference on Parallel Architectures and Compilation Techniques (PACT)*, pp. 167–176, IEEE, 2013.

[24] H. Wang, L.-S. Peh, and S. Malik, "Power-driven Design of Router Microarchitectures in On-chip Networks," in *Proceedings of MICRO*, 2003.

[25] J. Hu and R. Marculescu, "Application-specific Buffer Space Allocation for Networks-on-Chip Router Design," in *Proceedings of ICCAD*, 2004.

[26] C. W. Smullen, V. Mohan, A. Nigam, S. Gurumurthi, and M. R. Stan, "Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches," in *Proceedings of HPCA*, 2011.

[27] Z. Sun, X. Bi, H. H. Li, W.-F. Wong, Z.-L. Ong, X. Zhu, and W. Wu, "Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme," in *Proceedings of MICRO*, 2011.

[28] X. Guo, E. Ipek, and T. Soyata, "Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing," in *Proceedings of ISCA*, 2010.

[29] A. Driskill-Smith, D. Apalkov, V. Nikitin, X. Tang, S. Watts, D. Lottis, K. Moon, A. Khvalkovskiy, R. Kawakami, X. Luo, A. Ong, E. Chen, and M. Krounbi, "Latest Advances and Roadmap for In-Plane and Perpendicular STT-RAM," in *Proceedings of IMW*, 2011.

[30] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate Data Types for Safe and General Low-Power Computation," in *Proceedings of the 32nd ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2011)*, pp. 164–174, 2011.

[31] S. Venkataramani, V. K. Chippa, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Quality Programmable Vector Processors for Approximate Computing," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*, pp. 1–12, 2013.

[32] J. S. Miguel, J. Albericio, A. Moshovos, and N. E. Jerger, "Doppelganger: A Cache for Approximate Computing," in *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*, pp. 50–61, 2015.

[33] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Architecture Support for Disciplined Approximate Programming," *SIGPLAN Not.*, vol. 47, no. 4, pp. 301–312, 2012.

[34] D. S. Khudia, B. Zamirai, M. Samadi, and S. Mahlke, "Rumba: An Online Quality Management System for Approximate Computing," in *Proceedings of the 42nd Annual International Symposium on Computer Architecture (ISCA-42)*, pp. 554–566, 2015.

[35] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural Acceleration for General-Purpose Approximate Programs," in *Proceedings of the 45th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-45)*, pp. 449–460, 2012.

[36] A. Guzhva, S. Dolenko, and I. Persiantsev, "Multifold Acceleration of Neural Network Computations Using GPU," in *Proceedings of the 19th International Conference on Artificial Neural Networks: Part I (ICANN 2009)*, pp. 373–380, 2009.

[37] M. Creel and M. Zubair, "High Performance Implementation of an Econometrics and Financial Application on GPUs," in *Proceedings of International Conference on High Performance Computing, Networking, Storage and Analysis (SCC 2012)*, pp. 1147–1153, 2012.

[38] O. A. Aguilar and J. C. Huegel, “Inverse Kinematics Solution for Robotic Manipulators Using a CUDA-Based Parallel Genetic Algorithm,” in *Proceedings of the 10th Mexican International Conference on Advances in Artificial Intelligence - Volume Part I (MICAI 2011)*, pp. 490–503, 2011.

[39] S. Sidiroglou-Douskos, S. Misailovic, H. Hoffmann, and M. Rinard, “Managing Performance vs. Accuracy Trade-offs with Loop Perforation,” in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering (ESEC/FSE 2011)*, pp. 124–134, 2011.

[40] M. Samadi, J. Lee, D. A. Jamshidi, A. Hormati, and S. Mahlke, “SAGE: Self-tuning Approximation for Graphics Engines,” in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*, pp. 13–24, 2013.

[41] M. Samadi, D. A. Jamshidi, J. Lee, and S. A. Mahlke, “Paraprox: Pattern-Based Approximation for Data Parallel Applications,” in *Proceedings of the 19th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XIX)*, pp. 35–50, 2014.

[42] C. Alvarez, J. Corbal, and M. Valero, “Fuzzy Memoization for Floating-Point Multimedia Applications,” *IEEE Trans. Comput.*, vol. 54, no. 7, pp. 922–927, 2005.

[43] C. Álvarez, J. Corbal, and M. Valero, “Dynamic Tolerance Region Computing for Multimedia,” *IEEE Trans. Computers*, vol. 61, pp. 650–665, 2012.

[44] T. Moreau, M. Wyse, J. Nelson, A. Sampson, H. Esmaeilzadeh, L. Ceze, and M. Oskin, “SNNAP: Approximate Computing on Programmable SoCs via Neural Acceleration,” in *Proceeedings of the 21st IEEE International Symposium on High Performance Computer Architecture (HPCA-21)*, pp. 603–614, 2015.

[45] Z. Du, A. Lingamneni, Y. Chen, K. V. Palem, O. Temam, and C. Wu, "Leveraging the Error Resilience of Neural Networks for Designing Highly Energy Efficient Accelerators," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 34, pp. 1223–1235, 2015.

[46] A. Yazdanbakhsh, J. Park, H. Sharma, P. Lotfi-Kamran, and H. Esmaeilzadeh, "Neural Acceleration for GPU Throughput Processors," in *Proceedings of the 48th International Symposium on Microarchitecture (MICRO-48)*, pp. 482–493, 2015.

[47] A. Sampson, J. Nelson, K. Strauss, and L. Ceze, "Approximate Storage in Solid-State Memories," in *Proceedings of the 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-46)*, pp. 25–36, 2013.

[48] S. Liu, K. Pattabiraman, T. Moscibroda, and B. G. Zorn, "Flikker: Saving DRAM Refresh-power Through Critical Data Partitioning," in *Proceedings of the 16th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS-XVI)*, pp. 213–224, 2011.

[49] Y. Jin, K. H. Yum, and E. J. Kim, "Adaptive Data Compression for High-performance Low-power On-chip Networks," in *Proceedings of the 41st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-41)*, pp. 354–363, 2008.

[50] R. Das, A. K. Mishra, C. Nicopoulos, D. Park, V. Narayanan, R. R. Iyer, M. S. Yousif, and C. R. Das, "Performance and Power Optimization Through Data Compression in Network-on-Chip Architectures," in *Proceedings of the 14th International Conference on High-Performance Computer Architecture (HPCA-14)*, pp. 215–225, 2008.

[51] N. Jiang, D. U. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. E. Shaw, J.-H. Kim, and W. J. Dally, "A Detailed and Flexible Cycle-Accurate Network-on-

Chip Simulator," in *International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pp. 86–96, IEEE, 2013.

[52] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Comput. Archit. News*, vol. 39, pp. 1–7, 2011.

[53] R. Kumar, A. Martínez, and A. González, "Dynamic Selective Devectorization for Efficient Power Gating of SIMD Units in a HW/SW Co-Designed Environment," in *International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pp. 81–88, IEEE, 2013.

[54] E. J. Kim, K. H. Yum, G. M. Link, N. Vijaykrishnan, M. Kandemir, M. J. Irwin, M. Yousif, and C. R. Das, "Energy Optimization Techniques in Cluster Interconnects," in *International Symposium on Low Power Electronics and Design (ISLPED)*, pp. 459–464, ACM, 2003.

[55] V. Soteriou and L.-S. Peh, "Design-Space Exploration of Power-Aware On/Off Interconnection Networks," in *International Conference on Computer Design (ICCD)*, pp. 510–517, IEEE, 2004.

[56] G. Kim, J. Kim, and S. Yoo, "Flexibuffer: Reducing Leakage Power in On-Chip Network Routers," in *Design Automation Conference (DAC)*, pp. 936–941, IEEE, 2011.

[57] H. Matsutani, M. Koibuchi, D. Wang, and H. Amano, "Run-Time Power Gating of On-Chip Routers Using Look-Ahead Routing," in *Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 55–60, IEEE Computer Society Press, 2008.

[58] R. Das, S. Narayanasamy, S. K. Satpathy, and R. G. Dreslinski, “Catnap: Energy Proportional Multiple Network-on-Chip,” in *ACM SIGARCH Computer Architecture News*, vol. 41, pp. 320–331, ACM, 2013.

[59] J. Zhan, Y. Xie, and G. Sun, “NoC-Sprinting: Interconnect for Fine-Grained Sprinting in the Dark Silicon Era,” in *Proceedings of the 51st Annual Design Automation Conference (DAC)*, pp. 1–6, ACM, 2014.

[60] A. Kumar, L.-S. Peh, P. Kundu, and N. K. Jha, “Express Virtual Channels: Towards the Ideal Interconnection Fabric,” in *ACM SIGARCH Computer Architecture News*, vol. 35, pp. 150–161, ACM, 2007.

[61] A. Kodi, A. Louri, and J. Wang, “Design of Energy-Efficient Channel Buffers with Router Bypassing for Network-on-Chips (NoCs),” in *International Symposium on Quality Electronic Design (ISQED)*, pp. 826–832, IEEE, 2009.

[62] U. Y. Ogras and R. Marculescu, “Application-Specific Network-on-Chip Architecture Customization via Long-Range Link Insertion,” in *International Conference on Computer-Aided Design (ICCAD)*, pp. 246–253, IEEE, 2005.

[63] U. Y. Ogras and R. Marculescu, “"It’s a Small World After All": NoC Performance Optimization via Long-Range Link Insertion,” *IEEE Transaction on Very Large Scale Integration Systems*, vol. 14, no. 7, pp. 693–706, 2006.

[64] S. J. Hollis, C. Jackson, P. Bogdan, and R. Marculescu, “Exploiting Emergence in On-Chip Interconnects,” *IEEE Transactions on Computers*, vol. 63, no. 3, pp. 570–582, 2014.

[65] L.-S. Peh and W. J. Dally, “A Delay Model and Speculative Architecture for Pipelined Routers,” in *International Symposium on High-Performance Computer Architecture (HPCA)*, pp. 255–266, IEEE, 2001.

[66] J. Duato, "A New Theory of Deadlock-Free Adaptive Routing in Wormhole Networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 12, pp. 1320–1331, 1993.

[67] W. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2003.

[68] D. DiTomaso, A. Kodi, and A. Louri, "QORE: A Fault Tolerant Network-on-Chip Architecture with Power-Efficient Quad-Function Channel Buffers," in *Proceedings of HPCA*, 2014.

[69] Z. Wang, D. A. Jimenez, C. Xu, G. Sun, and Y. Xie, "Adaptive Placement and Migration Policy for an STT-RAM-Based Hybrid Cache," in *Proceedings of HPCA*, 2014.

[70] X. Wu, J. Li, L. Zhang, E. Speight, R. Rajamony, and Y. Xie, "Hybrid Cache Architecture with Disparate Memory Technologies," in *Proceedings of ISCA*, 2009.

[71] H. Jang, B. S. An, N. Kulkarni, K. H. Yum, and E. J. Kim, "A Hybrid Buffer Design with STT-MRAM for On-Chip Interconnects," in *Proceedings of NOCS*, 2012.

[72] B. Del Bel, J. Kim, C. Kim, and S. Sapatnekar, "Improving STT-MRAM Density Through Multibit Error Correction," in *Proceedings of DATE*, 2014.

[73] H. Naeimi, C. Augustine, A. Raychowdhury, S.-L. Lu, and J. Tschanz, "STTRAM Scaling and Retention Failure," *Intel Technology Journal*, vol. 17, 2013.

[74] Y. C. Chengen Yang, Yunus Emre and C. Chakrabarti, "Improving Reliability of Non-Volatile Memory Technologies Through Circuit Level Techniques and Error Control Coding," *EURASIP Journal on ASP*, vol. 2012, 2012.

[75] S. Ma, N. E. Jerger, and Z. Wang, "Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-chip," in *Proceedings of HPCA*, 2012.

[76] D. Strukov, "The Area and Latency Tradeoffs of Binary Bit-Parallel BCH Decoders for Prospective Nanoelectronic Memories," in *Proceedings of ACSSC*, 2006.

[77] A. Jog, A. K. Mishra, C. Xu, Y. Xie, N. Vijaykrishnan, R. Iyer, and C. R. Das, "Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs," in *Proceedings of DAC*, 2012.

[78] N. D. Rizzo, M. DeHerrera, J. Janesky, B. Engel, J. Slaughter, and S. Tehrani, "Thermally Activated Magnetization Reversal in Submicron Magnetic Tunnel Junctions for Magnetoresistive Random Access Memory," *Applied Physics Letters*, vol. 80, p. 2335, 2002.

[79] Z. Diao, Z. Li, S. Wang, and Y. Ding, "Spin-Transfer Torque Switching in Magnetic Tunnel Junctions and Spin-Transfer Torque Random Access Memory," *Journal of Physics:Condensed Matter*, vol. 19, p. 165209, 2007.

[80] A. Nigam, C. Smullen, V. Mohan, E. Chen, S. Gurumurthi, and M. Stan, "Delivering on the Promise of Universal Memory for Spin-Transfer Torque RAM (STT-RAM)," in *Proceedings of ISLPED*, 2011.

[81] A. Raychowdhury, D. Somasekhar, and T. Karnik, "Design Space and Scalability Exploration of 1T-1STT MTJ Memory Arrays in the Presence of Variability and Disturbances," in *Proceedings of IEDM*, 2009.

[82] D.Bedau, H.Liu, J.-J.Bouzaglou, A.D.Kent, J.Z.Sun, J.A.Katine, E.E.Fullerton, and S.Mangin, "Ultrafast Spin-Transfer Switching in Spin Valve Nanopillars with Perpendicular Anisotropy," *Applied Physics Letters*, vol. 96, p. 022514, 2010.

[83] Y. Cao, T. Sato, M. Orshansky, D. Sylvester, and C. Hu, "New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Simulation," in *Proceedings of IEEE Custom Integrated Circuits Conference*, 2000.

[84] X. Dong, C. Xu, Y. Xie, and N. P. Jouppi, "NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory," in *IEEE Transactions on CAD*, 2012.

[85] ITRS, "International Technology Roadmap for Semiconductors: 2009 Executive Summary." http://www.itrs.net/Links/2009ITRS/Home2009.htm.

[86] K. Swaminathany, R. Mukundrajany, N. Soundararajan, and V. Narayanan, "Towards Resilient Micro-Architectures: Datapath Reliability Enhancement using STT-MRAM," in *Proceedings of ISVLSI*, 2011.

[87] T.Dunn and A.Kamenev, "Optimization of the Current Pulse for Spin-Torque Switches," *Applied Physics Letters*, vol. 98, p. 143109, 2011.

[88] A. Kumar, L.-S. Peh, and N. Jha, "Token Flow Control," in *Proceedings of MICRO*, 2008.

[89] W.J.Dally and B.Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2003.

[90] D. DiTomaso, R. Morris, A. Kodi, A. Sarathy, and A. Louri, "Extending the Energy Efficiency and Performance With Channel Buffers, Crossbars, and Topology Analysis for Network-on-Chips," *IEEE Transactions on VLSI*, vol. 21, pp. 2141–2154, 2013.

[91] D. Harris, *Skew-Tolerant Circuit Design*. Morgan Kaufmann, 2000.

[92] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: Dependency-Driven Trace-Based Network-on-Chip Simulation," in *Proceedings of NoCArc*, 2010.

[93] D. U. Becker, "Efficient Microarchitecture for Network-on-Chip Routers," in *Ph.D. Dissertation*, Stanford University, 2012.

[94] T. Moscibroda and O. Mutlu, "A Case for Bufferless Routing in On-chip Networks," in *Proceedings of ISCA*, 2009.

[95] N. Goswami, B. Cao, and T. Li, "Power-performance Co-optimization of Throughput Core Architecture using Resistive Memory," in *Proceedings of HPCA*, 2013.

[96] M. K. Qureshi, V. Srinivasan, and J. A. Rivers, "Scalable High Performance Main Memory System Using Phase-Change Memory Technology," in *Proceedings of ISCA*, 2009.

[97] M.-T. Chang, P. Rosenfeld, S.-L. Lu, and B. Jacob, "Technology Comparison for Large Last-Level Caches: Low-Leakage SRAM, Low Write-Energy STT-RAM, and Refresh-Optimized eDRAM," in *Proceedings of HPCA*, 2013.

[98] L. Chen and T. M. Pinkston, "NoRD: Node-Router Decoupling for Effective Power-gating of On-Chip Routers," in *Proceedings of MICRO*, 2012.

[99] C. Fallin, C. Craik, and O. Mutlu, "CHIPPER: A Low-complexity Bufferless Deflection Router," in *Proceedings of HPCA*, 2011.

[100] A. K. Mishra, X. Dong, G. Sun, Y. Xie, N. Vijaykrishnan, and C. R. Das, "Architecting On-Chip Interconnects for Stacked 3D STT-RAM Caches in CMPs," in *Proceedings of ISCA*, 2011.

[101] S. E. Raasch, N. L. Binkert, and S. K. Reinhardt, "A Scalable Instruction Queue Design Using Dependence Chains," in *Proceedings of ISCA*, 2002.

[102] G. Kucuk, D. Ponomarev, and K. Ghose, "Low- Complexity Reorder Buffer Architecture," in *Proceedings of ICS*, 2002.

[103] G. Kucuk, D. Ponomarev, O. Ergin, and K. Ghose, “Reducing Reorder Buffer Complexity Through Selective Operand Caching,” in *Proceedings of ISLPED*, 2003.

[104] L. A. Lozano C. and G. R. Gao, “Exploiting Short-Lived Variables in Superscalar Processors,” in *Proceedings of MICRO*, 1995.

[105] T.-F. Chen and J.-L. Baer, “Effective Hardware-Based Data Prefetching for High-Performance Processors,” *IEEE Transactions on Computers*, vol. 44, pp. 609–623, 1995.

[106] A. Sampson, A. Baixo, B. Ransford, T. Moreau, J. Yip, L. Ceze, and M. Oskin, “Accept: A Programmer-Guided Compiler Framework for Practical Approximate Computing,” *University of Washington Technical Report UW-CSE-15-01*, vol. 1, 2015.

[107] R. St. Amant, A. Yazdanbakhsh, J. Park, B. Thwaites, H. Esmaeilzadeh, A. Hassibi, L. Ceze, and D. Burger, “General-purpose Code Acceleration with Limited-precision Analog Computation,” in *Proceeding of the 41st Annual International Symposium on Computer Architecuture (ISCA-41)*, pp. 505–516, 2014.

[108] P. Zhou, B. Zhao, Y. Du, Y. Xu, Y. Zhang, J. Yang, and L. Zhao, “Frequent Value Compression in Packet-based NoC Architectures,” in *Proceedings of the 2009 Asia and South Pacific Design Automation Conference (ASP-DAC 2009)*, pp. 13–18, 2009.

[109] J. Zhan, M. Poremba, Y. Xu, and Y. Xie, “Leveraging Delta Compression for End-to-End Memory Access in NoC Based Multicores,” in *2014 19th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 586–591, Jan 2014.

[110] A. R. Alameldeen and D. A. Wood, “Frequent Pattern Compression: A Significance-Based Compression Scheme for L2 Caches,” *Dept. Comp. Scie., Univ.*

Wisconsin-Madison, Tech. Rep, vol. 1500, 2004.

[111] B. Agrawal and T. Sherwood, "Ternary CAM Power and Delay Model: Extensions and Uses," *IEEE Trans. Very Large Scale Integr. Syst.*, pp. 554–564, 2008.

[112] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building Customized Program Analysis Tools with Dynamic Instrumentation," in *Proceedings of the 26th ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI 2005)*.

[113] C. Bienia, *Benchmarking Modern Multiprocessors*. PhD thesis, Princeton University, January 2011.

[114] D. A. Bader and K. Madduri, "Design and Implementation of the HPCS Graph Analysis Benchmark on Symmetric Multiprocessors," in *Proceedings of the 12th International Conference on High Performance Computing (HiPC 2005)*, pp. 465–476, 2005.

[115] J. Leskovec and A. Krevl, "SNAP Datasets: Stanford Large Network Dataset Collection." `http://snap.stanford.edu/data`, June 2014.

[116] N. Muralimanohar, R. Balasubramonian, and N. Jouppi, "Optimizing NUCA Organizations and Wiring Alternatives for Large Caches with CACTI 6.0," in *Proceedings of the 40th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO-40)*, pp. 3–14, 2007.