HIGH PERFORMANCE ON-CHIP INTERCONNECTS DESIGN FOR FUTURE MANY-CORE ARCHITECTURES

A Dissertation

by

HYUNJUN JANG

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,	Paul Gratz
	Rabi Mahapatra
	Duncan M. Walker
Head of Department,	Dilma Da Silva

December 2015

Major Subject: Computer Engineering

ABSTRACT

Switch-based Network-on-Chip (NoC) is a widely accepted inter-core communication infrastructure for Chip Multiprocessors (CMPs). With the continued advance of CMOS technology, the number of cores on a single chip keeps increasing at a rapid pace. It is highly expected that many-core architectures with more than hundreds of processor cores will be commercialized in the near future. In such a large scale CMP system, NoC overheads are more dominant than computation power in determining overall system performance. Also, for modern computational workloads requiring abundant thread level parallelism (TLP), NoC design for highly-parallel, many-core accelerators such as General Purpose Graphics Processing Units (GPGPUs) is of primary importance in harnessing the potential of massive thread- and data-level parallelism. In these contexts, it is critical that NoC provides both low latency and high bandwidth within limited on-chip power and area budgets.

In this dissertation, we explore various design issues inherent in future many-core architectures, CMPs and GPGPUs, to achieve both high performance and power efficiency. First, we deal with issues in using a promising next generation memory technology, Spin-Transfer Torque Magnetic RAM (STT-MRAM), for NoC input buffers in CMPs. Using a high density and low leakage memory offers more buffer capacities with the same die footprint, thus helping increase network throughput in NoC routers. However, its long latency and high power consumption in write operations still need to be addressed. Thus, we propose a hybrid design of input buffers using both SRAM and STT-MRAM to hide the long write latency efficiently. Considering that simple data migration in the hybrid buffer consumes more dynamic power compared to SRAM, we provide a lazy migration scheme that reduces the

dynamic power consumption of the hybrid buffer.

Second, we propose the first NoC router design that uses only STT-MRAM, providing much larger buffer space with less power consumption, while preserving data integrity. To hide the multicycle writes, we employ a multibank STT-MRAM buffer, a virtual channel with multiple banks where every incoming flit is seamlessly pipelined to each bank alternately. Our STT-MRAM design has aggressively reduced the retention time, resulting in a significant reduction in the latency and power overheads of write operations. To ensure data integrity against inadvertent bit flips from the thermal fluctuation during the given retention time, we propose a cost-efficient dynamic buffer refresh scheme combined with Error Correcting Codes (ECC) to detect and correct data corruption.

Third, we present schemes for bandwidth-efficient on-chip interconnects in GPGPUs. GPGPUs place a heavy demand on the on-chip interconnect between the many cores and a few memory controllers (MCs). Thus, traffic is highly asymmetric, impacting on-chip resource utilization and system performance. Here, we analyze the communication demands of typical GPGPU applications, and propose efficient NoC designs to meet those demands.

ACKNOWLEDGEMENTS

First of all, I would like to thank my adviser, Dr. Eun Jung Kim, for her consistent and careful guidance. Her advice made it possible to complete my PhD study. I am truly grateful for her encouragement and constant motivation throughout this work. Also, I would like to thank my committee members, Dr. Paul Gratz, Dr. Rabi Mahapatra, and Dr. Duncan M. Walker. Many thanks also go to Dr. Ki Hwan Yum for his support throughout my PhD study, especially for his careful feedback on research papers. Second, I would like to thank all of the previous and current members of the High Performance Computing Laboratory, especially Minseon Ahn, Baiksong An, Lei Wang, Rahul Boyapati, Jagadish Chandar Jayabalan, Wen Yuan, Rohan Kansal, Nikhil Kulkarni, Sagar Narayanan, Kyunghoon Kim, and Jiayi Huang for supporting and helping my research. Last but not least, I am especially grateful to my wife, Ara Cho, and parents for their incredible support, patience and trust for me. Without their dedication and belief, I couldn't have completed this study.

TABLE OF CONTENTS

Page

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

The ever-increasing power consumption and diminishing returns in the performance of unicore processor have led to the advent of many-core architectures. This many-core trend may lead to hundreds of cores integrated on a single chip. As the number of core counts increases, a scalable, flexible, and high bandwidth on-chip communication fabric becomes critically important. In this context, switch-based Network-on-Chip (NoC) are fast replacing buses and crossbars as the pervasive communication fabric for many-core chips. In such an NoC, an on-chip router is attached to every node and adjacent nodes are connected via local on-chip wiring. This NoC-based many-core architecture has been prevalent in Chip Multiprocessors (CMPs) domains. Also, especially for modern computational workloads such as graphics and data-intensive scientific applications necessitating abundant thread level parallelism (TLP), NoC designs for highly-parallel, many-core accelerators such as General Purpose Graphics Processing Units (GPGPUs) have emerged as an important field of research. For high performance many-core architectures, it is critical that NoC supplies high bandwidth at ultra-low latencies within a tight power and area budgets.

This dissertation addresses a number of design challenges regarding high performance NoCs for many-core architectures to achieve high performance and power efficiency. First, we present a new design of NoC router based on a next-generation memory technology, Spin-Transfer Torque Magnetic RAM (STT-MRAM) in CMPs. STT-MRAM has high density and near-zero leakage power, thus adopting STT-MRAM in NoC has significant merits in improving throughput with less power consumption. However, its long latency and high power consumption in write operations compared to those of SRAM need to be addressed. Thus, we propose a hybrid de-

sign of input buffers combining both SRAM and STT-MRAM to hide the long write latency. Also, considering that simple data migration in the hybrid buffer consumes more dynamic power compared to SRAM, we design a lazy migration scheme allowing the flit migration only when the network load exceeds a certain threshold, which helps to reduce the dynamic power consumption significantly. Simulation results show that the proposed scheme enhances the throughput by 21% on average.

Second, 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. Our STT-MRAM design also has aggressively reduced the retention time, resulting in a significant reduction in the latency and power overheads of write operations. To ensure data integrity under the limited retention time and random bit flips of STT-MRAM, we employ cost-efficient dynamic buffer refresh schemes combined with Error Correcting Codes (ECC) that are extensively used in memories and storage devices to tolerate both transient and static errors. Simulation results show that the proposed STT-MRAM NoC router improves throughput by 20.7% and achieves 17% savings in the total router power with minimum hardware overheads compared with an SRAM based NoC router.

Third, we present bandwidth-efficient on-chip interconnects designs for GPGPUs. Unlike CMP systems, where traffic tends to be uniform across the cores communicating with distributed on-chip caches, the communication in GPGPUs is highly asymmetric, mainly between many compute cores and a few memory controllers (MCs).

Thus the MCs often become hot spots, leading to skewed usage of the NoC resources and network bottlenecks. We quantitatively analyze the impact of network traffic patterns in GPGPUs with different MC placements and dimension order routings. Then, based on this analysis, we suggest VC monopolizing and partitioning schemes which dramatically improve NoC resource utilization without causing protocol deadlocks. We also investigate the impact of XY, YX, and XY-YX routing algorithms under diverse MC placements. Simulation results show the proposed NoC schemes improve overall GPGPU performance by up to 64.7% over baseline. Compared to the top performing prior work, our VC monopolizing and partitioning schemes achieve a performance gain of 25% with a simple MC placement policy.

2. HYBRID BUFFER DESIGN WITH STT-MRAM FOR ON-CHIP INTERCONNECTS *

2.1 Introduction

With the continued advance of CMOS technology, the number of cores on a single chip keeps increasing at a rapid pace. And it is highly expected that many-core architectures with more than hundreds of processor cores will be commercialized in the near future. In a large-scale chip multiprocessor (CMP) system, network overheads are more dominant than computation power in determining overall system performance. While shared buses provide networking performance enough for a small number of CMP nodes, they cannot be good solutions for many-core systems due to the limitation on scalability. Accordingly, switch-based networks-on-chip (NoCs) are being adopted as an emerging design trend in many-core CMP environments. Since all components in a chip including processors, caches and interconnects must compete for limited area and power budgets, resources available for NoCs are tightly constrained compared to off-chip interconnects. Moreover, network performance becomes more significant with the increasing scale of CMP systems. Therefore, a new and innovative NoC design that can guarantee better performance with limited resources is necessary for many-core systems.

The advance of memory technology has ushered in new non-volatile memory (NVM) designs that overcome the drawbacks of existing memories such as SRAM or DRAM. Among them, Spin-Torque Transfer Magnetic RAM (STT-MRAM) is being regarded as a promising technology for a number of advantages over the conventional

RAMs. STT-MRAM is a next-generation memory that uses magnetic materials as the main information carrier. It achieves lower leakage power and higher density compared to the existing SRAM. Also, STT-MRAM shows higher endurance compared to other NVM techniques such as Phase Change Memory (PCM) or Flash, which makes STT-MRAM more attractive for on-chip memories that must tolerate much more frequent write accesses compared to off-chip memories. However, one of the biggest weaknesses of STT-MRAM is long write latency compared to SRAM. Since the fast access time of memories on a chip must be guaranteed and cannot be negotiable, the slow write operations of STT-MRAM limit its popularity, even though it shows competitive read performance. Another serious drawback of STT-MRAM is high power consumption in write operations. This issue of high power consumption in STT-MRAM must be resolved in NoCs due to the limited power budgets.

Despite these weaknesses, using STT-MRAM in the NoC design has significant merits since an on-chip router can incorporate larger input buffers compared to SRAM with the same area budget because of the higher density of STT-MRAM. Larger input buffers contribute to improving the throughput of NoC, which results in the enhancement of overall system performance. However, the aforementioned challenges must be addressed first to exploit the benefit of STT-MRAM in NoC. Since the input buffer of an on-chip router must handle arriving flits on time, it is impossible in reality to use STT-MRAM without additional technique to hide the long write latency. Moreover, addressing the high write power issue of STT-MRAM is mandated in NoC environments.

In this study, we explore the design issues of adopting STT-MRAM in on-chip interconnects. First, by relaxing the non-volatility of STT-MRAM, the latency as well as the power consumption in write operations can be reduced at the sacrifice of

the retention time [34, 59]. Based on the observation of intra-router latency of flits, we find out that the retention time needed for input buffers in NoC can be significantly shortened. We exploit the write latency reducing technique [34] in the input buffers of on-chip routers, and decrease the latency to less than 2ns that corresponds to 6 cycles in 3GHz clock frequency. Then we propose a hybrid design of input buffers combining both SRAM and STT-MRAM. By allowing each arriving flit to be stored in the SRAM buffer first and then migrated to STT-MRAM, the write latency of STT-MRAM is effectively hidden, thus increasing network throughput.

Simply migrating each flit from SRAM to STT-MRAM buffer causes significant power consumption due to the high write power of STT-MRAM, compared to existing SRAM-based input buffers. So we design a lazy migration scheme that allows the flit migration only when the network load exceeds a certain threshold, which helps to reduce the power consumption significantly. Simulation results show that the hybrid input buffers improve the network throughput by 21% in synthetic workloads and 14% in SPLASH-2 parallel benchmarks on average compared to pure SRAM-based buffers with the same area overheads. Also, the lazy migration scheme contributes to power reduction by 61% on average compared to the simple migration scheme that always migrates flits from SRAM to STT-MRAM.

The remainder of this study is organized as follows. We discuss related work in Section 2.2, followed by the performance and power model of STT-MRAM in Section 3.2.2. In Section 3.4, we explain the hybrid buffer design using STT-MRAM in detail. Section 3.5 presents simulation results and analysis, and finally Section 3.9 summarizes our work and makes conclusions.

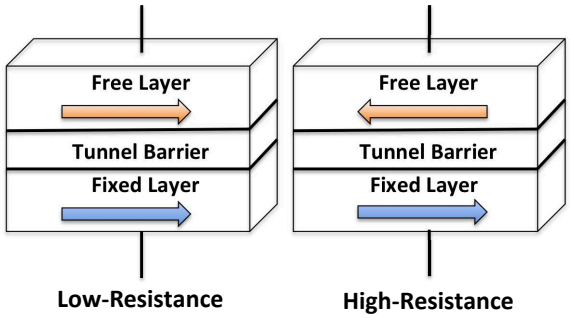


Figure 2.1: The Two States of an MTJ Module

2.2 Related Work

Since there has been no prior work using STT-MRAM in NoC design, we only summarize the relevant studies of STT-MRAM technologies as well as the application of NVM to diverse system domains such as processors and memories.

2.2.1 STT-MRAM

STT-MRAM is a next generation memory technology that takes advantage of magnetoresistance for storing data. It uses a Magnetic Tunnel Junction (MTJ), the fundamental building block, as a binary storage. An MTJ comprises a three-layered stack: two ferromagnetic layers and an MgO tunnel barrier in the middle. Among them, the fixed layer located at the bottom has a static magnetic spin, the spin of the electrons in the free layer at the top is influenced by applying adequate current through the fixed layer to polarize the current, and the current is passed to the free layer. Depending on the current, the spin polarity of the free layer changes either parallel or anti-parallel to that of the fixed layer. The parallel indicates a zero state, and the anti-parallel a one state. Figure 2.1 depicts the two parallel and anti-parallel states of an MTJ module. A single MTJ module is coupled with a transistor to form a basic memory cell of STT-MRAM called a 1T-1MTJ cell.

2.2.2 Utilizing NVMs in Processors and Memories

Several schemes have been proposed to provide architectural support for applying NVMs to system components. Jog *et al.* [34] proposed to achieve better write performance and energy consumption of STT-MRAM-based L2 cache through adjusting data retention time of STT-MRAM. Similarly, Smullen *et al.* [59] reduced the write latencies as well as dynamic energy of STT-MRAM by lowering the retention time for designing on-chip caches. In [47], they integrated STT-MRAM into on-chip caches in a 3D CMP environment and proposed a mechanism of delaying cache accesses to busy STT-MRAM banks to hide long write latency. Prior to that, Sun *et al.* [61] stacked MRAM-based L2 caches on top of CMPs and reduced overheads through read-preemptive write buffer and hybrid cache design using both SRAM and MRAM. Guo *et al.* [26] resolved the design issues of microprocessors using STT-MRAM in detail for more power-efficient CMP systems.

PCM also has been constantly explored to replace existing SRAM or DRAM-based memory systems. Due to its lower endurance compared to SRAM or STT-MRAM, PCM is mainly adopted for off-chip memories rather than on-chip caches. Several designs of PCM-based main memory were discussed in [73, 53, 42]. In [52], adaptive write cancellation and write pausing policies were proposed to reduce energy and improve performance. Zhou *et al.* [72] suggested a new memory scheduling scheme that allows Quality-of-Service (QoS) tuning through request preemption and row buffer utilization.

2.3 Performance and Power Model of STT-MRAM

As an area model of STT-MRAM, we use ITRS 2009 projections [32] as well as the model used in [26], where a 1T-1MTJ cell size is $30F^2$ in the 32nm technology. When we assume that an SRAM cell size is approximately $146F^2$ with the same technology, one SRAM cell can be substituted by at least four STT-MRAM cells under the same area budget. Also, about 3.2ns of write latency can be achieved with $30F^2$ STT-MRAM cell size [26]. It corresponds to 10 cycles in 3GHz clock frequency, which is quite long for on-chip routers compared to SRAM that completes both read and write accesses in a single cycle. Reducing retention time from 10 years to 10ms guarantees the same write latency with one third of original write current needed [34]. Using lower current is beneficial in terms of area overheads because it facilitates to implement STT-MRAM cells with smaller transistors, which reduces actual cell area.

In this study, we slightly increase write current to reduce this write latency of STT-MRAM further. The write latency reduces from 3.2ns to 1.8ns through increasing the write current from 50μA to 75μA under 125 °C of a temperature. Note that even this increased current is far less than the original current needed for 10 years of retention time, while maintaining the same STT-MRAM cell size, $30F^2$. Also, the increased current does not hurt write energy consumption since the MTJ switching time decreases accordingly [26]. As a result, the write latency decreases from 10 to 6 cycles in 3GHz clock frequency. The increased write current may hurt the performance in terms of read latency. However, we verify that the reduction of write latency from 3 to 1.8ns affects the read latency to only a small extent [59]. Therefore, we can assume that the increased read latency can still be covered by a single cycle, considering the original read delay of 122ps [26], which is far shorter than 333ps, a

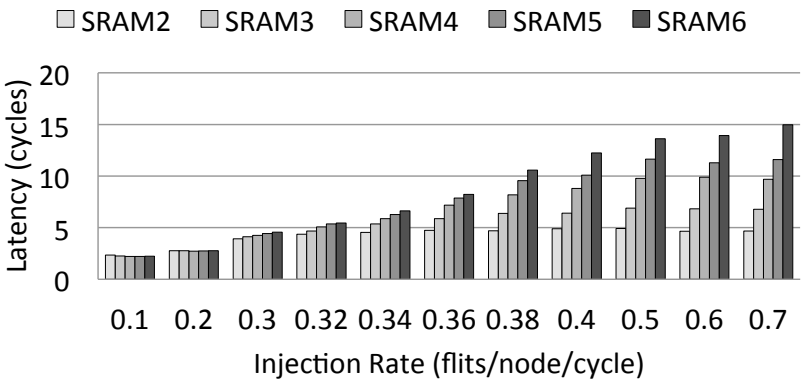


Figure 2.2: Maximum Intra-Router Latency of an On-Chip Router (*SRAM#: SRAM Buffer Size per VC)*

cycle time in 3GHz clock frequency.

The relaxed retention time of 10ms may hurt the reliability of data stored in an STT-MRAM buffer, if the retention time is shorter than the intra-router delay of a flit, defined by the time difference between arrival time at the buffer and departure time in a router. Figure 2.2 depicts maximum intra-router latency for different injection rates ranging from 0.1 to 0.7 with various SRAM buffer sizes per VC, under uniform random synthetic workloads. We observe that the latency does not go up beyond 16 cycles, and it is almost negligible compared to 10ms, which corresponds to more than 30 million cycles in 3GHz clock frequency [1]. Hence, it is confirmed that even the reduced retention time is completely enough to hold a flit in STT-MRAM buffers safely. For the read and write energy model of STT-MRAM, we conservatively adopt the same parameters from [26], 0.01pJ and 0.31pJ per bit for read and write, respectively. Note that these are based on 3.2ns of write latency, so actual write energy becomes smaller after decreasing the latency to 1.8ns.

[1]Note that in deadlock situations, packets can stay in the network forever. In this study, we adopt deadlock-free routing algorithms, thus avoiding such situations.

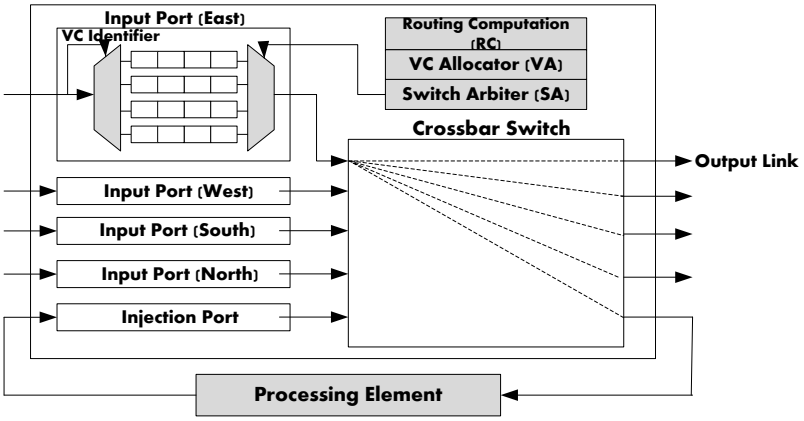


Figure 2.3: Generic Router Architecture

2.4 An On-Chip Router Architecture with Hybrid Buffer Design

In this section, we describe a generic router architecture and a buffer structure in NoC and present our hybrid buffer design that maximizes the mutually complementary features of the two different memory technologies, SRAM and STT-MRAM, while minimizing the drawbacks of STT-MRAM, the long latency and high power consumption in write operations.

2.4.1 Generic Baseline Router Architecture

The generic NoC router architecture is depicted in Figure 2.3. It is based on the state-of-the-art speculative router architecture [51]. Each arriving flit goes through 2 pipeline stages in the router: routing computation (RC), VC allocation (VA) and switch arbitration (SA) at the first cycle, and switch traversal (ST) at the second cycle. A lookahead routing scheme [23] is adopted, which generates routing information of the downstream router for an incoming flit prior to the buffer write, thus removing the RC stage from the critical path. Each router has multiple VCs per input port and uses flit-based wormhole switching [14]. Credit-based VC flow control [13] is

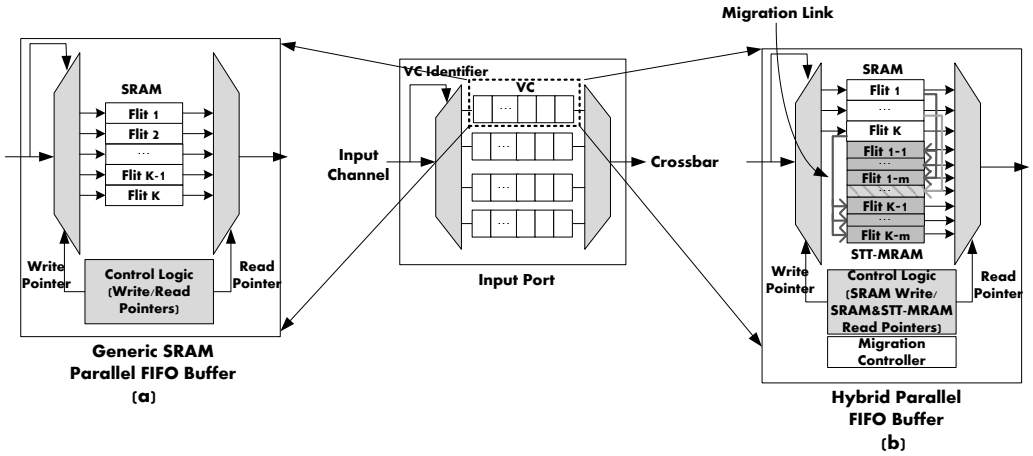


Figure 2.4: A Generic SRAM Input Buffer (a) and a Hybrid Input Buffer (b)

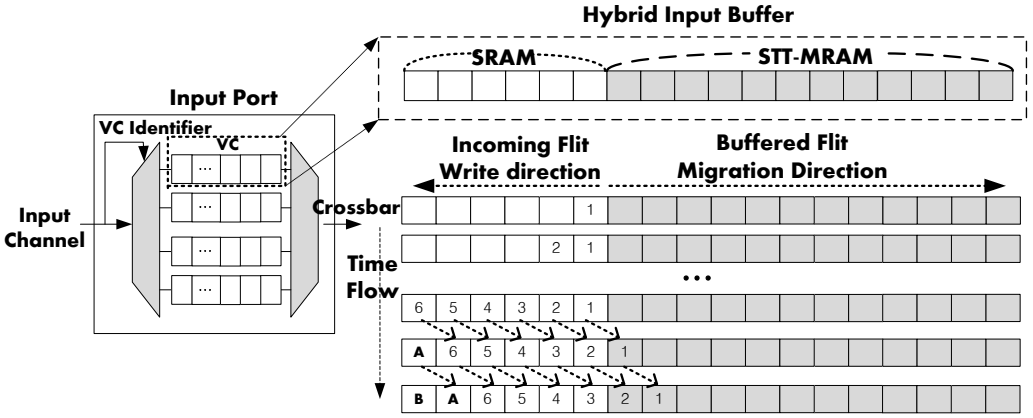


Figure 2.5: Simple Flit Migration Scheme in Hybrid Buffer Design

adopted to provide the back-pressure from downstream to upstream routers, thus controlling flit transmission rate to prevent packet loss due to buffer overflow.

Due to the limited area and power resources and ultra-low latency requirements, on-chip routers rely on very simple buffer structure. VC-based NoC routers consist of a number of FIFO buffers per input port where each FIFO corresponds to a VC as illustrated in Figure 2.4(a). Each input port has v VCs, each of which has a k-flit FIFO buffer. Current on-chip routers have small buffers to minimize area overheads, thus v and k are much smaller than in macro networks. The necessity for ultra-low latency leads to a parallel FIFO buffer design as shown in Figure 2.4. Contrary to a serial FIFO implementation, the parallel structure eliminates unnecessary intermediate processes for a flit to traverse all buffer entries until it leaves the buffer [70]. This fine-grained control requires more complex logic, which manages read and write pointers to keep the FIFO order. The read and write pointers in the parallel FIFO registers control an input demultiplexer and an output multiplexer. The write pointer points to the tail of the queue, and the read pointer points to the head of the queue. For a read operation, the flit pointed by the head is selected and transmitted to a crossbar input port. Similarly, write operation leads the incoming flit to be written to the location pointed by the tail pointer. The pointers are promptly updated after each read or write operation. After a read operation, once the head is overlapped with the tail, the buffer becomes empty. After a write operation, likewise, if the tail moves to the same position pointed by the head, the buffer is full.

2.4.2 An On-Chip Router Architecture with Hybrid Buffer Design

In this section, we show an on-chip router architecture with hybrid buffer design that combines SRAM and STT-MRAM. The hybrid design aims to maximize advantages inherent in different memory technologies in a synergistic fashion for per-

formance improvement while consuming power economically. The key idea is inspired by the nature of STT-MRAM that provides 4 times more buffer space than SRAM under the same area constraint due to its higher density characteristics [26, 74]. The increased buffer size contributes to making on-chip routers have spacious rooms for buffering, thus boosting the overall network throughput with no additional area overheads compared to a pure SRAM-based input buffer.

Figure 2.4(b) depicts the proposed hybrid input buffer of a VC. Compared to the pure SRAM buffer shown in Figure 2.4(a), the STT-MRAM is attached to each VC in parallel with the SRAM buffer. Each SRAM buffer entry is connected to m dedicated STT-MRAM buffer entries through separate migration links. The hybrid parallel FIFO buffer maintains read/write pointers. An incoming flit is first written to the SRAM buffer, thus the write pointer points to SRAM buffer entries only. But an outgoing flit may leave from either SRAM or STT-MRAM and the read pointer covers the entire buffer, both SRAM and STT-MRAM buffer entries.

A migration controller triggers the flit migration and determines if a certain flit is ready to be migrated to STT-MRAM. VC flow control is performed based on the availability of SRAM in downstream routers, meaning that the availability of STT-MRAM is not considered, because a write operation to STT-MRAM cannot finish in a single cycle.

2.4.2.1 Simple Flit Migration Scheme

The key design goal of the hybrid input buffer is to guarantee seamless read and write operations in every cycle to achieve higher throughput with an increased buffer size. To serve this purpose, we devise a flit migration scheme, which seamlessly migrates buffered flits from SRAM to STT-MRAM to secure more SRAM buffer space for incoming flits, while hiding the long write latency of STT-MRAM.

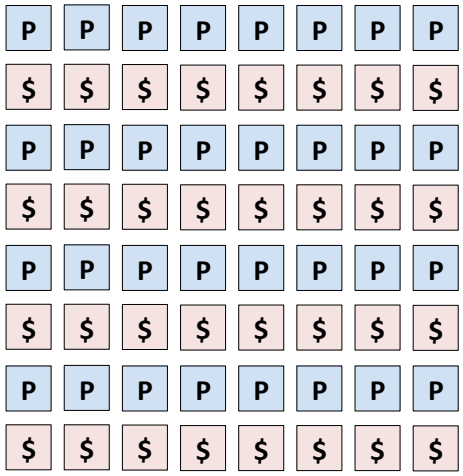


Figure 2.6: CMP Layout

Figure 2.5 depicts an example of the migration scheme, where each VC consists of 6 SRAM and 12 STT-MRAM buffer entries. The STT-MRAM buffer write latency is assumed to be 6 cycles. When an incoming flit arrives, it is written to the SRAM buffer first, and the migration from SRAM to STT-MRAM begins immediately. Supposing that a new flit arrives every cycle, the SRAM buffer becomes full eventually in the 6th cycle. At the same time, the first flit is migrated to STT-MRAM successfully and one SRAM buffer entry becomes available. Then a subsequent incoming flit occupies the released SRAM buffer entry with no additional timing delay. Note that Figure 2.5 illustrates the concept in a logical way, and no physical shift occurs except the migration from SRAM to STT-MRAM. The placement of flits in STT-MRAM is logical and is not the physical placement described in Figure 2.4(b).

2.4.2.2 Power-Efficient Lazy Migration

In the simple migration scheme explained in the previous section, the migration begins immediately as soon as an incoming flit arrives at the SRAM buffer. The simple migration wastes lots of power in a low network load because most of the flits

initially written to SRAM leave the buffer in the middle of migration to STT-MRAM.

Based on this observation, we propose a **lazy migration scheme**, which selectively triggers the migration of a flit based on the estimated network load per VC in the on-chip router. The network load is indirectly estimated by tracking the number of flits in the SRAM buffer. If the ratio of the number of flits in the SRAM buffer to the total SRAM buffer size exceeds a certain predefined threshold level, the flit migration is performed for every subsequent incoming flit as long as the the ratio exceeds the threshold. In this way, we can save total write power associated with the migration operation.

To implement the lazy migration scheme, the migration controller is augmented to keep track of the flits in the SRAM buffer and triggers the migration adaptively. The write power is reduced by up to 79% in a low network load compared to the simple migration, which will be discussed in detail in Secton 3.5.

Table 2.1: CMP System Configuration

System Parameters	Details
Clock frequency	3GHz
# of processors	32
L1 I and D caches	direct-mapped 32KB (L1I) 4-way 32KB (L1D), 1 cycle
L2 cache	16-way 16MB, 20 cycles 32 banks, 512 KB/bank
Cache block size	64B
Coherence protocol	Directory-based MSI
Memory latency	300 cycles
Flit size	16B
Packet size	1 flit (Benchmark-control) 5 flits (Benchmark-data) 4 flits (Synthetic)

Table 2.2: SRAM and STT-MRAM Parameters

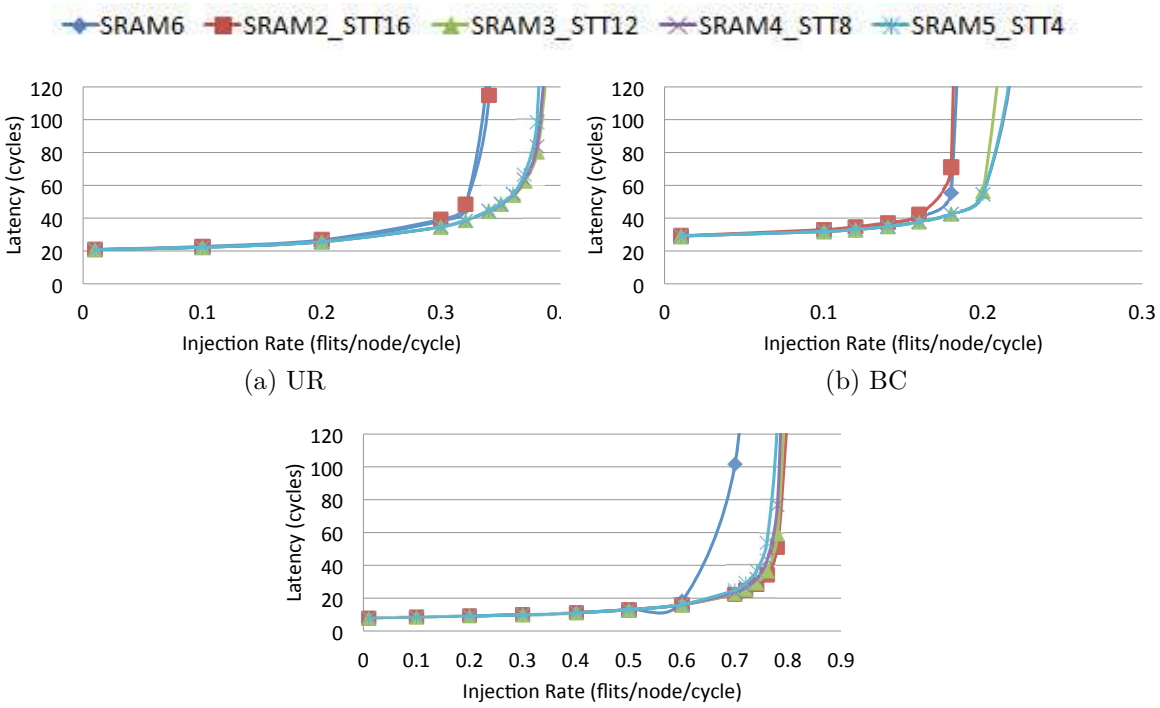
Parameter	SRAM	STT-MRAM
Read Energy (pJ/flit)	5.25	3.826
Write Energy (pJ/flit)	5.25	40.0
Leakage Power (mW)	0.028	0.005

2.5 Performance Evaluation

In this section, we evaluate the proposed hybrid on-chip router to examine how much it improves the overall network performance while reducing the power consumption in NoC, using several benchmarks and synthetic workloads.

2.5.1 System Configuration

A cycle-accurate NoC simulator is used to conduct the detailed evaluation of the proposed scheme. It implements the pipelined router architecture with VCs, a VC arbiter, a switch arbiter and a crossbar. Under the 32nm process technology, all simulations are performed in an 8x8 network having 32 out-of-order processors and 32 L2 cache banks on a single chip as shown in Figure 2.6. The network is



(a) UR

(b) BC

(c) NN

Figure 2.7: Performance Comparison with Synthetic Workloads

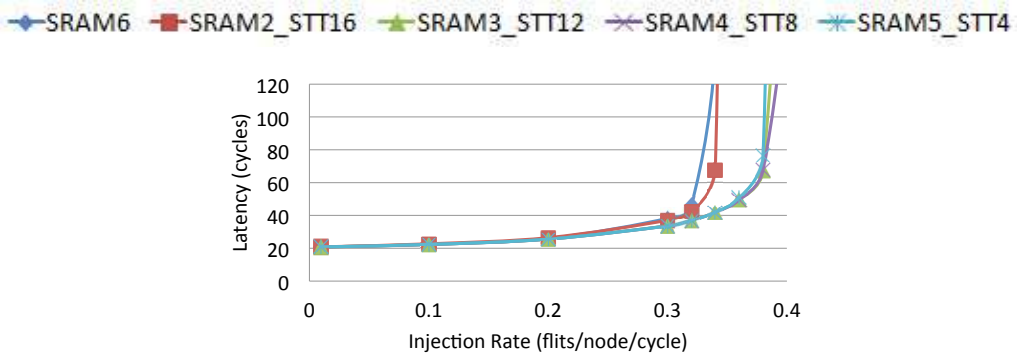
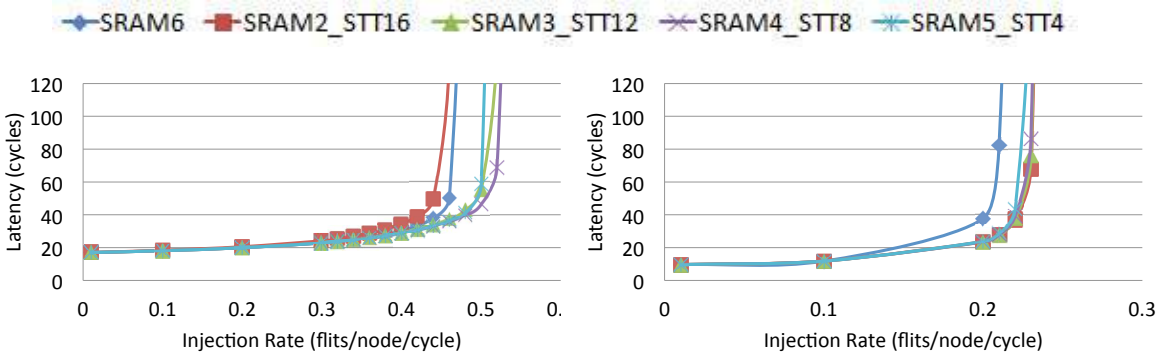
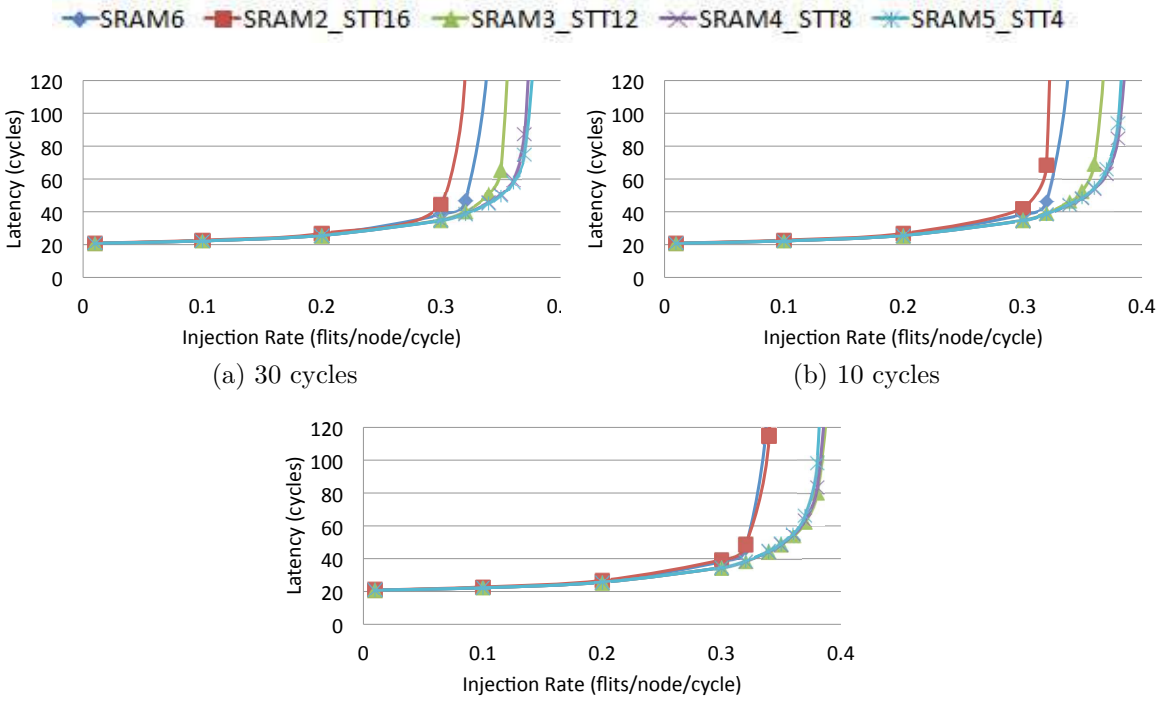


Figure 2.8: Performance Comparison with O1TURN Routing Algorithm



(a) 2D-Torus (b) Flattened Butterfly

Figure 2.9: Performance Comparison with Different Topologies



(a) 30 cycles (b) 10 cycles

(c) 6 cycles

Figure 2.10: Performance Comparsion with Various STT-MRAM Write Latencies

equipped with 2-stage speculative routers with lookahead routing [23]. 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, and k may vary with different buffer configurations. A dimension order routing algorithm, XY, and O1TURN [58] are used with wormhole switching flow control.

A variety of synthetic workloads are used to measure the effectiveness of the hybrid on-chip router: uniform random **(UR)**, bit complement **(BC)** and nearest neighbor **(NN)**. To evaluate the proposed schemes under realistic environments, we also use SPLASH-2 [57] parallel benchmark traces. The traces are obtained using Simics [46], a full system simulation platform. Table 3.1 specifies the detailed CMP configuration we use to run benchmarks.

We use Orion 2.0 [35] to estimate router power consumption. In addition, parameters shown in Table 3.2 are cited from [32, 26], for both SRAM and STT-MRAM. The unit of parameter for the leakage power is mW per 1-flit buffer. Throughout this study, the size of SRAM and STT-MRAM buffers are denoted by *SRAM#* and *STT#*, respectively. As stated in Section 2.4.2, STT-MRAM provides 4 times more buffer space compared to SRAM under the same area budget, thus *SRAM1* is equal to *STT4*. Unless otherwise stated, the write latency of STT-MRAM is 6 cycles based on the analysis in Section 3.2.2.

2.5.2 Performance Analysis with Synthetic Workloads and Benchmarks

Figure 2.7 shows performance improvement for various hybrid input buffer configurations compared to the pure SRAM buffer, under UR, BC and NN traffic patterns. All results are measured under the same area budget, *SRAM6* per VC, for input buffers. In all cases, the hybrid design shows throughput improvement by 18% for UR, 28% for BC, and 17% for NN on average. These results indicate that although

the STT-MRAM write latency is longer than that of SRAM, the performance loss is offset by the increased buffer size due to the high density of STT-MRAM, thus resulting in performance improvement.

We also evaluate the hybrid design using O1TURN [58] routing algorithm as well as various topologies: 2D-torus and flattened butterfly [38]. Figure 2.8 shows the performance with O1TURN in the 8x8 2D-mesh topology, where the overall throughput increases by 15% on average, while Figure 2.9 shows that the throughput is increased in 2D-torus and flattened butterfly by 13% and 15%, respectively.

To examine the impact of different write latencies of STT-MRAM on network performance, we conduct experiments under 2D-mesh and XY routing algorithm. Figure 2.10 shows the performance in terms of packet latency with 3 different write latencies of STT-MRAM: 30, 10, and 6 cycles. It clearly indicates that the overall network performance is affected by the duration of STT-MRAM write operation. Among the different hybrid configurations, *SRAM2_STT16* shows the worst performance. This is because the SRAM buffer space is too small to retain the incoming flits for sufficient period of time for migration, 6 cycles, which makes the simple flit migration scheme less efficient. Thus, the long write latency of STT-MRAM is not effectively hidden, resulting in the early saturation of the network. As shown in Figure 2.2, every flit stays in the buffer for at least 3 cycles. So the SRAM buffer size should be greater than or equal to 3 to run the migration scheme seamlessly.

If the write latency is long, 30 cycles, the performance is mostly determined by the SRAM size. This is because the long write latency lowers the possibility for flits to be migrated to the STT-MRAM buffer before network saturation. Therefore, *SRAM5_STT4* shows the best throughput improvement. On the contrary, if the write latency is sufficiently short, 6 cycles, the performance is greatly impacted by the total buffer size including both SRAM and STT-MRAM except the *SRAM2_STT16* case.

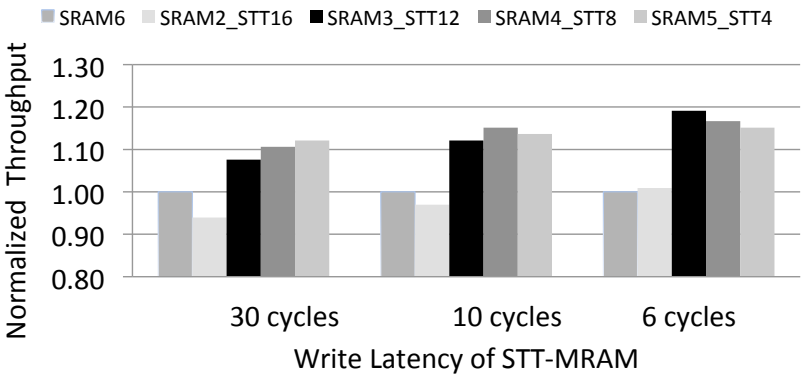


Figure 2.11: Throughput with Different STT-MRAM Write Latencies

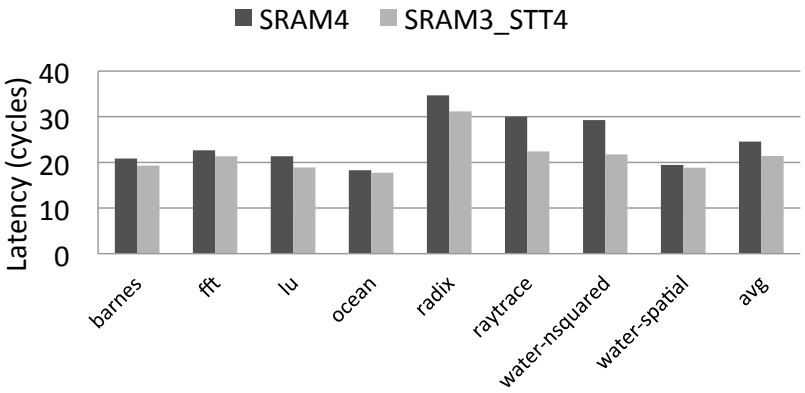


Figure 2.12: SPLASH-2 Benchmark Results

Thus, *SRAM3_STT12* shows the highest throughput compared to other configurations.

To make a clear quantitative comparison of relative performance of the 3 different write latencies, we show network throughput normalized to the *SRAM6* in Figure 2.11, based on the results in Figure 2.10. Figure 2.11 confirms the aforementioned analysis. In case of a relatively long write latency, 30 cycles, the hybrid input buffer having the largest SRAM buffer outperforms the others by up to 11% compared to the pure *SRAM6* buffer. Likewise, in case of a low write latency, 6

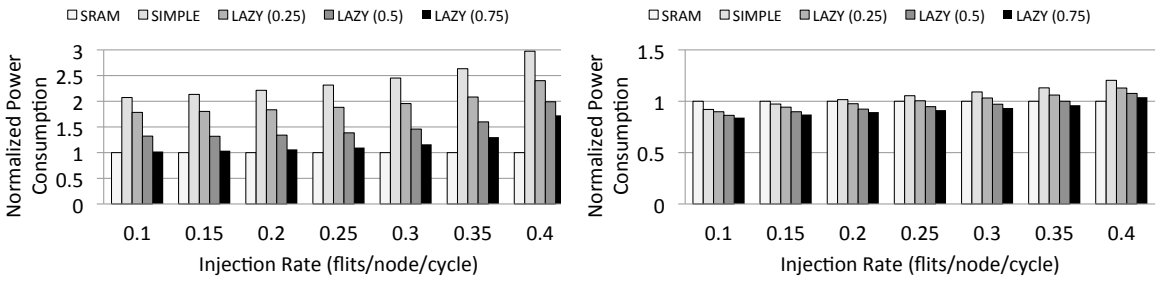
cycles, except the *SRAM2_STT16* case, the one having the largest total buffer size, *SRAM3_STT12* beats the other configurations by up to 18% in terms of network throughput.

Figure 2.12 shows the average network latency with SPLASH-2 benchmark traces. We assume *SRAM4* per VC as an area budget, the same as a cache block size. In general, the hybrid input buffer outperforms the pure SRAM-based one, by approximately 14% on average. Specifically, *water-nsquared* shows the best improvement by 34.5% while *ocean* shows the least improvement by 3.2%. The amount of improvement varies depending on the traffic patterns. We observe that in the benchmarks showing higher improvement, hot spots exist in their communication, whereas in the benchmarks with slight performance improvement, communication is evenly spread across the whole network.

Finally, we make a sensitivity analysis of the number of buffer entries in NoC routers. Under two different area budgets, *SRAM4* and *SRAM6*, we compare the throughput of the pure SRAM-based buffer and the hybrid buffer that shows the best performance. As the budget decreases from *SRAM6* to *SRAM4*, the amount of improvement coming from the hybrid buffer increases by approximately 5.5%. This trend indicates that the hybrid buffer is more beneficial as the area budget in CMP environments becomes tighter.

2.5.3 Power Analysis

Since power is one of the main issues in the NoC router design, we evaluate power consumption of the hybrid input buffer and compare the effect of the two migration schemes explained in Section 3.4. Figure 2.13a compares the dynamic buffer power consumption of 4 different migration schemes in *SRAM3_STT12*: simple and lazy with 3 different thresholds (0.25/0.5/0.75). All results are normalized to that of the



(a) Dynamic Power Consumption of Input Buffers (b) Total Power Consumption of Routers

Figure 2.13: Comparison of Power Efficiency

pure SRAM-based buffer, *SRAM6*. The lazy migration scheme with the threshold 0.75 consumes significantly less amount of power, by 53% on average, compared to the simple migration scheme. In a low network load (0.1), the power consumption of the lazy migration scheme with the threshold 0.75 is almost equivalent to that of the baseline SRAM. In a high network load (0.4), however, the flit migration occurs more frequently in the hybrid buffer due to the highly congested network. Accordingly, the migration lowers the possibility of reducing the dynamic power, thus increasing the power consumption of the lazy migration by up to 1.7x more than the baseline SRAM.

Figure 2.13b compares the total router power consumption of the 4 migration schemes that includes both leakage and dynamic power consumption of all routers across the network. In a low network load (0.1), the total power consumption of routers with the hybrid buffer is less than that of routers with the pure SRAM buffer by 16%. This is due to much less leakage power consumption of STT-MRAM compared to SRAM as shown in Table 3.2. As the network gets more congested, however, the hybrid buffer consumes more power compared to the baseline SRAM buffer. In a high network load (0.4), for instance, the lazy migration scheme with the threshold 0.75 consumes more power by up to 4% compared to the baseline SRAM.

2.6 Conclusions

In the deep sub-micron era, NoC has surfaced as a promising solution to reduce wiring delays and provide significant scalability in future many-core architecture. The NoC performance is heavily affected by the routers' buffer size and their efficient utilization. In this study, we have proposed a hybrid input buffer design using STT-MRAM with SRAM to achieve better network throughput with marginal power overheads in on-chip interconnection networks. The high density of STT-MRAM facilitates to accommodate larger buffer compared to the conventional SRAM under the same area budgets. Through the flit migration schemes, the long write latency of STT-MRAM is effectively hidden while minimizing the power overheads.

Simulation results indicate performance improvement of around 21% and 14% on average under the synthetic workloads and benchmarks, respectively, compared to the conventional on-chip router with the SRAM input buffer.

For future work, we intend to devise an STT-MRAM-aware routing algorithm and provide an architectural support to reduce the overall power consumption and latency further.

3. DESIGN AND ANALYSIS OF STT-MRAM ROUTER: TOWARDS POWER-EFFICIENT AND RELIABLE ON-CHIP INTERCONNECTS

3.1 Introduction

Switch-based Network-on-Chip (NoC) has become a popular architecture orchestrating chip-wide communication in Chip Multiprocessors (CMPs). NoC should be carefully designed due to its inherent constraints of the restricted power and area budgets in a chip. NoC consumes 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 [18]. Buffers also consume significant dynamic power [65], and this consumption increases rapidly as data flow rates increase. Consequently, 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) [59, 62] 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. STT-MRAM is CMOS-compatible, and provides virtually infinite write endurance [26] 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 [21]. 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 [66]. A region-based hybrid cache [69] with small fast SRAM and large slow MRAM mitigates performance degradation and energy overheads. For NoC routers, an SRAM/STT-MRAM hybrid buffer [33] 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 [16]. Even under a high thermal stability, however, we cannot totally avoid the occurrence of bit flips because of the stochastic nature of STT-MRAM [49]. 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 [16, 12]. 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 study, 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 multi-bank 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 [62], 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 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 [45]. 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

[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.

[2] See Section 3.5 for detailed system configuration.

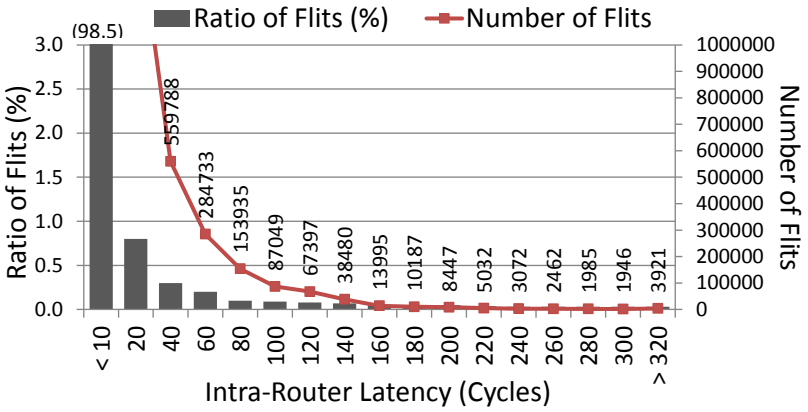


Figure 3.1: Per-Application Intra-Router Latency Distribution (*canneal* in PARSEC Benchmarks)

kept valid by triggering refreshes in a timely manner. Note that 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 [60]. ECCs detect and correct data corruption, thus mitigating the impacts of random bit flips on NoCs.

The main contributions of this study are as follows:

- 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 (17%) and performance improvement (20.7%) compared to a conventional SRAM router.
- 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.

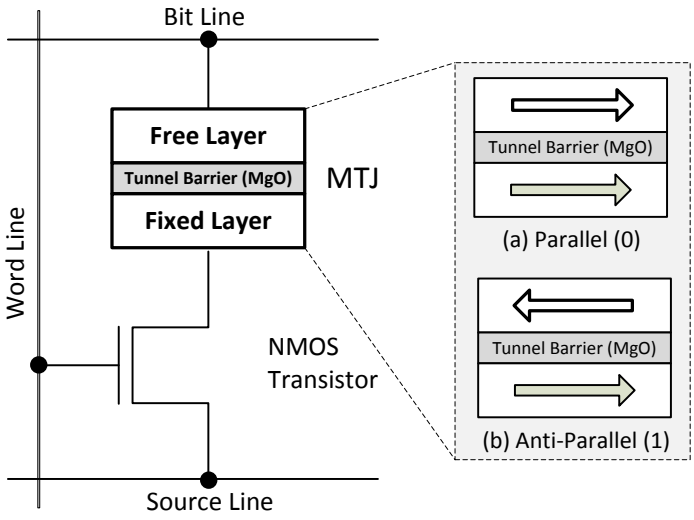


Figure 3.2: STT-MRAM Cell Structure

3.2 Background

In this section, we review key features of STT-MRAM and analyze design trade-offs 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.

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 STT-MRAM Design Considerations

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 [34, 59]. The *data retention time*, T_{ret}, of an MTJ is defined as follows [56].

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

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 [17]. 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 [50], leading to reduced retention-time STT-MRAM [59].

3.2.2.2 Switching Current and Switching Time

In a *precessional switching mode* [55] 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

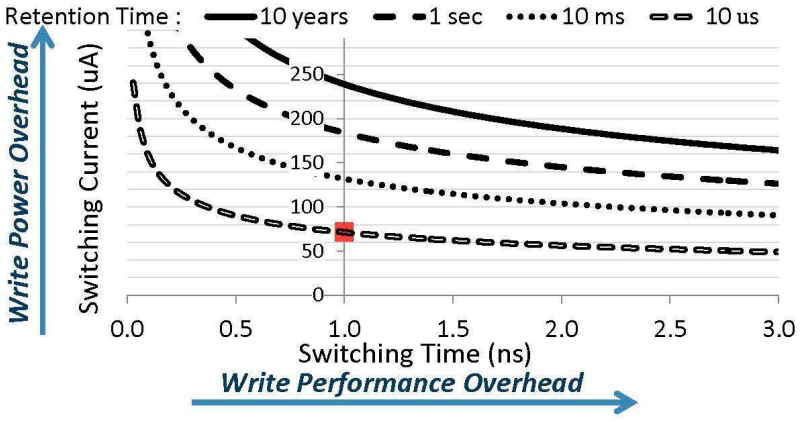


Figure 3.3: The Relationship between Switching Current and Switching Time for Different MTJ Retention Times

between the magnetization vector of the free layer and the easy axis [62]. 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 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 [34, 59, 62], 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 [15]) based on MTJ device equations [34] and simulation with the PTM model [8] 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 [20] and simulation results

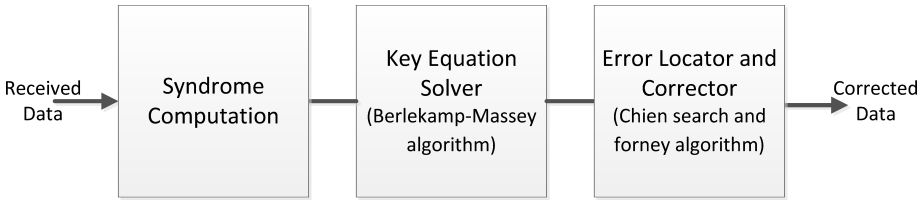


Figure 3.4: BCH ECC Decoder Block Diagram

are shown in Table 3.2.

3.2.2.3 Cell Area

As an area model of STT-MRAM, we refer to ITRS projections [32] as well as the model used in [26], 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 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 [21, 63]. 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.

3.2.2.5 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 [49]. The retention failure rate ($P_{retFail}$) shown in Equation 3.4 [16] 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 [16] 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 [60]. 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.

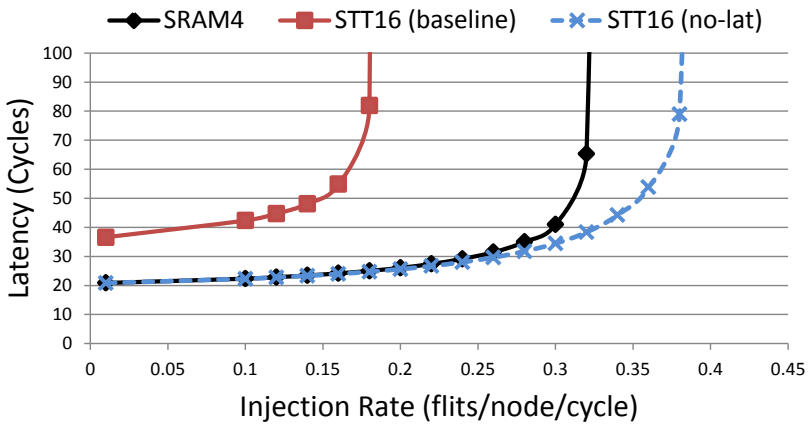


Figure 3.5: Performance Comparison between SRAM and STT-MRAM Routers under the Same Area Budget

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 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.

3.3.2 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 (as described in Section 3.2), leading to performance degradation due to flits corrupted, while increasing the retention time will negatively affect write performance and energy. Considering these trade-offs, 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 [34, 59, 62], 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 [15, 63], but as shown in Figure 3.3, it requires rather strong

currents[3], and is far from the optimal efficiency [64]. 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.

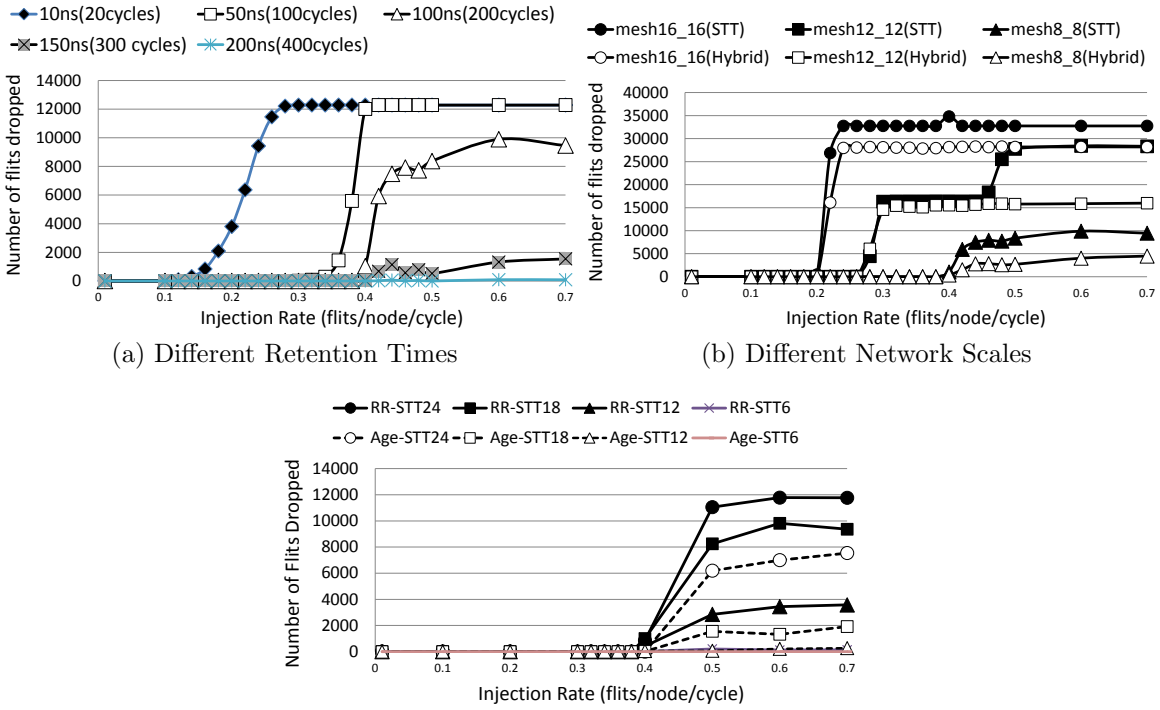
Our proposed scheme exploits these observations to accelerate STT-MRAM write speeds with less power consumption. In Section 3.4.2, we propose router architectures that effectively hide the multicycle write latencies of STT-MRAM.

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.

3.3.3 Avoiding Flit Losses

A key challenge incurred by the retention time reduction is to overcome the potential flit losses occurring when flits overstay a given retention time in STT-MRAM buffers. We observe three main NoC factors (Applications, Retention Times, and Network Scales) affecting the validity of flits in an STT-MRAM router. As shown in Figure 3.6(a), under the retention time of 100 ns, overstayed flits get dropped. Each benchmark application has different traffic patterns over its lifetime, thus showing different number of flits dropped. In particular, *canneal*, *fluidanimate*, and *dedup* show higher flit dropping rates than others due to their bursty traffic characteristics. Figure 3.6(b) shows that, under uniform random (UR) traffic, as the retention time of STT-MRAM gets decreased, maintaining the validity of flits in

[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.



(a) Different Retention Times

(b) Different Network Scales

(c) Different SA and Buffer Sizes (#flits/VC)

Figure 3.6: Various Factors Affecting the Number of Flits Dropped

the STT-MRAM buffer gets difficult, especially when networks become congested. Increased network scale also affects the number of flits dropped due to increased network contention as shown in Figure 3.6(c). At the same injection rate, under UR, 3.3 times more flits get dropped in a (16x16) mesh, on average, compared with those of an (8x8) mesh. From these observations, hence, it is imperative to take appropriate actions applicable to NoCs to prevent such flit losses.

There are several studies addressing these data losses in STT-MRAM through various refresh techniques [34, 59, 62], but their application domains are only limited to caches, which are totally different from buffers in an NoC router. Caches have relatively large capacities and require longer data retention times compared with small-sized FIFO input buffers. Multiple retransmission schemes [22, 28] minimizing

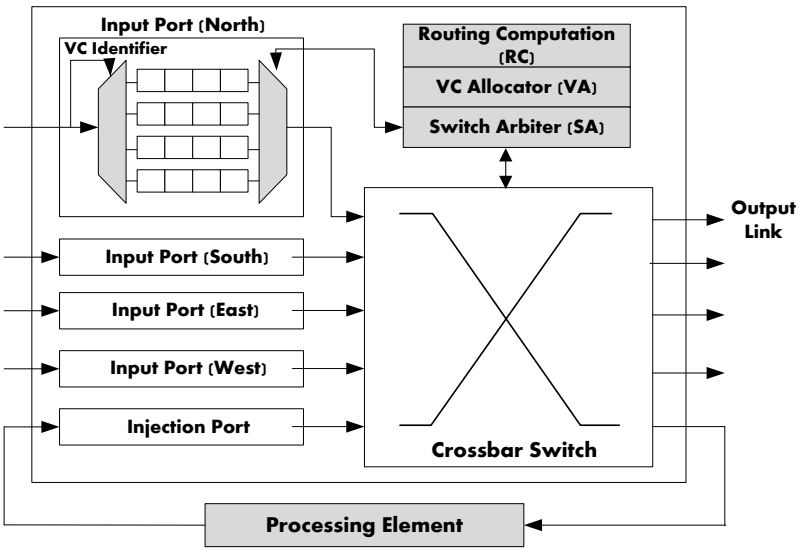


Figure 3.7: Baseline Router Architecture

buffering requirements have been proposed for bufferless routing. They cannot be directly applicable, since unlike bufferless routers, our work requires much larger outside buffering (for injection and reassembly) to hold all the packets in-fly in the network. This is because the number of packets in-fly are significantly greater in our scheme due to the larger buffer space available in the network by virtue of STT-MRAM. In Section 3.4.3, we propose cost-efficient dynamic buffer refresh schemes eliminating flit dropping with less refresh overheads.

3.3.4 Performance Impact Analysis

Here we quantitatively analyze the impact of our scheme in views of saturation throughput and latency in NoCs. For simplicity, we assume a 2D mesh network with a dimension-ordered routing algorithm.

3.3.4.1 Impact on Saturation Throughput

Our STT-MRAM scheme is effective in improving network saturation throughput, which is the data rate in bits per second which the network can accept prior to saturation. Ideally, the network throughput is maximized under perfect flow control and perfect load balancing from a routing algorithm. In reality, however, actual network throughput is affected by NoC factors such as the efficiency of a switch arbitration (matching between requests and available resources), the depth of buffers per VC, and the number of VCs per PC including the number of banks per VC proposed in our scheme.

To achieve a high throughput, we can inject/eject packets at a high rate while holding more packets under such a high packet injection/ejection rate. In our scheme, we provide extra buffering spaces holding more packets in routers, thus even at higher injection rates, network can accept additional packets, maintaining a high network throughput.

We characterize the saturation network throughput ($throughput_{sat}$), as being proportional to the buffer depth per each VC (B_c), the number of VCs per each input port (N_{vc}), the number of banks per VC (n_{bnks}), the packet length (pkt_{ln}), the efficiency of switch arbitration (SW_{eff}) and the factor of imperfect load balancing (P_{htspt}) that quantify the probability of network hotspots.

$$throughput_{sat} \propto \frac{B_c \cdot SW_{eff} \cdot N_{vc}^{\alpha}}{n_{bnks} \cdot pkt_{ln} \cdot P_{htspt}} \tag{3.5}$$

where network throughput is directly proportional to B_c due to additional buffering allowing for higher injection rates. Increasing N_{vc} improves the throughput not only by increasing the buffering capacity but by alleviating head-of-line (HoL) blocking, which is quantified by the factor α, where α is equal to 1, in an ideal NoC where

no HoL blocking occurs. Under such an ideal NoC, throughput is directly proportional to N_{vc}. Network throughput is also improved by boosting the efficiency of SW_{eff} and balancing loads across network, reducing the probablity of occurrence of hot spots, P_{htspt}. The number of banks per VC (n_{bnks}) is adjusted accordingly considering the given write latency of STT MRAM; Higher write latency needs more banks, reducing the available buffering capacity due to the increased logic area overheads, thus negatively impacting network throughput. As the length of the packet (pkt_{ln}) increases, flits are more likely to spread across multiple routers under congested network, impeding subsequent packets from proceeding to their destinations, thus degrading the network throughput. Providing deeper buffer spaces (B_c), however, we can offset such negative impact on throughput. The throughput impacts of aforemetioned factors (B_c, N_{vc}, n_{bnks}, and pkt_{ln}) are empirically shown in Section 3.5.2 and Section 3.5.3.

3.3.4.2 Impact on Latency

We quantify the impact of our proposed scheme on write latency, based on NoC model [3]. Average packet latency is defined as the average time taken for a packet to reach its destination. The equation below quantifies the packet latency ($T_{s->d}$) from source (s) to destination (d).

$$T_{s->d} = t_{node->s} + \sum_{vc} t_{s->d,vc} + t_{serial} \tag{3.6}$$

where $t_{node->s}$ denotes the time a packet waits prior to its header flit injection into its router attached to a processor, $t_{s->d,vc}$ is the time a packet waits inside VCs across intermediate routers (propagation delay) prior to being transmitted to downstream routers, and t_{serial} signifies serialization delay of a packet, which is affected by the length of a packet. Since our scheme related to the latency of a buffer (VC) write,

we quantify $t_{s->d,vc}$ (router latency at a VC) like below.

$$t_{s->d,vc} = wait_{vc+1} + \sum_{\tilde{vc}} t_{link,\tilde{vc}} + t_{transfer} \tag{3.7}$$

$$t_{transfer} = \begin{cases} 1 & : SRAMbuffer \\ n_{stt} & : STT-MRAMbuffer \end{cases} \tag{3.8}$$

where $wait_{vc+1}$ indicates the time taken for a packet to be allocated to a specific VC (vc+1) in a downstream router, $t_{link,\tilde{vc}}$ denotes the impact of multiple VCs (represented by summation over $\tilde{vc}$) competing for a same physical link, , and $t_{transfer}$ is the time required for a flit to be transferred to a downstream router (the write latency of the input buffers). As shown in equation 3.8, under SRAM, it takes 1 cycle (transfer time), but for n_{stt}-cycle STT-MRAM, the $t_{transfer}$ of the packet latency increases accordingly as n_{stt} increases. A chosen STT-MRAM technology affects n_{stt}-cycle write latency, possibly increasing overall network latency, which can be handled by schemes proposed in Section 3.4.

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 3.7, which is similar to that used by Kumar *et al.* [41] employing several features for latency reduction, including speculation [68] 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 3.11(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

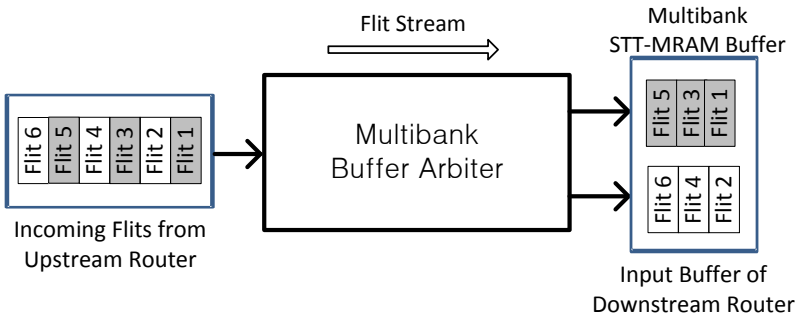
For conventional SRAM buffers, incoming flits are written to their designated buffers with no delay due to the short SRAM write latency. On the contrary, when we replace SRAM with STT-MRAM, only a single flit can be written to a buffer every n cycles, which causes subsequent incoming flits to be delayed. 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.

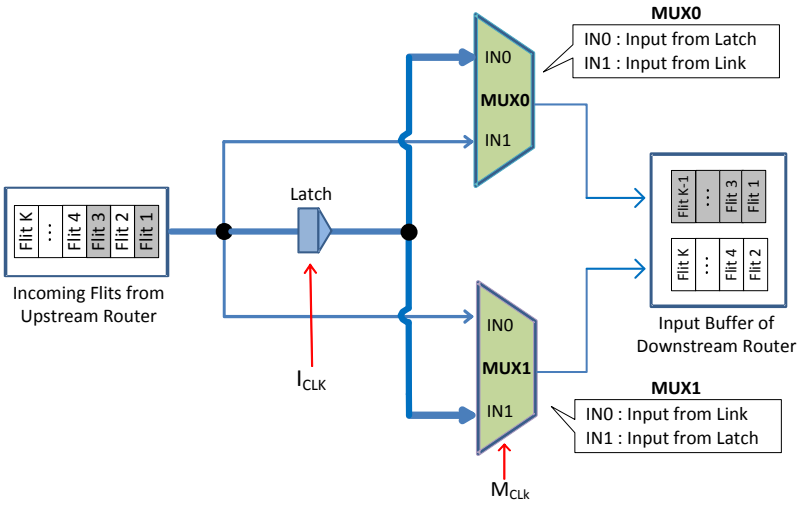
3.4.2.1 Multibank STT-MRAM Buffer

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 [18, 19] 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 study is controlled by a control block (as in the Channel buffer [19]) 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 3.8(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 3.8(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. I_{clk} and M_{clk} are control signals originating from the control block in



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



(b) Multibank Buffer Arbiter Internal Structure

Figure 3.8: Multibank STT-MRAM Buffer

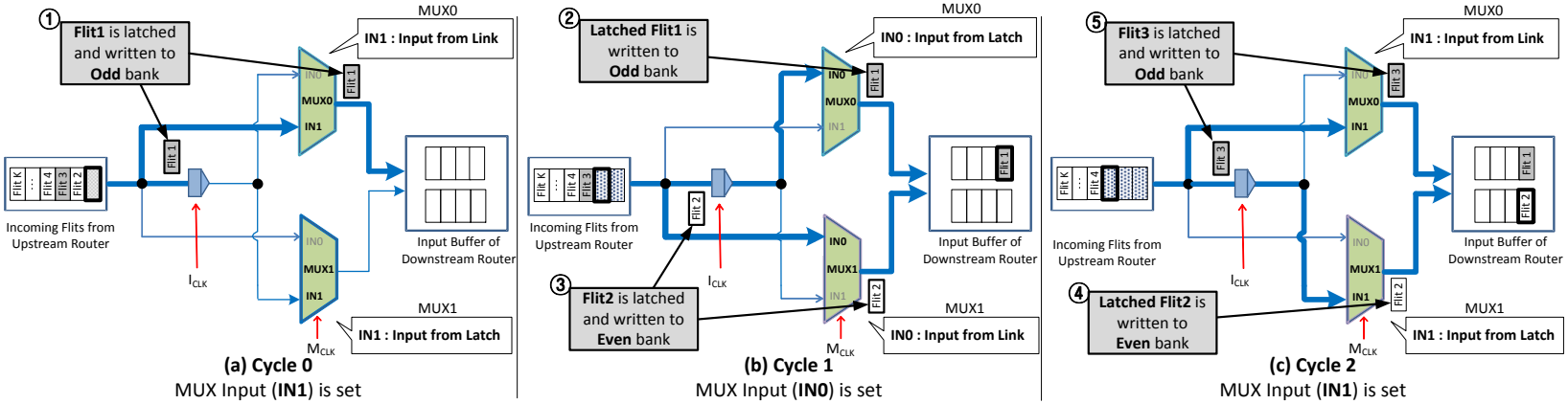


Figure 3.9: Dual-Bank STT-MRAM Buffer Example (Sequence of Operations: ① ~ ⑤)

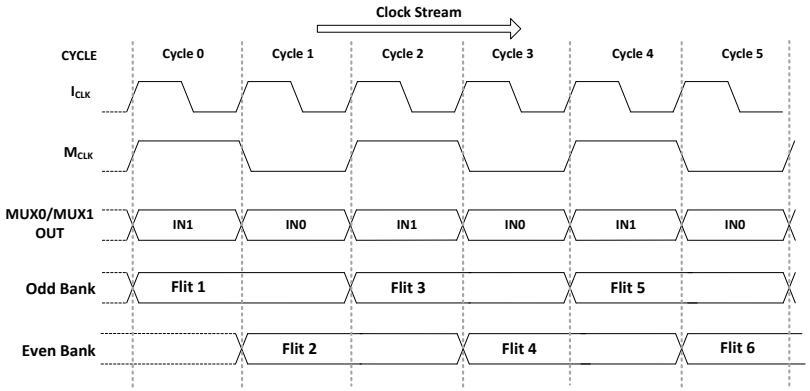


Figure 3.10: Timing Diagram Corresponding to Figure 3.9

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 [33], 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 performance. To avoid this, we incorporate the buffer bypassing logic [65] 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

[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 [27].

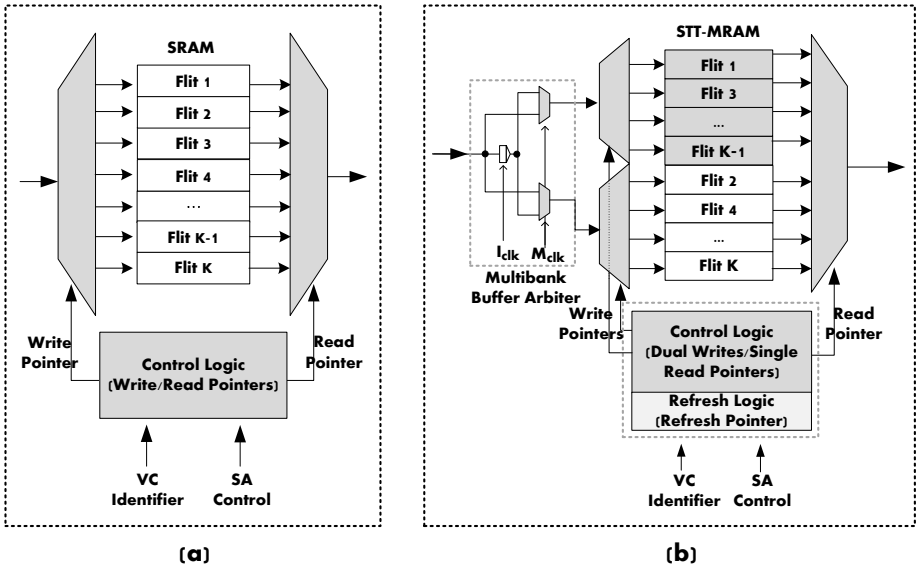


Figure 3.11: A Baseline SRAM Input Buffer (a) and a Dual-Bank STT-MRAM Input Buffer (b)

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 3.12. 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 3.9 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.

• **Timing Diagram:** Figure 3.10 shows a timing diagram corresponding to the Dual-Bank STT-MRAM buffer example in Figure 3.9. I_{clk} indicates the interconnect clock synchronized with the simple latch, and M_{clk} indicates the MUX clock synchronized with each MUX. The two MUXes are controlled by a common select signal that changes once every cycle, and have a clock of half the frequency of I_{clk}. The input signal of MUX0 and MUX1 is alternating on a cycle-by-cycle basis as shown in Cycle 0 to Cycle 5. During Cycle 0, half of $Flit1$ is first written to the Odd bank, and then during the next cycle, $Flit1$ completes its write process. The consecutive arriving flits go through the same write process to each bank.

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 3.11(a)), the proposed multi-bank 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 3.11(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 3.13(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 3.13(b)), the incoming flit is written to the location pointed by the tail pointer in one of the banks. For a read operation (Figure 3.13(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 oper-

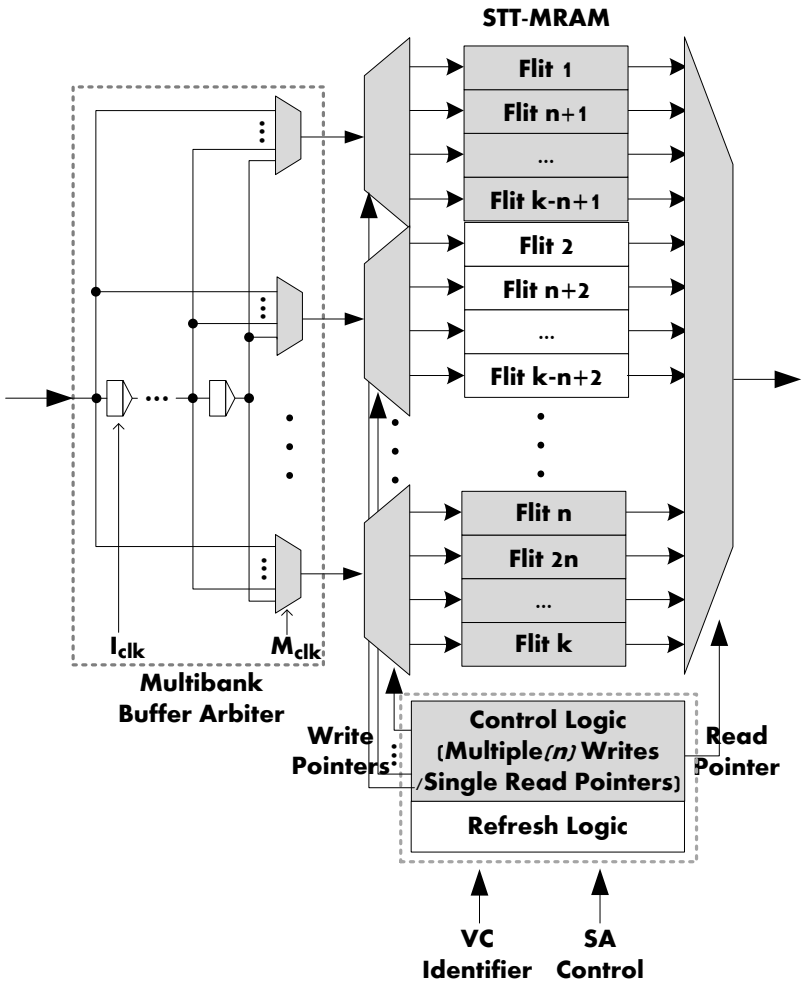


Figure 3.12: A General Multibank STT-MRAM Buffer (k: Total Number of Flits Buffered, To Hide n-cycle Write Latencies, n-1 Latches and n Banks Are Needed.)

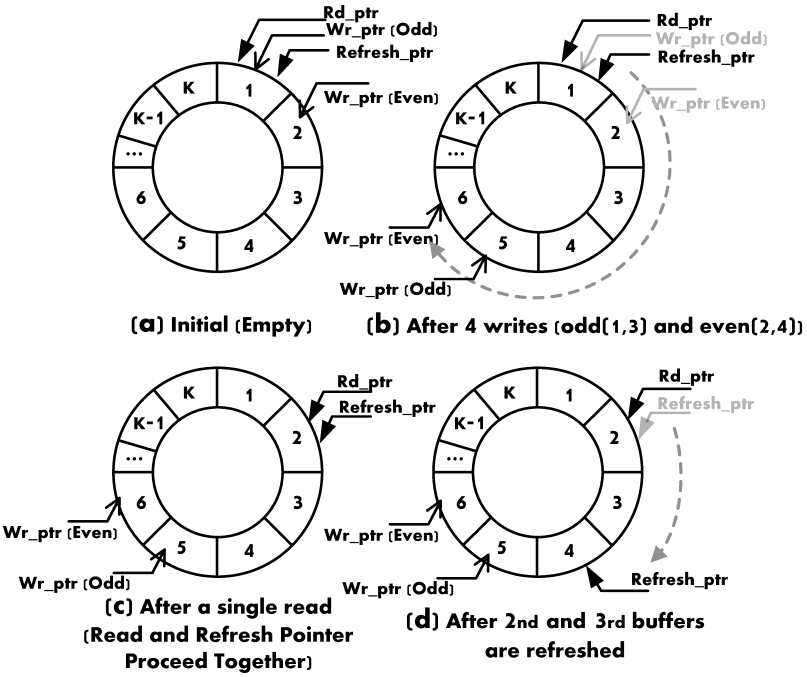


Figure 3.13: Circular Queue for Dual-Bank STT-MRAM Buffer (*Assuming all errors are correctable / Sequence: (a) ~(d))*

ation. The refresh pointer, *Refresh_ptr*, shown in Figure 3.13(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; For instance, such an issue could arise in Figure 3.13(d) where read operations follow right after refreshes.

Single-flit control packets are dominant in PARSEC benchmarks (78.7% on av-

erage) [45], thus taking significant fraction of errors, while data packets (consisting of a single head/tail and multiple body flits) take less fraction of errors compared to that of control packets. Note that the relative ratio of correctable packets to uncorrectable packets can be controlled by adopting a proper refresh policy through which we can minimize overall uncorrectable packets, which is described in Section 3.4.3.1.

Once parts of control or data packets get corrupted beyond ECC capability (*uncorrectable*), nodes need to nullify corresponding packets to make room for incoming flits while triggering retransmission of 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 multiple flits, such recovery processes need to be handled carefully because data packets can span multiple nodes as network gets congested. In such a case, we need to notify corresponding nodes of the corruption, and thus the corrupted packets are to be nullified inside each router. To handle aforementioned cases, we add two status bits to VCs inside a router for monitoring each flit: a *valid-bit* indicating the validity of a flit and a *in-order-bit* indicating temporal write order of flits, and thus we can properly guide read and write pointers; to be specific, initially, when a flit is written into a buffer, *valid-bit* and *in-order-bit* are set to one (1), and the refresh pointer set the *valid-bit* to zero (0) once a flit becomes *uncorrectable*. If the *valid-bit* is zero (0), read pointer skips reading the flit and write pointer overwrite the *invalid* flit with an incoming flit. For nullifying flits residing in a neighbor router, a *dummy tail* flit is sent and release VC reservation made by the corrupted packet, and thus another packet can use the VC. The virtual channel may also include pointers to the flits of the packet that are "buffered on the current node" and "the number of flit buffers available on the next node".

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 impor-

[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.

tant 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 [60]. Basically, SECDED is sufficient to mask any single bit error, thus fitting in the 8-bit parity field[6] for a 128-bit flit [68], 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 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 3.14, 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-

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

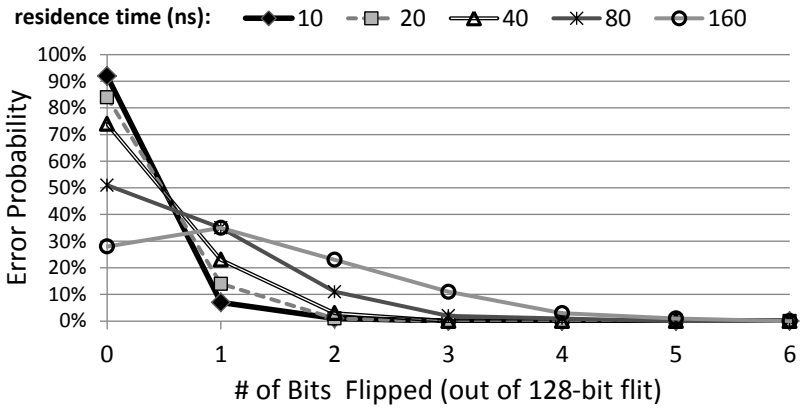


Figure 3.14: Probability of the Number of Bits Flipped (*Note that the sum of error probabilities under a specific residence time is 100 %*)

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 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 3.13, which is controlled by a VC refresh logic shown in Figure 3.11(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 3.13(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 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 3.15 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 Buffer:** Refresh buffer can be made using either an external refresh buffer (**ERB**) of SRAM, or an internal refresh buffer (**IRB**) of STT-MRAM. The ERB requires two writes: an initial write to the ERB and another subsequent write to the original FIFO buffer. To decrease such redundant write overheads in the ERB, we can relocate the refresh buffer inside the FIFO buffer and when a flit needs to get refreshed, the flit is moved to the IRB, and the IRBs keep progressing through the FIFO buffer as flits get refreshed. Note that the IRB is hidden from the upstream router to prevent the VA stage in the upstream router from allocating these buffers to flits. It is also noted that the impact of refresh overheads on the network throughput is negligible due to such seamless refresh operations. Also, since flits arrive at VCs at different points of time, no two flits have the same timestamp across all VCs in the same PC. Accordingly, we only need a single refresh buffer per PC for the refresh operation, and we assume a single flit can be refreshed per cycle in each PC.

•**Refresh Coverage:** As a means of keeping the integrity of data, prior study [49] also suggests to periodically *sweep* the memory for error correction, but they refresh only data turned out to be corrupted (**Selective Refresh**) 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

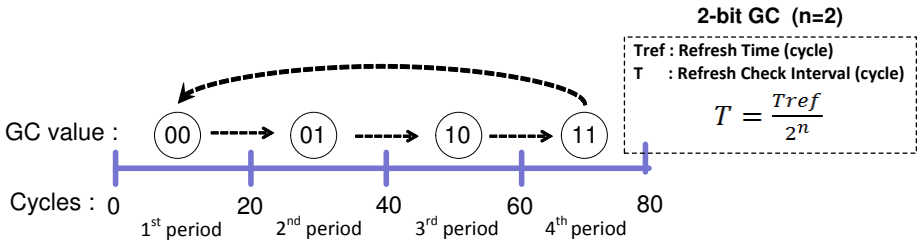


Figure 3.15: An Example of a 2-bit Global Counter (GC) Refresh Logic (*Assuming refresh time is 80 cycles (40 ns in 2 GHz))*

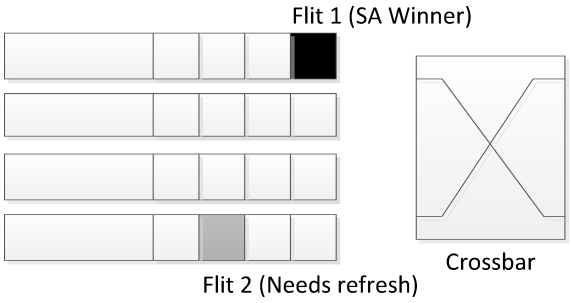


Figure 3.16: Concurrent Error Protection Example

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 3.14. Thus, to maintain low error probability in NoCs, we propose to trigger refresh even for currently valid flits in a buffer (**Enforced Refresh**), 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.

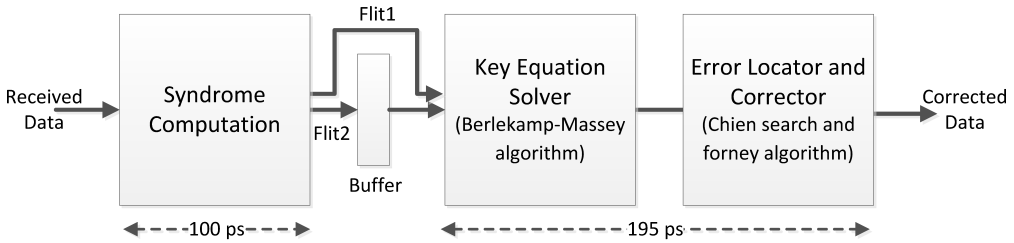


Figure 3.17: Two-Phase ECC for Concurrent Error Protection

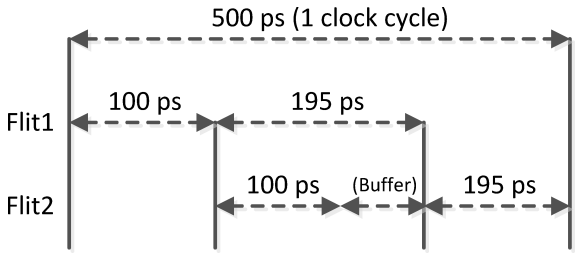
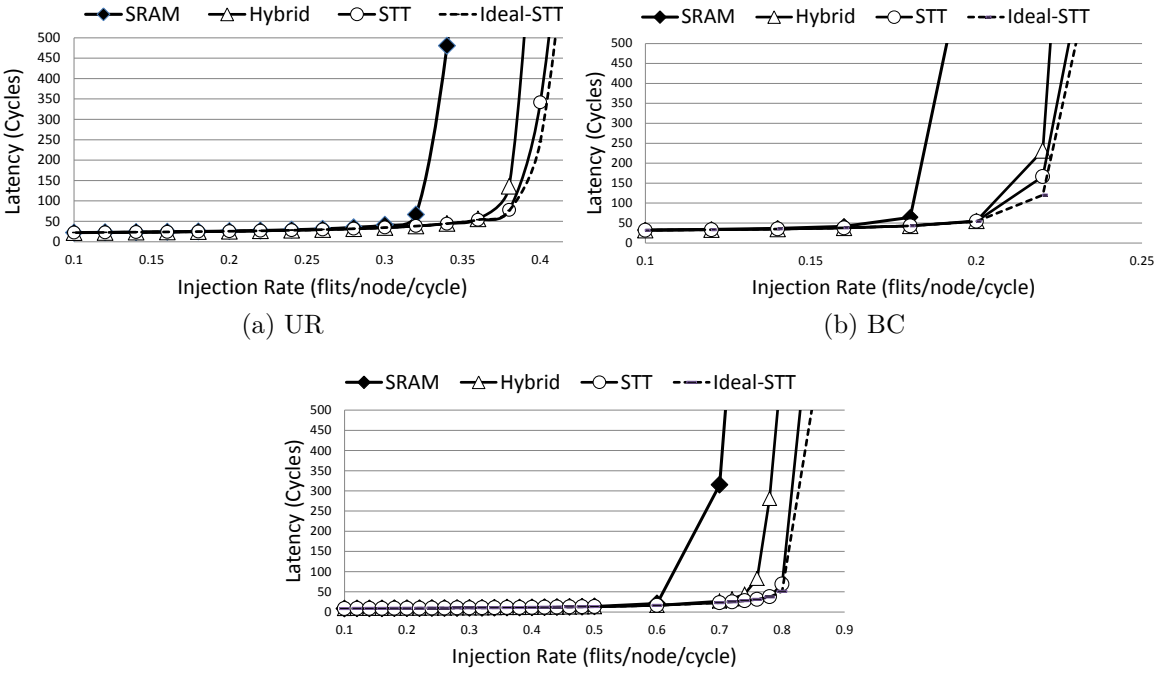


Figure 3.18: Timing Diagram of Concurrent Error Protection based on Two-Phase ECC

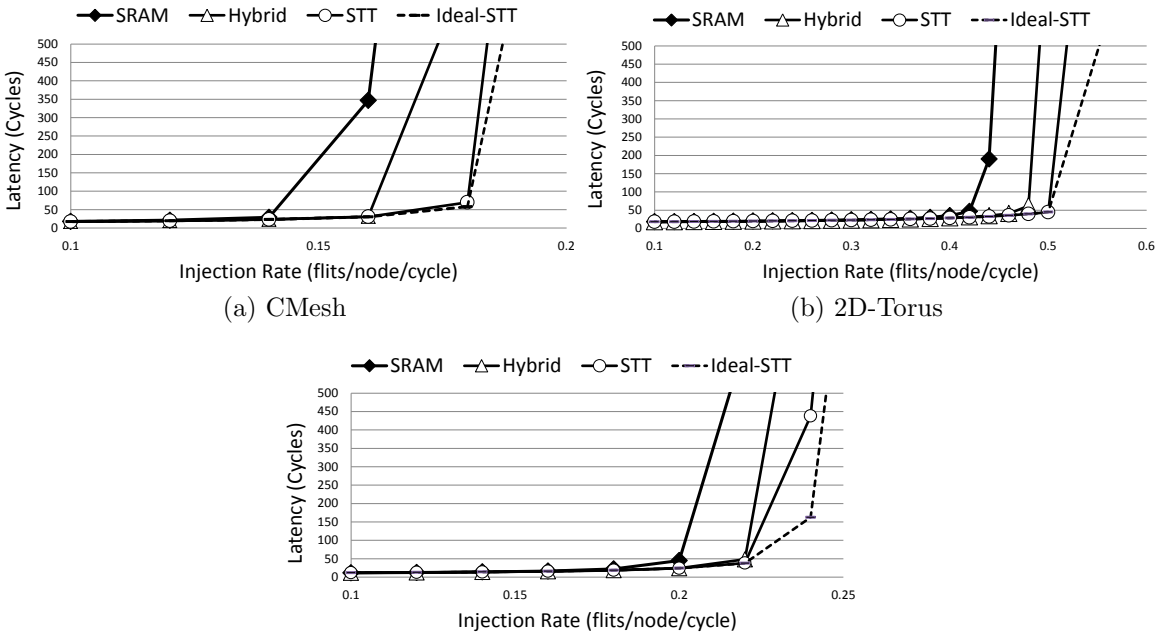


(a) UR

(b) BC

(c) NN

Figure 3.19: Performance Comparison with Different Synthetic Workloads



(a) CMesh

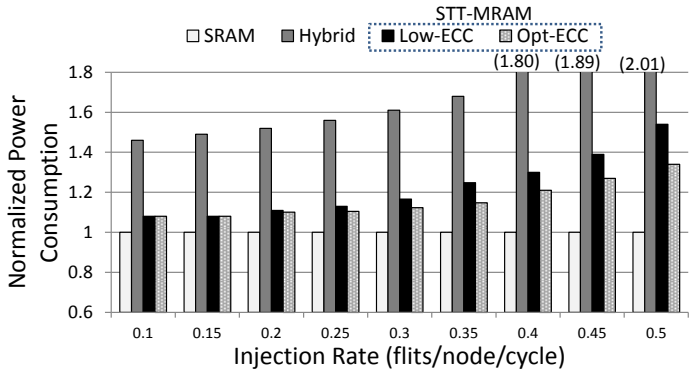
(b) 2D-Torus

(c) Flattened Butterfly

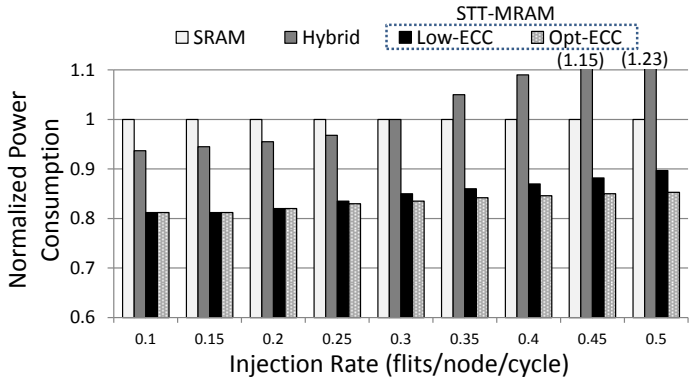
Figure 3.20: Performance Comparison with Different Topologies

3.4.3.3 Two-Phase ECC for Concurrent Error Protection

To ensure data integrity in NoCs, it is not only necessary to periodically refresh flits as detailed in Section 3.4.3.1, but flits also need to be checked for corruption and corrected if necessary before crossbar traversal in ST stage. This is because transferred flits which turn out to be *uncorrectable* in a downstream router, consume unnecessary power for crossbar and link traversal from an upstream router while occupying buffer spaces till next refresh interval in the downstream router. In avoiding such negative effects, we observe cases where concurrent error check and corrections are necessary at the same cycle; for instance, as shown in Figure 3.16, a flit, a winner in SA stage gets ready for error checks prior to crossbar traversal, while at the same time another flit in a buffer gets ready to be refreshed according to the refresh logic described in Section 3.4.3.2. To handle this, simply adopting two separate ECC



(a) Dynamic Buffer Power Consumption



(b) Total Router Power Consumption

Figure 3.21: 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.)

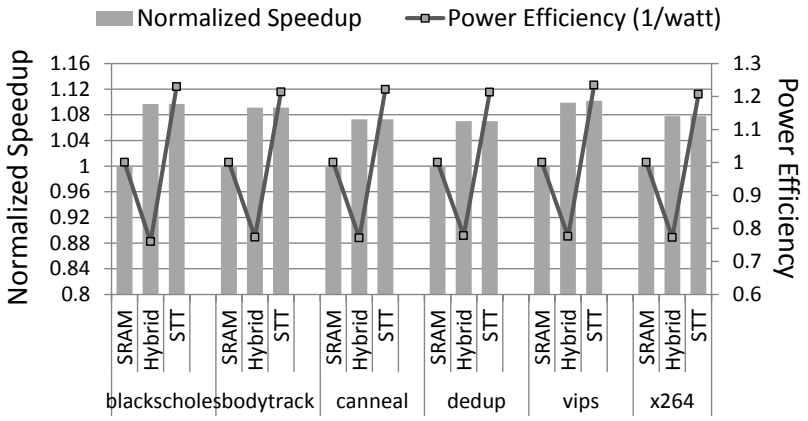


Figure 3.22: PARSEC Benchmark Results (Power and Performance Graphs)

modules per input port is disadvantageous in that each individual ECC sacrifices buffer spaces while incurring additional power overheads. Thus, we propose a **two-phase ECC** enabling a single ECC per input port to deal with concurrent error protection while avoiding potential network delay caused when sequentially checking for errors in multiple flits. In the two-phase ECC, ECC decoding logics are logically divided into two phases (detection and correction) using an intermediate latch as shown in Figure 3.17 and each phase is seamlessly pipelined and runs in parallel for different flits. Specifically, as shown in Figure 3.18, each phase is utilized in parallel by two flits. Once a flit 1 (SA winner) finishes a detection and enters a correction phase, the detection phase is immediately available and utilized by a subsequent flit, flit 2. According to our timing analysis, the proposed concurrent error protection can be done in a single cycle in 2GHz clock frequency, incurring no performance penalty.

Note that the intermediate latch serves as a temporal storage holding the subsequent flit and once the flit (SA winner) is detected as error-free and thus no need for correction, the subsequent flit seamlessly enter the correction phase, if necessary, without being latched. It is also noted that if a flit is turned out to be *uncorrectable* through the detection phase, we drop the corresponding packet from the network and send a retransmission request to a source. For packet dropping, if a part of a packet had already been transmitted, a *dummy tail* is sent, nullifying the packet in the downstream router.

3.5 Evaluation

3.5.1 System Configuration

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 [30] 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 [7] 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 [26, 33]. Note that the unit of the leakage power is mW per 1-flit buffer. Throughout this study, 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 ($SRAM4$ is equal to $STT16$). For the STT-MRAM buffer schemes, however, due to the extra area overheads incurred

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 [33] utilizes 10 ms.)*

by additional circuitry for the MBA shown in Figure 3.8 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

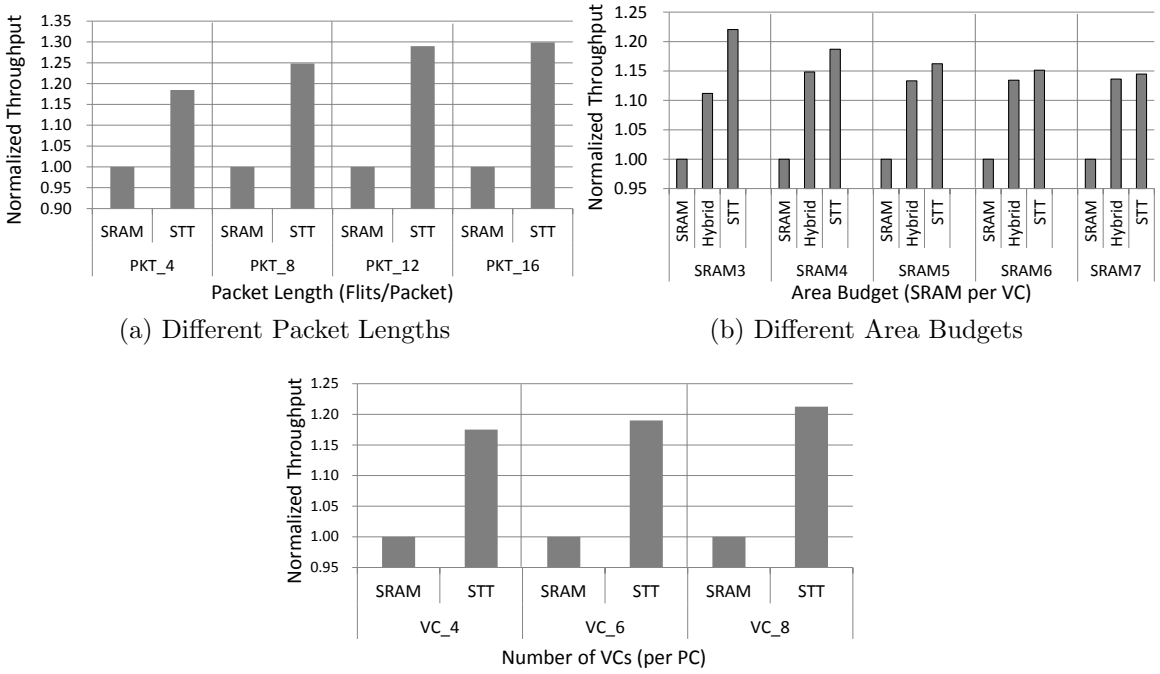
Figure 3.19 shows performance results of four different buffer configurations: the SRAM buffer, the Hybrid buffer, the proposed STT-MRAM buffer, and an ideal STT-MRAM buffer having no write delays with significantly large buffer spaces un-

der 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 [33], 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 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 3.20 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 3.21(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 3.21(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.



(a) Different Packet Lengths

(b) Different Area Budgets

(c) Different Number of VCs

Figure 3.23: Sensitivity Analysis

Figure 3.21(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 [34, 69]. 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 3.22 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 3.22, the relative performance improvement of the proposed scheme over the SRAM baseline is not comparable to those shown in Figure 3.19 (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 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 3.23 to examine their effects on NoC throughput. Figure 3.23(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 base-

line 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 3.23(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 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 3.23(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.

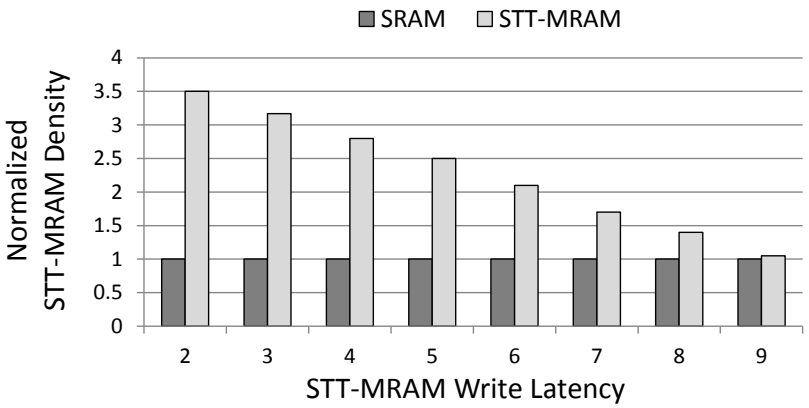


Figure 3.24: Normalized STT-MRAM Density under the Same Per-Router Area Budget

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) [48], and Whole Packet Forwarding (WPF) [45].

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, 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

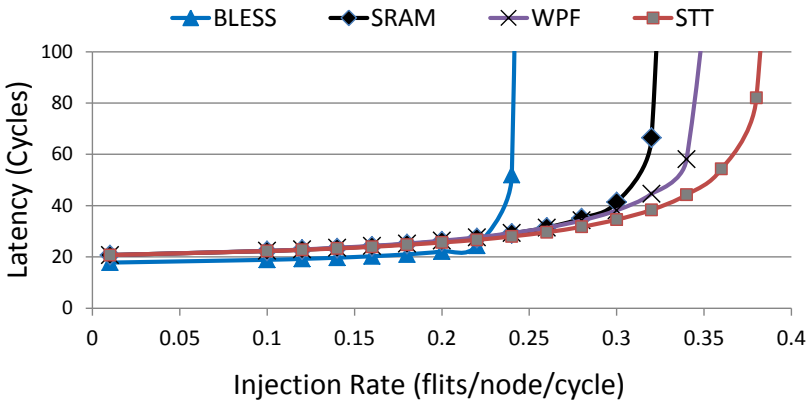


Figure 3.25: Comparisons with BLESS [48] and WPF [45] (UR)

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 3.24). 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 [48, 45]. We compare the power and performance benefits of our scheme with them. BLESS [48] reduces buffer power and area overheads by eliminating router buffers, and handles network contention by deflecting contending flits to any free output port. In our evaluation, the performance overheads of BLESS

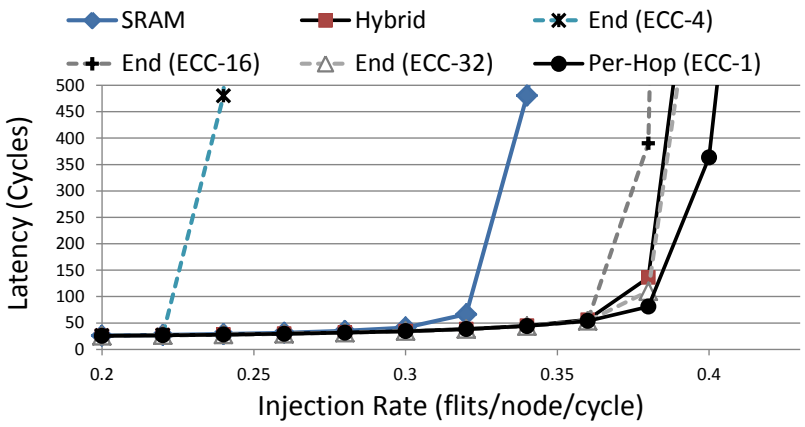


Figure 3.26: Comparisons between Different ECC Schemes (End-to-End vs. Per-Hop)

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 3.25. 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 [45] 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 3.25, 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.

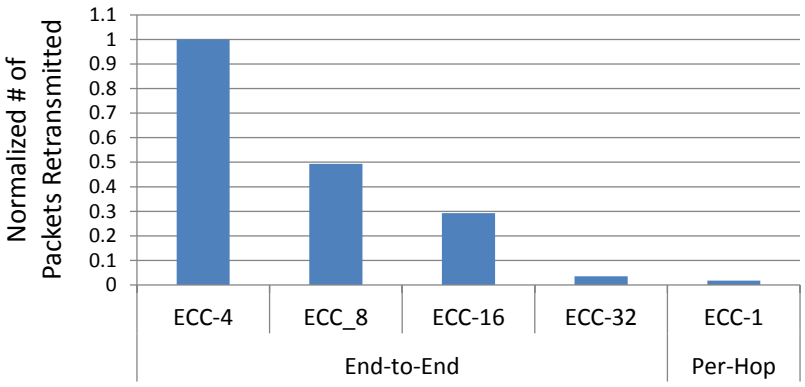


Figure 3.27: Normalized Number of Packets Retransmitted under Different ECC Schemes

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 study, 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 3.26 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 3.27. 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.* [26] detailed STT-MRAM based architectural techniques to offer power-efficient and scalable microprocessors. Goswami *et al.* [24] proposed STT-MRAM based GPGPU architectures and hybrid shared memory for power-performance optimizations. In [34, 59, 62], 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.

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 [10], 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 [22], it has higher packet latency overall, resulting in degraded performance.

3.8 Other Applications and Future Work

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 [54]. 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 [40, 39], and exploring the lifetimes of variables in programs [44], 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 [11] 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.

3.9 Conclusions

In this study, 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.

4. BANDWIDTH-EFFICIENT ON-CHIP INTERCONNECT DESIGNS FOR GPGPUS *

4.1 Introduction

General Purpose Graphics Processing Units (GPGPUs) have emerged as a cost-effective approach for a wide range of high performance computing workloads which have a high thread level parallelism (TLP) [36]. GPGPUs are characterized by numerous programmable computational cores which allow for thousands of simultaneous active threads to execute in parallel. The advent of parallel programming models, such as CUDA and OpenCL, makes it easier to program graphics/non-graphics applications, making GPGPUs an excellent computing platform. The growing quantity of parallelism and the fast scaling of GPGPUs have fueled an increasing demand for performance-efficient on-chip fabrics finely tuned for GPGPU cores and memory systems [4, 37].

Ideally, the interconnect should minimize blocking by efficiently exploiting limited network resources such as virtual channels (VCs) and physical channels (PCs) while ensuring deadlock freedom. Networks-on-Chip (NoCs) have been useful in chip-multiprocessor (CMP) environments due to their scalability and flexibility. Although NoC design has matured in this domain [31, 67], NoC design for GPGPUs is still in its infancy. Only a handful of works have examined the impact of NoC design in GPGPU systems [4, 37, 43, 71].

Unlike CMP systems, where traffic tends to be uniform across the cores communicating with distributed on-chip caches, the communication in GPGPUs is highly

asymmetric, mainly between many compute cores and a few memory controllers (MCs). Thus the MCs often become hot spots [4], leading to skewed usage of the NoC resources such as wires and buffers. Specifically, heavy reply traffic from MCs to cores potentially causes a network bottleneck, degrading the overall system performance. Therefore, when we design a bandwidth-efficient NoC, the asymmetry of its on-chip traffic must be considered. In prior work [4, 5, 37], the on-chip network is partitioned into two independent, equally divided (logical or physical) subnetworks between different types of packets to avoid cyclic dependencies that might cause protocol deadlocks. Due to the asymmetric traffic in GPGPUs skewed heavily towards reply packets, however, such partitioning can lead to imbalanced use of NoC resources given in each subnetwork. Thus, it fails to maximize the system throughput, particularly for memory-bound applications requiring a high network bandwidth to accommodate many data requests. The throughput-effectiveness is a crucial metric for improving the overall performance in throughput-oriented architectures, thus designing a high bandwidth NoC in GPGPUs is of primary importance. In the GPGPU domain, this is the first study evaluating and analyzing the mutual impacts of different MC placements and routing algorithms on system-wide performance. We observe that the interference from disparate types of GPGPU traffic can be avoided by adopting the *bottom* MC placement with proper routing algorithms, obviating the need of physically partitioned networks.

The contributions of this work are as follows: First, we quantitatively analyze the impact of network traffic patterns in GPGPUs with different MC placements and dimension order routing algorithms. Then, motivated by this detailed analysis, we propose VC monopolizing and partitioning schemes which dramatically improve NoC resource utilization without causing protocol deadlocks. We also investigate the impact of XY, YX, and XY-YX routing algorithms under diverse MC placements.

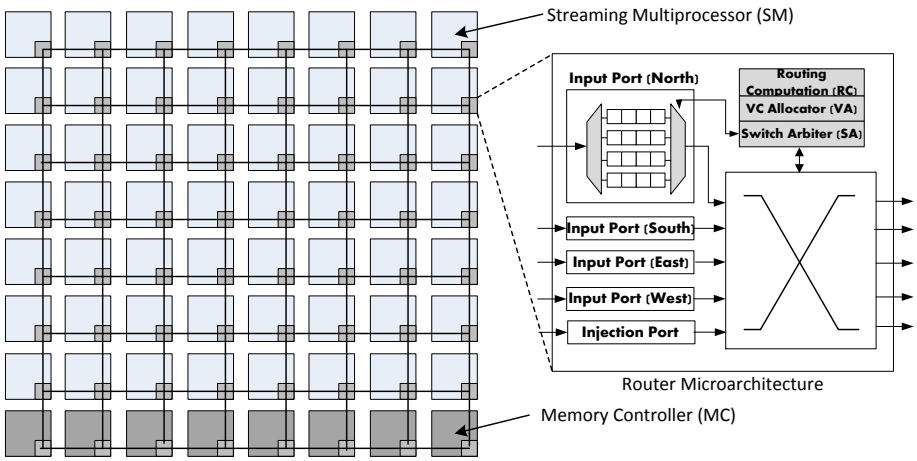


Figure 4.1: GPGPU NoC Layout and Router Microarchitecture. (The NoC layout consists of many SMs and a few MCs, each of which contains an NoC router.)

4.2 Background

In this section, we describe the baseline GPGPU architecture and NoC router microarchitecture in detail.

4.2.1 Baseline GPGPU Architecture

A GPGPU consists of many simple cores called streaming multiprocessors (SMs), each of which has a SIMT width of 8. The baseline GPGPU architecture consists of 56 SMs and 8 MCs as shown in Figure 4.1. Also, as shown in Figure 4.2, each SM is associated with a private L1 data cache, read-only texture/constant caches, and register files along with a low latency shared memory. Every MC is associated with a slice of the shared L2 cache for faster access to the cached data. We assume write-back polices for both L1 and L2 caches [5], and minimum L2 miss latency is assumed to be 120 cycles. We assume a 2D mesh to connect cores and MCs as in Figure 4.1 due to its advantages of scalability, simplicity and regularity [4].

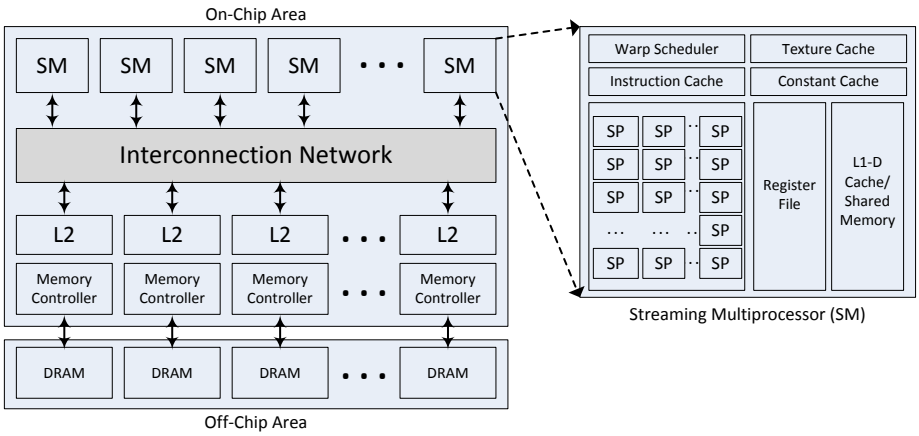


Figure 4.2: GPGPU Microarchitecture and Streaming Multiprocessor (SM)

4.2.2 Baseline NoC Router Architecture

Figure 4.1 shows the baseline NoC router, which has 5 I/O ports to connect the SMs to L2 cache and MCs in a GPGPU. The router is similar to that used by Kumar *et al.* [41] employing several features for latency reduction, including speculation 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 backpressure from downstream to upstream routers, which controls flit transmission rate to avoid buffer overflows.

4.3 Designing Bandwidth-Efficient NoCs in GPGPUs

Here, we analyze the GPGPU workload NoC traffic characteristics and their impact on system behavior. Based on this analysis, we propose VC monopolization and asymmetric VC partitioning to achieve higher effective bandwidth.

4.3.1 GPGPU On-Chip Traffic Analysis

4.3.1.1 Request and Reply Traffic

Prior work shows on-chip data access patterns to be more performance critical than data stream size in GPGPUs [25]. Further, these traffic patterns are inherently *many-to-few* (in the request network, from the many cores to the few MCs) and *few-to-many* (in the reply network, from the MCs back to the cores) [4]. As shown in Figure 4.3 *MC-to-core*, the reply network sees much heavier traffic loads than *core-to-MC*, the request network. This is because the request network consists of many short packets (read requests) mapped into a single flit and fewer long packets (write requests) mapped into 3∼5 flits. The reply network consists of many long packets (read reply) mapped into 5 flits and relatively a few short packets (write reply) mapped into a single flit. Figure 4.4 shows that on average around 63% of packets are read replies. Exceptionally, *RAY*, contains more request packets than reply packets, due to a write demand in this application.

In general, the ratio between request and reply traffic can be derived as follows. Considering the overall injection rate as λ at each node, we denote the ratio of read and write requests by r and w, respectively, and the sum of r and w equals one because request consists of only two types: read and write. The length of each packet can be divided into two groups: a short packet (L_s) representing read request and write reply, and a long packet (L_l) including read reply and write request. The amount of request traffic T_{rqs} is the sum of read and write requests and likewise, the

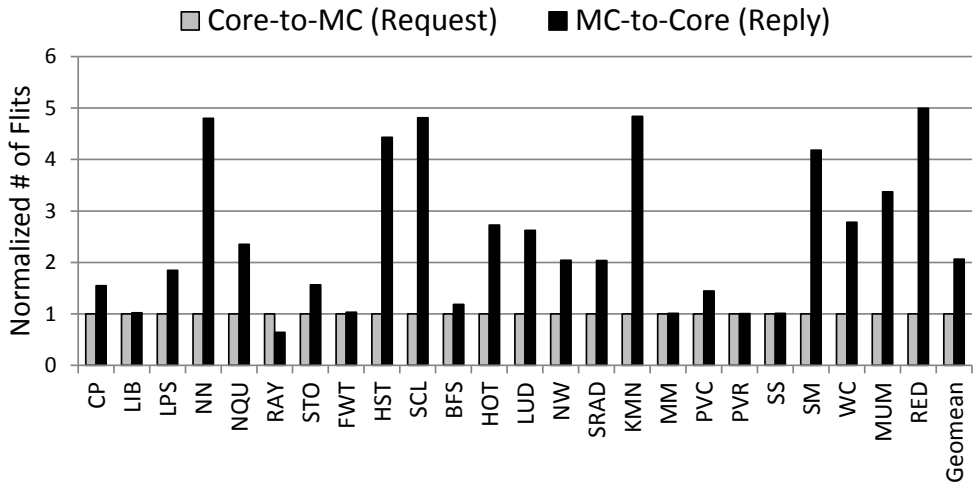


Figure 4.3: Normalized Traffic Volumes Between Cores and MCs.

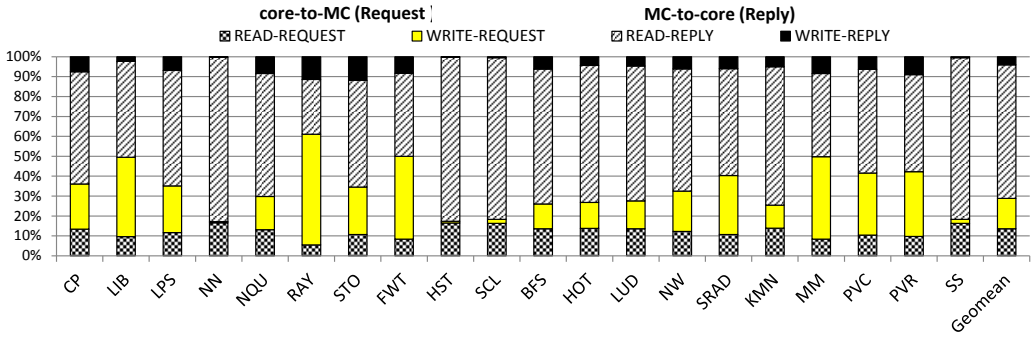


Figure 4.4: Packet Type Distribution for GPGPU Benchmarks

amount of reply traffic T_{rep} is the sum of read and write replies.

$$T_{rqs} = \lambda \cdot r \cdot \ L_s + \lambda \cdot w \cdot \ L_l$$
$$T_{rep} = \lambda \cdot r' \cdot \ L_l + \lambda \cdot w' \cdot \ L_s \tag{4.1}$$

where r' and w' denote ratios of replies for read (r) and write (w) requests, respectively. Since a single request is always followed by a single reply, the ratio between r and w is identical to that of r' and w'. Here, the ratio of reply to request (R) is derived by dividing T_{rep} by T_{rqs}. Thus, according to Figure 4.3, R equals around two since the traffic volume of reply packets is two times higher than that of request packets.

Figure 4.5 illustrates the traffic, T_{rqs} and T_{rep}, in an (NxN) mesh network with k MCs. In this figure, we take an example of $N = k = 4$, and XY routing with the bottom MC placement. Each arrow represents the direction of traffic and the associated coefficient number denotes the link utilization toward that direction. By multiplying the coefficient with either T_{rqs} or T_{rep}, we can approximate the amount of traffic towards a specific direction. For vertical links, a coefficient value is determined by the row location. For example, each core located at the 1st row (*i=1*) uses its south output port 4 times. If a core sends request packets to N MCs at the bottom row, the south port of a router associated with the core is utilized by N times. Thus, for a core located at the i^{th} row in an (NxN) mesh, the request coefficient towards south direction becomes $N \cdot i$. Similarly, we can utilize the core's column in deriving coefficient values for horizontal links. If a core j is located at the j^{th} column in the mesh network, $N - j$ MCs are on the east side of this core. To access these $N - j$ MCs, all cores located from the 1^{st} to the j^{th} columns must use the east output port of core j. Therefore, the coefficient for east port of j^{th} core is set by $j \cdot (N - j)$. In

a similar way, we analyze the coefficient values for all directions as follows.

$$
\begin{aligned}
C_{south} &= N \cdot i \\
C_{north} &= N \cdot (i-1) \\
C_{east} &= j \cdot (N-j) \\
C_{west} &= (N-j+1) \cdot (j-1)
\end{aligned}
\tag{4.2}
$$

The quantitative analysis above proves three important facts about the characteristics of GPGPU network traffic. First, reply traffic is much heavier than request traffic as empirically shown in Figures 4.3 and 4.4. Second, both request and reply traffic get congested as they approach to the MCs located at the bottom. The growing coefficient values of C_{south} and C_{north} from Equation 4.2 support this argument. Third, request and reply traffic do not get mixed on horizontal or vertical links as shown in Figures 4.5(a) and 4.5(b).

4.3.1.2 Memory Controller Placement

One way of alleviating traffic congestion is to move the MCs. Different MC placements help improve network latency and bandwidth by spreading the processor-memory traffic, balancing the NoC load. While prior work on MC placement shows the performance improvement to be gained from MCs placement in CMPs [2], this work does not analyze how MC placement affects the average hop count for GPGPUs. Here, we conduct a detailed quantitative analysis of GPGPU MC placement policies and show how distributed MC location can improve the NoC efficiency in GPGPUs.

The most clear advantage of distributed MC placement is the reduced average number of hops from cores to MCs, as shown in Figure 4.6. Comparing with other MC placements, specifically, the diamond MC placement shown in Figure 4.6(d)

allows more cores to access MCs with fewer hops. Since we are using dimension order routing, there is only one unique path from each core to a given MC. The row and column number of i^{th} MC are assumed to be $row_{m,i}$ and $col_{m,i}$, respectively. In the same way, the row and column number of j^{th} core are $row_{c,j}$ and $col_{c,j}$. With this notation, for an (NxN) mesh network with N number of MCs and $(N^2 - N)$ cores, the average number of hops from cores to MCs can be estimated as follows:

$$\begin{aligned} H_{avg} &= \frac{\sum (\text{Vertical + Horizontal hops})}{\text{Total number of paths}} = \frac{H_{vert} + H_{hori}}{(N^2 - N)N} \\ &= \frac{\sum_{j=1}^{N^2-N} \sum_{i=1}^{N} |row_{m,i} - row_{c,j}| + |col_{m,i} - col_{c,j}|}{N^2(N-1)} \end{aligned} \tag{4.3}$$

Note that Equation 4.3 is a general form of the equation applicable to any MC placement. H_{vert} and H_{hori} represent the aggregated number of hops for vertical and horizontal directions. We summarize H_{vert} and H_{hori} of different MC placements in Table 4.1.

Based on Table 4.1, we find that sorting the MC placement diagrams in the order of decreasing average number of hops yields the following order: *bottom, edge, top-bottom,* and *diamond*. This analysis also corresponds with results from prior work [2], which reported the best performance improvement with *diamond* MC placement. Although the *diamond* placement shows the least number of hops, we show that other MC placement policies can outperform the *diamond* placement by adopting VC monopolizing and different routing algorithms in Section 4.4.2.

4.3.1.3 Traffic Analysis for Different MC Placements

In this section, we show numerical traffic analysis for different MC placements. For simplicity, we do traffic analysis for XY request traffic only. Also, we assume

that the number of MCs is a multiple of four (*i.e.*, $N = 4k$). The same analysis can be applied to other routing algorithms and reply traffic.

Bottom MC - Horizontal Traffic. Since we use XY routing, the horizontal movement occurs before the vertical movement. Therefore, the cores located at the same row affect the amount of horizontal traffic. Let's take an example of arbitrary core located at the i^{th} row and j^{th} column and name it as core (i,j). For the bottom MC placement, Figure 4.6a shows that there are (N-b) MCs placed at the east side of core (i,j) and j cores are placed at the west side including core (i,j) itself. Therefore, core (i,j)'s east port will be used (N-j)j times. Similarly, there are (j-*1*) MCs on the west side of core (i,j) and (N-j+*1*) cores are placed at the west side including core (i,j) itself. Therefore, core (i,j)'s west port will be used (b-*1*)(N-b+*1*) times. To summarize, the overall horizontal traffic of a core (i,j) is

$$(N-j)j + (j-1)(N-j+1) \tag{4.4}$$

Since there are N number of cores on each row, each row has the following amount of horizontal traffic.

$$\begin{aligned}
&\sum_{j=1}^{N}(N-j)j + (j-1)(N-j+1) \\
=&\sum_{j=1}^{N} -2j^2 + 2j(N+1) - (N+1) \\
=&-2 * \frac{N(N+1)(2N+1)}{6} + 2(N+1) * \frac{N(N+1)}{2} - N(N+1) \\
=&\frac{N(N+1)(N-1)}{3}
\end{aligned} \tag{4.5}$$

The bottom MC placement has (N-1) core rows and a single MC row. Since there is no horizontal request traffic between MCs, the total amount of horizontal traffic is

$$\begin{aligned} &\frac{N(N+1)(N-1)}{3} * (N-1)rows \\ =&\frac{N(N+1)(N-1)^2}{3} \end{aligned} \tag{4.6}$$

Bottom MC - Vertical Traffic. In XY routing, the vertical movement occurs after the horizontal movement. Therefore, the amount of vertical traffic is also relevant to the cores located at the other rows in the network. Let's take another example with core (i,j). Since each row has N number of cores and each column has one MC at the bottom, N number of vertical requests will pass the south port of core (i,j). In addition, each row above core (i,j) also generates same N vertical requests towards south. Therefore, each core at the $i^(th)$ row has Ni amount of vertical traffic. Since we have N columns in the network, the total amount of vertical traffic is

$$\begin{aligned} &N * \sum_{i=1}^{N} Ni \\ =&N^2 * \sum_{i=1}^{N} i \\ =&N^2 * \frac{N(N-1)}{2} \\ =&\frac{N^3(N-1)}{2} \end{aligned} \tag{4.7}$$

Top-Bottom MC - Horizontal Traffic. All MC's horizontal location is same as the bottom MC placement. Therefore, the total amount of horizontal traffic remains

same as Equation 4.6.

Top-Bottom MC - Vertical Traffic. Unlike the bottom MC placement, there are N number of cores whose horizontal location is same as that of MCs. These cores are located at the 1^{st} row and i^{th} row. Since $N/2$ of MCs are located at the same row, these cores generate $N/2$ amount of vertical traffic. Other cores located between the 2^{nd}nd row and $(i-1)^{th}$ row have same amount of vertical traffic as the bottom MC placement. Therefore, the total amount of vertical traffic is

$$\begin{aligned} & N * \Big(\sum_{i=1}^{N-1} \frac{N}{2} + N(i-1) \Big) \\ =& \frac{N^2(N-1)}{2} + N^2 * \sum_{i=1}^{N-1} (i-1) \\ =& \frac{N^2(N-1)}{2} + N^2 * \frac{(N-1)(N-2)}{2} \\ =& \frac{N^2(N-1)^2}{2} \end{aligned} \tag{4.8}$$

Comparing Equation 4.7 and Equation 4.8 shows that the top-bottom MC placement has less amount of vertical traffic than that of bottom MC placement. Therefore, we can expect higher performance for the top-bottom MC placement.

Edge MC - Horizontal Traffic. Unlike previous MC placements, the horizontal location of MCs is different in the edge MC placement. Figure 4.6b shows that there are $N/2$ MCs from 1^{st} to $N/4^{th}$ column and another pair of $N/2$ MCs from $(N-N/4+1)^{th}$ column to N^{th} column. The 1^{st} row and the N^{th} row have different amount of horizontal traffic compared to other rows ($2^{nd}\sim(N-1)^{th}$ row) since those two rows have $N/4$ MCs at each side of the edge. Considering these facts, the 1^{st} and N^{st} rows have following horizontal traffic.

$$\Big(\sum_{j=1}^{N-\frac{N}{2}} Nj + \sum_{j=1}^{\frac{N}{4}} 2 * (\frac{N}{4} - j) * (N - \frac{N}{2}) \Big) * 2rows = \frac{3}{8} N^3 \tag{4.9}$$

From the 2^{nd} to $N - 1^{th}$ row, each row has following horizontal traffic.

$$\begin{aligned} &\Big(\sum_{j=1}^{N-\frac{N}{4}} Nj + \sum_{j=1}^{\frac{N}{4}} 4 * (\frac{N}{4} - j) + \sum_{j=(N-\frac{N}{4}+1)}^{N} 4 * (N - j)j \Big) \\ =&\frac{N(19N^2 - 16)}{48} N^3 \end{aligned} \tag{4.10}$$

Therefore, the total amount of horizontal traffic is

$$\begin{aligned} &\frac{3}{8} N^3 + \frac{N(19N^2 - 16)}{48} N^3 * (N - 2)rows \\ =&\frac{N(19N^3 - 20N^2 - 16N + 32)}{48} \end{aligned} \tag{4.11}$$

Since subtracting Equation 4.11 from Equation 4.5 is always larger than zero, we can conclude that the edge MC placement has larger amount of horizontal traffic. In addition, the amount of vertical traffic is same for both edge and top-bottom placements, we can expect that the top-bottom MC placement will have less amount of XY-request traffic and better performance improvement.

Edge MC - Vertical Traffic. All MC's vertical location is same as the top-bottom MC placement. Therefore, the total amount of horizontal traffic remains same as Equation 4.8.

Diamond MC - Horizontal and Vertical Traffic. The main advantage of the Diamond MC placement is that total traffic is well distributed across the GPU net-

work by distributing MCs in each quadrant. The Diamond MC placement allows each quadrant to have same number of MCs. However, we found that it is difficult to do traffic analysis for the Diamond MC placement since there are many different ways to place MCs inside of each quadrant. Rather than deriving a definite set of traffic analysis, we approximate both horizontal and vertical traffic of Diamond MC placement by half of the vertical traffic of Top-Bottom MC placement. This is a reasonable approximation since each quadrant of Top-Bottom MC placement also has the same number of MCs. The major difference is that the Diamond MC placement distributes network traffic by locating MCs at the center of the quadrant. In other words, the Diamond MC placement is a special case of Top-Bottom MC placement where MCs are located at the center of the quadrant rather than the top or bottom of the quadrant. Based on this observation and simulation results, we summarize the traffic analysis in Table 4.1.

4.3.2 Proposed Design

4.3.2.1 VC Monopolizing and Asymmetric VC Partitioning

Request and reply packets in GPGPUs compete for NoC resources such as VCs and PCs. When the resources are naïvely shared by both packets, avoiding protocol deadlock requires that reply packets must not compete for the same resources as request packets. To avoid this, prior studies [5, 4, 37] suggest partitioning NoCs equally into two parts for the different types of traffic: one network carries request packets and the other network reply packets. Creating two parallel physical networks [37] incurs significant hardware overheads due to the twofold increase in the number of routers and wire resources. To this overhead, we employ a virtual network partitioning, where the network is divided virtually by two separate sets of VCs dedicated for request-reply traffic under one physical network.

MC placement	H_{vert}	H_{hori}
Bottom	$\frac{N^3(N-1)}{2}$	$\frac{N(N+1)(N-1)^2}{3}$
Edge	$\frac{N^2(N-1)^2}{2}$	$\frac{N(19N^3-20N^2-16N+32)}{48}$
Top-Bottom	$\frac{N^2(N-1)^2}{2}$	$\frac{N(N+1)(N-1)^2}{3}$
Diamond	$\approx \frac{N^2(N-1)^2}{4}$	$\approx \frac{N^2(N-1)^2}{4}$

Table 4.1: The Average Number of Vertical/Horizontal Hops under Different MC Placements in an (NxN) Mesh

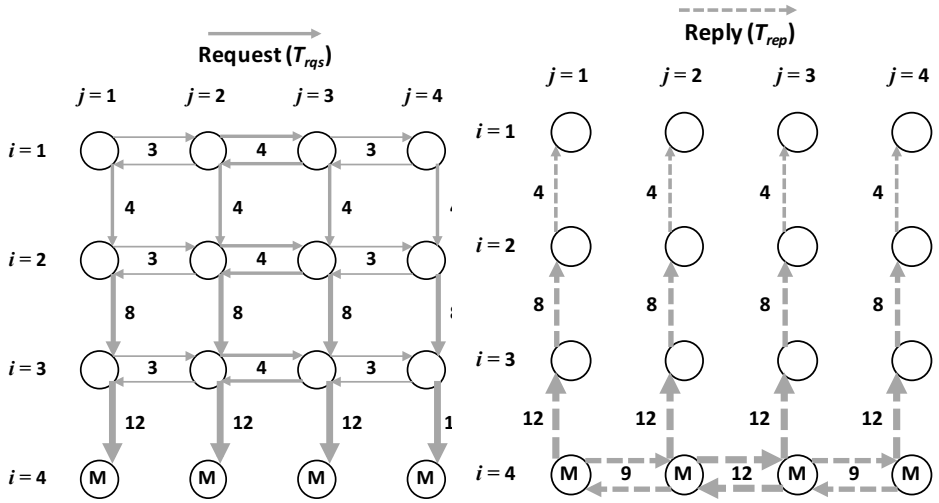
However, when all MCs are located at the bottom, request and reply traffic are not overlapped with dimension ordered routing as shown in Figure 4.5. Therefore, there is no need to split networks to avoid protocol deadlock. Thus, all the VCs can be *fully monopolized* by either request or reply packets, providing more buffer resources for each type of traffic, thus helping improve overall system performance. On the other hand, VC monopolizing is not feasible when VCs have mixed request and reply traffic, as shown in Figure 4.10(c). These mixed VCs must be partitioned into request and reply packets to avoid protocol deadlock. In this case, we propose *asymmetric VC partitioning* which assigns more VCs to reply traffic. Since reply traffic generally requires much more network bandwidth than request traffic, moving

VC resources from the request to the reply improves the overall system performance while maintaining the same overall NoC area and power budget. The detailed evaluation is described in Section 4.4.2.

4.3.2.2 Routing Algorithms

A routing algorithm is one of the critical factors in achieving bandwidth-efficient NoC, influencing the amount of traffic each link will carry. Routing contributes to the reduction in network contention (hot spots) when combined with an appropriate MC placement. To find the performance-optimal combination of a routing algorithm and an MC placement, we analyze the impact of different dimension order routing algorithms (XY, YX, and XY-YX [2]) under the different MC placements shown in Figure 4.6. For example, under our baseline MC placement, *bottom MC*, shown in Figure 4.6(a), XY routing incurs increased network contention mainly due to the high volume of reply traffic between MCs, thus degrading overall system performance. Alternatively, XY-YX routing which leads request packets to follow XY routing, while reply packets to follow YX routing, helps achieve significant performance improvement because heavy traffic between MCs due to reply packets is entirely eliminated as shown in Figure 4.10. Since the request traffic in YX routing still generates contention between MCs, the performance improvement of YX routing is less than that of XY-YX routing. However, the reply traffic in YX routing does not cause any communication between MCs since the reply traffic always traverses to the Y direction first. Therefore, XY-YX or YX routing is effective in load-balancing the processor-memory traffic in a 2D mesh topology with the *bottom* MC placement scheme[1]. On other MC placement schemes, we find routing algorithms have little impact on overall performance. Relevant simulation results are detailed in Section 4.4.

[1] we do not consider adaptive routing because of the increased critical path delays in a router [2] and degraded row buffer locality caused by not preserving the order of request packets to MCs [71].



(a) XY Request

(b) XY Reply

(c) XY Request + XY Reply

Figure 4.5: Network Traffic Example with XY Routing. *(Note that request (a) and reply (b) traffic take different paths, thus traffic does not mix on horizontal and vertical links.)*

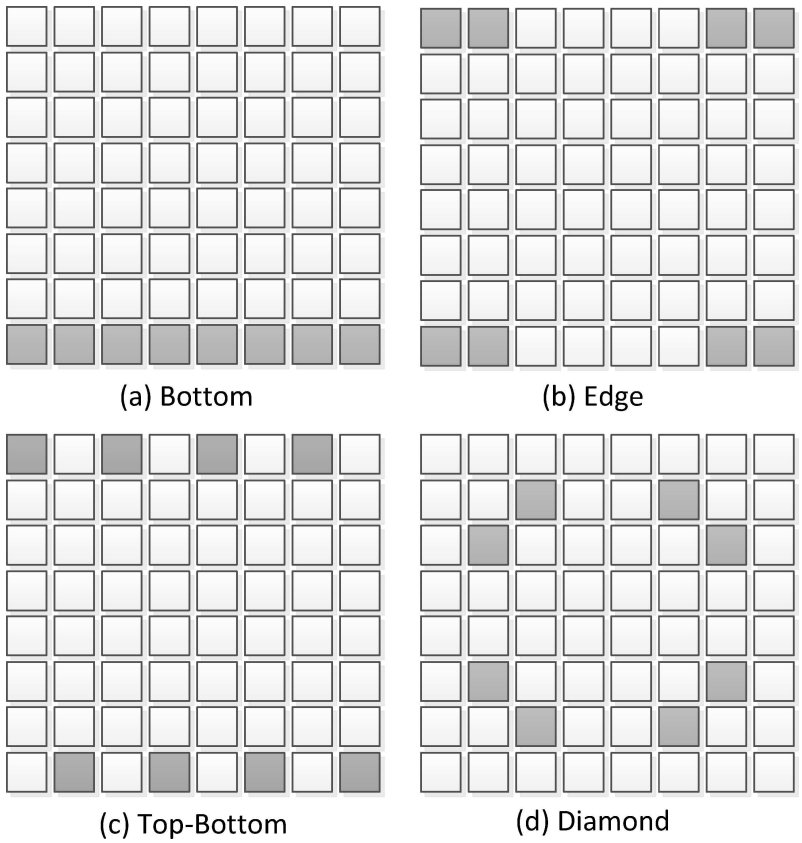
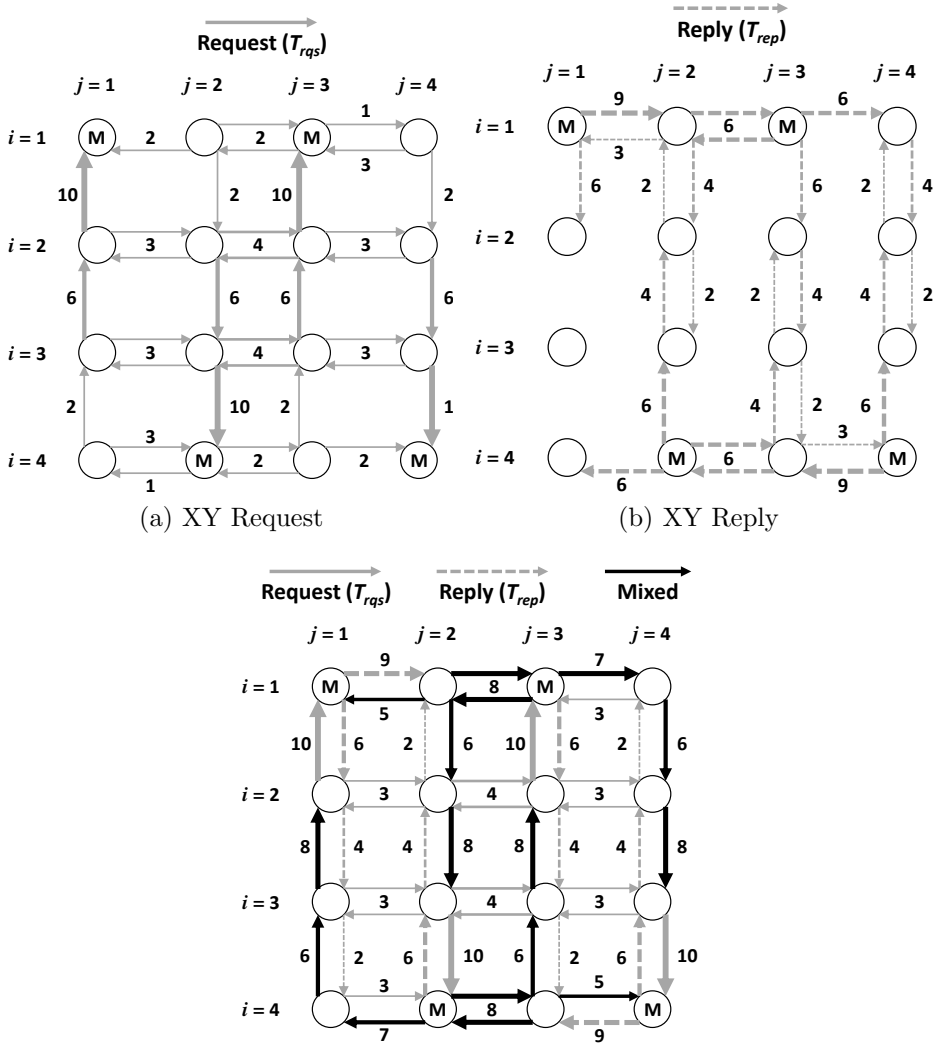


Figure 4.6: Different MC Placements. *(Shaded tiles represent MCs co-located with GPGPU cores.)*

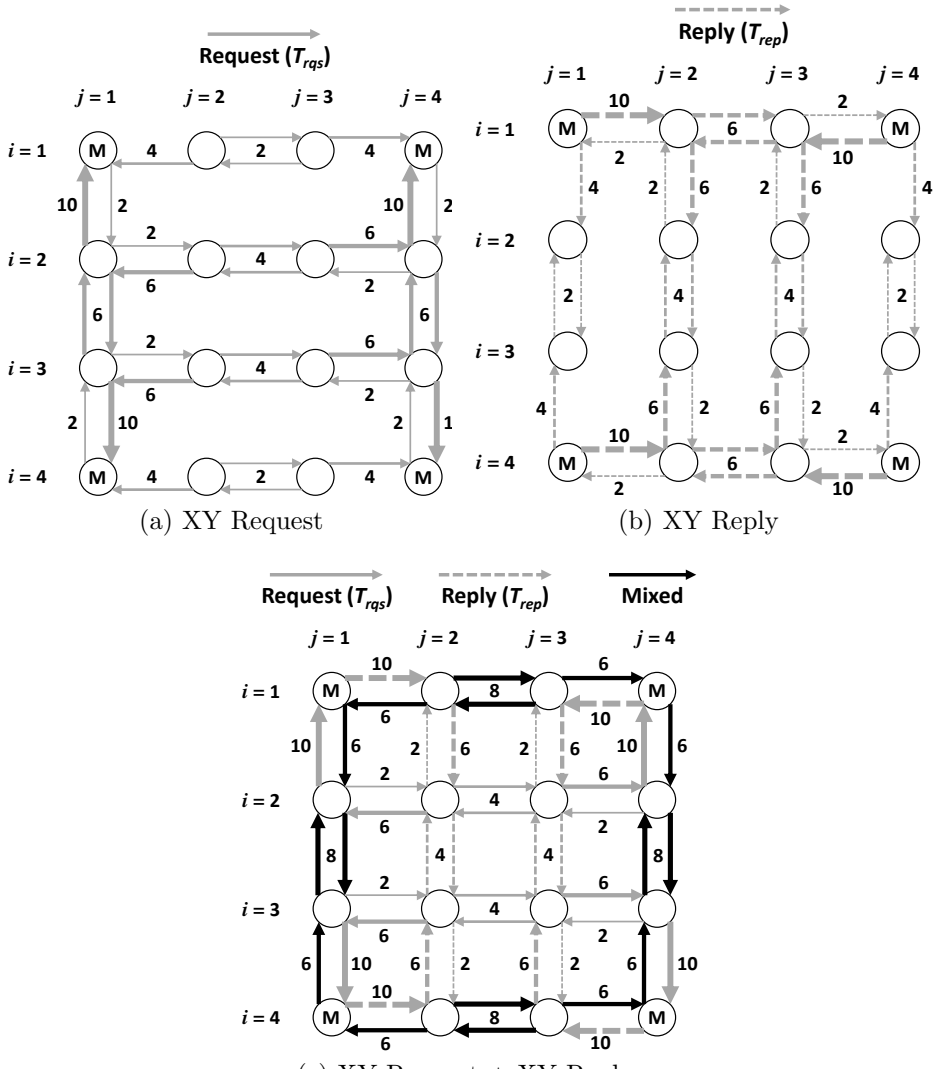


(a) XY Request

(b) XY Reply

(c) XY Request + XY Reply

Figure 4.7: Network Traffic Example with Top-Bottom MC and XY Routing.

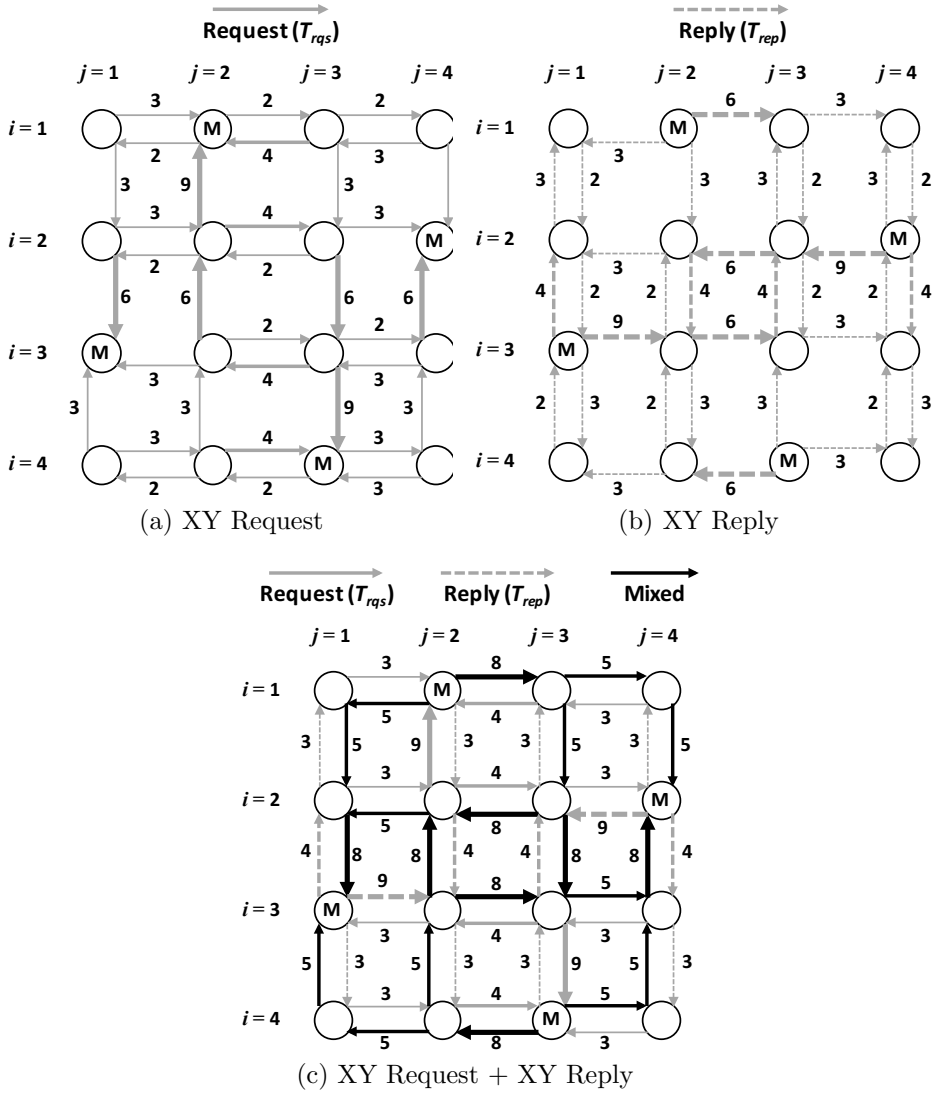


(a) XY Request

(b) XY Reply

(c) XY Request + XY Reply

Figure 4.8: Network Traffic Example with Edge MC and XY Routing.

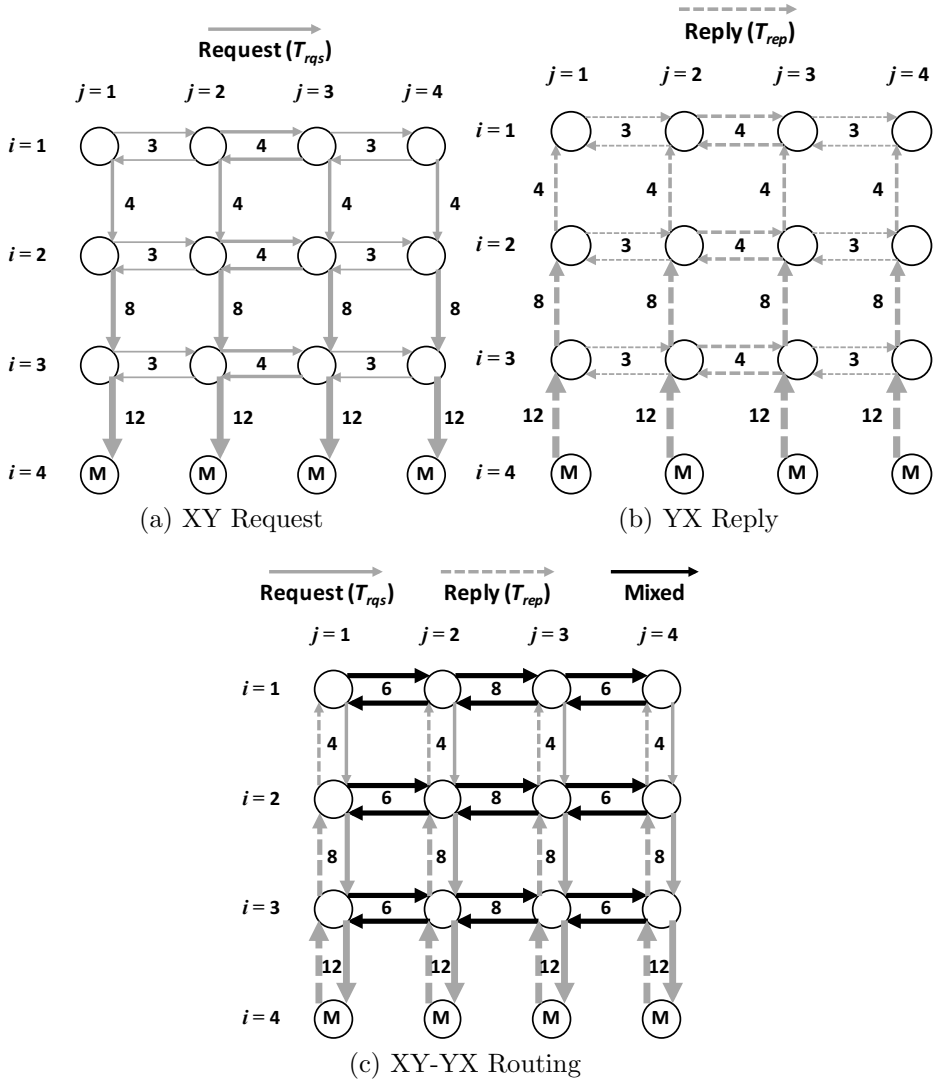


(a) XY Request

(b) XY Reply

(c) XY Request + XY Reply

Figure 4.9: Network Traffic Example with Diamond MC and XY Routing.



(a) XY Request (b) YX Reply

(c) XY-YX Routing

Figure 4.10: Network Traffic Example with XY-YX Routing. (*Note, request/reply traffic is mixed on horizontal links.)*

4.4 Performance Evaluation

In this section, we evaluate schemes proposed in Section 4.3 with the aim of developing a high performance NoC, optimized for use in GPGPUs. We also analyze simulation results in detail using a wide variety of GPGPU benchmarks.

4.4.1 Methodology

The NoC designs and MC placement schemes examined here are implemented in GPGPU-Sim [6]. The simulator is flexible enough to capture the internal design of GPGPU and our target architecture has similarities to NVIDIA's FermiGTX 480. Figure 4.1 shows the NoC router microarchitectures modeled in GPGPU-Sim. A 2D mesh network is used to connect SMs, caches, and MCs. To prevent protocol deadlock, the baseline NoC (Table 4.2) is built with a single physical network with two separate VCs for handling request and reply traffic. We evaluate our schemes with a wide range of GPGPU workloads such as CUDA SDK [1], ISPASS [5], Rodinia [9], and MapReduce [29]. Each benchmark suite is structured to span a range of parallelism and compute patterns, providing feature options that help identify architectural bottlenecks and fine tune system designs.

4.4.2 Performance Analysis

4.4.2.1 Impact of Network Division

As described in Section 4.3.2.1, we advocate for a single physical network with separate virtual networks for request and reply packets. To avoid protocol deadlock, we increase the number of VCs per port, where different types of packets traverse on-chip networks via different VCs. It is noted that additional VCs employed to avoid a protocol deadlock can affect the critical path of a router since VC allocation is the bottleneck in the router pipeline [37]. However, we observe that two separate

System Parameters	Details
Shader Core	56 Cores, 1400 MHz, SIMT width = 8
Memory Model	8 MCs, 924 MHz
Interconnect	8 x 8 2D Mesh, 1400 MHz, XY Routing
Virtual Channel	2 VCs per Port
VC Depth	4
Warp Scheduler	Greedy-then-oldest (GTO)
MC placement	Bottom
Shared Memory	48KB
L1 Inst. Cache	2KB (4 sets/4 ways LRU)
L1 Data Cache	16KB (32 sets/4 ways LRU)
L2 Cache	64KB per MC (8-way LRU)
Min. L2 / DRAM Latency	120 / 220 cycles

Table 4.2: System Configuration

VCs under a single physical network degrades system performance less than 0.03% in geometric mean across 25 benchmarks. This observation leads us to use separate VCs with a single physical network instead of two physical networks requiring more hardware resources.

4.4.2.2 Impact of Routing Algorithms

While maintaining the same number of VCs with reduced network resources, we observe that alternative routing algorithms can significantly improve the overall system performance. Figure 4.11 shows the speed-up obtained with YX and XY-YX, normalized against the baseline XY. YX and XY-YX with the *bottom* MC placement scheme achieve a speedup of 39.3% and 64.7%, respectively. As discussed in Section 4.3.2.2, the improvement mainly comes from mitigated traffic congestions between MCs. The heavy reply traffic generated from MCs is the main factor causing performance bottlenecks in NoCs. In this context, XY-YX routing outperforms YX routing by more than 25%. This is because unlike YX routing, XY-YX completely

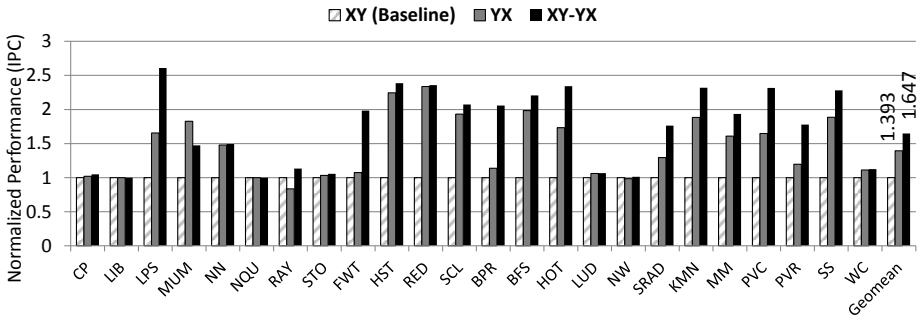


Figure 4.11: Speed-up with Routing Algorithms *(Normalized to baseline XY)*

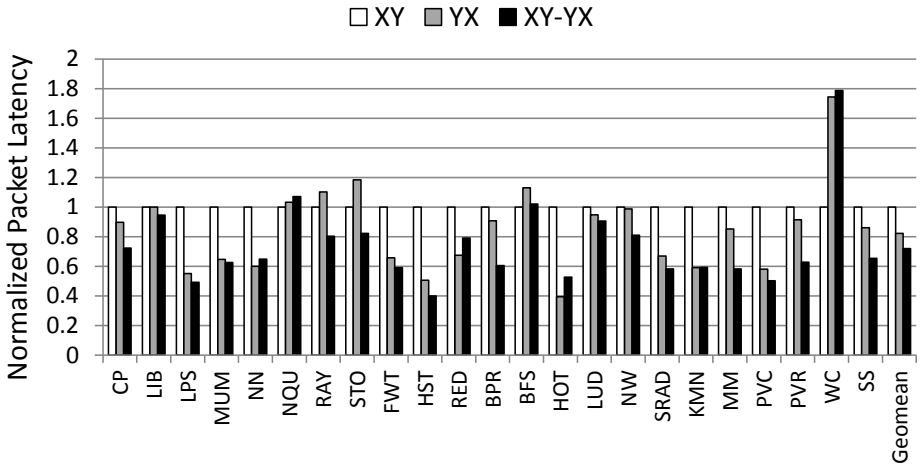


Figure 4.12: Normalized Packet Latency under Different Routing Algorithms

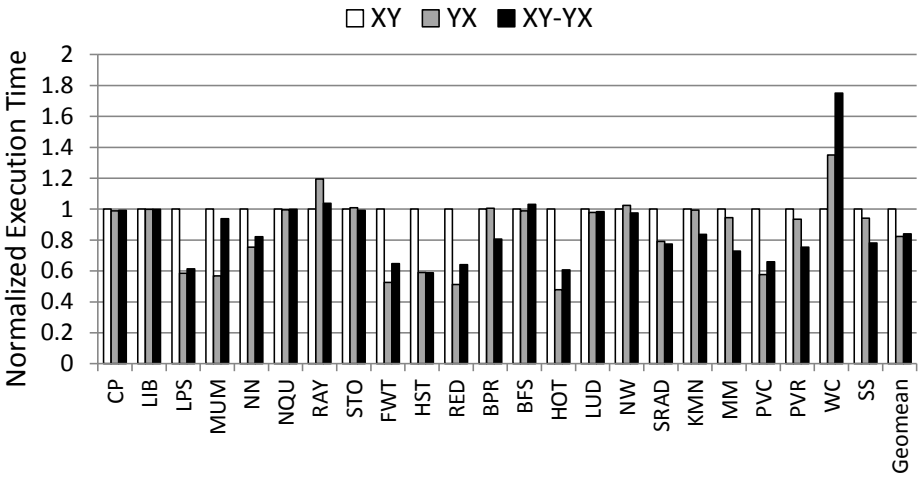


Figure 4.13: Normalized Execution Time under Different Routing Algorithms

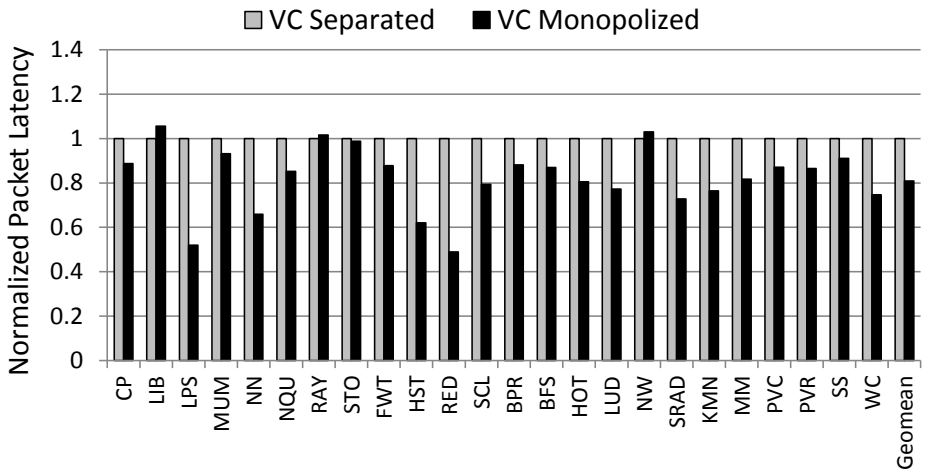


Figure 4.14: Normalized Packet Latency under VC Monopolizing Scheme

removes the resource contention between MCs by providing different routing paths for request and reply packets as illustrated in Figure 4.10. In the context of NoC, Figure 4.12 shows the overall network latency under different routing algorithms normalized to the baseline XY routing. On average, XY-YX routing shows less packet latency compared to that of XY and YX routings by 27% and 10.9%, respectively. This is mainly due to the absence of network contention between MCs. Figure 4.13 shows the comparison of total execution time that is normalized to that of XY routing algorithm for each benchmark application. It is interesting to note that YX and XY-YX routing show comparable total execution time even under the difference in their respective NoC characteristics such as packet latencies shown in Figure 4.12.

4.4.2.3 VC Monopolizing

As illustrated in Figure 4.5, request and reply packets never overlap with each other in any dimension under XY or YX routing with the *bottom* MC placement, allowing VC monopolization as described in Section 4.3.2.1. Figure 4.16 shows the impact of VC monopolizing on system performance under different routing algorithms.

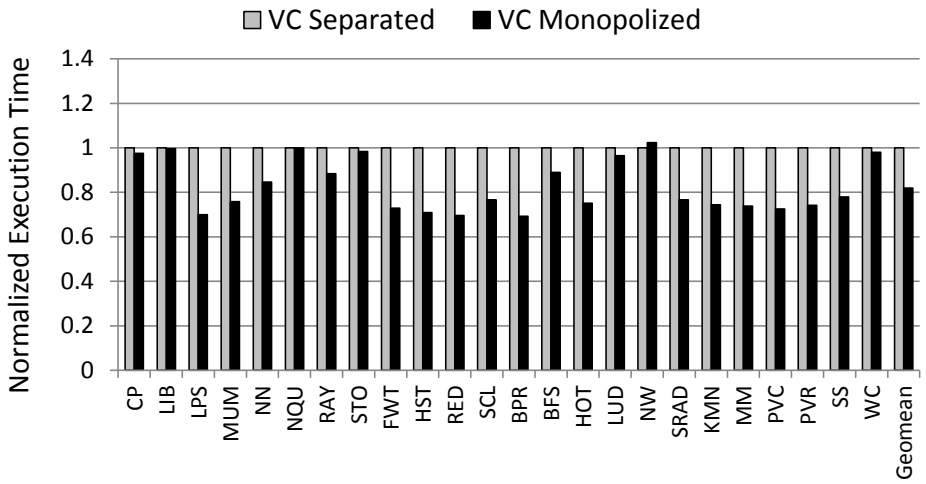


Figure 4.15: Normalized Execution Time under VC Monopolizing Scheme

Monopolized VCs lead XY and YX to achieve 43.8% and 88.9% (= 39.3% from YX + 49.6% from monopolization) of speed-up in geometric mean, respectively. Note that unlike XY and YX routing, XY-YX routing still requires separate VCs in horizontal links to prevent protocol deadlock because different types of packets get potentially mixed while moving along the horizontal links as illustrated in Figure 4.10. This limits the number of VCs that can be monopolized (*partial monopolizing*) because only the VCs located in the vertical links can be fully monopolized in XY-YX routing. Accordingly, partially monopolized XY-YX routing shows less performance improvement at 85.4% (= 64.7% from XY-YX + 20.7% from monopolization), compared to that of the fully monopolized scheme with YX routing. Furthermore, across different MC placements (*edge*, *top-bottom*, and *diamond*), VC monopolizing is effective in achieving better performance improvement, as detailed below. In terms of NoC, VC monopolizing scheme is also effective in reducing the network packet latency compared to that of VCs separated for disparate traffic as shown in Figure 4.14 where VC monopolizing reduces packet latency by 19.7% on average. Accordingly, as shown in

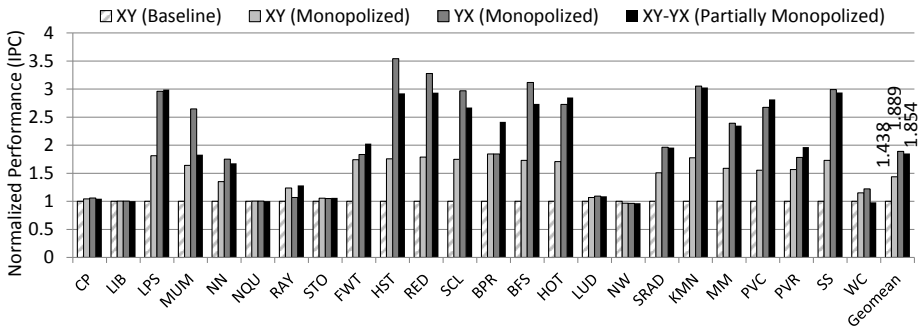


Figure 4.16: Speed-up with VC Monopolized Scheme *(Normalized to XY routing with VC separated for each traffic)*

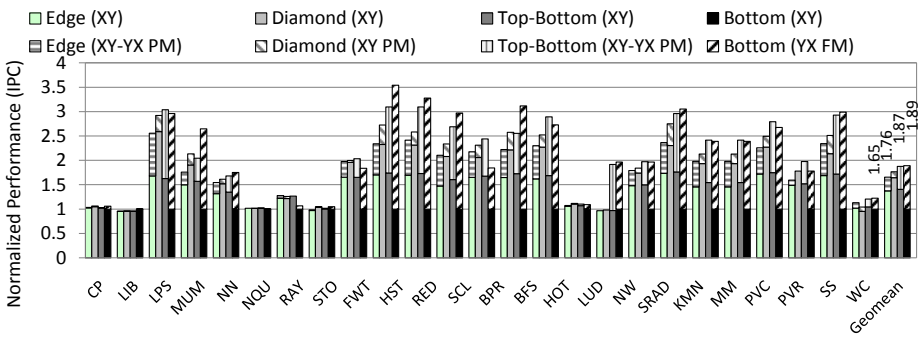


Figure 4.17: Speed-up with Different MC Placements with Routing Algorithms *(PM: Partial Monopolizing, FM: Full Monopolizing, Normalized to bottom MC+XY routing)*

Figure 4.15, the overall execution time for each benchmark is also reduced by 19.2%, on average, when applying VC monopolizing scheme.

4.4.2.4 Impact of MC Placement Scheme

In our baseline, we simply put all MCs at the bottom row in 8×8 2D mesh. As the network traffic in GPGPUs gets skewed towards the MCs in this scheme, the *bottom* MC placement causes high network congestion near the MCs, thus degrading performance. One way to alleviating such traffic congestion is to locate MCs

sparsely across the network. Figure 4.17 shows the impact of different MC placements on overall system performance. Note that each MC placement is simulated with three different routing algorithms (XY, YX, and XY-YX). In Figure 4.17, we pick the routing algorithm showing the highest performance improvement for each MC placement scheme. In the figure we see that MC location has a significant impact on performance. This is because, with distributed MC placements, request and reply packets are spread across multiple locations of the on-chip network rather than converging to the bottom row. Compared to the bottom MC placement, the average performance speedup is 37.3%, 64.4%, and 40.4% for the *edge*, *diamond*, and *top-bottom* placements, respectively. And when applying the VC monopolizing scheme in combination with different MC placements, additional 28.3%, 12%, and 47.3% performance improvement (65.6%, 76.4%, and 87.7% in total) are achieved with the *edge*, *diamond*, and *top-bottom* placements, respectively. Here it is worthwhile to note that our baseline bottom MC placement combined with YX routing and fully monopolized VCs shows the highest performance improvement (89.4%) and even outperforms the prior top-performing work, diamond MC placement, by 25% (= 89.4% - 64.4%), even though the diamond has the least number of hops as analyzed in Section 4.3.1.2. This proves the performance effectiveness of the VC monopolizing scheme described earlier in Section 4.3.2.1.

4.4.2.5 Asymmetric VC Partitioning

Mixed request and reply traffic limits the possibility of using the *VC monopolizing* scheme. Different routing algorithms such as XY-YX routing or dispersed MC placements like *diamond* cause the mix of traffic in the middle of the links. In such network configurations, we apply asymmetric VC partitioning between request and reply packets as we detailed in Section 4.3.2.1. To show the impact of asymmet-

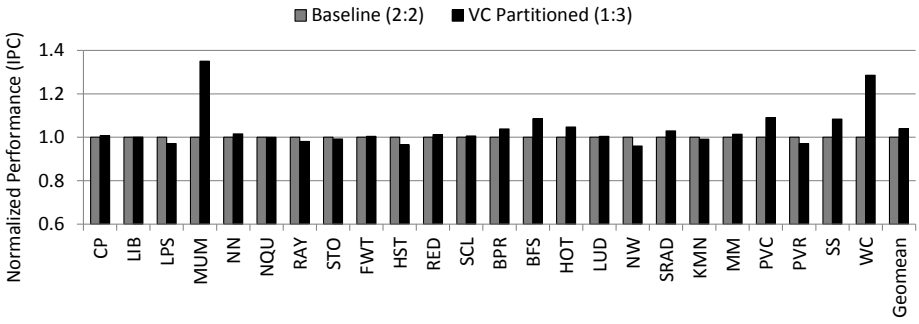


Figure 4.18: Speed-up with Asymmetric VC Partitioning *(Request:Reply = 1:3)*

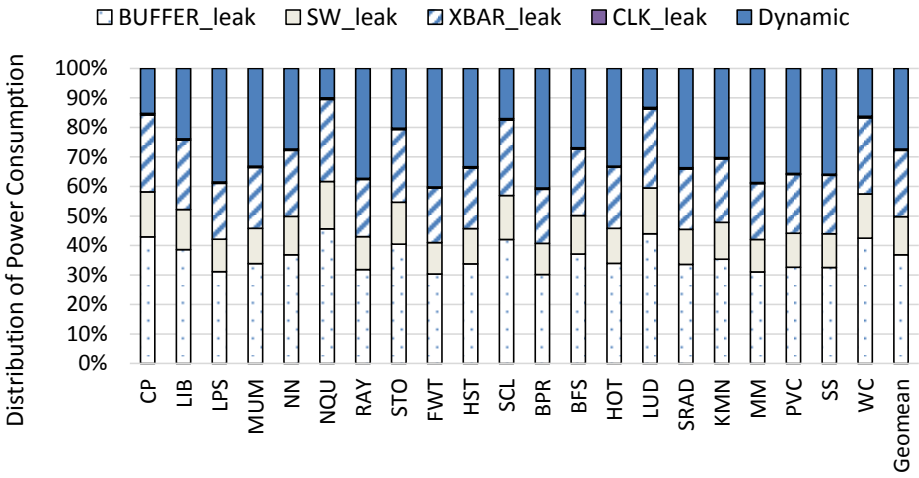


Figure 4.19: Distribution of Power Consumption in NoC

ric VC partitioning, we assume the number of VCs per port are four. Figure 4.18 shows a speed-up with asymmetric VC partitioning, where only one VC is assigned to request packets and other VCs are assigned to reply packets. For XY-YX routing, the asymmetric partitioning improves the performance by 3.9% in geometric mean. Since reply packets are usually heavier than request packets, assigning more VCs to reply packets is beneficial assuming enough VCs already exist. Note that asymmetric VC partitioning is effective in enhancing performance across all MC placements.

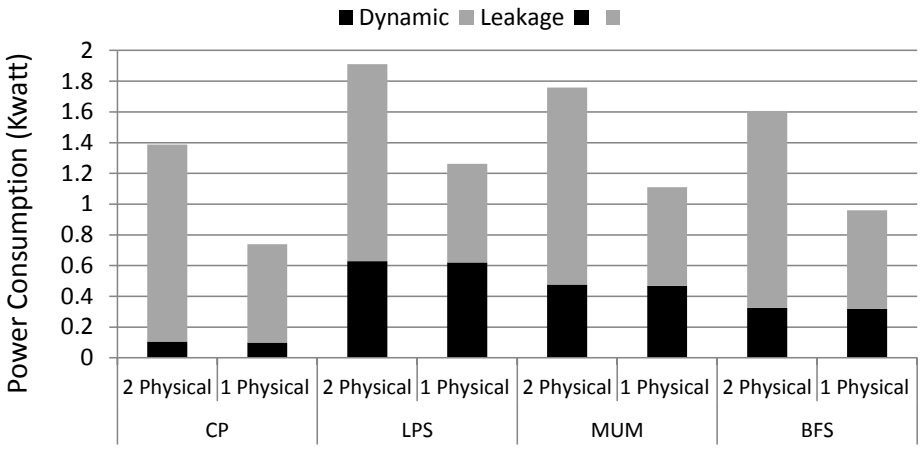


Figure 4.20: Network Power Breakdown under 2 Physical vs. 1 Physical Network

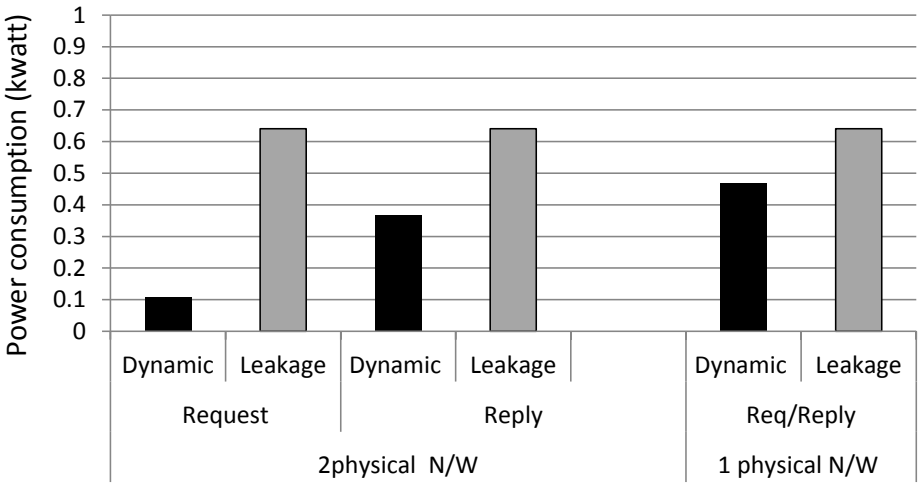


Figure 4.21: Detailed Network Power Breakdown (MUM benchmark) under 2 Physical vs. 1 Physical Network

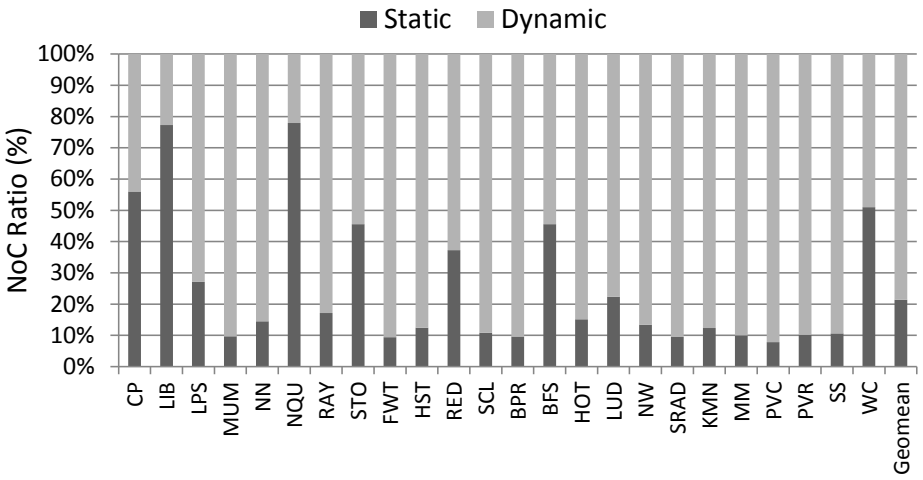


Figure 4.22: NoC Static vs. Dynamic Ratio

4.4.3 Power Analysis

Power is one of the critical metrics that should be considered in designing power-efficient NoCs in GPGPUs. In this section, we analyze the impact of different physical channel configurations, routing algorithms, and VC monopolizing scheme on NoC power consumption.

4.4.3.1 Distribution of Power Consumption

Figure 4.19 shows the detailed distribution of power consumption in NoC. Most of the power is consumed by leakage power (72%) aside from dynamic power. On-chip router buffer and crossbar are the major leakage power consumer for NoC, consuming 37% and 25% leakage power, respectively.

4.4.3.2 Impact of Network Division

As detailed in Section 4.4.2, a single physical channel having multiple VCs is more effective in reducing the total power consumption in NoCs compared to that of a separate physical channel. Figure 4.20 shows the total power consumption when adopting

different physical channel configurations. The separate physical channel contributes to increasing the leakage power due to the redundant use of physical channels and on-chip routers, thus consuming almost doubled leakage power compared to that of the single physical channel. The amount of dynamic power consumption varies across different benchmarks due to different network loads inherent in each benchmark. Specifically, for benchmarks showing high degree of network dynamicity in NoC such as LPS and MUM, more dynamic power are consumed than other benchmarks. (Figure 4.22 shows such network dynamic characteristics unique for each benchmark.) Figure 4.21 shows detailed network power consumption under different physical channel configurations. Since the single physical channel configuration use only 2 VCs for request and reply traffic, the individual dynamic power consumption is higher than that of the separate physical network. However, due to the significant leakage power consumption and higher total dynamic power consumption of the separate physical network, the overall power consumption is much higher than that of a single physical network.

4.4.4 Sensitivity Analysis

4.4.4.1 Impact of Arbitration Policy (VA/SA)

Different arbitration policy in on-chip routers affect the performance of GPGPUs. Figure 4.23 shows the different performance when applying different arbitration policy in VA and SA stage in on-chip routers. Overall, age-based VA and SA shows higher performance than that of RR because of the priority given to the oldest flit waiting in buffers in on-chip routers.

4.4.4.2 Impact of Router Buffer Depths

Increasing network buffering capacity is effective in enhancing performance in GPGPUs. Figure 4.24 shows the performance results under three different buffer

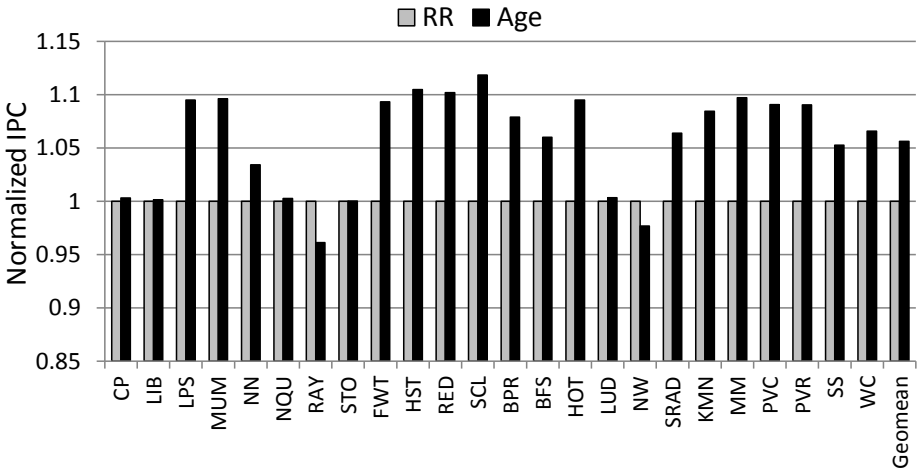


Figure 4.23: Normalized IPC under Different Arbitration Policy (RR vs. Age)

depths such as 4 flits, 8 flits, and 16 flits per VC. Increasing buffer depth helps to increase the amount of traffic retained in NoC, thus helping to increase overall network throughput, leading to better performance in terms of IPC for each benchmark. Specifically, on average, increasing the buffer depth by 4 times, BUF4 to BUF16, leads to 11.3% performance improvement .

4.4.4.3 Impact of Router Pipeline Latencies

In a conventional on-chip router, a flit goes through 4 stages pipeline consisting of RC, VA, SA, and ST as mentioined in Section 4.2, where each stage takes a single clock cycle. Such a pipeline stage delay impacts the overall NoC overheads in GPGPUs. To hide the multiple pipeline stage delays, we employ schemes such as lookahead routing and speculation, thus the total delay can be reduced from 4 to 2 cycles. Figure 4.25 shows the impact of different pipeline stage delays on the overall GPGPU performance for each benchmark. As a flit goes through less pipeline stage delays, the overall performance gets improved such that 2 cycle pipelines improve performance by 9.7% on average, compared to that of 4 cycle pipelines.

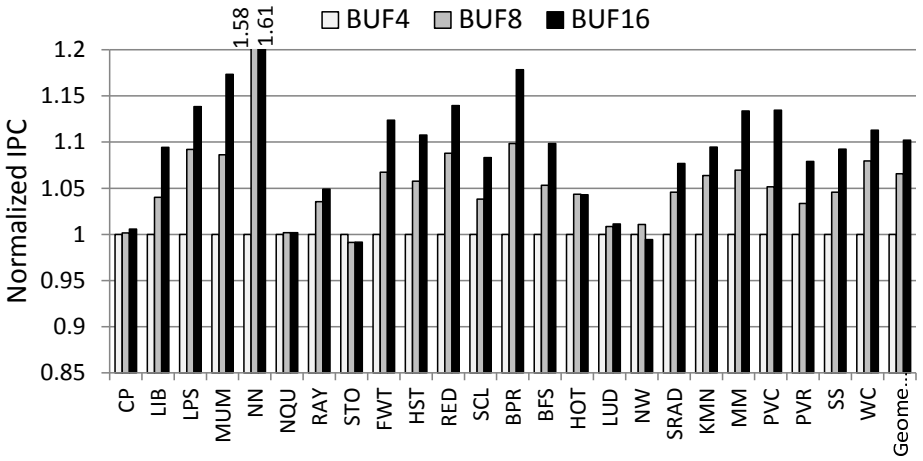


Figure 4.24: Normalized IPC under Different Buffer Depth

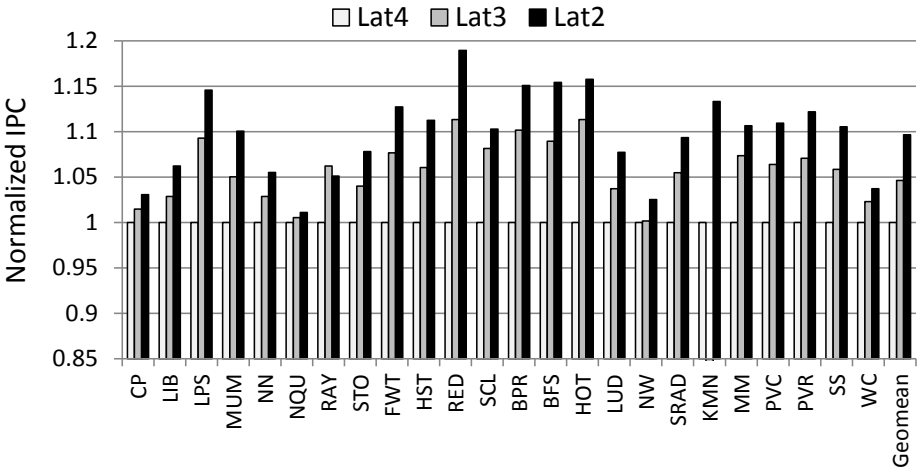


Figure 4.25: Normalized IPC under Different Router Pipeline Latencies

4.5 Related Work and Conclusions

Yuan et al. [71] proposed a complexity-effective memory scheduler for GPU architectures. NoC routers of the proposed mechanism reorder packets to increase row-buffer locality in the memory controllers. As a result, a simple in-order memory scheduler can perform similar to a much more complex out-of-order scheduler. Bakhoda et al. [4] proposed a throughput-effective NoC for GPU architectures. Due to many cores with a smaller number of memory controllers, a many-to-few traffic pattern is dominant in GPUs. To optimize such traffic, they used a half router, which disallows turns, to reduce the complexity of the network while increasing the injection bandwidth from the memory controllers to provide burst data read. Kim *et al.* propose [37] lightweight high frequency NoC router architectures, which reduces the pipeline delay in routers, achieving high on-chip network bandwidth, and reducing energy consumption due to the simplified architecture. In a heterogeneous system, Lee *et al.* [43] propose feedback based VC partitioning between CPU and GPU applications. The proposed technique dedicates a few VCs to CPU and GPU applications with separate injection queue. VC partitioning in heterogeneous architectures is effective at preventing interference between the CPU and GPU applications. However, it requires a feedback mechanism to dynamically partition VCs on each router which might be additional overhead in a heavily loaded network such as GPGPUs.

In this study, we analyze the unique characteristics of on-chip communication within GPGPUs under a variety of benchmark applications. We find that the *many-to-few* and *few-to-many* traffic between cores and MCs create a significant bottleneck, leading to the inefficient use of NoC resources in on-chip interconnects. We show the improved system performance based on VC monopolizing and asymmetric VC partitioning under diverse MC placements and dimension ordered routing algorithms.

5. CONCLUSIONS

On-chip interconnects for CMPs and GPGPUs have brought new challenges of overcoming the inherent constraints of the restricted power and area budgets in a chip. To address such challenges, three schemes have been proposed to design high performance, power-efficient, and reliable NoCs.

First, we propose a hybrid input buffer design using STT-MRAM with SRAM to achieve better network throughput with marginal power overheads in onchip interconnection networks. The high density of STT-MRAM facilitates to accommodate larger buffer compared to the conventional SRAM under the same area budgets. Through the flit migration schemes, the long write latency of STT-MRAM is effectively hidden while minimizing the power overheads.

Second, we present 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 withmultiple 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.

Third, we analyze the unique characteristics of on-chip communication within GPGPUs under a wide range of benchmark applications. We find that the many-to-few and few-to-many traffic patterns between cores and MCs create a severe bottleneck, leading to the inefficient use of NoC resources in on-chip interconnects. We show significant improvements in GPGPUs based on the proposed schemes.

REFERENCES

[1] NVIDIA CUDA SDK. `https://developer.nvidia.com/gpu-computing-sdk`.

[2] Dennis Abts, Natalie D. Enright Jerger, John Kim, Dan Gibson, and Mikko H. Lipasti. Achieving Predictable Performance Through Better Memory Controller Placement in Many-Core CMPs. In *Proceedings of ISCA*, 2009.

[3] Vikram S. Adve and Mary K. Vernon. Performance Analysis of Mesh Interconnection Networks with Deterministic Routing. *IEEE Transactions on PDS*, 5, 1994.

[4] Ali Bakhoda, John Kim, and Tor M. Aamodt. Throughput-effective on-chip networks for manycore accelerators. In *Proceedings of MICRO*, 2010.

[5] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *ISPASS*, pages 163–174, 2009.

[6] Ali Bakhoda, George L. Yuan, Wilson W. L. Fung, Henry Wong, and Tor M. Aamodt. Analyzing CUDA Workloads Using a Detailed GPU Simulator. In *Proceedings of ISPASS*, 2009.

[7] Daniel U. Becker. *Efficient Microarchitecture for Network-on-Chip Routers*. Ph.D. Dissertation, 2012.

[8] Yu Cao, Takashi Sato, Michael Orshansky, Dennis Sylvester, and Chenming Hu. New Paradigm of Predictive MOSFET and Interconnect Modeling for Early Circuit Simulation. In *Proceedings of IEEE Custom Integrated Circuits Conference*, 2000.

[9] Shuai Che, Michael Boyer, Jiayuan Meng, David Tarjan, Jeremy W. Sheaffer, Sang-Ha Lee, and Kevin Skadron. Rodinia: A Benchmark Suite for Heterogeneous Computing. In *Proceedings of IISWC*, 2009.

[10] Lizhong Chen and Timothy M. Pinkston. NoRD: Node-Router Decoupling for Effective Power-gating of On-Chip Routers. In *Proceedings of MICRO*, 2012.

[11] Tien-Fu Chen and Jean-Loup Baer. Effective Hardware-Based Data Prefetching for High-Performance Processors. *IEEE Transactions on Computers*, 44:609–623, 1995.

[12] Yu Cao Chengen Yang, Yunus Emre and Chaitali Chakrabarti. Improving Reliability of Non-Volatile Memory Technologies Through Circuit Level Techniques and Error Control Coding. *EURASIP Journal on ASP*, 2012, 2012.

[13] W. J. Dally. Virtual-Channel Flow Control. *IEEE Trans. Parallel Distrib. Syst.*, 3:194–205, March 1992.

[14] W. J. Dally and C. L. Seitz. Deadlock-Free Message Routing in Multiprocessor Interconnection Networks. *IEEE Trans. Comput.*, 36:547–553, May 1987.

[15] 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*, 96:022514, 2010.

[16] B. Del Bel, Jongyeon Kim, C.H. Kim, and S.S. Sapatnekar. Improving STT-MRAM Density Through Multibit Error Correction. In *Proceedings of DATE*, 2014.

[17] Zhitao Diao, Zhanjie Li, Shengyuang Wang, and Yunfei Ding. Spin-Transfer Torque Switching in Magnetic Tunnel Junctions and Spin-Transfer Torque Random Access Memory. *Journal of Physics:Condensed Matter*, 19:165209, 2007.

[18] 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.

[19] D. DiTomaso, R. Morris, AK. 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*, 21:2141–2154, 2013.

[20] Xiangyu Dong, Cong Xu, Yuan Xie, and Norman P. Jouppi. NVSim: A Circuit-Level Performance, Energy, and Area Model for Emerging Nonvolatile Memory. In *IEEE Transactions on CAD*, 2012.

[21] 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.

[22] Chris Fallin, Chris Craik, and Onur Mutlu. CHIPPER: A Low-complexity Bufferless Deflection Router. In *Proceedings of HPCA*, 2011.

[23] M Galles. Scalable Pipelined Interconnect for Distributed Endpoint Routing: The SGI SPIDER Chip. In *Proceedings of Hot Interconnect 4*, 2009.

[24] Nilanjan Goswami, Bingyi Cao, and Tao Li. Power-performance Co-optimization of Throughput Core Architecture using Resistive Memory. In *Proceedings of HPCA*, 2013.

[25] Nilanjan Goswami, Ramkumar Shankar, Madhura Joshi, and Tao Li. Exploring GPGPU Workloads: Characterization Methodology, Analysis and Microarchitecture Evaluation Implications. In *Proceedings of IISWC*, 2010.

[26] Xiaochen Guo, Engin Ipek, and Tolga Soyata. Resistive Computation: Avoiding the Power Wall with Low-Leakage, STT-MRAM Based Computing. In *Proceedings of ISCA*, 2010.

[27] David Harris. *Skew-Tolerant Circuit Design*. Morgan Kaufmann, 2000.

[28] Mitchell Hayenga, Natalie Enright Jerger, and Mikko Lipasti. SCARAB: A Single Cycle Adaptive Routing and Bufferless Network. In *Proceedings of MICRO*, 2009.

[29] Bingsheng He, Wenbin Fang, and Qiong Luo. Mars: A MapReduce Framework on Graphics Processors. In *Proceedings of PACT*, 2008.

[30] Joel Hestness, Boris Grot, and Stephen W. Keckler. Netrace: Dependency-Driven Trace-Based Network-on-Chip Simulation. In *Proceedings of NoCArc*, 2010.

[31] Yatin Hoskote, Sriram Vangal, Arvind Singh, Nitin Borkar, and Shekhar Borkar. A 5-GHz Mesh Interconnect for a Teraflops Processor. *IEEE Micro*, 27:51–61, 2007.

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

[33] Hyunjun Jang, Baik Song An, Nikhil Kulkarni, Ki Hwan Yum, and Eun Jung Kim. A Hybrid Buffer Design with STT-MRAM for On-Chip Interconnects. In *Proceedings of NOCS*, 2012.

[34] Adwait Jog, Asit K Mishra, Cong Xu, Yuan Xie, N. Vijaykrishnan, Ravishankar Iyer, and Chita R. Das. Cache Revive: Architecting Volatile STT-RAM Caches for Enhanced Performance in CMPs. Technical Report CSE-11-010, The Pennsylvania State University CSE Dept., June 2011.

[35] Andrew B. Kahng, Bin Li, Li-Shiuan Peh, and Kambiz Samadi. ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration. In *Proceedings of DATE*, 2009.

[36] Onur Kayiran, Adwait Jog, Mahmut T. Kandemir, and Chita R. Das. Neither More Nor Less: Optimizing Thread-level Parallelism for GPGPUs. In *Proceedings of PACT*, 2013.

[37] Hanjoon Kim, John Kim, Woong Seo, Yeongon Cho, and Soojung Ryu. Providing Cost-effective On-Chip Network Bandwidth in GPGPUs. In *Proceedings of ICCD*, 2012.

[38] John Kim, James Balfour, and William Dally. Flattened Butterfly Topology for On-Chip Networks. In *Proceedings of MICRO*, 2007.

[39] Gurhan Kucuk, Dmitry Ponomarev, Oguz Ergin, and Kanad Ghose. Reducing Reorder Buffer Complexity Through Selective Operand Caching. In *Proceedings of ISLPED*, 2003.

[40] Gurhan Kucuk, Dmitry Ponomarev, and Kanad Ghose. Low- Complexity Reorder Buffer Architecture. In *Proceedings of ICS*, 2002.

[41] A Kumar, Li-Shiuan Peh, and N.K. Jha. Token Flow Control. In *Proceedings of MICRO*, 2008.

[42] Benjamin C. Lee, Engin Ipek, Onur Mutlu, and Doug Burger. Architecting Phase Change Memory as a Scalable DRAM Alternative. In *Proceedings of ISCA*, 2009.

[43] Jaekyu Lee, Si Li, Hyesoon Kim, and Sudhakar Yalamanchili. Adaptive virtual channel partitioning for network-on-chip in heterogeneous architectures. *ACM Trans. Des. Autom. Electron. Syst.*, 18:48:1–48:28, 2013.

[44] Luis A. Lozano C. and Guang R. Gao. Exploiting Short-Lived Variables in Superscalar Processors. In *Proceedings of MICRO*, 1995.

[45] Sheng Ma, Natalie Enright Jerger, and Zhiying Wang. Whole Packet Forwarding: Efficient Design of Fully Adaptive Routing Algorithms for Networks-on-chip. In *Proceedings of HPCA*, 2012.

[46] P. S. Magnusson, M. Christensson, J. Eskilson, D. Forsgren, G. Hallberg, J. Hogberg, F. Larsson, A. Moestedt, B. Werner, and B. Werner. Simics: A Full System Simulation Platform. *Computer*, 35(2):50–58, 2002.

[47] Asit K. Mishra, Xiangyu Dong, Guangyu Sun, Yuan Xie, N. Vijaykrishnan, and Chita R. Das. Architecting On-Chip Interconnects for Stacked 3D STT-RAM Caches in CMPs. In *Proceedings of ISCA*, 2011.

[48] Thomas Moscibroda and Onur Mutlu. A Case for Bufferless Routing in On-chip Networks. In *Proceedings of ISCA*, 2009.

[49] Helia Naeimi, Charles Augustine, Arijit Raychowdhury, Shih-Lien Lu, and James Tschanz. STTRAM Scaling and Retention Failure. *Intel Technology Journal*, 17, 2013.

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

[51] Li-Shiuan Peh and William J. Dally. A Delay Model and Speculative Architecture for Pipelined Routers. In *Proceedings of HPCA*, 2001.

[52] Moinuddin K. Qureshi, Michele M. Franceschini, and Luis A. Lastras-montao. Improving Read Performance of Phase Change Memories via Write Cancellation and Write Pausing. In *Proceedings of HPCA*, 2010.

[53] Moinuddin K. Qureshi, Vijayalakshmi Srinivasan, and Jude A. Rivers. Scalable High Performance Main Memory System Using Phase-Change Memory Technology. In *Proceedings of ISCA*, 2009.

[54] Steven E. Raasch, Nathan L. Binkert, and Steven K. Reinhardt. A Scalable Instruction Queue Design Using Dependence Chains. In *Proceedings of ISCA*, 2002.

[55] Arijit Raychowdhury, Dinesh Somasekhar, and Tanay Karnik. Design Space and Scalability Exploration of 1T-1STT MTJ Memory Arrays in the Presence of Variability and Disturbances. In *Proceedings of IEDM*, 2009.

[56] 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*, 80:2335, 2002.

[57] S.C.Woo, M.Ohara, E.Torrie, J.P.Singh, and A.Gupta. The SPLASH-2 Programs: Characterization and Methodological Considerations. In *Proceedings of ISCA*, 1995.

[58] Daeho Seo, Akif Ali, Won-Taek Lim, Nauman Rafique, and Mithuna Thottethodi. Near-Optimal Worst-Case Throughput Routing for Two-Dimensional Mesh Networks. In *Proceedings of ISCA*, 2005.

[59] Clinton W. Smullen, Vidyabhushan Mohan, Anurag Nigam, Sudhanva Gurumurthi, and Mircea R. Stan. Relaxing Non-Volatility for Fast and Energy-Efficient STT-RAM Caches. In *Proceedings of HPCA*, 2011.

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

2006.

[61] Guangyu Sun, Xiangyu Dong, Yuan Xie, Jian Li, and Yiran Chen. A Novel Architecture of the 3D Stacked MRAM L2 Cache for CMPs. In *Proceedings of HPCA*, 2009.

[62] Zhenyu Sun, Xiuyuan Bi, Hai (Helen) Li, Weng-Fai Wong, Zhong-Liang Ong, Xiaochun Zhu, and Wenqing Wu. Multi Retention Level STT-RAM Cache Designs with a Dynamic Refresh Scheme. In *Proceedings of MICRO*, 2011.

[63] Karthik Swaminathany, Ravindhiran Mukundrajany, Niranjan Soundararajan, and Vijaykrishnan Narayanan. Towards Resilient Micro-Architectures: Datapath Reliability Enhancement using STT-MRAM. In *Proceedings of ISVLSI*, 2011.

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

[65] Hangsheng Wang, Li-Shiuan Peh, and Sharad Malik. Power-driven Design of Router Microarchitectures in On-chip Networks. In *Proceedings of MICRO*, 2003.

[66] Zhe Wang, Daniel A. Jimenez, Cong Xu, Guangyu Sun, and Yuan Xie. Adaptive Placement and Migration Policy for an STT-RAM-Based Hybrid Cache. In *Proceedings of HPCA*, 2014.

[67] David Wentzlaff, Patrick Griffin, Henry Hoffmann, Liewei Bao, Bruce Edwards, Carl Ramey, Matthew Mattina, Chyi-Chang Miao, John F. Brown III, and Anant Agarwal. On-Chip Interconnection Architecture of the Tile Processor. *IEEE Micro*, 27:15–31, 2007.

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

[69] Xiaoxia Wu, Jian Li, Lixin Zhang, Evan Speight, Ram Rajamony, and Yuan Xie. Hybrid Cache Architecture with Disparate Memory Technologies. In *Proceedings of ISCA*, 2009.

[70] Alexandre V. Yakovlev, Albert M. Koelmans, and Luciano Lavagno. High-Level Modeling and Design of Asynchronous Interface Logic. *IEEE Design and Test of Computers*, 12:32–40, 1995.

[71] George L. Yuan, Ali Bakhoda, and Tor M. Aamodt. Complexity Effective Memory Access Scheduling for Many-core Accelerator Architectures. In *Proceedings of MICRO*, 2009.

[72] Ping Zhou, Yu Du, Youtao Zhang, and Jun Yang. Fine-Grained QoS Scheduling for PCM-based Main Memory Systems. In *Proceedings of IPDPS*, 2010.

[73] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. A Durable and Energy Efficient Main Memory Using Phase Change Memory Technology. In *Proceedings of ISCA*, 2009.

[74] Ping Zhou, Bo Zhao, Jun Yang, and Youtao Zhang. Energy Reduction for STT-RAM Using Early Write Termination. In *Proceedings of ICCAD*, 2009.