DISSERTATION

RELAX: CROSS-LAYER RESOURCE MANAGEMENT FOR RELIABLE NOC-BASED 2D AND 3D MANYCORE ARCHITECTURES IN THE DARK SILICON ERA

Submitted by

Venkata Yaswanth Raparti

Department of Electrical and Computer Engineering

In partial fulfillment of the requirements

For the Degree of Doctor of Philosophy

Colorado State University

Fort Collins, Colorado

Fall 2019

Doctoral Committee:

Advisor: Sudeep Pasricha

Anura Jayasumana
Wim Bohm
Ryan Kim

ABSTRACT

RELAX: CROSS-LAYER RESOURCE MANAGEMENT FOR RELIABLE NOC-BASED 2D AND 3D MANYCORE ARCHITECTURES IN THE DARK SILICON ERA

Emerging 2D and 3D chip-multiprocessors (CMPs) are facing numerous challenges due to technology scaling that impact their reliability, power dissipation, performance, and security. With growing parallelism in applications and the increasing core counts, traditional resource management frameworks and critical on-chip components such as networks-on-chip (NoC) and memory controllers (MCs) do not scale well to efficiently cope with this new and complex design space of CMP design. Several phenomena are affecting the reliability of CMPs. For instance, device-level phenomena such as (Bias Temperature Instability) BTI and (Electro Migration) EM lead to permanent faults due to aging in CMOS logic and memory cells in computing cores and NoC routers of CMPs. Simultaneously, alpha particle strikes (soft errors) and power supply noise (PSN) impacts lead to transient faults across CMP components. There have been several attempts to address these challenges at the circuit and micro-architectural levels, such as guard-banding and over-provisioning of resources to the CMP. However, with increasing complexity in the architecture of today's CMPs, mechanisms to overcome these challenges at the circuit and micro-architectural levels alone, incur large overheads in power and performance. Hence, there is a need for a system-level solution that utilizes control knobs from different layers and manages the CMP reliability in runtime to efficiently minimize the adverse effects of these failure mechanisms while meeting performance and power constraints.

Network-on-chip (NoC) has become the defacto communication fabric in CMP architectures. There are different types of NoC topologies and architectures that are tailored for different CMP platforms based on their communication demands. The most used topology is 2D/3D mesh-based NoC with a deadlock-free turn-model based routing scheme as it has demonstrated to be scaling well with the increasing core count. However, with unprecedented reliability and security challenges in CMP designed at the sub-nanometer technology node, the basic turn-model routing is proved to be inefficient to provide seamless communication between cores and other on-chip components. This demands for a more reliable NoC solution in 2D, and 3D CMPs.

Another critical criterion while designing a CMP is NoC throughput and power consumption in CMPs with integrated manycore accelerators. Manycore accelerator platforms operate on thousands of threads with hundreds of thread blocks executing several kernels simultaneously. The core-to-memory data generated in accelerators is very high compared to a traditional CPU processor. This leads to congestion at memory controllers that demands a high bandwidth NoC with high power and area overheads, which is not scalable as a number of cores in the accelerator increases. High volumes of read reply data in manycore accelerator platforms necessitate intelligent memory scheduling along with low latency NoC to resolve the memory bottleneck issue. Mechanisms to overcome these challenges require complex architectures across CMP interconnection fabric that are designed and integrated at various global locations. Unfortunately, such global fabrication of CMP processors makes them vulnerable to security threats due to hardware Trojans that may be inserted in third-party (3PIP) NoCs.

We address these issues by designing a cross-layer resource management framework called *RELAX* that enhances performance and security of NoC-based 2D and 3D CMPs, while meeting a

diverse set of platform constraints related to the lifetime of the CMP, dark silicon power, fault tolerance, thermal and real-time application performance. At the OS-level, we have developed several techniques such as lifetime aware application mapping heuristic, adaptive application degree of parallelism (DoP), slack aware checkpointing, and aging aware NoC path allocation. At the system level, we propose dynamic voltage scheduling (DVS), and a low power checkpointing mechanism to meet the dark silicon power and application deadline constraints. At the architectural level, we introduce several novel upgrades to the architectures of NoC routers, memory controllers (MCs), and network interfaces (NIs) to improve the performance of NoC-based CMPs while minimizing the power dissipation and mitigating security threats from hardware Trojans.

ACKNOWLEDGEMENTS

I would like to thank all the individuals whose encouragement and support have made the completion of this thesis possible.

First and foremost, I would like to express my sincere gratitude to my advisor, Prof. Sudeep Pasricha, who has guided me through the process of doctoral study with his insightful and valuable advice. It was only with his encouragement and motivation I was able to explore some of the complex yet exciting problems in manycore processor design optimization. He ensured that I improve my attention to detail while conducting research, which benefited me in both academic and non-academic activities. His devotion to both my research and my personal development were invaluable to me throughout my doctoral studies. He nurtured the analytical mindset and gave sufficient time and opportunities to realize my true potential in accomplishing my Ph.D. goals. I appreciate all the help, guidance, and inspiration I received from Prof. Pasricha, who made it possible for me to survive the trial of graduate school with unforgettable memories and broadened horizons.

I would like to take this opportunity to thank the respected members of my Ph.D. committee, Prof. Anura Jayasumana, Prof. Ryan Kim, and Prof. Wim Bohm. Their feedback helped me to rediscover my research and refine my work from different perspectives. I also much appreciate all the help I received from my mentors at Micron Technology, Netapp, and Samsung — Dr. Jiangli Zhu, Dharmesh Thakkar, and Vinod Muthu — for their feedback in helping me shape my doctoral studies and my career.

Furthermore, my special thanks to my research partner Dr. Nishit Kapadia, whose collaboration helped me understand the research challenges in my initial days of doctoral studies by gaining valuable insights on circuit and power delivery network (PDN) aging. A big thanks to my dear friends Dr. Sai Vineel Reddy Chittamuru, Dr. Ishan Thakkar, and Dr. Daniel Dauwe for their support and cooperation through intellectually stimulating conversations that boosted my morale while solving complex research problems. This list cannot be complete without mentioning the company and the help from my current and former lab mates in Prof. Pasricha's EPIC lab: Vipin Kumar Kukkala, Saideep Tiku, Yi Xiang, Yong Zhou, Ninad Hogade, Shoumik Maiti, Febin Sunny, Asif Anwar Baig Mirza, and Kamil Khan.

I am blessed to have a wonderful family – my father Sankar Rao Raparti, my mother Rama Raparti, my brother Pradyumna, and my in-laws – for their support that empowered me to pursue my Ph.D. Their generosity and humility have made me continually strive to be a better person. Last but not least, I would like to thank my wife, Pavithra Raghavan for staying by me amidst my stressful rants and long all-nighters. She made sure that I am well fed and made me feel home away from home. I am highly indebted to her love.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

LIST OF RESEARCH PUBLICATIONS

JOURNAL PUBLICATIONS:

- V.Y. Raparti, N. Kapadia, S. Pasricha, “ARTEMIS: An Aging-Aware Runtime Application Mapping Framework for 3D NoC-based Chip Multiprocessors”, **IEEE Transactions on Multi-Scale Computing Systems (TMSCS), 2017.** ***(Selected as Featured Paper for Apr-Jun 2017 issue)***
- V.Y. Raparti, S. Pasricha, “RAPID: Memory-Aware NoC for Latency Optimized GPGPU Architectures,” **IEEE Transactions on Multi-Scale Computing Systems (TMSCS), 2018**
- V.Y. Raparti, S. Pasricha, "Approximate NoC and Memory Controller Architectures for GPGPU Accelerators," **IEEE Transactions of Parallel and Distributed Systems (TPDS)** (*under review*)**, 2019.**

CONFERENCE PUBLICATIONS:

- N. Kapadia, V.Y. Raparti, S. Pasricha, “ARTEMIS: An Aging-Aware Run-Time Application Mapping Framework for 3D NoC based Chip Multiprocessors,” **IEEE/ACM International Symposium on Networks-on-Chip (NOCS), 2015.**
- V.Y. Raparti, S. Pasricha, “A Cross-Layer Runtime Framework for Checkpoint-based Soft-Error and Aging Management in SoCs,” **SRC Techcon, Sep 2016.**

- V.Y. Raparti, S. Pasricha, “CHARM: A Checkpoint-based Resource Management Framework for Reliable Multicore Computing in the Dark Silicon Era,” **IEEE International Conference on Computer Design (ICCD), Oct 2016.**
- V.Y. Raparti, S. Pasricha, “PARM: Power Supply Noise Aware Resource Management for NoC based Multicore Systems in the Dark Silicon Era,” **IEEE/ACM Design Automation Conference (DAC), San Francisco, CA, Jun. 2018.**
- V.Y. Raparti, S. Pasricha, “Memory-Aware Circuit Overlay NoCs for Latency Optimized GPGPU Architectures,” **IEEE International Symposium on Quality Electronic Design (ISQED), Mar. 2016.**
- V.Y. Raparti, S. Pasricha, “DAPPER: Data Aware Approximate NoC for GPGPU Architectures,” to appear, **IEEE/ACM International Symposium on Networks-on-Chip (NOCS), Torino, Italy, Oct 2018.** ***(Best Paper Award)***
- V.Y. Raparti, S. Pasricha, “Lightweight Mitigation of Hardware Trojan Attacks in NoC-based Manycore Computing,” **IEEE/ACM Design Automation Conference (DAC), Las Vegas, NV, USA, Jun. 2019**

1. INTRODUCTION

This chapter outlines the challenges of modern-day manycore chip multiprocessor (CMP) design and the necessity to address these challenges at different levels of abstraction using runtime resource management techniques, as well as novel circuit and micro-architectural innovations in networks-on-chip (NoC) and memory controllers (MCs). This chapter also gives a general overview of the contributions of this dissertation.

1.1. MOTIVATION FOR NOC-BASED CMP DESIGN

With technology scaling, there is an increasing availability of resources on electronic chips to support extensive parallel computing. The growth in on-chip resources is facilitated by the rising transistor density on-chip with each new generation as shown in **Figure 1**.

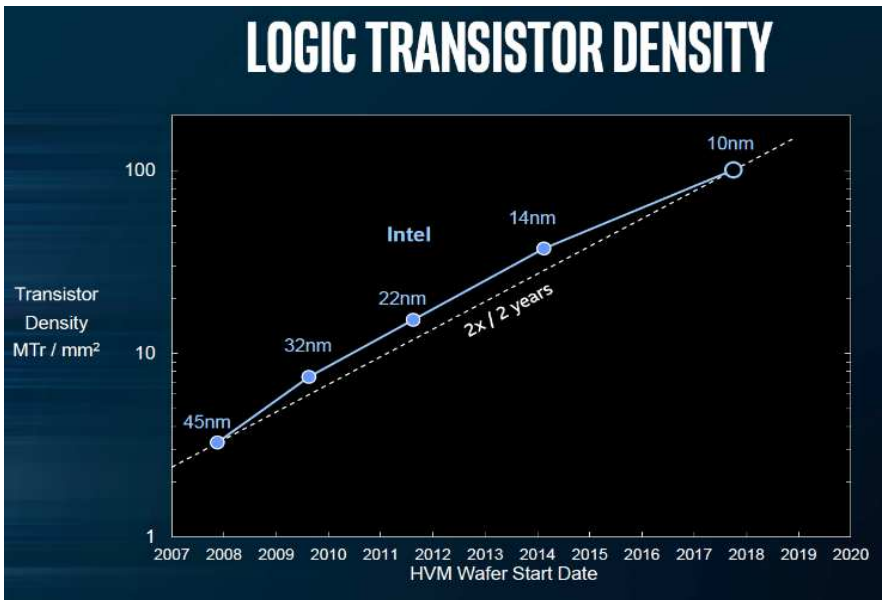


Figure 1 Moore's law indicating growth in transistor density with technology scaling [1]

Design engineers have attempted to extract higher instruction-level parallelism (ILP) by modifying core micro-architectures with ILP-driven enhancements and increasing the on-chip cache memory sizes. However, such complex single-core processors yield only 40% increase in performance for every 50% increase in circuit area [2]. This poor yield has led to the era of manycore computing where the emphasis is on using multiple small processing elements (cores) with dedicated memories to execute in parallel and deliver 70-80% higher performance for every 50% increase in circuit area [2]. The advent of such manycore processors (i.e., CMPs) has resulted in a growing interest among researchers and engineers to leverage the parallelism by developing special hardware architectures with hundreds to thousands of cores, such as general purpose parallel accelerators and neural processors [3], as shown in Figure 2. Each core in a CMP has a private and shared memory to cache the data coming from main memory for computation. Software frameworks such as NVIDIA CUDA [4], and OpenCL [5] and OpenMP [6] have been developed to facilitate the parallelization of applications that leverage the available CMP resources for faster execution.

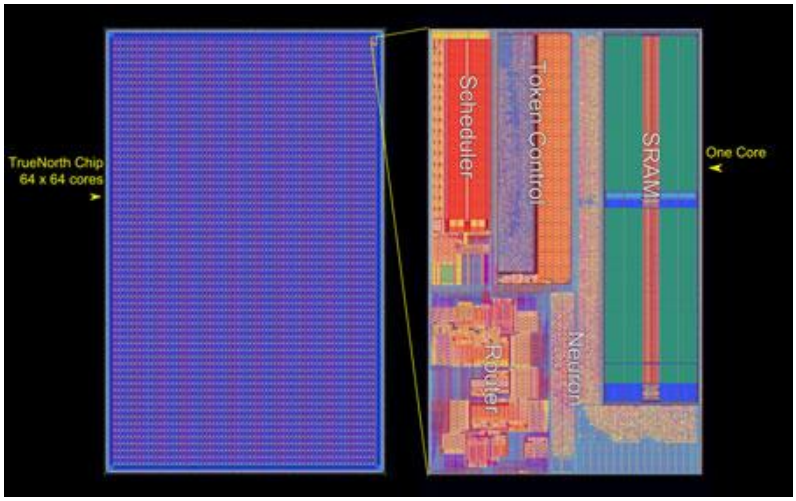


Figure 2 IBM TrueNorth DNN Chip with 4096 cores [3]

However, there are two main challenges associated with the efficient execution of parallel applications on emerging chip multiprocessors (CMPs) with billions of transistors and hundreds of cores and other on-chip components.

Firstly, with the continued scaling down in dimensions and the rise in density of transistors, the logic and memory cells of a manycore processor are prone to faults due to thermal hotspots, rapid wear out due to aging, increased soft error rate, and voltage fluctuations, as a result of inherent non-ideal physical phenomena in fabricated transistors. *As CMPs become ubiquitous in the domains of defense, aerospace, healthcare, and consumer electronics, it is crucial to provide a reliable computing platform that can guarantee application quality of service (QoS), even in the presence of various types of faults.*

Secondly, the inherently parallel nature of today's applications, together with their ever-increasing core count results in high core-to-core and core-to-memory traffic. In the era of cloud computing with rigid service level agreements (SLAs) between clients and original equipment manufacturers (OEMs), clients cannot tolerate a slowdown in performance and downtime of resources that are guaranteed in the SLA. CMPs are also shared by multiple client applications with slices of compute and memory resources are distributed across different clients according to their compute and I/O requirements. The rise in data-parallel applications that utilize shared on-chip resources demands CMPs with high bisectional bandwidth to support the seamless multi-application execution. Bus based on-chip communication fabrics fail to meet such bandwidth requirements. Hence CMPs are integrated with networks-on-chip (NoCs) to improve their bisectional bandwidths (e.g. IBM TrueNorth, STMicroelectronics STHORM [7], [8]). But traditional 2D mesh-based NoC architectures [9] that are used in these chips do not offer high throughput for memory-intensive applications which limits the performance of CMPs with

integrated GPGPU accelerators. Such bandwidth challenges require new NoC and MC architectures that can cope with high bandwidth and low latency demands [10] together with intelligent scheduling of application tasks on the cores. CMPs should also ensure the security of critical application data with the integration of third party (3PIP) NoCs for on-chip communication. *This compels CMPs to have on-chip communication fabrics that are tailor-made for different traffic patterns as well as throughput and security requirements while consuming low overall CMP power and area overheads.*

1.2. RELIABILITY CHALLENGES IN CMP DESIGN

With shrinking transistor sizes, the reliability and lifetime of CMPs are adversely affected by a plethora of device-level challenges. In the following sub-sections, we briefly discuss the significant phenomena that limit the reliability of CMPs.

1.2.1 AGING IN CIRCUITS AND POWER DELIVERY NETWORK

Aging in circuits due to phenomena such as bias temperature instability (BTI) [11], hot carrier injection (HCI) [12], and electromigration (EM) [13] lead to a slowdown or permanent faults in the logic circuits and power delivery network of CMPs. The effect of aging is observed as a rise in CMOS transistor threshold voltage (V_T), which results in a higher delay of logic gates that are in the critical path of the circuit. Figure 3 shows the illustration of BTI and HCI phenomena on a CMOS inverter under stress-destress cycles and varied switching activities. BTI degradation is directly proportional to the duration of stress window, and HCI is directly proportional to the switching activity of the transistor.

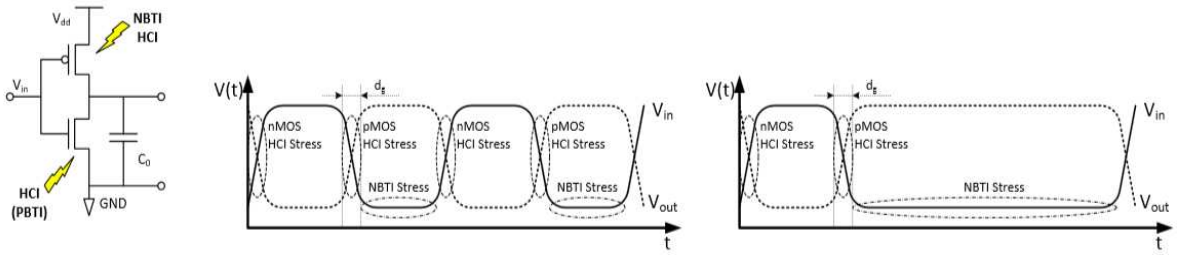


Figure 3 Time windows of hot carrier injection (HCI) and bias temperature instability (BTI) in a CMOS inverter [14]

Electromigration (EM) is another device level phenomenon that leads to a rise in resistance of interconnect and power delivery network over time, which results in open or short circuits. EM is the phenomenon that is most dominant in wires that carry larger unidirectional currents such as those in the power delivery network (PDN) [15]. Figure 4 shows the effect of EM over time in a copper wire. High current flow causes the internal electric field to knock off ions in copper wires and replace the atoms with voids that increase the resistance of the wire and over time causes an open circuit. In some cases, the knocked off atoms get attracted to the nearby conducting wires leading to closed circuits. The increase in resistance in the PDN leads to a degradation in supply voltage (V_{dd}) [16] which leads to further degradation of performance. With aggressive technology scaling, the impact of BTI, HCI, and EM on circuit and PDN aging are exacerbated leading to premature chip failure if no proper measures are taken during design and runtime.

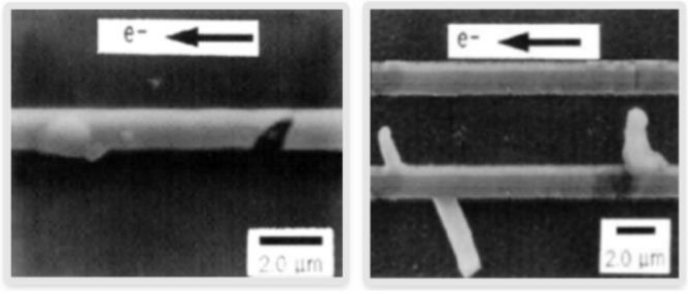


Figure 4 Creation of void (open circuit) and hillock (closed circuit) by electromigration in Cu wire [17]

1.2.2 SOFT ERRORS

Soft errors are single event upsets (SEUs) and single event transients (SETs) that are caused due to the interaction of high energy particles such as protons, neutrons, and alpha particles or heavy ions with logic gates or memory cells. When an alpha particle strikes a PN-junction, as shown in Figure 5, it produces a funnel of electron-hole pairs that cause a current pulse that leads to a single or multiple bit flip event (soft error). With scaling down of technology nodes, the critical charge required for single or multiple bit flips is reduced, leading to higher soft error rates (SERs) [18]. Studies also showed that soft error rate (SER) is inversely proportional to the supply voltage (V_{dd}) as low supply voltage further lowers the critical charge required for SEUs [19]. SETs will become a persistent problem with chips that are designed to operate at high frequency, as the probability that the glitch due to an alpha particle strike propagates to a storage element is higher at a higher clock speed [20]. However, commercial CMPs are designed to operate at low V_{dd} and high frequency to gain the best power performance yield from the chip. Designing purely for performance results in CMPs that are prone to high soft error rates (SERs).

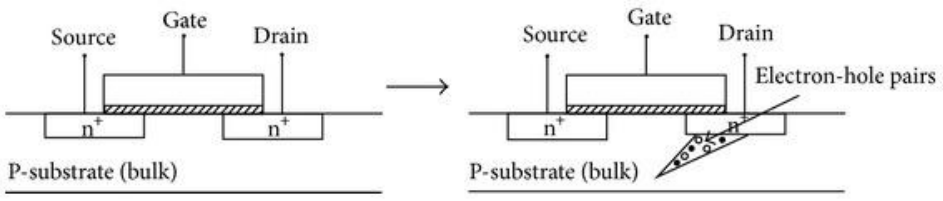


Figure 5 Illustration of alpha particle strike on an NMOS transistor [21]

1.2.3 POWER SUPPLY NOISE

With shrinking transistor size, circuits are designed to run at a higher operating clock frequency to extract the best performance out of CMPs. However, components (e.g., cores) that operate at high clock frequency also introduce noise in the V_{dd} and ground power lines, which is

called Power Supply Noise (PSN) [22]. PSN is caused by the resistive drop (IR) and inductive droop ($L \times \Delta I/\Delta t$) in V_{dd} and ground power lines as shown in Figure 6 that is proportional to the resistivity and switching activity of the wires supplying current to the circuits. PSN leads to voltage emergencies (VEs) that, if left unchecked, can change the outcomes of applications executed on CMPs designed at sub-10nm technology. The occurrence of VEs is even more severe in cores and NoC routers that operate in near-threshold voltage (V_T) regions in low power computing platforms.

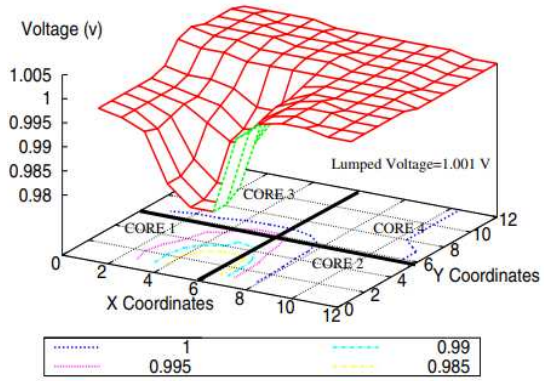


Figure 6 Illustration of voltage droop at core 1 due to switching activity in the neighboring cores in a 4-core NoC-based CMP [23]

1.2.4 DARK SILICON

The execution of parallel applications generates thermal-hotspots due to the higher transistor packing density of modern-day CMPs even with lower dynamic per-transistor power dissipation. In particular, some applications with compute-intensive threads lead to hotspots, as shown in Figure 7. These thermal hotspots exacerbate aging in circuits and PDNs, leading to shortening of the lifespan of the CMP. Hence, CMPs require an efficient cooling mechanism that draws additional power to bring down the on-chip temperature. A CMP that generates high temperature also draws high power for the cooling mechanism and increases the operational cost of the data

center that employs it. Hence, there is a need to limit the amount of heat generated by CMPs to lower the overall operational cost and increase the lifetime of the CMP. A thermal design power constraint (TDP) is often set to overcome this thermal challenge. In most CMPs today, at any given time, large sections of the chip remain inactive not to exceed this TDP. This phenomenon, called dark-silicon (because a part of the silicon chip must remain off or "dark" to meet TDP constraints), is growing as technology scales. By placing limits on how many components get activated at any given time, dark silicon constraints help cope with the power challenge but require new approaches to ensure performance quality of service (QoS) guarantees.

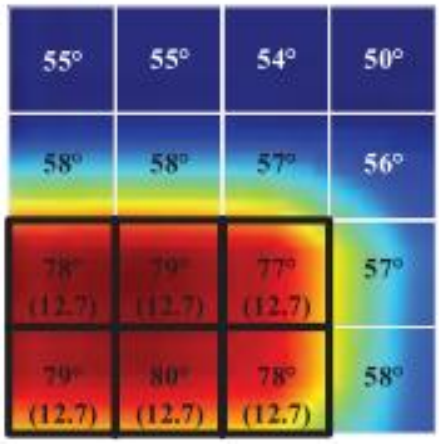


Figure 7 Thermal hotspots in a 16-core CMP due to a compute-intensive workload with 6 tasks running on adjacent cores (on the bottom left) [24]

1.3. MEMORY BOTTLENECK PROBLEM IN MANYCORE ACCELERATORS

Today's CMPs are frequently integrated with general-purpose GPU accelerators (GPGPUs) on the same die or as extension cards that can be accessed via interfaces such as PCIe to enable execution of data-parallel applications in the domains of machine learning, big data, and pattern recognition [4]. GPGPUs are utilized in tandem with CPUs to extract performance out of a CMP that executes emerging applications. However, the growing parallelism in manycore GPGPUs

leads to high communication traffic between cores and main memory (DRAM). Traditional 2D mesh-based NoCs cannot handle the traffic conditions of GPGPUs which have *many-to-few* traffic pattern in the request packets, and *few-to-many* pattern in the reply packets, as shown in Figure 8. Such skewed on-chip traffic in GPGPUs leads to long wait times for reply data in the memory controller (MC) that is connected to DRAM. This phenomenon is referred to as the memory bottleneck problem [10].

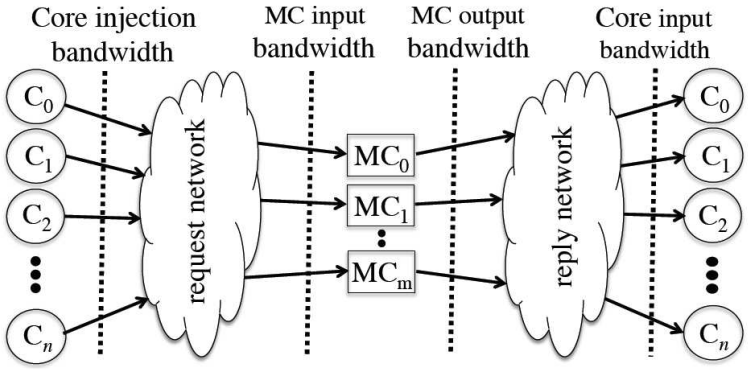


Figure 8 Many-to-few and Few-to-many on-chip traffic in GPGPUs. C nodes are Cores, and MC nodes are memory controllers [25]

There is an urgent need for novel NoC and MC architectures to resolve this memory bottleneck problem in manycore accelerators as confirmed by prior work [10] [26]. However, NoCs in GPGPUs incurs high power and area overheads unlike NoCs in CPUs, due to the higher bandwidth requirements in GPGPUs. In deep submicron technology nodes with higher transistor density, having a power-hungry NoC may violate the dark silicon power budget of the CMP, making the manycore accelerators inefficient. *Thus, there is a need to innovate and design new energy-efficient NoC and MC architectures to extract maximum QoS out of manycore GPGPU accelerators in CMPs.*

1.4. SNOOPING ATTACKS IN 3PIP NOC BASED CMPS

Lastly, today's CMPs have hundreds of components that are designed, integrated, and fabricated at global facilities. Figure 9 illustrates the different stages involved in the design cycle of a system on chip (SoC). In the first stage, the specifications and the functional design aspects of an SoC are finalized. In the second stage, design engineers develop register transfer level (RTL) intellectual property (IP) blocks that are combined with the IPs of third party vendors and sent to the SoC design integration (third) stage. In the third stage, all the IP blocks are integrated and validated for functional correctness and sent to the manufacturing and post-silicon testing (fourth) stage. The third and the fourth stages are repeated for several iterations until the chip matches the specifications, and finally the chip is released for packaging and deployment (final stage). With growing complexity in SoC design, designers are opting for third party (3PIP) NoCs, e.g., NoC architectures from Arteris [27]. These 3PIP NoCs are ideal candidates for inserting hardware Trojans [28] inside the trusted hardware at the integration and fabrication stages of the IC design, as shown in Figure 9, to carry out security attacks through an untrusted accomplice malicious software. These hardware Trojans are malicious circuits that can be used to execute different types of attacks such as denial-of-service attacks (DoS), side-channel attacks on shared resources, or data snooping attacks. Especially with the cutthroat competition in the semiconductor industry that is backed by geopolitical rivalries, there is a growing utilization of hardware Trojans (HTs) in consumer electronic devices to spy on classified information [29] of the rival nation.

A secure network-on-chip (NoC) is crucial for safeguarding sensitive data of parallel applications from leaking to malicious agents. However, traditional security enhancement techniques, such as key-based encryption and decryption mechanisms increase NoC latency and power consumption [30]. *Hence, in CMPs, there is a critical need for unique lightweight security*

enhancement mechanisms that safeguard the applications executing on the CMP from hardware Trojan attacks, without sacrificing overall application performance.

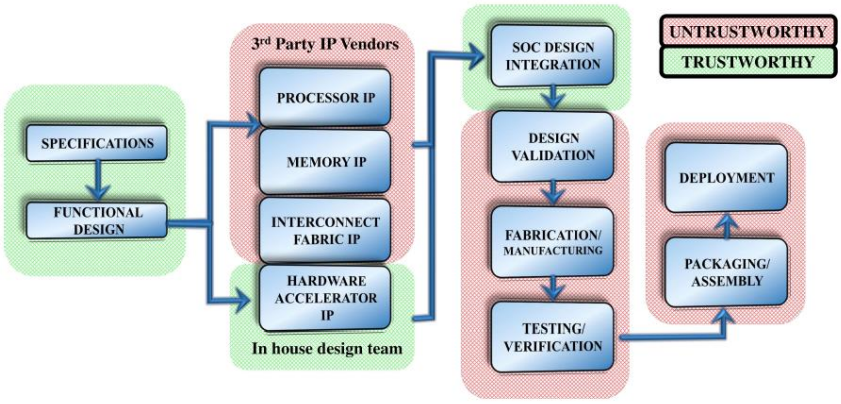


Figure 9 Illustration of a system on chip (SoC) design flow with trusted and untrusted stages

1.5. DISSERTATION OVERVIEW

In summary, there is a crucial need for a holistic framework in CMPs that can cope with the conflicting nature of CMP design goals and challenges. Such a framework is not easy to conceptualize because of the complex inter-dependencies between design decisions, constraints, and optimization goals. For example, to enhance application performance, designers may run parallel applications at a high degree of parallelism (DoP; i.e., with more threads), which leads to an increase in transient faults due to soft errors. Such execution needs to occur at a high V_{dd} to minimize transient faults. However, this leads to faster chip aging (reduced lifetime), and pressures on the dark-silicon power budget that may end up ultimately reducing application throughput on the CMP. To address the above-mentioned problems, the main contribution of this dissertation is the design of a novel cross-layer resource management framework (*RELAX*) that enhances the performance of NoC-based 2D and 3D CMPs, while meeting a diverse set of platform constraints (e.g., dark silicon, reliability, thermal, security, and application execution deadlines).

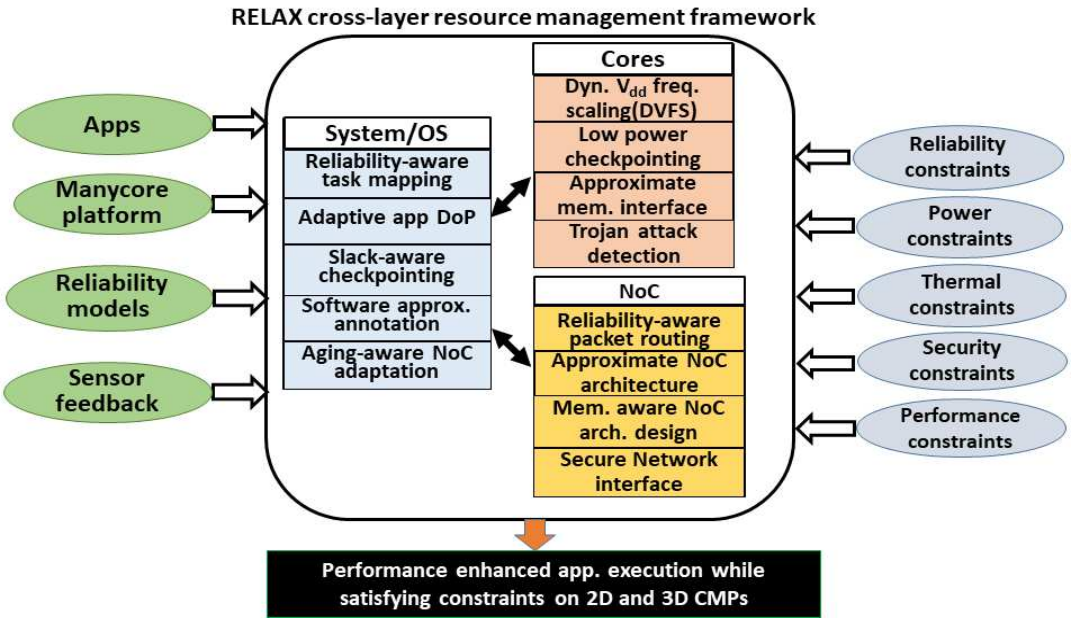


Figure 10 Overview of RELAX: A cross-layer resource management framework for secure, reliable, and performance optimized NoC-based CMP

Figure 10 shows a high level overview of the *RELAX* framework with our published contributions describing various facets of this framework in [31] [32] [33] [34] [35] [36] [37] [38] [39]. *RELAX* is a cross layer framework that combines system/OS level, core level, and NoC and memory interface level innovations towards the design of a NoC based CMP. The *RELAX* framework takes inputs from application metadata, platform performance counters, reliability models, and on-chip sensors to enhance the application performance on 2D and 3D CMPs while satisfying system level and chip level constraints such as (lifetime, power budget, thermal, security, and application deadlines). Each layer receives runtime feedback from other layers and together they jointly enhance the CMP performance without violating the chip-level and system-level constraints. The rest of this dissertation is organized as follows:

In chapter 2, we address the reliability challenges due to aging in 3D NoC based CMPs. We propose a novel aging-aware resource management framework called *ARTEMIS* [31] that

maximizes the work performed over the lifetime of the CMP. In *ARTEMIS*, we propose a runtime application mapping and DVS-scheduling framework that can adapt to different aging scenarios to extend the lifetime of a 3D NoC-based CMP. *ARTEMIS* considers aging in both circuits (V_T-degradation) and power delivery network (IR-drop), unlike any other prior work on application scheduling in CMPs. Furthermore, *ARTEMIS* also does a novel aging enabled NoC routing path allocation to produce a balanced core-router aging profile to extend the lifetime of the NoC. In addition, this framework also meets chip-wide power constraints, thus finding applicability in contemporary power-constrained CMP designs.

In chapter 3, we address the challenge of meeting application quality of service (QoS) in the presence of permanent faults due to aging and transient faults due to bit flips in 2D NoC based CMPs. As explained at the beginning of this section, it is challenging to minimize aging and mitigate soft errors simultaneously as their respective solutions have counter-productive effects on each other. Hence, we propose a novel low power checkpointing based runtime framework called *CHARM* [32] to meet the target lifetime of a CMP in the presence of soft-errors and application deadline constraints. *CHARM* dynamically manages application degree of parallelism (DoP), V_{dd} and execution frequency, to minimize the overheads of checkpointing-and-rollback mechanism required for error correction due to transient faults. *CHARM* also incorporates a reliability-aware NoC routing scheme that balances the NoC router and core aging in the presence of soft-errors and application deadlines. As a result, *CHARM* achieves higher application throughput compared to best known prior work without violating the chip-wide dark silicon power budget.

In chapter 4, we propose a runtime framework called *PARM* [33] that minimizes peak power supply noise in low power CMPs that operate at *near-threshold* voltages. *PARM* reduces the total number of voltage emergencies (VEs) caused due to power supply noise (PSN) in the presence of

a dark silicon power budget and application deadline constraints. *PARM* assigns application mapping regions, DVS schedules, and adaptable DoP to applications arriving at runtime, to minimize the variation in switching activity between cores, which is the root cause of PSN in CMPs. Besides, *PARM* also encapsulates a PSN-aware NoC routing scheme called *PANR* that trades off communication latency with switching activity of the NoC routers to balance the switching variations between cores and NoC routers with high switching activities. This results in lowering the error correction overhead incurred by the checkpointing-and-rollback mechanism required to correct the faults caused by VEs.

In chapter 5, we analyze the root cause of the memory bottleneck problem in GPGPU based CMPs and propose a memory-aware, and latency optimized, fast overlay NoC called *RAPID* [34]. In *RAPID,* we divide the NoC into a request and reply plane. We propose a novel NoC router architecture in the reply plane of the NoC called *hinge router* which is tailor-made for the many-to-few and few-to-many traffic pattern in manycore GPGPU accelerators. Besides, we also propose a novel memory controller called *burst MC* which is enhanced to prioritize the incoming burst of read requests that create congestion at the MC-NoC interface. The proposed NoC and MC architectures consume lesser energy than the state-of-the-art architectures that address the memory bottleneck problem.

In chapter 6, we further analyze the traffic pattern of GPGPU accelerators and resolve the memory bottleneck problem. We utilize the approximate computing paradigm to address the memory bottleneck in GPGPUs by proposing a data-aware approximate NoC architecture called *DAPPER* and a novel memory controller architecture called *AMC* [39]. In this contribution also, we divide the on-chip communication traffic into request and reply planes. In the request plane, we leverage the row buffer locality (RBL) and bank level parallelism (BLP) of the requests and

design a new scheduler that minimizes the DRAM latency. In the reply plane, we leverage the approximability of the data waiting in the MC output queues to *coalesce* the reply data and reduce the NoC traffic between MCs and GPGPU cores. We then utilize the fast overlay circuit architecture to transmit the coalesced data to destination cores in less than 3 clock cycles, which is significantly lower than in conventional NoC architectures. *DAPPER+AMC* improves NoC and MC throughput while minimizing the NoC and DRAM latency as well as total energy consumed by manycore GPGPU accelerators without sacrificing more than 1% of output correctness.

In chapter 7, we focus on resolving security vulnerabilities in CMPs that use 3PIP NoC [38]. Data snooping is a significant threat in 3PIP NoCs. Hardware Trojans that are embedded in 3PIP NoC during design/fabrication/integration stages can potentially duplicate the sensitive application data and send it to a listener core that is executing a malicious application. In this chapter, we first demonstrate how a low power hardware Trojan can be placed in a network interface (NI) of a 3PIP NoC that consumes low area overheads, making it difficult to detect using functional verification and side channel analysis. We then propose a novel snooping invalidation module called *SIM* that is located between network interface (NI) and NoC router to detect the duplicate packets that are injected by the Trojan in the malicious NI and mitigate the snooping attack. Furthermore, we also propose a threshold activated NoC snooping detection module called *THANOS*, which is located between trusted cores and the untrusted 3PIP NoC to detect the source of an on-going snooping attack. *THANOS* is also designed to be hard to reverse engineer or get tampered by physical inspection attacks.

Chapter 8 concludes this dissertation. We summarize our comprehensive body of research in this chapter and also make recommendations for future work.

2. ARTEMIS: AN AGING-AWARE RUNTIME APPLICATION MAPPING FRAMEWORK FOR 3D NOC-BASED CHIP MULTIPROCESSORS

In emerging 3D NoC-based chip multiprocessors (CMPs), aging in circuits due to bias temperature instability (BTI) stress is expected to cause gate-delay degradation that, if left unchecked, can lead to untimely failure. Simultaneously, the effects of electromigration (EM) induced aging in the on-chip wires, especially those in the 3D power delivery network (PDN), are expected to notably reduce chip lifetime. A commonly proposed solution to mitigate circuit-slowdown due to aging is to hike the supply voltage; however, this increases current-densities in the PDN due to the increased power consumption on the die, which in turn expedites PDN-aging. We thus note that mechanisms to enhance lifetime reliability in 3D NoC-based CMPs must consider circuit-aging together with PDN-aging. In this chapter, we propose a novel runtime framework (*ARTEMIS*) for intelligent dynamic application-mapping and voltage-scaling to simultaneously manage aging in circuits and the PDN, and enhance the performance and lifetime of 3D NoC-based CMPs. We also propose an aging-enabled routing algorithm that balances the degree of aging between NoC routers and cores, thereby increasing the combined lifetime of both. Our framework also considers dark-silicon power constraints that are becoming a major design challenge in scaled technologies, particularly for 3D stacked CMPs. Our experimental results indicate that *ARTEMIS* enables the execution of 25% more applications over the chip lifetime compared to state-of-the-art prior work.

2.1. MOTIVATION AND CONTRIBUTION

Bias Temperature Instability (BTI) is the most dominant physical phenomenon that degrades the maximum switching rate of transistors under long periods of voltage stress in emerging chip

multiprocessors (CMPs) [11]. BTI causes gradual circuit slowdown over the operational lifetime of the electronic chip. For systems manufactured at technology nodes below 45nm, BTI-induced delay-degradation can be quite significant [40], [41]. The principal effect of such a circuit-aging mechanism is to increase circuit-threshold voltage (V_T), which results in higher circuit-delay. From a system-level perspective, such V_T-degradation causes slowdown in critical paths of processor-cores and network-on-chip (NoC) routers, thereby limiting overall system performance. With increasing demand for reliable CMPs with longer lifetimes in non-consumer domains such as aerospace, defense, automobile, and health, prolonging the useful CMP lifetime will be very beneficial.

Additionally, electromigration (EM) in metal wires on the chip leads to increased interconnect resistance over time in CMPs. This phenomenon is most dominant in power delivery network (PDN) wires that carry larger unidirectional currents compared to signal wires [15], [12]. The increased resistance of the power-grid results in higher IR-drops in the PDN, which causes further circuit slowdown due to degradation of supply voltage [16]. These adverse effects of EM are expected to be particularly severe in 3D CMPs that possess limited number of power-pins and higher current densities [42]. Also, with process technology scaling, this problem is exacerbated due to the reduction in cross-sections of metal wires, which causes further increase in PDN current-densities [15].

To mitigate BTI-induced delay degradation, while maintaining circuit operation at a minimum clock frequency (i.e., minimum performance level), one solution is to hike the supply voltage [43] adaptively over time based on the degree of circuit-aging. However, doing so increases current-densities in the PDN due to the increased power dissipated in the chip as a result of the voltage-hike. High current densities end up causing faster EM-induced PDN-aging [44],

hastening circuit-slowdown. Hiking supply voltage also increases VT-degradation, further increasing the rate of circuit-aging.

As aging reduces the viable lifetime of current and emerging CMPs, it is becoming increasingly important to consider it during the design process. Unfortunately, designers today are more focused on meeting performance requirements, and resort to either using costly hardware guardbands to minimize the effect of performance variations on a die, or employing large supply voltage guardbands to ensure a reliable voltage supply, that ends up increasing power densities and peak temperature of chip, resulting in shortening chip lifetime. Practical and low-cost solutions to enhance lifetime are thus becoming essential, especially in dense 3D CMPs fabricated in scaled technologies. As noted earlier, such solutions must also consider the interdependence between BTI-induced circuit aging, supply voltage, and EM-induced PDN-aging.

Yet another challenge facing CMP designers is the rise in on-chip power dissipation. The slowdown of power scaling with technology scaling, due to leakage and reliability concerns [45], [46], has led to high chip power-densities, giving rise to the dark-silicon phenomenon, whereby a non-negligible fraction of the chip must be shut down at any given time to satisfy the chip power-budget. With the extent of dark-silicon increasing with every technology-generation [47], [48], designs are becoming increasingly power-limited rather than area-limited. Therefore, runtime power-saving techniques such as dynamic voltage scaling (DVS) are of paramount importance to extract much needed performance given a stringent chip-wide power-budget.

To simultaneously address all the above mentioned challenges related to aging, power dissipation, and performance facing chip designers, in this chapter we propose *a novel runtime aging-aware application-mapping framework called ARTEMIS*. Our framework is intended for 3D

NoC-based CMPs and aims to increase the useful work performed over the lifetime of these chips, while meeting the dark-silicon power-budget (DS-PB) and application performance goals. The novel contributions of in this chapter are summarized below:

- We propose a novel runtime application-mapping and DVS-scheduling framework that can adapt to different aging scenarios to extend the lifetime of a 3D NoC-based CMP;
- As the impacts of PDN-aging and circuit-aging (for cores and NoC routers) on system-performance are correlated, our framework considers aging in these components, unlike any prior work, while making mapping decisions to alleviate system aging;
- Our methodology to evaluate system-aging and the resulting maximum-attainable performance accounts for progressive effects of IR-drops due to PDN-aging, VT-degradation due to circuit-aging, and temperature profiles over the chip lifetime;
- We design a novel symmetric aging-enabled NoC routing path allocation (SAR) heuristic to produce a balanced core-router aging profile to extend the lifetime of the NoC. In addition, SAR efficiently trades-off aging with network-congestion in the NoC;
- Our framework also meets chip-wide power constraints, thus finding applicability in contemporary power-constrained (dark-silicon afflicted) multicore designs.

2.2. RELATED WORK

In recent years, several researchers have proposed run-time and design-time application mapping techniques to address the problem of circuit-aging in CMPs. Tiwari et al. [43] suggest mapping high-power tasks onto faster (less-aged) cores and low-power tasks onto slower cores, thereby "hiding" the aging in the chip. At the same time, they propose to lessen aging by scaling the supply voltage or the threshold voltage. Feng et al. [49] perform "local wear-leveling" by

scheduling tasks on cores while considering circuit-aging in sub-core components. But such wear-leveling approaches where "younger" cores are prioritized over aged cores without considering application-performance (frequency) requirements lead to higher leakage power dissipation as faster cores are also leakier, which expedites aging. *Thus, in the dark-silicon regime where performance is closely tied to power, wear-leveling techniques are usually sub-optimal*.

Some of the recent works propose aging-aware frameworks that discretize target lifetime of the chip into finer lifetime constraints, and perform runtime management to satisfy a pre-defined target lifetime and system performance goal. For instance, Mintarno et al. [41] use frequency, voltage, and cooling power as control parameters to optimize the energy-efficiency of a system while meeting lifetime targets. Paterna et al. [50] propose a linear-programming based task-allocation solution to optimize energy, while [51] performs voltage tuning over shorter time-intervals to meet aging constraints over longer time-intervals. But these techniques are either too time consuming to be viable for runtime decision making, or require comprehensive knowledge of future application characteristics, which may not be available in many environments where CMPs are used. In [52], M. H. Haghbayan et al. have proposed a lifetime aware runtime mapping framework that maps tasks on to cores to meet the dark-silicon power budget and satisfy the target reliability till the end of the chip lifetime. However, they have not considered other viable resource management approaches such as Dynamic Voltage Scaling (DVS) to execute more applications within a given dark- silicon power budget. They have also considered a 2D CMP where the impact of electro-migration (EM) is not predominant. However, recent chip designers and manufacturers are gravitating towards 3D CMPs, where the degradation (due to EM) in power delivery networks (PDN) can potentially slow down performance of on-chip components such as cores and NoC-routers. Having redundant cores is a viable solution in a processor with a small core count. But, as

the core count increases, redundancy based schemes will incur very high power and area overheads, which is impractical in dark-silicon power constrained chips. In [47], Kapadia et al. have proposed a process variation aware framework for application mapping to improve the reliability due to soft errors in the dark-silicon era.

More recent works have considered on-chip temperature profile as a prime contributor to circuit-aging and propose techniques to reduce the peak on-chip temperature with thermal-aware mapping techniques. Gnad et al., in [53], have proposed a framework that uses an offline generated frequency degradation table for various applications to estimate the aging-induced frequency degradation in cores before mapping, for efficient aging and dark-silicon management on a 2D CMP. Their framework has objectives similar to *ARTEMIS*, but when applied to a 3D CMP, it ignores PDN-aging. Further, they aim to minimize circuit-aging and power consumption in cores, while ignoring other on-chip components such as NoC. In [54], Singh et al., have proposed a mapping technique to reduce the energy consumption and the peak on-chip temperature while improving the application throughput of 3D video processing applications on 3D CMPs. Even though thermal-aware mapping minimizes aging to an extent in 2D CMPs, mitigating aging in cores and PDN simultaneously in 3D CMPs requires a mapping technique that is cognizant of degradation profiles of both cores and PDN along with the prior knowledge of workload that is executed on them. In [55], [56], Pasricha et al. have proposed a fault tolerant and energy efficient NoC routing scheme for 2D and 3D NoC based systems In [57], Rehman et al. have proposed a framework to address the reliability challenges due to soft-error induced failures. Their framework aims at leveraging the knowledge of on-chip variation and aging profiles to efficiently choose the hardware-software reliability mitigation and application mapping techniques, to improve the reliability of 2D CMPs. Unlike their framework, we balance circuit- as well as PDN-aging and

increase the useful lifetime of 3D CMPs in the presence of dark-silicon. In [58], a novel temperature-model is proposed that considers both spatial and temporal dependencies. This model is used by a design time task scheduling and operation point assignment mechanism, to jointly optimize reliability and energy of multimedia applications represented using streaming data flow graphs (SDFG). However, a design time task assignment does not efficiently capture a run-time aging profile caused by dynamic workload characteristics found in most of today's CMPs. In [59], a run-time task scheduling framework is proposed to minimize the communication energy consumed, without effecting the throughput of multimedia applications on heterogeneous multicore systems in the presence of permanent and intermittent failures. In [60], a scenario-aware fault injection model is proposed to model intermittent failures caused by wear-out mechanisms on 2D MultiProcessor-System-on-Chips (MPSoCs). They further propose a wear-out-aware application mapping technique that maps applications based on the intermittent failure rate, which is an indicator of chip aging. However, this work ignores the presence of dark silicon power constraint that is prevailing in most of today's multicore processors. Also, all these works do not consider the impact of aging in PDN. *To the best of our knowledge, this work is the first work on lifetime-aware application mapping at runtime that considers the impact of PDN-aging on the lifetime of the chip, and is also tailored for the power-constrained dark-silicon design regime.*

Aging is a concern not just for computation cores but also for NoC fabrics that connect these cores together. But very few works have investigated design-techniques that extend the service life of the NoC fabric. Bhardwaj et al. [61] have proposed an aging-aware adaptive routing algorithm that routes packets along the paths that are both less congested and experience smaller aging stress. But the authors do not consider aging in compute-cores. *Our work represents one of the first efforts*

to extend the useful lifetime of the entire chip by producing a balanced core-router aging profile, with our proposed symmetric aging-enabled routing path allocation (SAR) heuristic.

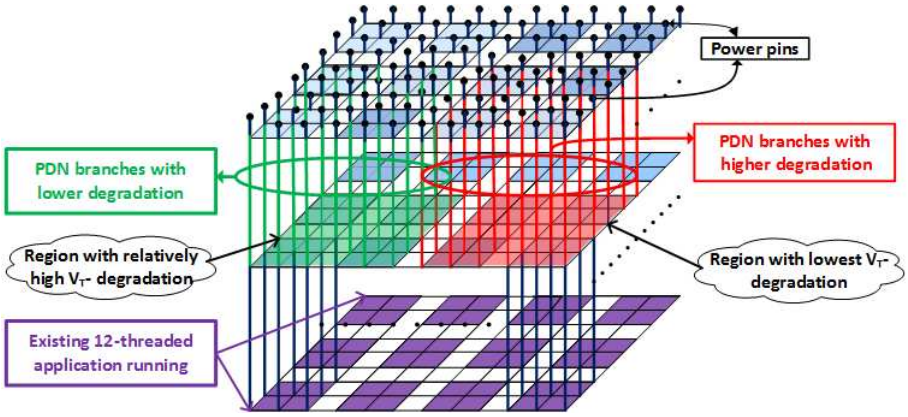


Figure 11 Example of a 3D package for a 36-core CMP (4x3x3 3D-mesh) with a regular 3D power-grid that has 108 external power-pins and 108 grid-points per tier (16 grid-points supplying to each core). Not all vertical branches of PDN are shown, for brevity.

2.3. MOTIVATION

In this section, we illustrate the advantages of our *ARTEMIS* framework with the help of a small example. We consider a scenario in which applications arrive at runtime to be executed on a 36-core CMP with a core capable of executing a single thread (task) at a time. The example assumes a 4-thread application being mapped on to the cores of a 3D-CMP at time t, when a 12-thread application is already executing on the bottom tier (shown in purple in Figure 11). In a 3D-CMP, a 2D region of tiles in a central layer with less V_T-degradation can have a relatively high PDN degradation due to the current flowing through the PDN in middle layer, to supply the applications that ran in the bottom layer. The variation in PDN degradation is depicted by red and green colored tiles and lines in Figure 11 where red PDN lines supplied more current to applications in the bottom layer than green PDN lines. When an aging-aware wear-leveling technique based on prior work [43], [49] is used, the application would be mapped to the rectangular region (shown in red) containing cores with the least V_T-degradation, i.e., the region

with the youngest cores. Observe that the vertical PDN branches supplying to this region have high degradation (higher resistance values due to past stress). Alternatively, although the green rectangular region has more V_T-degradation, the PDN IR-drops sustained by it are lower than the red region. By always prioritizing mapping of applications on to the youngest cores, without considering the resulting impact on EM-induced degradation in the PDN, resistances of already stressed PDN-wires would be further increased, thus exacerbating PDN-degradation. Therefore, *ARTEMIS* considers both V_T-degradation and PDN-degradation while making mapping decisions to limit PDN-degradation while guaranteeing that performance and power constraints are met on a 3D NoC-based CMP.

Additionally, the maximum frequency (f_{max}) of a core is affected by both the PDN IR-drops (which affects V_{dd}) as well as V_T-degradation:

$$\boldsymbol{f_{max}} = \frac{\boldsymbol{\mu(V_{dd}-V_T)^{\alpha}}}{\boldsymbol{C_0.V_{dd}}} \qquad \text{... (1)}$$

where α and μ are technology-dependent constants, and C_0 is switching capacitance of the critical path **[62]**. Approximate values of these constants are listed in Table 1.

Table **1**. Thus, any mapping solution obtained without consideration of IR-drops experienced by cores can potentially lead to undesirable timing-errors.

In summary, the wear-leveling based application-mapping approach (i.e., always choosing the youngest cores) that is used in several prior works would increase leakage power, and hence temperature, thus resulting in higher circuit-aging. In addition, higher leakage power dissipation would cause increased supply currents to be drawn from the PDN resulting in higher PDN-aging. In contrast, *ARTEMIS* prioritizes older (slower) cores that can support application frequency

constraints without requiring hiking of V_{dd}-levels. The next section presents our problem formulation, followed by details of the *ARTEMIS* framework in Section 2.2.5.

2.4. PROBLEM FORMULATION

2.4.1 MODELING BTI-INDUCED CIRCUIT AGING

In this work, we model circuit aging effects arising from BTI-induced circuit degradation, as BTI has been found to be one of the most dominant aging mechanisms in emerging semiconductor technologies. But, our framework is capable of supporting models of other aging-mechanisms (HCI, TDDB etc.) as well. Velamala et al. [63] [64] have shown that a Trapping/Detrapping (TD) based BTI model is capable of accurately predicting the degradation under a sequence of V_{dd}'s used in the DVS operation. They have noted that when the supply voltage is changed from a higher V_{dd} to lower V_{dd}, the circuit-degradation undergoes recovery; this recovery behavior is not captured by conventional Reaction-Diffusion (RD) models. Therefore, our analysis of circuit-aging over the CMP-lifetime is based on the long-term aging prediction model proposed in [63], which accounts for different V_{dd}-levels over time. We estimate the effective ΔV_T increase that a component (computation core or NoC router) experiences over a time-interval of t using equations (2) and (3):

$$\boldsymbol{\Delta V_T(t) = L.[A + Blog(1 + Ct)]} \quad \text{.. (2)}$$

$$\boldsymbol{L = K_1.\exp\left(\frac{-E_0}{kT}\right).\left\{exp\left(\frac{\beta V_1}{T_{ox}kT}\right).\alpha_1 + \cdots + exp\left(\frac{\beta V_S}{T_{ox}kT}\right).\alpha_S\right\}} \quad \text{.. (3)}$$

where V_i is the i^{th} V_{dd}-level utilized by the component for a time-duration α_i, S is the total number of allowable V_{dd}-levels, and $\{\alpha_1+\alpha_2+...+\alpha_S\} \leq t$. T is the average temperature of the component during corresponding α_i. We obtain A,B, and C and other parameter-values in equations (2) and

(3) by solving the equations given [11], [63]- [65] that are validated against silicon data. Values of constants used in equations (2) and (3) are listed in Table 1.

Table 1 Values of constants used in the models of BTI and EM aging

Constant	Description	Value
μ	Mobility of the charge carriers	1000 $cm^2/V.s$
α	Technology dependent constant	2
K_1	Poisson parameter for trap distribution	0.0075
E_0	Electric field across the channel	0.1897 eV
k, k_B	Boltzmann constant	0.000086
T_{ox}	Oxide thickness	1.4 nm
D_{eff}	Effective diffusivity	6.7×10^{-15}
σ_c	Critical stress	4.1×10^{6} Pa
Ω	Atomic volume	1.182×10^{-29} m^3
ρ	Resistivity of Cu	2.5×10^{-8} Ω.m
B	Effective bulk modulus for the Cu-dielectric system	10^9
e	Charge of an electron	1.602×10^{-19} C
Z_{eff}	Apparent effective charge number	5.0
L_{wire}	Length of the wire	50um

2.4.2 MODELING EM-INDUCED PDN AGING

To model the phenomena of void nucleation and void growth in every horizontal and vertical on-chip wire, particularly those in the PDN that are under the most stress, we use the EM model proposed in [66] for copper (Cu) interconnects in the power grid. equation (4) gives the time t_n at which the void nucleates:

$$\boldsymbol{t_n = \frac{K_t}{D_{eff}},\ K_t = \frac{\pi}{4}\left(\frac{(\sigma_c)^2 \Omega k_B T}{(eZ_{eff}\rho j)^2 B}\right)} \qquad \text{….. (4)}$$

Once the void nucleates at time t_n, then at an observation time t_0, the length of the void L_{void} is:

$$\boldsymbol{L_{void}(t_0) = \left(\frac{D_{eff}}{k_B T}\right) eZ_{eff}\rho j (t_0 - t_n)} \qquad \text{….. (5)}$$

The length of the void in a Cu wire increases with the product of electrical current and the time-duration for which the current flows through it. The length of the void in turn determines the increased resistance (ΔR) or degradation of the wire, which is given by:

$$\Delta \boldsymbol{R} = \boldsymbol{c}.\boldsymbol{R_0}\left(\frac{\boldsymbol{L_{void}}}{\boldsymbol{L_{wire}}}\right) \qquad \text{….. (6)}$$

where R_0 is the resistance, constant c depends on the resistivity and cross-sectional area, and L_{wire} is the wire-segment length. Table 1 lists the values we used for the constants in equations (6).

2.4.3 INPUTS, ASSUMPTIONS, AND PROBLEM OBJECTIVE

We have the following inputs to our problem:

- A 3D NoC-based CMP with a 3D mesh NoC, of dimensions (dim_x, dim_y, dim_z) and number of tiles $N = dim_x \times dim_y \times dim_z$ with each tile containing a compute core and a NoC router;
- A set S of candidate supply voltage (V_{dd}) levels for the chip;
- A chip-wide dark-silicon power budget (DS-PB);
- An application task graph for each application: vertices with task execution-times on compute cores and edges with inter-task communication volumes; execution time and volume values are assumed available from offline profiling;
- Degree of parallelism (DoP) of each application, and a set of n admissible rectangular/cuboidal shapes (x, y, and z dimensions) of regions that it could be mapped to $\{B_1, \ldots, B_n\}$; e.g., a tuple set {2×4×1, 4×2×1, 2×2×2} for an application with DoP=8;
- A minimum operating frequency (and thus a corresponding maximum execution time) constraint for each application;

- A regular 3D power grid, with $p \times p$ grid-points supplying to each core and $dim_x \times dim_y \times p \times p$ power-pins at the top of the 3D-die, and an air-cooled heat sink at the bottom of the 3D-die;

We make the following assumptions in our work:

- Applications can arrive in any order at runtime and must be mapped onto the 3D CMP while satisfying application-specific maximum execution time and minimum frequency constraints;
- There exists one-to-one mapping between tasks and cores, i.e., a core can execute only one task (thread) at any given time;
- Applications are mapped contiguously on non-overlapping rectangular (on single tier) or cuboidal (across multiple tiers) shaped regions of the 3D CMP for inter-application isolation and more optimized communication-profiles (as recommended in prior works such as [67]); thread-migration is not considered;
- A chip-wide supply voltage exists that can be scaled using DVS at runtime, as the overheads of implementing DVS at a per-core granularity are high for CMPs with very large core counts [68];
- Similar to prior-works (e.g., [41], [49], [51]) we assume presence of on-chip aging-sensors [69], [70] that provide aging information of compute-cores and NoC routers to our framework; we also assume voltage-sensors [71] at each power-input (PDN-grid-point) of a CMP-tile that track the severity of IR-drops (i.e., the degree of PDN-degradation on PDN-paths supplying to that core);
- From the on-chip sensors, runtime sensed data (at a per-tile granularity), in terms of threshold-voltage (V_T) distribution and maximum IR-drops, is available after every *epoch*; our aging models (discussed in Sections 2.4.1 and 2.4.2) emulate runtime readings from sensors on real

chips. We define an epoch as the time-period during which the aging profile of the chip can be assumed to be constant;

- The 3D CMP is rendered unusable (end of lifetime) when an incoming application is unable to be executed (i.e., when its minimum application frequency requirement cannot be supported) on any of the allowed rectangular/cuboidal regions, at any V_{dd} level without violating the DS-PB constraint, when there are no other applications running at that time.

Objective: Given the above inputs and assumptions, our objective with the *ARTEMIS* framework is to perform runtime application-mapping and DVS-scheduling on a given 3D NoC-based CMP platform such that the total number of applications executed over the lifetime of the chip is maximized, while all application-specific minimum operating frequency- and maximum runtime-constraints, as well as CMP platform-specific DS-PB constraints are satisfied.

2.5. ARTEMIS FRAMEWORK: OVERVIEW

This section explains the design flow of the framework that involves the actual mechanism of run-time mapping and DVS selection along with the simulation of circuit- and PDN-aging in cores and routers. In manufactured CMPs, compact aging sensors such as the ones proposed in [69], [70], and [72] can be used to measure the on-chip aging profile. These sensors are amenable to use in standard cell design with minimal area and power overhead. They can be implemented in large numbers along the top 10% of component's critical paths to collect high volume digital data on device degradation.

ARTEMIS is a run-time application mapping framework that processes applications from the top of the service queue, where the applications are stored as they arrive in the run-time. The run-

time framework receives periodic feedback (at the end of each epoch) from on-chip aging and power sensors (simulated models), that is used for aging- and dark-silicon-aware mapping and DVS scheduling. At any given time, the scaling-down of V_{dd} via DVS-scheduling to save power and to limit aging is limited by the frequency-constraints of the applications running on the 3D NoC-based CMP, whereas scaling-up of V_{dd} is constrained by the DS-PB.

The key aspects of our proposed framework are illustrated in Figure 12. The Run-time application-mapping consists of assigning the application task-graph on to a chosen rectangular- or cuboidal-shaped region of tiles on the 3D CMP admissible for the application (from the list $\{B_1, \ldots, B_n\}$), as well as performing routing path allocation of the intra-application communication-flows on the 3D NoC. The knowledge of the chip-aging profile is continuously utilized in the application-mapping and DVS scheduling steps.

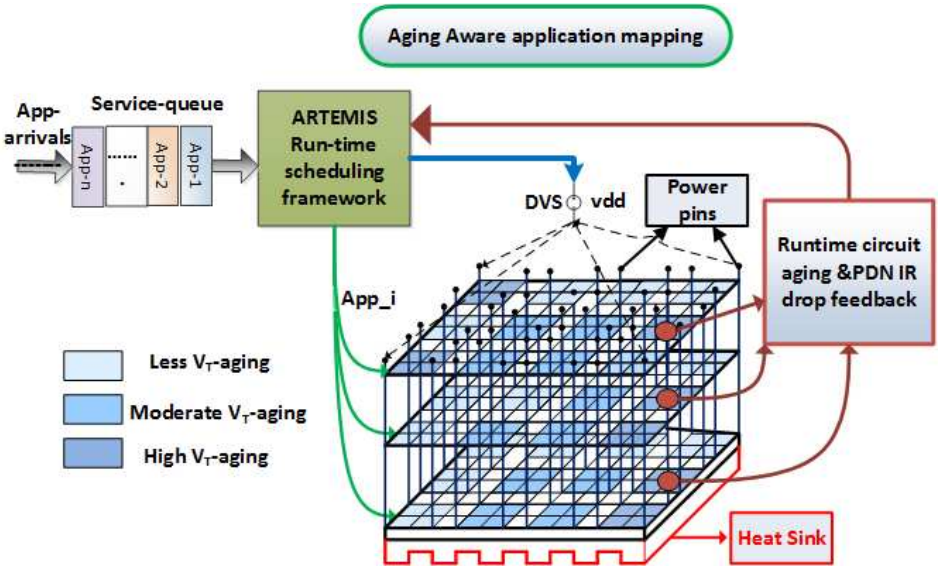


Figure 12 Overview of ARTEMIS runtime aging-aware application-mapping and DVS-scheduling framework

The *ARTEMIS* framework is executed in two nested procedures: *(i)* Aging-aware application mapping and DVS *(inner-loop)*; and *(ii)* Circuit- and PDN-aging analyses *(outer-loop)*. These procedures are discussed in Sections 2.5.1 and 2.5.2 respectively, and the design flows for the two procedures are shown in Figure 13 (a) and Figure 13 (b).

2.5.1 AGING-AWARE APPLICATION MAPPING AND DVS SCHEDULING

During each epoch at runtime, we assume that applications arrive for execution on the 3D NoC-based CMP. Suppose a sequence of *l* applications arrives in an epoch. Aging-aware application mapping and DVS module *(inner-loop)* is responsible for mapping these *l* applications onto the CMP during the epoch. If an application cannot be mapped immediately after it arrives, it is kept in a service queue, and mapped later. We assume processing of the service-queue on a first-come-first serve basis (although a priority-based processing approach could also be used). At the end of the epoch, aging information is updated from the on-chip circuit-aging sensors as well as the voltage sensors (this information is utilized by inner-loop during the next epoch). Subsequently, a new application sequence is serviced during each new epoch, and this process continues until the end of the lifetime of the 3D NoC-based CMP.

At the start of an epoch, the application-service queue is initialized to point to the first application (*app_ptr=0*), and the local time counter1 is initialized to zero, as shown in Figure 13 (a). Processing of the service-queue event is triggered, (i.e., new applications are serviced) when an application arrives or an existing one terminates. Once the event is triggered, an application instance is removed from the front of the queue and processed by the aging-aware mapping and V_{dd} selection phase (Figure 13 (a); discussed in section 2.5.1.1). Applications from the queue continue to be processed one-by-one until an "application stall" event is detected. An application

in a service queue can be stalled only due to the following reasons: (i) available tile constraints on the 3D-die; (ii) DS-PB constraint; or (iii) application frequency constraints for the given degradation profile of the 3D-CMP. Note that if an application is stalled when there are no other applications running, i.e., the chip-degradation (V_T- and PDN-degradation combined) precludes it from meeting the application-frequency constraints, the 3D NoC-based CMP is considered as no longer usable and has reached its end of life.

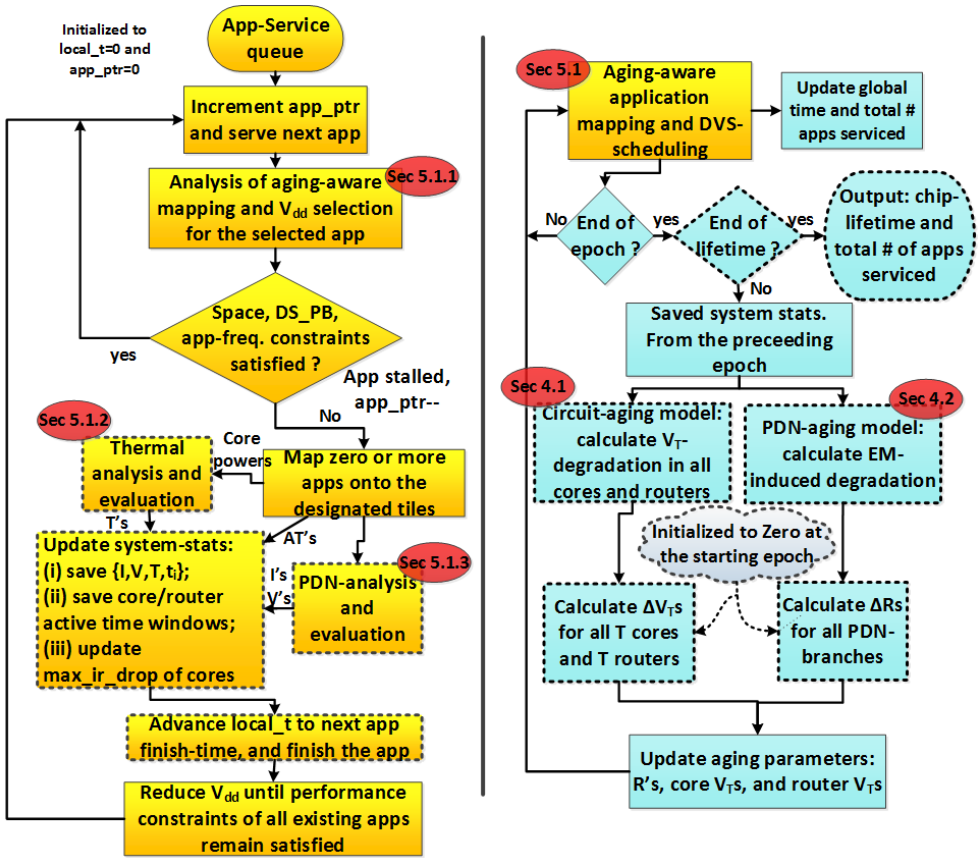


Figure 13 ARTEMIS design-flow: (a) Aging-aware application-mapping and DVS scheduling (inner loop; section 2.5.1.1); (b) Circuit- and PDN-aging analyses (outer loop; section 2.5.1.2). The boxes with dotted outlines are used as part of our aging-simulation framework; however, these steps are not required on real hardware where runtime aging information is assumed to be available from on-chip sensors.

When an "application stall" is detected or the application-service-queue becomes empty for the current epoch, the application(s) that have been processed by the mapping/selection phase (discussed in section 2.5.1.1) are mapped on to the appropriate tiles chosen by the phase. At this time (local time), either one or more new applications are mapped on to the 3D-CMP or an application just ended (which triggered the service-queue), thus the steady-state computation-profile (i.e., CMP tile-power values, the resulting supply currents in the PDN, and thermal-profile) of the 3D-CMP changes. To evaluate the new computation-profile, given the tile-power-distribution, thermal-analysis is performed to re-evaluate the thermal-profile (at per-tile granularity) and PDN-analysis is performed to evaluate all the branch currents and voltage-drops at all grid-points in the PDN. Also, a worst-case IR-drop value (WC-IR-drop, which is the maximum voltage-drop out of all grid-points supplying to a tile) is evaluated for each of the N tiles. The WC-IR-drop value is updated for each tile (at every change of computation-profile) over the chip-lifetime and continuously used to calculate the maximum-frequency of the tile (for a given V_{dd}) in the application-mapping step. The thermal- and PDN-analysis is discussed in sections 2.5.1.2 and 2.5.1.3, respectively. After the mapping/selection phase, thermal-analysis, voltage *(V)*, and temperature *(T)* values, as well as the WC-IR-drop values in the time-window t_i for this (i^{th}) computation-profile, is saved in the system-stats, as shown in Figure 13 (a).

Additionally, if one or more applications are mapped at the current local time, the active-times (AT's) of compute-cores and NoC routers are calculated for each newly mapped application. For each tile, these AT's could be represented as $\{C_j, R_j, t_j\}$, where C_j *and* R_j take values of '1' or '0' depending on whether the corresponding compute-core or NoC router is active during the time-window t_j. These AT's for compute-cores and NoC routers are also saved in system-stats. The

system-stats for all time-windows over the entire epoch duration are eventually utilized for aging-analyses (in the outer loop) at the end of the current epoch.

After updating system-stats, local time is advanced to the next application finish-time, and the corresponding application is completed, (Figure 13 (a)). As part of our DVS strategy to save power and limit aging, on completion of any application, we reduce V_{dd} to the lowest allowable level that would not introduce any violations in frequency constraints of existing (already running) applications.

2.5.1.1 APPLICATION-SPECIFIC MAPPING AND VDD-SELECTION

For the application under consideration, this phase consists of three steps: *(i)* circuit- and PDN-aging aware region selection and Vdd-selection, *(ii)* communication-aware task-to-tile mapping, and *(iii)* NoC routing path allocation. We describe these steps below.

(i) Circuit- and PDN-aging aware region selection and voltage-selection: In our framework, an application with a given DoP can be mapped on to rectangular or cuboidal regions on the 3D CMP, with shapes to be chosen from a pre-defined list $\{B_1, \ldots, B_n\}$ for that application. All intra-application communication is contained within t closed region, thus application-isolation is maintained and communication cross-interference is eliminated. Our heuristic in this step utilizes the V_T-degradation profile and the WC-IR-drop profile of the 3D CMP. The objective is to find the region on the 3D-mesh (with one of the admissible shapes) so as to: *(a)* minimize leakage-power; *(b)* minimize EM-induced degradation of PDN-paths supplying to cores with high WC-IR-drops; *(c)* satisfy the frequency-constraint of the application by all cores within the region; *(d)* satisfy the DS-PB. In other words, we search for CMP-regions with most circuit-aging that satisfy

minimum application-frequency constraints and have least WC-IR-drops. To this end, we define the following cost-function (Ψ) for joint optimization of leakage-power and PDN-degradation:

$$\boldsymbol{\Psi} = \sum_{\boldsymbol{k=1}}^{\boldsymbol{k=DoP}} \left\{ \boldsymbol{\alpha}. \left(\frac{\boldsymbol{\max\_V_T - V_{Tk}}}{\boldsymbol{\max\_V_T - nom\_V_T}} \right) + \boldsymbol{\beta} \left(\frac{\boldsymbol{WC\_IR\_drop_k}}{\boldsymbol{\max\_IR\_drop}} \right) \right\} \quad \ldots\ldots (7)$$

where, V_{Tk} is the effective V_T and $WC\text{-}IR\text{-}drop_k$ is the WC-IR-drop of the k^{th} core within the region of DoP cores; nom_V_T is the nominal (lowest) effective V_T-value of a core with no aging; and α and β are weighting coefficients. We define max_V_T as the maximum V_T value that the core can support for an ideal (zero) WC-IR-drop (at highest V_{dd}) while meeting the frequency-constraint of the application. Similarly, *max_IR_drop* is the maximum tolerable WC-IR-drop for a core for nominal V_T and highest V_{dd}.

Algorithm 1: Aging-aware region selection and V_{dd}-selection heuristic

Inputs: V_T-profile, WC-IR-drop profile, {B_1, …, B_n}

1: while ($V_{dd} \leq max\_V_{dd}$) **do {**
2: for each tile on the 3D NoC-based CMP **do {**
3: assume this tile to be at the minimum *x, y, z* coordinates of the region
4: for each shape in {B_1, …, B_n} **do {**
5: if all tiles (compute-cores and routers) satisfy app-frequency
6: check if DS-PB is satisfied
7: calculate Ψ, choose this shape if least Ψ **AND** DS-PB satisfied
8: else go to next shape B_i (step 4)
9: end if
10: } // end for each shape …
11: } // end for each tile …
12: if (no valid region found **AND** no DS-PB violation**)**
13: hike V_{dd}
14: end if
15: } //end while
16: if no valid region found
17: stall this application
18: end if

output: a valid region to map the application and V_{dd}-level, or "stall"

Algorithm 1 shows the pseudo code of our region- and V_{dd}- selection heuristic. The heuristic performs a simple exhaustive search over all tiles on the 3D-mesh and over all admissible shapes

$\{B_1, \ldots, B_n\}$ for the application under consideration. The V_T-profile and WC-IR-drop profile inputs are used for calculating the value of Ψ. The region with the least Ψ value that satisfies the frequency-constraints (with maximum frequency for the selected V_{dd} level calculated using equation (1)) and at the same time does not violate the DS-PB (given that existing applications have been running), is selected for mapping the application under consideration. If no region on the 3D-mesh is found to satisfy the frequency-constraints, we repeat the search for successively higher V_{dd}-levels (which can allow using a higher frequency as per equation (1) with a better probability of meeting frequency-constraints) until either a valid region with minimal Ψ is found or the DS-PB is violated. If no valid region is found, an "application stall" event is initiated.

We now present the theoretical time-complexity of this heuristic. At most N tiles (total tiles on the 3D NoC-based CMP) are considered for the prospective mapping region. Note that DoP of the application (relatively small integer c – treated as a constant) number of tiles are to be evaluated for frequency and leakage-power at each of these iterations. As, the number of candidate V_{dd}-levels $|S|$ as well as the number of admissible shapes n are expected to be small constant integers, our region-selection step runs in linear complexity with respect to the number of tiles, N: $O(cn|S|N)$.

(ii) Communication-aware task-to-tile mapping: After the region on the 3D CMP has been selected (of size equal to application-DoP), our mapping heuristic maps the appropriate application-task-graph on to the chosen CMP tiles. We utilize a fast and efficient communication-aware incremental-mapping approach (similar to that used in prior works such as [73] [74]) suitable for runtime use.

(iii) Symmetric aging-enabled routing path allocation (SAR): In this step, we map the communication-flows of the current application on to the designated cuboidal region on the 3D NoC-based CMP. We propose an aging-enabled and congestion-aware routing scheme (SAR) to

produce a balanced core-router aging profile and extend the lifetime of the 3D NoC. The main objective of *SAR* is to minimize the number of runtime scenarios where application-mapping on a given cuboidal region is precluded due to aging in routers. Note that an application can be mapped only if all tiles (each tile has a compute-core and a NoC-router) within the region under consideration satisfy the minimum application-frequency constraint. Prior work on aging- enabled routing (such as [61]) considers the aging in NoC-routers but does not consider the aging in compute-cores. Such an approach could lead to a somewhat imbalanced aging within tiles of the CMP, thus potentially preventing application mapping onto desirable CMP regions due to excessive aging in NoC routers. *SAR* on the other hand enables symmetric aging on individual tiles of the 3D-CMP to extend the service life of NoC routers. Additionally, SAR efficiently trades-off aging with network-congestion in the NoC by selecting routing paths to maximize NoC-lifetime while leveraging the knowledge of maximum execution time constraints of applications, i.e., the aging metric in the routing cost function is prioritized by varying degrees, given the time-slack available for application-completion.

To ensure a low-overhead implementation, path diversity, and deadlock freedom, our routing algorithm builds on the 4N-First turn model [55] for 3D-mesh NoCs. This routing algorithm is partially adaptive, and hence allows the flexibility to potentially select from among multiple next hop directions, at each router. We designed a cost-function for next-hop selection during routing that considers the difference between router-aging and core-aging ($router\_V_T - core\_V_T$) values to ensure balanced aging in CMP tiles. Moreover, as congestion in the NoC-links leads to excessive routing delays and thus longer application-runtimes, we prefer allocating flows to links with lesser communication-volumes. The following routing cost function (Rt_{cost}), which is a linear

combination of the two normalized metrics, is used to make routing decisions at each hop along the path:

$$\boldsymbol{Rt_{cost}} = \boldsymbol{\alpha_R}.\frac{(\boldsymbol{V_T\ difference})-(\boldsymbol{minimum\ V_T\ difference})}{\boldsymbol{range\ of\ V_T\ difference}}$$

$$+\ \boldsymbol{\beta_R}.\frac{(\boldsymbol{volume})-(\boldsymbol{minimum\ volume})}{\boldsymbol{range\ of\ volume}} \qquad \text{….. (8)}$$

where, α_R and β_R are weighting coefficients, V_T *difference* represents (*router*_V_T – *core*_V_T) of the candidate next hop router, and volume represents the existing communication-volume (already allocated while routing previous flows) on the link. *SAR* selects the next hop with the minimum routing-cost, Rt_{cost}, given in Eq. (8).

Communication delays are calculated from the application-frequency and NoC link bandwidths, and thus the current application-delay can be estimated from the already routed communication-flows. NoC routers and links in an application region run at the same frequency as the cores in the region (application-frequency). Note that the goal of *SAR* is to extend NoC lifetime while meeting application execution time constraints. Thus, the values of coefficients in Eq. (8), α_R and β_R, are re-evaluated after routing each flow, as shown below:

$$\beta_R = \{\textit{current app. delay}\}/\{\delta.(\textit{app. execution time constraint})\}$$

$$\alpha_R = (1 - \beta_R) \qquad \text{….. (9)}$$

Before any application communication flows are mapped to the NoC routers, we start with values α_R =1 and β_R =0. As flows are mapped and the estimated application-delay increases, the value of β_R increases (α_R decreases) proportionally until the application-delay reaches a significant fraction (δ) of the application execution time constraint. At this point (β_R =1 and α_R =0), *SAR* ceases to be aging-aware and routes on paths with minimum congestion exclusively, to meet the execution time constraint of the given application. *Algorithm 2* below summarizes our symmetric-aging enabled routing scheme.

Note that the given application is executed on the actual 3D-CMP platform only after the analysis for routing path allocation is performed. The turn model rules are implemented in each router using simple combinational logic. The next hop selection information at each NoC router is stored in small next-hop routing tables that enable quick selection of the most appropriate next-hop direction based on the source and destination of a packet. Even for the largest sized, 32-threaded applications mapped onto a {4x4x2} cuboid on the 3D CMP platforms we considered, we found that the upper bound on number of communication-flows (with unique source-destination pairs) needed to be routed through any router is 64, with our 3D turn-model based minimal routing scheme. Thus, a NoC router on the 3D CMP would need a next hop table of up to 64 entries. Assuming 3 bits for the output port and 6 bits for the source and destination each, the footprint of the NoC routing table is only 960 bits.

Algorithm 2: Symmetric aging-enabled routing path allocation

Inputs: Task-graph, execution time constraints, minimum frequency, task-mapping of current application, V_T-profile of compute-cores and routers

1: Initialize α_R=1 and β_R =0
2: for all communication-flows **do {**
3: for all hops on the minimal path **do {**
4: select the next hop with the least Rt_{cost} (equation 8)
5: } update α_R and β_R (equation 9)
6: }

output: all flows of the application allocated on the cuboidal CMP- region

As we consider communication intensive applications for execution on the 3D CMP, there is a need for deeper buffers at input and output channels to avoid severe network congestion and application slowdown. For such conditions, we provide each input/output channel with four virtual channels, each consisting of a buffer of size four flits. Hence, the overall size of the buffers is up

to 1.7KB, assuming the flit size to be 8B. Thus, the hardware overhead of implementing *SAR* is small (960 bits or 0.12KB) when compared to the total size of the buffers.

2.5.1.2 THERMAL-ANALYSIS AND EVALUATION

To perform thermal evaluation of a given computation-profile in our framework, we utilize the open-source thermal emulator 3D-ICE 2.2.5 [75] which supports steady-state thermal analysis of 3D ICs with a conventional air-cooled heat-sink. For the given power-profile, the tool outputs the core-temperatures (T's) on the 3D die.

2.5.1.3 PDN-ANALYSIS AND EVALUATION

The supply current drawn by each core is calculated from the core-power and selected V_{dd}-level. Given the supply current requirements of the N cores on the 3D-CMP, we created a linear programming (LP) formulation and used lp_solve [76] to solve for the grid-point voltages and currents flowing in the 3D regular power grid. This enables the updating of {V's, I's} in the power-grid and WC-IR-drops of cores in the 3D CMP, for the given time-window (t_i) of the computation-profile.

2.5.1.4 CIRCUIT- AND PDN-AGING ANALYSES

In the outer loop of our framework (Figure 13 (b)), we utilize system-stats generated by the inner loop over the last epoch to perform aging-analysis at the end of the epoch. Given system-stats for the last epoch, this analysis is used to calculate the rise in effective V_T values (ΔV_T's) of all cores and NoC routers on the 3D CMP, as well as the rise in resistance values (ΔR's) of all

vertical and horizontal PDN-branches, using the circuit- and PDN-aging information. The BTI-induced circuit-aging of compute-core and NoC router components are calculated (discussed in section 2.4.1) using the *V*'s and *T*'s experienced by these components during all of their AT's over an entire epoch. The EM-induced PDN-degradation in PDN-branches (discussed in section 2.4.2) is calculated using *I*'s for all computation-profiles of the epoch. As effects of EM are far less dominant in signal interconnects compared to PDN-interconnects [15], [12], we ignore EM-induced aging in the NoC-links and focus primarily on PDN interconnects.

Note that the active-time windows of compute-cores and routers {C_j's, R_j's, t_j's} may not be aligned with the chip-wide computation-profile windows {*V*'s, *T*'s, *I*'s, t_i's}; therefore, in circuit-aging calculations, the component AT's are required to be split into multiple time-windows where computation-profiles change. Also, at the start of the very first epoch, the *R*'s and V_T's are initialized with nominal values representing no degradation and the ΔR's and ΔV_T's are initialized to zero-values. Lastly, the updated aging profiles are leveraged to make mapping decisions in the next epoch. When the end of lifetime is encountered (discussed in section 2.5.1.1), the aging analyses procedure outputs the lifetime of the 3D-CMP in terms of both the total system-execution-time (global time) and the total number of applications serviced during this time (Figure 13 (b)).

2.6. EXPERIMENTAL STUDIES

2.6.1 EXPERIMENTAL SETUP

Our experiments were conducted using 13 different parallel application benchmarks taken from the well-known SPLASH-2 [77] and PARSEC [78] benchmark suites. We profiled the execution-time, power dissipation, and degree of memory-intensity of each application for

different application-DoPs by performing multicore simulations using the open-source tools SNIPER [79] and McPAT [80]. For each benchmark, the DoP resulting in highest performance was obtained from this profiling study and selected as the fixed DoP value for that benchmark. These DoP values ranged from 4 to 32. Note that increasing DoP beyond this baseline value for each benchmark resulted in lower performance, due to inter-thread synchronization and communication overheads.

We categorized the 13 benchmarks into two groups: *(i)* communication-intensive benchmarks - {*cholesky, fft, radix, raytrace, dedup, canneal, and vips*}; and *(ii)* compute-intensive benchmarks – *{swaptions, fluidanimate, streamcluster, blackscholes, radix, bodytrack, and radiosity}*. As radix has properties of both, we use it in both groups. In our analyses, we employ three types of application sequence groups as inputs to our framework: communication-intensive, compute-intensive, and mixed (using all 13 applications). We assume each application-sequence to have 100 randomly ordered application-instances selected from the respective group. To enhance the statistical significance of our results, we averaged results for five different randomly generated application-sequences for each group.

To simulate the chip-lifetime within a reasonable time, we extrapolate the effects of aging over 500 such sequences, making the total number of application-instances executed within an epoch to be approximately $l = 50{,}000$. Simulation times for *ARTEMIS* to simulate till the end of the lifetime were between 6 and 10 hours. The communication-intensive application workloads typically entailed larger simulation times because of longer chip lifetimes (see results and discussion in Section 2.6.2), compared to the computation-intensive workloads.

We consider a 60-core 3D-mesh NoC based CMP platform, with dimensions 5×4×3 ($dim_x \times dim_y \times dim_z$). Our SNIPER simulations for application-profiling capture performance and

power consumption at the 22nm process technology node. Seven operating voltage levels are used, ($|S|$=7): 0.7V, 0.75, 0.8V, 0.85V, 0.9V, 0.95V, and 1.0V. Frequency-requirements of different applications are set between 1.5GHz and 2GHz. The following region-dimensions-lists $\{B_1, \ldots, B_n\}$ for applications (for the given DoPs) are employed: {2×2×1} for DoP = 4, {4×2×1, 2×4×1, 2×2×2} for DoP = 8, {4×2×2, 2×4×2, 4×4×1} for DoP = 16, and {4×4×2} for DoP = 32. The dark-silicon power-budget (DS-PB) is conservatively set at 85W. The regular 3D-PDN power grid is modeled based on guidelines provided in [81]. With 20 cores on each tier, a total of 320 input power pins are used with n^2=16 grid-points for each core. Nominal (initial non-aged) values of branch resistances are assumed to be 50mΩ [81], with 25 μm^2 cross-sectional area.

For our circuit-aging calculations, we assume a nominal effective V_T of 0.3V for un-aged cores and routers. In our combined cost function calculations (Ψ in equation (7)) for the aging-aware region-selection heuristic, we use α=β=0.5 (empirically derived to achieve the longest lifetimes); *max_IR_drop* and max_V_T are set to 0.3V and 0.5V respectively, based on equation (1), with operating frequency requirement of 2GHz. In our *SAR* heuristic, we use δ=0.6, to calculate the value of β_R, for an appropriate trade-off between application performance and aging. In our experiments, an epoch interval can range between 25 to 35 days, depending on the power profile, execution-times, and average DoPs of the application workload, as well as the degree of aging in the chip. Given the relatively slow rate of aging, such an aging-measurement interval has been found to be appropriate for runtime frameworks . Also, as the overheads incurred due to employing aging sensors have been reported to be quite small (power dissipation of 84.7nW, sensing-latency of 100μs, and area of 77.3μm^2 per sensor at 45nm technology node) in [69], we ignore them in our calculations.

2.6.2 EXPERIMENTAL RESULTS

Our experiments compare three variants of the proposed *ARTEMIS* framework with two other runtime mapping approaches derived from prior work. These prior works are designed for 2D CMPs, so we extend them to 3D CMPs for a fair comparison. To investigate the effectiveness of the circuit-aging (leakage) and PDN-aging aware region-selection, voltage-selection, and mapping techniques, we adapt our *ARTEMIS* framework to use an *XYZ*-routing scheme (*ARTEMIS-XYZ*) and compare the results obtained with two other run-time mapping techniques that selects contiguous regions for mapping and use the same *XYZ* routing scheme: *(i) traditional worst-case guard-banding approach (WC-GB):* In this approach, region selection is done based on the runtime area constrained mapping approach from [67] that attempts to fit the maximum number of applications on the chip. To satisfy the application-performance requirements for an extended period of time, a high V_{dd}=1.0V is used at all times. This framework selects contiguous regions for mapping, to maximize the performance, and minimize the communication latency. However, it does not assume runtime inputs from aging-sensors to make mapping decisions and thus is not aging-aware; *(ii) wear-leveling approach with DVS (WL+DVS):* In this approach, contiguous region-selection for application-mapping is always done based on the lowest average V_T-degradation in cores, as proposed in [43], [49]; in addition, V_{dd} is opportunistically reduced when possible and adaptively hiked with aging to meet application performance constraints.

Additionally, we adapt our *ARTEMIS* framework to use an aging- and congestion-aware routing scheme *(ACR)* obtained from prior work in [35]. We also include results for our *ARTEMIS* framework with the proposed symmetric aging-enabled routing *(SAR)* scheme. Thus, the comparison between *ARTEMIS-XYZ, ARTEMIS-ACR*, and *ARTEMIS-SAR* allows us to determine the most effective 3D NoC routing approach that can help improve lifetime in 3D NoC-based

CMPs while meeting application performance and chip-wide power constraints. Finally, to test the efficiency of the *ARTEMIS* framework, all the experiments are conducted when the workload is high at a uniform inter-application arrival rate of ~1.5s while each application executes for few seconds on the 3D CMP.

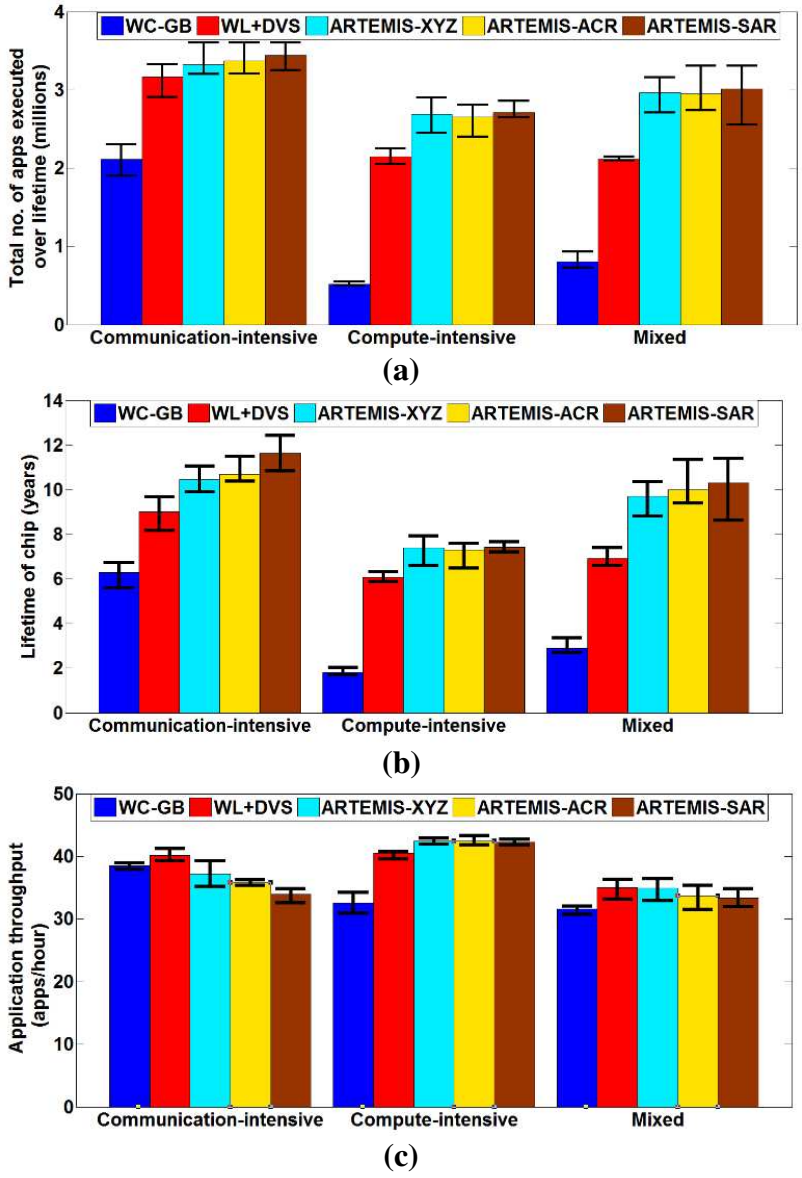


Figure 14 Results comparing *ARTEMIS* framework variants with other approaches from prior work, for workloads that combine various SPLASH-2 and PARSEC benchmarks: (a) Total number of applications serviced over lifetime, (b) lifetime (years), (c) application-throughput over lifetime (applications/hour)

Figure 14 (a) shows the total number of applications serviced over the chip lifetime, Figure 14 (b) shows total CMP-lifetime (total system-execution-time), and Figure 14 (c) shows the application-throughput extracted over the service-life of the CMP for all the compared frameworks, across the three different types of application-input-sequences. The error-bars shown in Figure 14 in all our plotted results represent the range of results across simulations with five different randomly generated application-sequences (with individual applications in the sequence derived from the SPLASH-2 and PARSEC benchmark suites, as discussed earlier). As expected, the WL+DVS framework outperforms the WC-GB approach that does not perform DVS. By intelligently selecting application-regions on the 3D-die with its region-selection heuristic, our *ARTEMIS* frameworks (*ARTEMIS-XYZ, ARTEMIS-ACR*, and *ARTEMIS-SAR*) achieve a notable reduction in leakage-power dissipation and reduce stress on the more highly degraded PDN-paths. The *ARTEMIS* frameworks produce 9%–40% (25% average) improvement in the total number of applications serviced over the next best framework, WL+DVS, as well as significant improvements in total CMP-lifetime, as can be seen from Figure 14 (a)-(b). For communication-intensive applications, we observed far less percentage of dark-silicon, approximately 0%–15% (depending on V_{dd}-levels and workload profiles), compared to compute-intensive applications where dark-silicon is approximately 10%–33%. A lower percentage of dark-silicon is indicative of more active cores running with less stress, whereas a higher percentage of dark-silicon indicates fewer active cores that are running with greater stress.

Thus, communication-intensive applications experience less aggressive aging (because of their lower %dark-silicon), which results in more of these applications being executed over the chip lifetime and a higher lifetime compared to compute intensive applications. Figure 14 (a)-(b) corroborate this observation. Also, most communication-intensive applications generate relatively

low current-densities in the PDN, i.e., PDN-degradation is slower relative to circuit-degradation, which limits the improvements obtained by the *ARTEMIS* frameworks for such applications, as can be observed from Figure 14 (a).

Next, we present an analysis of lifetime improvements obtained when our proposed symmetric aging-aware routing path allocation *(SAR)* heuristic is used with *ARTEMIS* (*ARTEMIS-SAR*), compared to the *ARTEMIS-XYZ* and *ARTEMIS-ACR* frameworks. Our *SAR* heuristic enables better balancing of aging between compute-cores and their associated NoC-routers. While executing communication-intensive workloads exclusively, where the rate of aging in routers is comparable to that of core-aging, SAR minimizes the number of runtime scenarios when mapping of an application is stalled due to aged routers, thereby extending the system-lifetime. Observe in Figure 14 (a) that *ARTEMIS-SAR* produces notable improvements in number of applications executed over lifetime, compared to *ARTEMIS-XYZ* (by 4%) and *ARTEMIS-ACR* (by 2.2%) with comparable application-throughput (as shown in Figure 14 (c)), for communication-intensive workloads. However, the choice of routing scheme has very little effect on lifetimes for compute-intensive and mixed workloads, where NoC-aging does not determine the service-life of the chip.

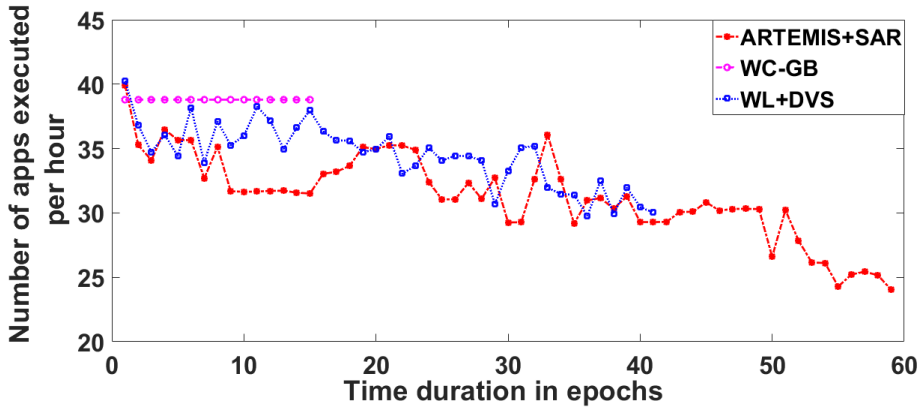
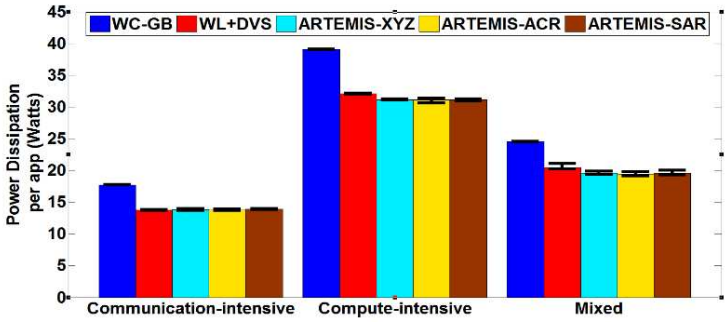


Figure 15 Results showing the comparison of application throughput of *ARTEMIS+SAR*, WL+DVS, WC-GB over their respective CMP lifetimes

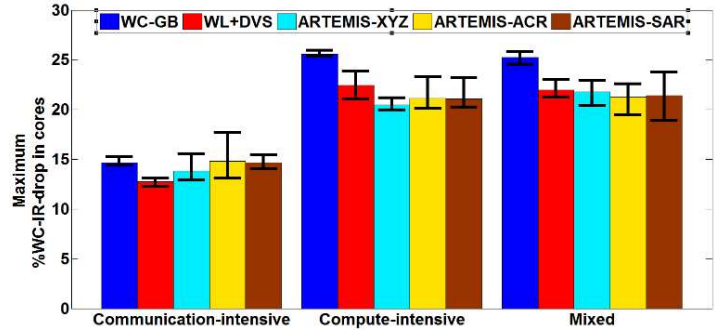
To further analyze how the behavior of different frameworks change over time, we show how application throughput varies across time in different frameworks in Figure 15. We consider *ARTEMIS* with *SAR* routing scheme and a mixed workload in this experiment. It can be seen from Figure 15 that WC-GB maintains a constant application throughput, while WL+DVS and *ARTEMIS+SAR* tradeoff throughput for an extended CMP lifetime. WL+DVS and *ARTEMIS+SAR* employs intelligent aging-aware application mapping for a graceful degradation of the CMP, and utilizes DVS to satisfy the application minimum frequency and DS-PB constraints. *ARTEMIS+SAR* further benefits from PDN-aging-aware intelligent mapping scheme that extends the useful lifetime of the CMP beyond that of WL+DVS, meanwhile achieving similar throughput.

We also show experimental results related to the power dissipation, PDN performance, and V_T degradation profile on the 3D CMP when using different mapping frameworks, in Figure 16. A comparison of the average power-dissipated per application over the chip lifetime is shown in Figure 16 (a). As expected, WC-GB framework, which does not utilize DVS, dissipates significantly more power. The leakage-optimizing mapping in *ARTEMIS* results in up to a 5.5% improvement for compute-intensive workloads (2.8% on average for all workloads) in total power/application over WL+DVS. We also analyze the distribution of percentage worst-case IR-drops (%WC-IR-drops) at the end of lifetime with different frameworks. Figure 16 (b) shows the maximum %WC-IR-drops obtained for different frameworks at the end of chip lifetime. The aging-unaware WC-GB framework maps applications such that some cores are more heavily loaded than others, thus resulting in the shortest lifetimes with high maximum WC-IR-drops. With our strategy to prioritize mapping on cores with less WC-IR-drops, *ARTEMIS* frameworks produce

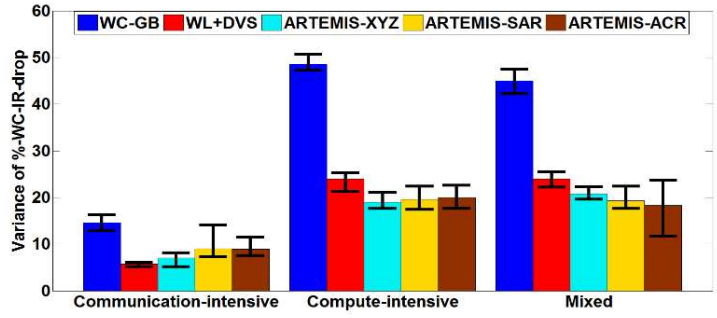
lower maximum %WC-IR-drop-values (by up to 9% lower), compared to WL+DVS, despite *ARTEMIS* having a longer lifetime and servicing a higher number of applications.



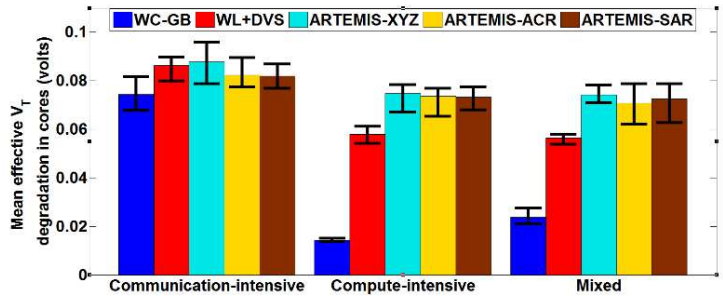
(a)



(b)



(c)



(d)

Figure 16 Results showing improvements for our circuit-aging (leakage) and PDN-aging aware region selection and Vdd selection heuristic in the ARTEMIS frameworks: (a) power dissipation per application, (b) maximum %WC-IR-drop at end of lifetime, (c) Variance of

%WC-IR-drop at end of lifetime, (d) Mean effective VT-degradation in compute-cores at end of lifetime

Figure 16 (c) shows the variance in the WC-IR-drop-distribution on the 3D chip obtained at the end of lifetime with different frameworks. A smaller variance of IR-drops with *ARTEMIS* frameworks (up to 24% lower compared to WL+DVS) signifies efficient management of PDN-aging that aides in improving the longevity of the PDN, and thus the entire chip. Figure 16 (d), shows the mean effective V_T-degradation in compute-cores at the end of lifetime, and provides additional insights into the lifetime improvements obtained with our circuit-aging (leakage) and PDN-aging aware region selection and V_{dd} selection heuristic. As discussed earlier, the V_T-values of circuit components increase with aging. Given the *nominal-V_T* of 0.3V at the start of lifetime, observe in Figure 16 (d) that the mean V_T-degradation values at the end of lifetime for *ARTEMIS* frameworks are significantly higher (by up to 30% for compute-intensive workloads) compared to the WL+DVS framework. By restricting the EM-induced PDN-degradation, *ARTEMIS* can extend the tolerable degree of circuit-aging (V_T-degradation) in compute-cores, while meeting the same performance constraints. Thus, the 3D CMP remains functional for much higher V_T-degradation with *ARTEMIS* compared to other approaches.

To obtain a more comprehensive understanding of the performance of the compared frameworks, we present snapshots of the 3D-CMP die at the end of lifetime, when using different frameworks. Figure 17 shows average values of V_T-degradation and figure 8 shows maximum WC-IR-drops in cores at the end of CMP lifetime after executing compute intensive workloads. We have only considered compute intensive workloads for this analysis because V_T-degradation and IR-drops due to these workloads are higher in cores due to their higher power dissipation (compared to mixed and communication-intensive workloads).

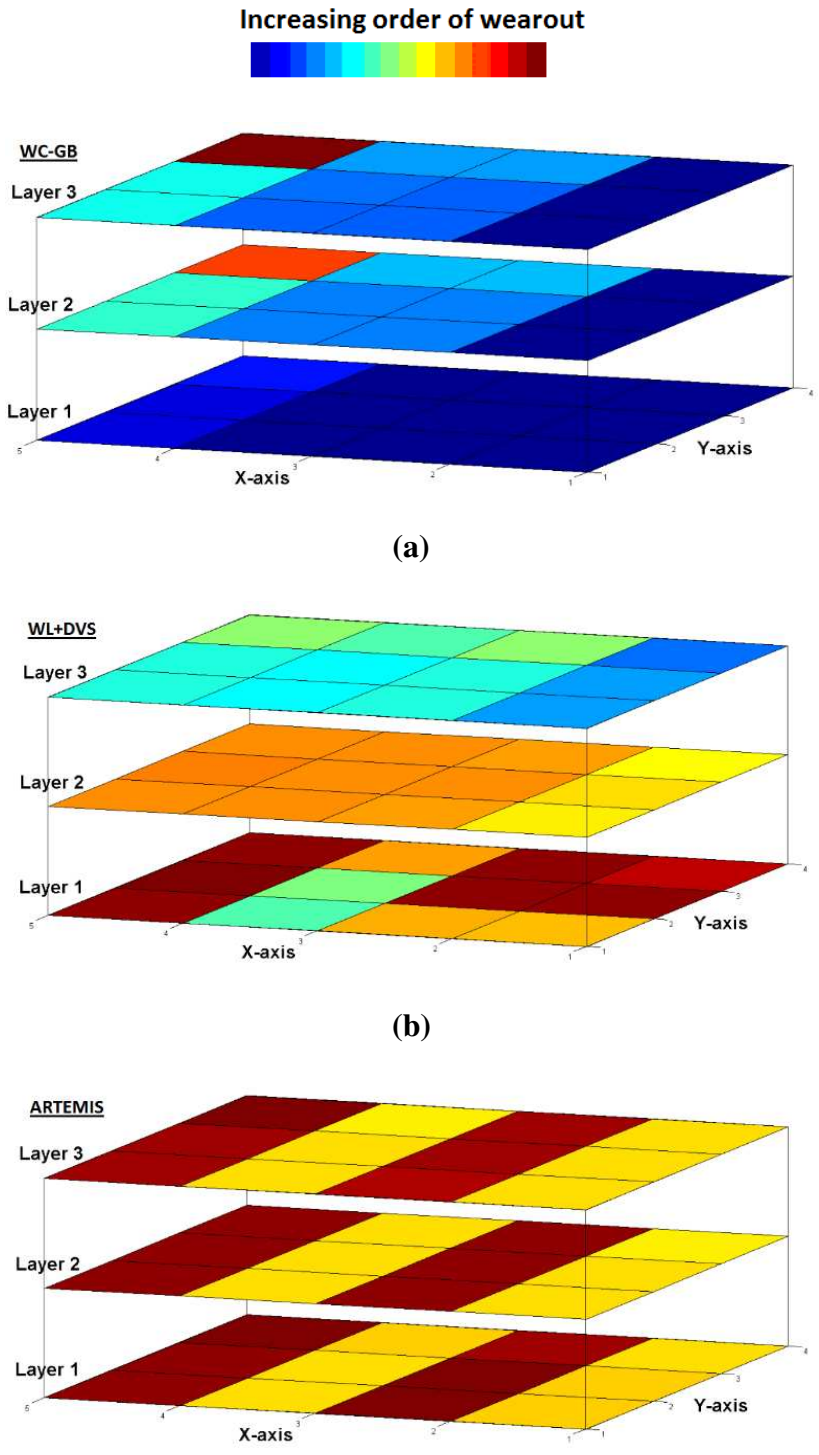


Figure 17 Surface plots showing the effective V_T degradation in cores of a 3D CMP at the end of their respective lifetimes using: (a) worst case guard-banding (WC-GB) technique, (b) wear leveling with DVS (WL+DVS) technique, and (c) proposed *ARTEMIS-SAR* framework.

Figure 17 (a), Figure 17 (b), Figure 17 (c) show the average V_T-degradation observed in each core at the end of CMP lifetime for WC-GB, WL+DVS, and *ARTEMIS-SAR* frameworks. We consider the end of lifetime as the condition when application performance constraints are not met at any CMP V_{dd} level, without violating the chip's dark-silicon power budget (DS-PB) constraint, and when no other application is currently running on the CMP.

Figure 17 (a) shows that when WC-GB fails, most of its cores are comparatively less aged than that of WL+DVS in Figure 17 (b) and *ARTEMIS* in Figure 17 (c). This is because WC-GB ambitiously tries to map the applications at a very high V_{dd} repeatedly on to the same, rapidly aging cores. Also, the PDN degradation in WC-GB is much higher than the other two frameworks in Figure 18 (a). As the degradation in PDN worsens, application performance constraints are not met within the allowed set of CMP V_{dd} levels. This results in end of lifetime condition in WC-GB, because when application-performance constraints are not met in a region, WC-GB does not exhaustively search for other mapping regions on the CMP even if no other applications are running simultaneously.

Figure 17 (c) shows that by using *ARTEMIS*, the chip has more degraded cores at the end of the lifetime compared to the other two frameworks. This indicates that all the cores have been well utilized by ARTEMIS till the time it failed. Better management of EM-degradation in PDNs (which is explained using Figure 18), and trading off application throughput for lifetime by *ARTEMIS* (shown in Figure 15) helps to extend the functional lifetime of each individual core on the 3D CMP till the end of its lifetime.

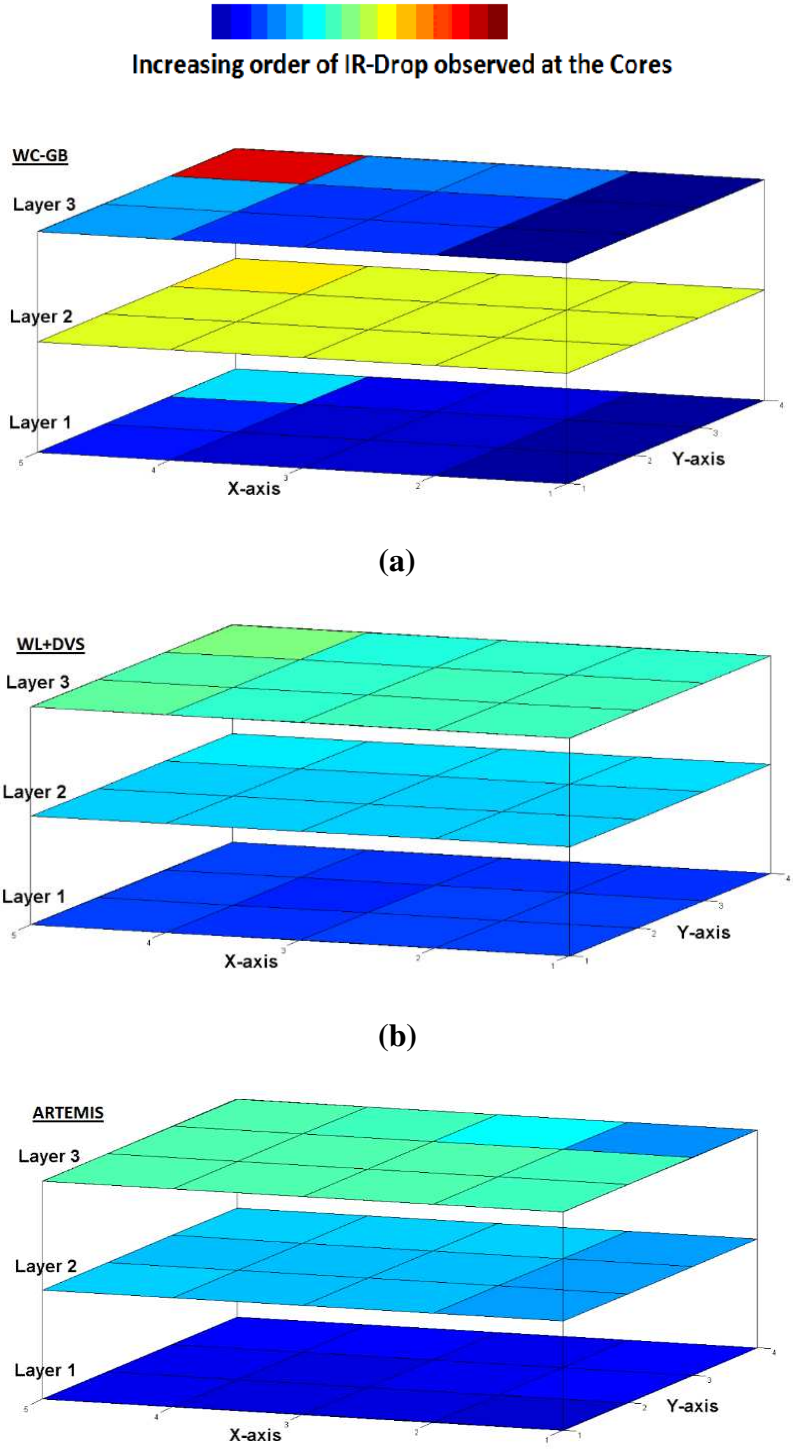


Figure 18 Surface plots showing the Worst-Case IR drops observed on each layer of a 3D CMP at the end of their respective lifetimes using: (a) worst case guard-banding (WC-GB) technique, (b) wear leveling with DVS (WL+DVS) technique, (c) proposed ARTEMIS framework.

Also, note that with *ARTEMIS*, cores are degraded in a pattern that is beneficial for contiguous mapping of applications till the CMP fails, unlike with WC-GB and with WL+DVS that leave fragmented regions of cores with varied degradation profiles. Figure 18 (a), 18 (b), 18 (c) show the Worst-Case-IR-drops (WC-IR-drops), due to EM of PDN, observed at the end of the CMP lifetime. The colors at each tile give a comparative visualization of IR-drop values logged at the input pins of the cores. From figures 18 (a), 18 (b) and 18 (c), it can be seen that WC-GB is completely unaware of PDN aging while mapping, and hence has some hotspots (red and brown tiles) due to excessive mapping of tasks on to the same tiles, even when other tiles are free and less degraded. WL+DVS show slightly higher WC-IR-drops to *ARTEMIS* at the end of the CMP lifetime. By integrating PDN-awareness and simultaneously managing Vdd and mapping regions intelligently, *ARTEMIS* makes it possible to use the CMP well beyond the V_T-degradation values (with similar PDN-degradation) observed using prior works (as shown in Figure 17(a), 17 (b), 17 (c)).

The aging aware mapping heuristic WL+DVS that is unaware of PDN degradation, tries to map applications on to tiles with less circuit degradation. Hence, the PDN ages at a faster rate in WL+DVS compared to *ARTEMIS*. But, as PDN degradation gets higher towards the end of the lifetime, applications do not meet their frequency constraint, and force a CMP V_{dd} hike at the time of mapping. Also, cores tend to dissipate more power as they get older, leading to DS-PB violation. This results in CMP reaching the end of lifetime condition faster. Hence PDN-aging is crucial for achieving longer lifetime in 3D CMPs. *ARTEMIS* is thus able to manage both circuit and PDN aging to extract more work out of the CMP in its lifetime.

2.7. CONCLUSIONS

In this chapter, we proposed an aging-aware application-mapping and DVS scheduling framework (*ARTEMIS*) that considers PDN-aging of 3D NoC-based CMPs in addition to circuit-aging (in NoC routers and cores) in both the performance and aging evaluation stages, and the lifetime optimization methodology. We have considered the analysis of *ARTEMIS* framework in a highly-constrained system with variable application execution time. Compared to a framework based on the best known prior work on aging-aware mapping techniques, *ARTEMIS* can service 25% more applications (on average) over the chip lifetime, which highlights its promise for emerging 3D-CMPs. As part of future work, we plan to explore support for variable process variations, and consider a service queue model that includes wait time of an application, for further improvements within our framework.

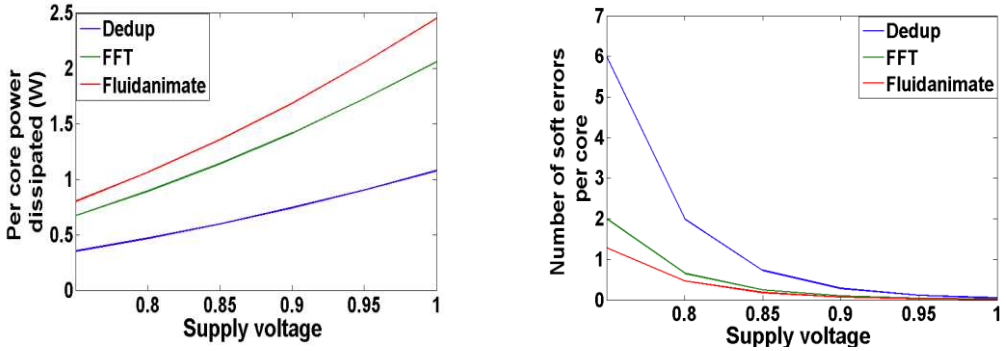
3. CHARM: A CHECKPOINT-BASED RESOURCE MANAGEMENT FRAMEWORK FOR RELIABLE MULTICORE COMPUTING IN THE DARK SILICON ERA

With increasing transistor miniaturization, circuit densities have drastically increased, and the critical charge, which is the minimum charge capable of a bit-flip in a memory- or a logic-cell, has significantly decreased [82] [18]. This phenomenon has caused newer process technologies to become more susceptible to transient-faults due to the effects of radiation, e.g., alpha-particle and neutron strikes. Simultaneously, circuit-aging due to phenomena such as Bias Temperature Instability (BTI) and Hot Carrier Injection (HCI) is becoming prominent for systems manufactured at technology nodes of 45nm and below [11] [40]. The principal effect of such a circuit-aging mechanism is to increase circuit-threshold voltage (V_T), which results in higher circuit-delay in cores and routers, thereby limiting overall system performance. At the same time, with the extent of dark-silicon increasing every technology-generation (30-50% for 22nm) [47] [48], chip multiprocessor (CMP) designs are becoming increasingly power-limited rather than area-limited.

Figure 19 highlights the intricate inter-dependence between various optimization metrics (power, performance, reliability), design-knobs (voltage, app-DoP, task-to-core mapping) and their effects on physical phenomena (soft-errors, circuit-aging). Figures 19 (a) and 19 (b) show the plots of supply voltage (V_{dd}) versus average core power consumed, and the number of soft-errors observed (we model soft-errors using the approach in [62]), for three applications of varying memory intensities and fixed DoPs. In modern power-constrained designs with increasing levels of dark-silicon, operating at lower voltages, and hence lower frequencies, can reduce the average power consumption of the application as shown in figure 19 (a). Additionally, low power techniques can potentially reduce the rate of aging on the die thereby extending useful lifetime of

the chip. However, the soft-error rate (SER) exponentially increases when we reduce the rate of circuit-aging with DVS, as shown in figure 19 (b).

Given a fixed power budget, power savings translate to higher available power-slacks on the CMP-die thereby yielding higher performance by either simultaneously executing additional applications or running applications at higher app-DoPs. Recent works such as [83] have shown that by varying the degree of parallelism (DoP) of applications at runtime to adapt to the execution environment of the CMP, significant benefits can be achieved in terms of application service-times and power dissipation. Moreover, application-DoP (app-DoP) also impacts the application soft-error reliability, aging footprint, and chip power budget. However, higher app-DoP further increases the probability of soft-errors during the execution of an application. Even with sophisticated re-execution techniques using checkpointing and rollback, such as those proposed in [84], the occurrence of soft-errors can incur significant overheads in terms of power dissipation and aging footprint – the very metrics that were expected to improve by employing low-power DVS techniques.



(a) **(b)**

Figure 19 (a) Per-core power consumed by three applications of varying memory intensities at different supply voltages (Vdd); (b) Soft-errors observed per-core, when executing applications at different supply voltages

To solve this intricate problem, in this chapter, we propose a novel system-level runtime soft-error and lifetime-reliability aware resource management framework (*CHARM*) that employs dynamically adaptable application degrees of parallelism (app-DoP), together with intelligent application mapping and DVS strategies to maximize the number of applications serviced over the target lifetime of a CMP, while meeting the chip-wide dark-silicon power budget (DS-PB) and application performance deadlines. For applications to recover from runtime soft-errors, we also integrate an error checkpointing and rollback scheme within our framework. Experimental analysis shows that *CHARM* not only achieves transient fault resilience, but also achieves an improvement of up to 2.5× in CMP lifetime, and up to 6× in number of applications executed over CMP lifetime compared to the state-of-the-art. Our novel contributions in this chapter are summarized as follows:

- we propose a novel runtime framework (*CHARM*) for application mapping and DVS that can adapt to different aging profiles of a chip and maximize the number of applications that meet their deadlines in the presence of soft-errors, over the chip lifetime;
- *CHARM* manages dynamically arriving applications by varying their application-DoPs as well as Vdd and execution frequency, based on queue pressure and app-slack time, to minimize checkpointing and rollback overheads, and also to minimize the aging footprint;
- our methodology of evaluating maximum attainable performance, in the presence of soft errors and system aging, accounts for computing the execution time overhead due to checkpointing and rollback recovery, as well as V_T degradation over the lifetime of the chip;
- we also propose a novel aging aware network-on-chip (NoC) routing path allocation scheme that balances the core-router aging profile;

- our framework does not violate chip-wide power constraints while mapping an application, thus finding applicability in contemporary power-constrained (dark-silicon afflicted) CMP designs.

3.1 RELATED WORK

Several efforts (e.g., [85] [86] [87]) have proposed design-time fault-tolerant task scheduling frameworks that assume fault-detection mechanisms implemented on the multicore platform. Hardening techniques such as task re-execution and replication are utilized in these works to probabilistically meet task-completion deadlines for real-time tasks of varying criticalities, while minimizing energy. However, these efforts do not consider circuit-aging due to BTI, HCI stress, dark-silicon challenges, runtime adaptation support, or application DoPs.

In recent years, some researchers have proposed runtime and design-time techniques to address the problem of circuit-aging in CMPs. For example, Tiwari et al. [43] propose mapping high-power tasks onto faster (less-aged) cores and low-power tasks onto slower cores, thereby "hiding" the aging on the chip. They also propose to mitigate aging by scaling the supply voltage or the threshold voltage. Kapadia et al. [36] perform application mapping and DVS scheduling on the CMP while considering circuit-aging in cores and EM-aging in the power delivery network (PDN). Such an approach where tasks are mapped to cores without considering application-performance requirements would lead to higher number of applications being dropped and increase waiting time in the service queue. Also, these works neither perform soft-error management nor consider the benefits of adaptable app-DoPs.

A recent work [83] on runtime application scheduling for CMPs employs dynamically adaptable application DoPs to minimize average application service times and energy, while meeting a chip-wide power budget and soft-error-reliability constraints. However, the work ignores the circuit-aging phenomena and does not perform any lifetime optimization. Moreover, the work does not consider error recovery or task re-execution mechanisms for reliable system operation, and also does not emphasize satisfying completion deadlines of applications.

To overcome the abovementioned concerns, we propose a novel runtime resource management framework *(CHARM)* that, for the first time, considers the combined effects of runtime management techniques on lifetime-reliability and soft-error reliability of the CMP, to maximize the number of applications meeting their completion-deadlines, given a fixed CMP target-lifetime in years.

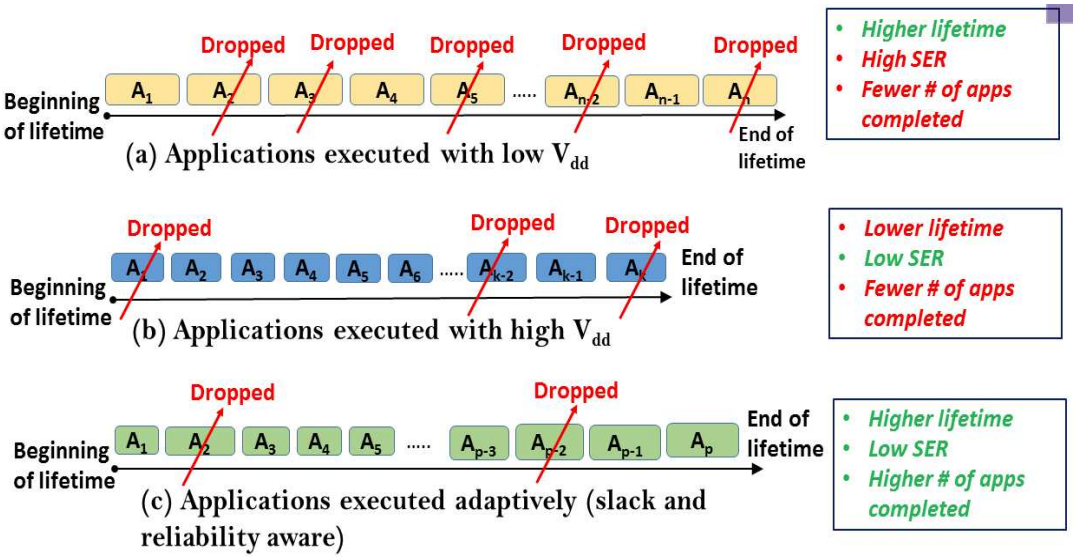


Figure 20 Motivation example of runtime application-scheduling using (a) representative of prior works [36], [83] where low V_{dd} is used to preserve chip lifetime; (b) where high V_{dd} is used to reduce SER probability; (c) where the V_{dd} and DoP are varied adaptively to reduce SER and maximize the number of completed applications within a target CMP lifetime.

3.2 MOTIVATION

In this section, we illustrate the advantages of our proposed *CHARM* framework with the help of a small example. We consider three scenarios in which applications arrive at runtime at a service-queue to be mapped onto the CMP. We call the chip as *aged* when the cores get slower due to degradation in threshold voltage V_T. An application can only tolerate up to a certain number of soft-errors beyond which the overheads of the recovery mechanism surpasses the available slack-time. When this happens, that application is *dropped* from the service queue without execution. Prior works [43] [36] try to minimize aging footprint of applications by mapping them at low V_{dd} and using DVS scheduling to increase chip lifetime without effecting the performance, which is shown in Figure 20 (a). But they do not consider the effects of DVS and DoP on soft-error rate (SER). SER increases exponentially with the decrease in supply voltage-V_{dd}, as given in equation (14) in Section 3.3.A. Hence the chip encounters higher SER and higher number of dropped applications. Intuitively, scheduling applications with high V_{dd} as shown in Figure 20 (b) reduces SER along with the application execution times. However, this leads to much lower chip lifetime and considerably lower number of applications serviced over the lifetime.

In contrast, *CHARM* dynamically selects combinations of {DoP, voltage/frequency, mapping region} for the applications based on *app-slack time*, *queue pressure,* and *aging profile* to jointly minimize run times and maximize number of applications that meet their deadlines, as shown in Figure 20 (c). This leads to lower SER overhead and more applications meeting their deadlines within the power budget. As the chip gets *aged, CHARM* extends chip lifetime by mapping applications with lower voltage/frequency and higher DoP. This is done amidst a rise in SER probability, to scale down the power consumption and temperature footprint of the chip and also minimize aging of cores. The proposed approach with *CHARM* leads to more cores being active

towards the end of the target chip lifetime, resulting in more number of applications meeting their deadlines over the lifetime.

3.3 PROBLEM FORMULATION

A. Models for Performance and Reliability Estimation

We begin by modeling the maximum frequency of a core based on its supply voltage as given in Eq. (10).

$$\boldsymbol{f_{max} = \frac{\mu(V_{dd}-V_T)^{\alpha}}{C_0.V_{dd}}} \quad \ldots\ldots (10)$$

where, V_{dd} and V_T are the supply voltage and effective threshold voltage of a core, α and μ are technology-dependent constants, and C_0 is switching capacitance of the critical path [62].

We model circuit-aging effects arising from BTI and HCI-induced circuit degradation, as these have been found to be the most dominant aging mechanisms in emerging semiconductor technologies. Velamala et al. [63] [64] have shown that a Trapping/Detrapping (TD) based BTI model is capable of accurately predicting the degradation under a sequence of V_{dd}'s used in the DVS operation. They have noted that when the supply voltage is changed from a higher V_{dd} to lower V_{dd}, the circuit undergoes recovery. Therefore, our analysis of circuit-aging over the CMP-lifetime is based on the long-term aging prediction model proposed in [63], which accounts for different V_{dd}-levels over time. We estimate the effective ΔV_T increase that a component (core or NoC router) experiences over a time-interval of t using Eq. (11), where A, B and C are fitting parameters that are constant.

$$\boldsymbol{\Delta V_T(t) = L.[A + Blog(1 + Ct)]} \quad .. (11)$$

$$\boldsymbol{L = K_1 . \exp\left(\frac{-E_0}{kT}\right) . \left\{ exp\left(\frac{\beta V_1}{T_{ox} kT}\right) . \alpha_1 + \cdots + exp\left(\frac{\beta V_S}{T_{ox} kT}\right) . \alpha_S \right\}} \quad .. (12)$$

In equation (12), V_i is the i^{th} V_{dd}-level utilized by the component for a time-duration α_i, S is the total number of allowable V_{dd}-levels, and $\{\alpha_1+\alpha_2+...+\alpha_S\} \leq t$. T is the average temperature of the component during corresponding α_i. We use parameter-values in equations (11) and (12) from [63] and [64] that are validated against real silicon data.

HCI occurs because of the irreversible deposits of charges (holes/electrons) generated in regions such as the gate oxide, over the lifetime of the transistor. This phenomenon also leads to degradation of the threshold voltage (V_T) which can be modeled as a function of stress-time [88]. The degradation model for V_T from [89] is:

$$\boldsymbol{\Delta V_T = A . t^m} \quad ..... (13)$$

where, A and m are technology dependent parameters, and t is the time under which a component is under stress. We consider m to be equal to ~0.5 which is accepted over a wide range of processing technologies.

Transient faults are the bit flips in logic devices due to transient phenomena such as charged alpha particle strikes. We model the soft error-rates (SER) as discussed in [90]. We define $\lambda(f)$, as the SER at a given frequency f by the equation below:

$$\boldsymbol{\lambda(f) = \lambda_0 . 10^{d.\left(\frac{1-f}{1-f_{min}}\right)}} \quad ..... (14)$$

where λ_0 is the SER corresponding to the highest frequency value (f_{max}). For compute cores we consider $\lambda_0 = 10^{-6}$ errors/sec and assume d=3 [83]. For NoC routers, we consider $\lambda_0 = 10^{-6}/3$ errors/sec as a router's area is approximately one third that of a core's area in the CMP platform we analyze in this work. We compute the probability of one or more faults occurring over an execution period τ using equation (15):

$$\boldsymbol{P(f,\tau) = 1 - e^{-\lambda(f).\tau}} \qquad \text{..... (15)}$$

$$\boldsymbol{E(f,\tau) = \int_0^{\tau} \tau.P(f,\tau).d\tau} \qquad \text{..... (16)}$$

Equation (16) gives the expected number of faults $E(f,\tau)$ observed in a given time interval [0, τ] during which *f* remains constant. We compute the number of faults in any given interval [τ_1, τ_2] as follows:

$$\boldsymbol{E[\tau_1,\tau_2] = E(f,\tau_2) - E(f,\tau_1)} \qquad \text{..... (17)}$$

We employ a checkpoint and rollback based error recovery mechanism as proposed in [91]. The number of checkpoints employed for a task is a function of its worst case execution time *L* and its deadline *D*. Using this technique, fault-free execution time of an application involves two overheads: checkpoint time-(C_i) and time to sanity check the processor state-(S_i). Occurrence of a failure further incurs time to retrieve the latest saved state and re-execution time-(R_i). The following equations show the overheads as a function of number of checkpoints:

$$\boldsymbol{T_i(n) = T_i + n.C_i + (n+1).S_i} \qquad \text{..... (18)}$$

$$\boldsymbol{F_i(n) = T_i + R_i + k.\left(\frac{C_i}{n+1}\right) + S_i} \qquad \text{..... (19)}$$

Equation (18) gives the fault-free execution time of a task *i* with *n* checkpoints ($T_i(n)$) where, T_i is the ideal execution time, $n.C_i$ represents checkpoint time and $(n+1).S_i$ represents sanity check time. Equation (19) gives the execution time of a task *i* in the presence of *k* faults ($F_i(n)$) The term $k.(C_i/n+1)$ represents the re-execution time for *k* faults. For our system, we tolerate up to three faults, *k*=3, as we have observed that the overhead of tolerating more faults per task in applications we analyze surpasses the available slack-time in the best case. *CHARM* decides *n* based on the deadline D_i of the task graph to be mapped. equation (20) gives the optimum number of checkpoints n_i assigned to a task *i*:

$$\boldsymbol{n_i \leq 2.\frac{C_i}{D_i - T_i - R_i - S_i} - 1} \qquad \text{..... (20)}$$

where, D_i is the deadline of the task i, T_i is the fault free execution time, R_i is the re-execution overhead and S_i is the sanity check overhead. The periodic checkpointing interval duration for each task i is thus T_i/n.

B. Inputs, assumptions and problem objective

We assume the following inputs to our problem:

- A 2D mesh NoC-based CMP of dimension $\{dim_x, dim_y, dim_z\}$ and number of tiles $N=dim_x \times dim_y \times dim_z$; each tile has a core and a router;
- A chip-wide dark-silicon power budget (DS-PB);
- An application task graph for each application; vertices with task execution-times on compute cores and edges with inter-task communication volumes; error-free execution times and volume values are assumed available from offline profiling;
- A set of candidate supply voltages $(V_{dd}) = \{V_1, V_2. \ldots V_n\}$ for each core; maximum frequency of a core for this V_{dd} is given by equation (10);
- Application task graphs for the set $P = \{P_1, P_2, \ldots P_\eta\}$ of DoPs for all applications; an application i possesses $|P_i|$ viable DoPs; an application has a maximum DoP value beyond which performance does not improve (or gets worse due to high synchronization overheads) - such sub-optimal DoP values are ignored;
- A set of permissible rectangular shapes of regions that an application can be mapped on to $\{B_1 \ldots B_n\}$; e.g., a set $\{2\times4, 4\times2\}$ for an application with DoP=8.

We make the following assumptions in our work:

- There exists one-to-one mapping between tasks and cores, i.e., a core can execute only one task (thread) at any given time;
- Applications are mapped contiguously on non-overlapping rectangular shaped regions of the 2D CMP for inter-application isolation and optimized communication-profiles (as recommended in prior works such as [67]); thread-migration is not considered;
- Per-core granularity of DVS is considered, to meet DS-PB and application performance demands, and facilitate runtime selection of DoP for the applications to avoid execution deadline violations;
- Similar to prior works [43] [36] we assume the presence of on-chip sensors to detect the V_T degradation along the critical path circuits of cores and routers due to aging and send that information to our framework; we also assume the presence of an on-chip error detection mechanism to detect soft-error events in cores and routers and initiate the application re-execution process from a checkpoint;
- On-chip aging sensors monitor the runtime V_T values of individual cores and routers at the end of each *epoch* and send the values to our framework, to enable smart mapping decisions; an *epoch* is defined as a time-period during which the aging profile of the chip is assumed to be constant or does not grow significantly;

Objective: Given the above inputs and assumptions, the objective of our *CHARM* framework is to dynamically determine application-specific mapping (region selection, task-to-core mapping, and NoC routing), DoP values, and checkpoint periods, as well as a per-core DVS schedule, to

maximize the number of applications that meet their execution-deadlines, while satisfying chip-wide DS-PB and tolerating up to *k* transient faults per application over a given chip target lifetime.

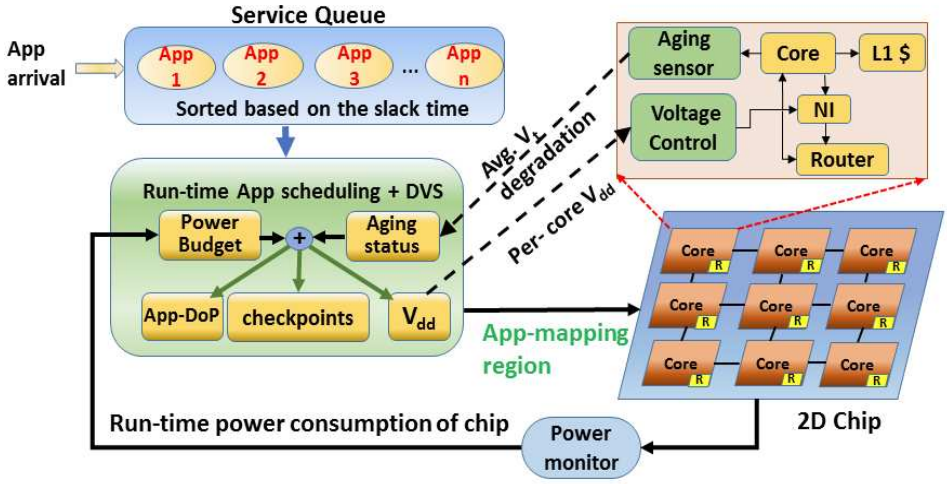


Figure 21 Overview of *CHARM* runtime app-DoP selection, reliability aware mapping, and DVS scheduling framework.

3.4 CHARM FRAMEWORK: OVERVIEW

The key aspects of our proposed framework are illustrated in Figure 21. *CHARM* makes decisions based on the runtime input it receives from on-chip aging sensors and *app-slack time* = {worst case execution time – deadline} available for an application waiting in the queue. *CHARM* intelligently prioritizes between lifetime and performance based on the available app-slack time and the chip degradation profile. *CHARM* dynamically selects the DoP, checkpoint period, and per-core V_{dd}, for an application's execution, based on runtime inputs and available slack. *CHARM* operates as two nested loops, *(i)* circuit-aging, lifetime and epoch management (Figure 22 (a), outer loop); and *(ii)* reliability-aware application mapping, DVS, and application-DoP selection (Figure 22 (b), inner loop). These two components are discussed in detail next.

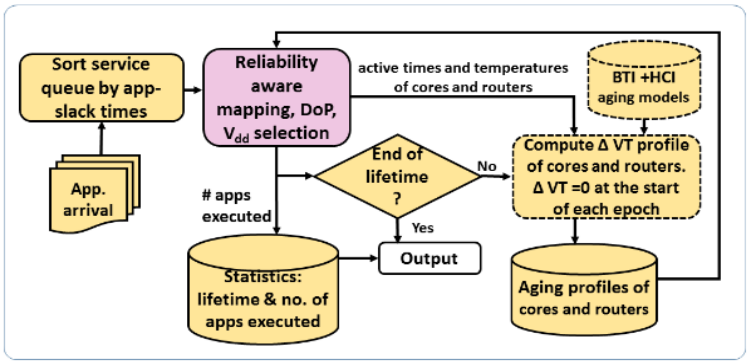
A. Circuit-aging, lifetime, and epoch management

As discussed earlier, the lifetime of a chip is divided into epochs. In each epoch, applications arrive to the CMP for execution. *CHARM* maps them immediately or keeps them in a service queue for mapping later. The applications waiting in the service queue are sorted at every occurrence of an *app-event* in the increasing order of their app-slack times. An app-event is defined as either the arrival of a new application to the CMP or the exit of a mapped application from the CMP. At an app-event, *CHARM* successively removes applications from the service queue and maps each one of them until there are insufficient number of consecutive idle cores on the chip to execute an application without violating the application deadline and the chip DS-PB constraints.

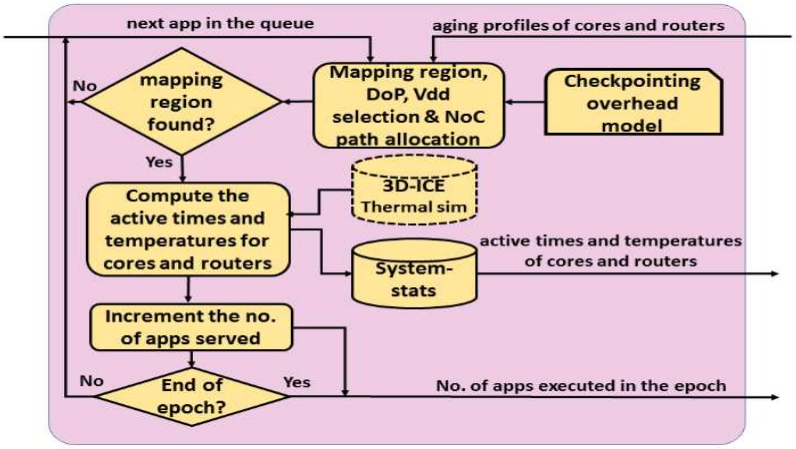
The inner loop performs reliability-aware mapping, app-DoP selection, and V_{dd} selection during an epoch are discussed in section 3.4.B (shown as the pink box in Figure 22 (a), with an expanded view in Figure 22 (b)). The output of this inner loop at the end of the epoch provides information about the activity on the chip over the epoch, such as number of applications executed in the epoch, the active-times (AT's) of compute-cores and NoC routers over the epoch, and the thermal profile of these components over the epoch. Given these system-stats for the last epoch, the rise in effective V_T values (ΔV_T's) of all cores and NoC routers on the CMP (i.e., extent of BTI and HCI-induced circuit aging) is calculated as discussed in section 3.3.A using the V_{dd}'s and temperatures experienced by these components during all of their AT's over the entire epoch. The computed ΔV_T's are saved and passed on to the inner loop for reliability aware mapping, DoP and V_{dd} selection (section 3.4.B) in the next epoch.

Note that at the start of the very first epoch, the V_T's are initialized with nominal values representing no degradation and the ΔV_T's are initialized to zero-values. When the end of lifetime condition is encountered, the framework stops mapping applications, and outputs the lifetime of

the chip along with the total number of applications executed over the lifetime. *We consider the chip as failed (end of lifetime) when an application is dropped despite there being no other application running on the CMP and when the overall chip aging-profile (sum of tile V_T values) is beyond a certain threshold.*



(a)



(b)

Figure 22 *CHARM* design-flow: (a) circuit-aging, lifetime and epoch management (outer-loop, Section 3.4.A); (b) reliability aware mapping, DVS and app- DoP scheduling scheme (inner-loop, Section 3.4.B); blocks shown with dotted outlines are simulated models used in our work, and are a proxy for on-chip sensors that will get the information at runtime in a real system.

B. Reliability aware mapping, DVS and app-DoP selection

Before mapping, *CHARM* assigns checkpointing periods for each application as given by Eq. (20). Subsequently, the worst case execution time of an application is estimated, given the overheads of checkpointing and rollback recovery mechanism, using Eq. (18), (19). If the worst-case execution time cannot meet the application-slack time using the available on-chip resources, that application is dropped from the service queue. For any application under consideration, this stage consists of three phases, *(i)* region selection, app-DoP and V_{dd} selection, *(ii)* communication-aware task-to-tile mapping, and *(iii)* NoC routing path allocation. We describe each of these steps below.

(i) region selection, app-DoP and voltage-selection: In our framework, an application with a given DoP can be mapped on to a rectangular region on the 2D CMP, with shapes to be chosen from a pre-defined list $\{B_1, \ldots, B_n\}$ for that application. All intra-application communication is contained within the region, thus application-isolation is maintained and communication cross-interference between applications is eliminated. Our heuristic in this step utilizes the runtime V_T-degradation profile of the CMP, which is given as:

$$\Omega = \sum_{k=1}^{k=N} \boldsymbol{V_{Tk}}, \qquad \ldots\ldots\ldots (21)$$

where V_{Tk} is the average V_T value of the k^{th} tile (and includes core-V_T and router-V_T for the tile), and N is the total number of tiles on the CMP. The objective of our heuristic changes according to the value of Ω*: when the value is greater than a threshold ζ the objective is to preserve the lifetime of the CMP, otherwise the objective is to maximize the number of applications that complete before their deadlines.* When the objective is to preserve lifetime, we define a metric ψ to select the rectangular region on the CMP:

$$\psi = \sum_{k=1}^{k=DoP} \frac{\max\_V_T - V_{Tk}}{\max\_V_T - nom\_V_T} \qquad \text{..... (22)}$$

where, V_{Tk} is the same as in Eq. (21) for cores within the region; and nom_V_T is the nominal (lowest) effective V_T-value of a core with no aging. We define max_V_T as the maximum V_T value that the core can support (at highest V_{dd}). In order to preserve lifetime, *CHARM* selects a region with the least ψ that satisfies the application's deadline and DS-PB constraints. This is done because tiles dissipate higher leakage power when they are less aged, resulting in higher on-chip temperatures that cause further aging of the chip. Hence, it is intuitive to choose aged-cores that satisfy the application deadline, to preserve the overall lifetime of the chip.

Algorithm 3: Reliability-aware region, DoP and V_{dd}-selection heuristic

Inputs: V_T-profile, {P_1, … P_η}, {B_1, …, B_n}, {V_1, ….V_n}, DS-PB

1: for each DoP in {P_1,..P_η} & each V_{dd} in {V_1,...V_n} & each tile in CMP,
do{
2: **for** each shape in {B_1, …, B_n} **do {**
3: list _time.insert(P_i,B_i,V_i)
4: list_age.insert(P_i,B_i,V_i)
5: **} // end for** each shape
6: }**// end for**
7: if (Ω < ζ) **and** (app-slack time is less than τ) **{**
8: ptr = list _time.begin( )
9: } // end if
10: else if ((app-slack is greater than τ) **or** (Ω ≥ ζ) **{**
11: ptr = list _age.begin( )
12: } // end if
13: while(ptr != list _time.end( )) **do{**
14: check if CMP meets DS-PB with the ptr ->(P_i,B_i,V_i)
15: **if (**DS-PB constraint not met): ptr++
16: } // end while
17: if ((P_i,B_i,V_i) is not found) drop the application
output: a valid region to map the application, app-DoP value and V_{dd}-level for cores in the selected region; or application being dropped

Algorithm 3 shows the pseudo code for our region, DoP and V_{dd} selection heuristic. The heuristic performs a search over the available per-core supply voltages (V_1, … V_n), application DoPs {P_1, ... P_η} and the permissible mapping regions {B_1,..., B_n} for the application. Aging

profiles (V_T) of cores and routers, permissible DoPs P_i, permissible shapes B_i and per-core voltages are given as inputs to *Algorithm 3*. With these inputs and the DS-PB constraint, our heuristic aims to find a suitable {DoP, mapping region and V_{dd}} *combination* for an application under consideration for mapping to the CMP.

For any candidate combination of a given application, the heuristic estimates the number of soft-errors, as per Eq. (17), and computes an *estimated executed time*, as per Eq. (19). If the *estimated execution time* for a combination cannot meet the application deadline, the combination is not considered for mapping. The mapping heuristic does an exhaustive search over combinations of all the DoP (P_i), V_{dd} (V_i) and mapping regions (B_i) and sorts them into two ordered lists (lines 1-5 in *Algorithm 3*). The first list, *list_time*, is in the increasing order of the *estimated execution time*, for a (P_i,B_i,V_i) combination. The second list, *list_age*, is in the increasing order of the region's aging profile ψ, given by Eq. (22). If the total aging profile of the CMP ($\boldsymbol{\Omega}$ from Eq. (21)), is less than a threshold ζ, and if the app-slack time is less than τ, the heuristic prioritizes faster app execution time and finds a suitable candidate from *list_time* (lines 7-9). In all other cases, it prioritizes lifetime preservation and finds a suitable candidate in the *list_age* (lines 10-12). The heuristic then starts at the beginning of the list, where the best combination is saved, and checks if that meets the DS-PB constraint. If not, it iterates to the next combination in the list (lines 13-16). If none of the combinations satisfy the DS-PB constraint, the application is dropped from the service queue (line 17).

When the application arrival rate is very high, our aim is to map more applications on the CMP. *Hence, when the queue pressure is above a threshold 'χ' (queue-pressure > χ), the application under consideration is mapped over a smaller DoP range,* as given by Eq. (23), (24):

$$\boldsymbol{Q.size > \chi : P = \{P_1, P_2, \ldots P_k\}} \qquad \ldots\ldots (23)$$

$$\boldsymbol{Q.size \leq \chi : P = \{P_1, P_2, \ldots P_N\}} \quad \ldots\ldots (24)$$

where *Q.size* gives the size of the service queue, and $\{P_1, P_2, \ldots P_k\}$ is a trimmed down list of permissible DoPs, with $P_k < P_N$.

We now present the theoretical time complexity of our heuristic. At most N tiles (total tiles on the 2D NoC-based CMP) are considered for the prospective mapping region. Note that the permissible DoPs $|D|$, V_{dd} levels $|S|$ for the cores and number of admissible shapes n are very small constant integers, and the DoP of the application (relatively small integer c – treated as a constant) number of tiles are to be evaluated at each of these iterations. With these assumptions, our region/*Vdd*/DoP-selection heuristic effectively runs in linear-time complexity with respect to the number of tiles, N: $O(c.n.|D|.|S|.N)$, and is very suitable for fast execution at runtime with low overhead.

(ii) Communication-aware task-to-tile mapping: After the mapping region for an application has been selected (of size equal to app-DoP), our heuristic proceeds to map the application's task-graph on to the CMP tiles. We use a fast and efficient task-to-tile incremental-mapping approach (similar to that used in prior works such as [73]) suitable for use at runtime, which aims to minimize communication between cores. We consider the earliest deadline first (EDF) task scheduling scheme for each application task graph and map that task-graph on to a selected rectangular shaped region of tiles on the 2D CMP;

(iii) Wear-out balancing routing path allocation (WBR): After the task to tile mapping step, we map the communication-flows of the current application on to the NoC in the selected region on the CMP. We propose an aging- and congestion-aware routing scheme (WBR) to balance the core-router aging profile and extend the lifetime of the NoC routers along with that of the cores. WBR trades-off aging with network-congestion in the NoC by selecting routing paths to maximize NoC-

lifetime while leveraging the knowledge of maximum execution time constraints of the mapped application.

To ensure an implementation with low-overhead, path diversity, and deadlock freedom, our routing algorithm builds on the Odd-Even partially adaptive turn model routing scheme [92] for 2D-mesh NoCs. The Odd-Even scheme typically presents multiple routing options at each hop. The key idea of our routing algorithm is to select the next hop (and hence a path) in such a way that it either preserves lifetime of the routers or reduces congestion. We designed a cost-function for next-hop selection during routing that considers the difference between router-aging and core-aging ($router\_V_T - core\_V_T$) values to ensure balanced aging in CMP tiles. Moreover, as congestion in the NoC-links leads to excessive routing delays and thus longer application-runtimes, we also prefer allocating communication flows to links with lesser communication-volumes. The following routing cost function (Rt_{cost}), which is a linear combination of the two normalized metrics, is used to make routing decisions at each hop along the path:

$$\boldsymbol{Rt_{cost} = \alpha_R . \frac{(V_T\ difference) - (minimum\ V_T\ difference)}{range\ of\ V_T\ difference}}$$

$$\boldsymbol{+ \beta_R . \frac{(volume) - (minimum\ volume)}{range\ of\ volume}} \quad \text{..... (25)}$$

where, α_R and β_R are weighting coefficients, $V_T\ difference$ represents ($router\_V_T - core\_V_T$) of the candidate next hop router, and *volume* represents the existing communication-volume (already allocated while routing previous flows) on the link. WBR selects the next hop with the minimum routing-cost, Rt_{cost}, given in Eq. (25). The values of the coefficients in Eq. (25), α_R and β_R, are re-evaluated after routing each flow, as shown below:

$$\boldsymbol{\beta_R = \{current\ app.\ delay\}/\{\delta.(app.\ execution\ time\ constraint)\}}$$

$$\boldsymbol{\alpha_R = (1 - \beta_R)} \quad \text{. .. (26)}$$

Before any application communication flows are mapped through the NoC routers, we start with values α_R =1 and β_R =0. As flows are mapped and the estimated application-delay increases, the value of β_R increases (and α_R decreases) proportionally until the application-delay reaches a significant fraction (δ) of the application execution time constraint. At this point (β_R =1 and α_R =0), WBR ceases to be aging-aware and routes on paths with minimum congestion exclusively, to satisfy the execution time constraint of the given application. ***Algorithm 4*** below summarizes our wear-out balancing NoC routing scheme.

Algorithm 4 Wear-out balancing routing path allocation

Inputs: Task-graph, execution time constraints, minimum frequency, task-mapping of current application, V_T-profile of compute-cores and routers

1: Initialize α_R=1 and β_R =0
2: for all communication-flows **do**
3: for all hops on the minimal path **do**
4: select the next hop with the least Rt_{cost} as per Eq.(25)
5: update α_R and β_R as per Eq. (26)
6: end for
7: end for
output: all flows of the application allocated in the CMP-region

The paths that are allocated for each communication flow are stored in lightweight routing tables, in each NoC router. In this work, the largest size of an application footprint (DoP) is 32, which has an upper bound of 64 communication flows through any router. Hence, each router will have to store the next hop information for at most 64 paths (64 entries in a table). Assuming 2 bits for the output port and 6 bits for the source and destination each (for the 60-core CMP we consider), the footprint of the NoC routing table is only 768 bits. Thus, the hardware overheads of implementing WBR are low and reasonable.

3.5 EXPERIMENTAL STUDIES

A. Simulation setup

We conducted experiments on 13 different parallel applications from the SPLASH-2 [93] and PARSEC benchmark suites. These applications are executed on the cycle-accurate SNIPER [79] multicore simulator together with McPAT [94] to obtain their steady state power, compute time, and communication intensity across various app-DoPs, V_{dd} and clock frequencies. The DoP values we used ranged from 4 to 32 beyond which most of the applications were observed to have lower performance due to high communication (synchronization) overheads. The *CHARM* framework is implemented in C++ and is integrated with the SNIPER simulator, models for aging and soft-errors, and checkpointing and rollback support. *CHARM* also uses the open-source thermal emulator 3D-ICE [75] to simulate the temperature profile of the cores and routers from the steady state power values, assuming a conventional air-cooled heat-sink. For the given power-profile, 3D-ICE outputs the core and router temperatures on the die.

We categorized the 13 benchmarks into two groups: *(i)* memory-intensive benchmarks - {*cholesky, fft, radix, raytrace, dedup, canneal, and vips*); and *(ii)* compute-intensive benchmarks – {*swaptions, fluidanimate, streamcluster, blackscholes, radix, bodytrack, and radiosity*}. As *radix* has properties of both, we use it in both groups. In our analyses, we employ three types of application sequence groups as inputs to our framework: memory-intensive, compute-intensive, and mixed (using all 13 applications). Each application-sequence has 100 randomly ordered application-instances selected from the respective group. To enhance the statistical significance of our results, we averaged results over four different randomly generated application-sequences for each group. To simulate the chip-lifetime within a reasonable time, we extrapolate the effects of

aging over 500 such sequences, making the total number of application-instances executed within an epoch to be approximately l = 50,000. We experimented with three different inter-application arrival rates of 1.4s, 2.8s and 5s. We also analyzed the impact of different soft-error rates.

We considered a 2D CMP with 60 homogeneous tiles, fabricated at 22nm. Each tile has an x86 core, a NoC router, and a private L1 cache. The tiles are arranged in a 10×6 mesh layout. The V_{dd} values supported by each tile (core + router) are between 0.75V-1V, in steps of 0.05V. Note that as all the cores on which a parallel application is mapped are operated at the same V_{dd} and clock frequency, and as there is no inter-application communication, there are no overheads due to voltage level converters and clock synchronizers during application execution. We considered the computation frequencies using Eq. (10) and obtain the respective application execution time from the SNIPER simulation data. The dark-silicon power budget (DS-PB) is assumed to be 80W for our selected 60 core chip at the 22nm node. We also considered the runtime overhead for our framework. The total execution time for *CHARM* on one of the cores is 0.36ms, which is insignificant compared to the average application execution time (~seconds).

For computing the circuit-aging, we assumed the nominal V_T of each core and router to be 0.3V at the beginning of the chip lifetime. We consider a tile to be unusable after its average V_T goes beyond 0.57V. Above that value, the maximum operating frequency of the core cannot meet any of the applications' deadline constraints. The chip aging-footprint threshold ζ depends on workload types and arrival rates. The ζ value was empirically derived, as discussed in the next section.

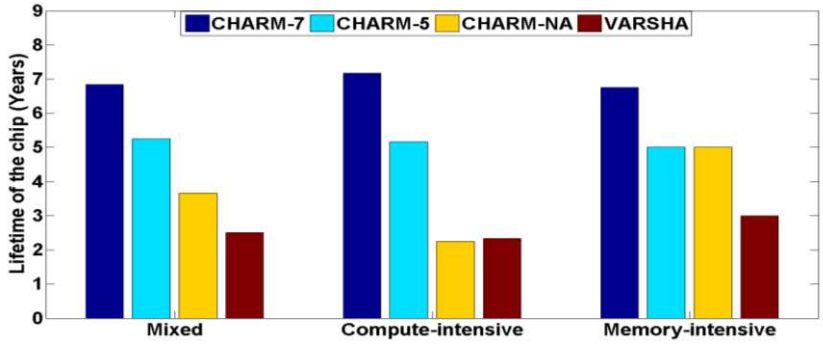
In the WBR routing heuristic, we use δ=0.6, to calculate the value of β_R, for an appropriate trade-off between application performance and aging. In our experiments, an epoch interval can

range between 25 to 35 days, depending on the power profile, execution-times, and average DoPs of the application workload, as well as the degree of aging in the chip. Also, as the overheads incurred due to employing aging sensors have been reported to be quite small (power dissipation of 84.7nW, sensing-latency of 100µs, and area of 77.3µm^2 per sensor at 45nm technology node [69]), we ignore them in our analysis.

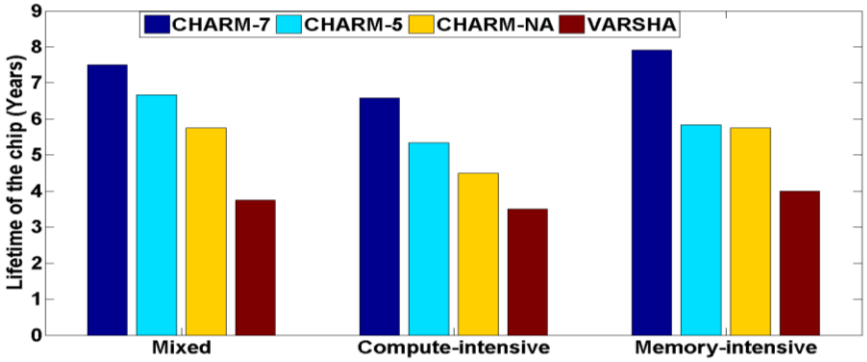
B. Simulation Results

We compare our *CHARM* framework against an enhanced version of a prior work *VARSHA* [83] that tries to optimize the energy and performance of a multicore chip while meeting dark-silicon power constraints as well as satisfy reliability and performance constraints. The work however does not employ any checkpointing and rollback based correction, nor does it consider circuit-aging phenomenon while mapping. While *VARSHA* does not consider per-core DVS, we have enhanced *VARSHA* with per-core DVS in our analysis for a fair comparison with *CHARM* which also assumes per-core DVS support. We explore three variants of our *CHARM* framework: *CHARM-5*, which is designed for a target lifetime of 5 years; *CHARM-7*, which is designed for a target lifetime of 7 years; and *CHARM-NA*, which has no target lifetime. *CHARM-NA* thus only has the soft-error prevention mechanism, and aims for high V_{dd} and app-DoP to get the best execution speeds throughout the chip lifetime. We simulated and analyzed the lifetime of the chip, total number of applications executed over the lifetime, average power dissipated by applications, and the number of functional cores at the end of the lifetime, for the four frameworks. Figure 23 shows the lifetimes of the CMP for the different frameworks. *CHARM-7* and *CHARM-5* are designed to achieve their target lifetimes of 7 and 5 years respectively. This is made possible by changing the threshold value ζ for different target lifetimes and application arrival rates. For

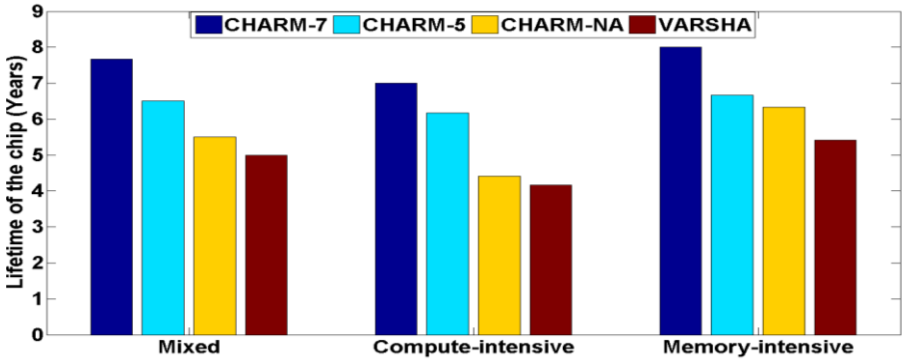
CHARM-7, ζ is empirically derived to be approximately 21.5V for arrival-rate-1, 22.5V for arrival-rate-2, and 23V for arrival-rate-3. Similarly for *CHARM-5* it is approximately 27V for arrival-rate-1, 28.5V for arrival-rate-2, and 30V for arrival-rate-3.



(a)



(b)



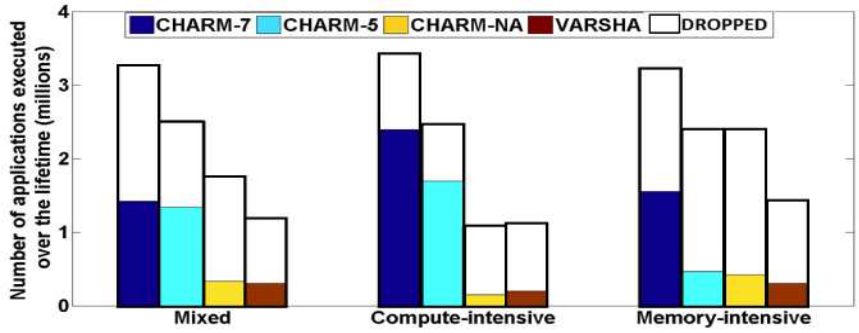
(c)

Figure 23 Comparison of lifetimes of the chip for different frameworks across different workloads and arrival rates: (a) Arrival-rate-1, inter-app duration is 1.4s (b) Arrival-rate-2, inter-app duration is 2.8s (c) Arrival-rate-3, inter-app duration is 5s

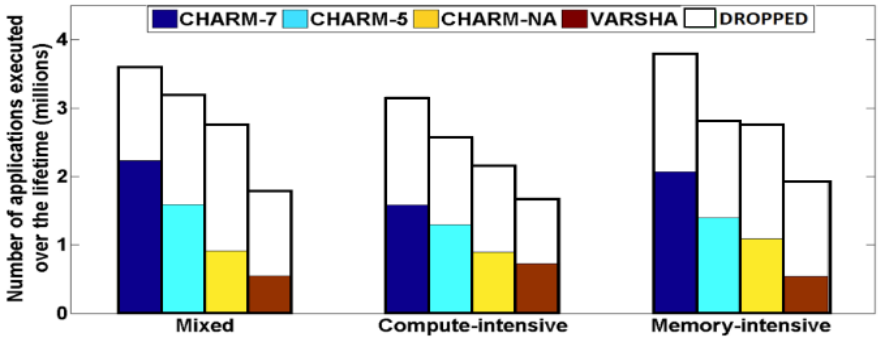
From Figure 23 (a), Figure 23 (b) and Figure 23 (c) it is evident that *CHARM-5* and *CHARM-7* either meet or come very close to achieving their target lifetimes of 5 years and 7 years respectively. Both *CHARM-7* and *CHARM-5* intelligently adapt their mapping phases to save lifetime or optimize performance according to the available slack time and threshold constraints. Without a target lifetime, *CHARM-NA* optimizes primarily for performance while *VARSHA* optimizes for energy, leading to their lower lifetimes. In particular, *CHARM-7* achieves 50-100% improvement in lifetime compared to *CHARM-NA*, and up to 2× improvement compared to *VARSHA*.

Figure 24 shows the total number of applications executed over the lifetime of the chip for the four frameworks. At all the three arrival rates, *CHARM-7* executes a higher number of applications than any other framework, achieving up to 2× improvement compared to *CHARM-NA* and up to 6× improvement compared to *VARSHA,* in the number of applications executed. This is due primarily to the higher lifetime constraint for *CHARM-7* and the ability of our proposed heuristics to manage circuit aging to meet this constraint, while reducing the number of dropped applications compared to *CHARM-NA* and *VARSHA*. *CHARM-5* executes 2× more applications than *CHARM-NA* and *VARSHA* for compute-intensive and mixed workloads but gives results comparable to *CHARM-NA* for memory-intensive workloads when the arrival rate is high (Figure 24 (a)). This is because, although memory-intensive apps consume less power, they run for longer durations and have shorter *app-slack* times compared to compute-intensive apps. When the arrival rate is high, *CHARM-5* drops higher number of applications in the latter part of its lifetime owing to its inability to find a mapping region that meets application deadlines while preserving the lifetime at the same time. This leads to a similar number of applications executed in *CHARM-5* and *CHARM-NA*. However, when the arrival rate is low (Figure 24 (c)), *CHARM-5* surpasses the

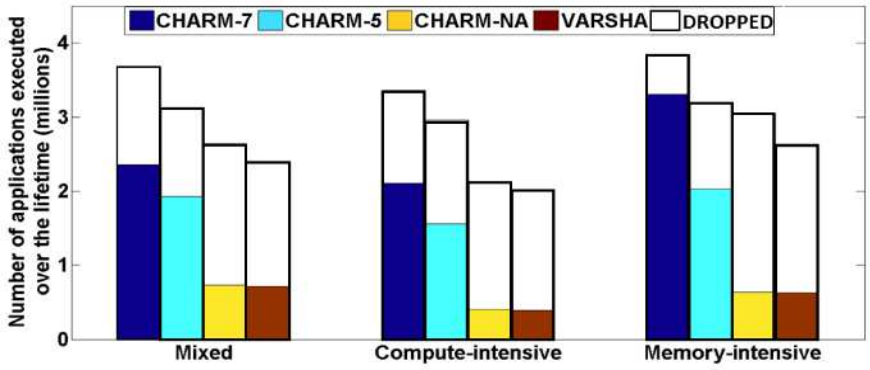
applications executed by *CHARM-NA* and *VARSHA* by over 2.5×. *CHARM-7* executes 10-40% higher number of applications compared to *CHARM-5,* due primarily to the longer lifetime achieved with *CHARM-7* which provides more opportunities to execute greater number of applications. *A*s *CHARM-NA* prioritizes performance by executing applications at higher V_{dd} and DoP, and *VARSHA* executes applications at very high V_{dd} to safeguard the applications from soft-errors in the absence of checkpointing and rollback recovery, both frameworks suffer from relatively lower lifetimes and application execution counts. Running at higher V_{dd} also leads to higher power dissipation and fewer number of applications running on the CMP to avoid violating DS-PB constraints. As a result, in *CHARM-NA* and *VARSHA*, the waiting time in the service queue is much higher, and a larger number of applications get dropped due to missed deadlines.



(a)



(b)



(c)

Figure 24 Results comparing the number of applications executed by different frameworks across different workloads and arrival rates. (a) Arrival-rate-1, inter-app duration is 1.4s (b) Arrival-rate-2, inter-app duration is 2.8s (c) Arrival-rate-3, inter-app duration is 5s

Figure 25 shows the average power dissipated per application by the four different frameworks. When the arrival rate is high (Figure 25 (a)), *CHARM-7* dissipates 50-80% less power per application than both *CHARM-NA* and *VARSHA* with different workload types. *CHARM-5* achieves 50-80% improvement in power compared to both CHARM-NA and VARSHA. This is because of the queue-pressure threshold (Section 3.4.B), which does not allow high DoPs for applications, leading to more number of apps being mapped simultaneously and lesser average power dissipated per application, for *CHARM-7* and *CHARM-5*. When the arrival-rate is moderate (Figure 25 (b)), *CHARM-7* achieves 70% improvement compared to *CHARM-NA* and 80% improvement compared to *VARSHA,* and *CHARM-5* achieves an improvement of 25% compared to *CHARM-NA* and 30% compared to *VARSHA,* for compute intensive workloads.

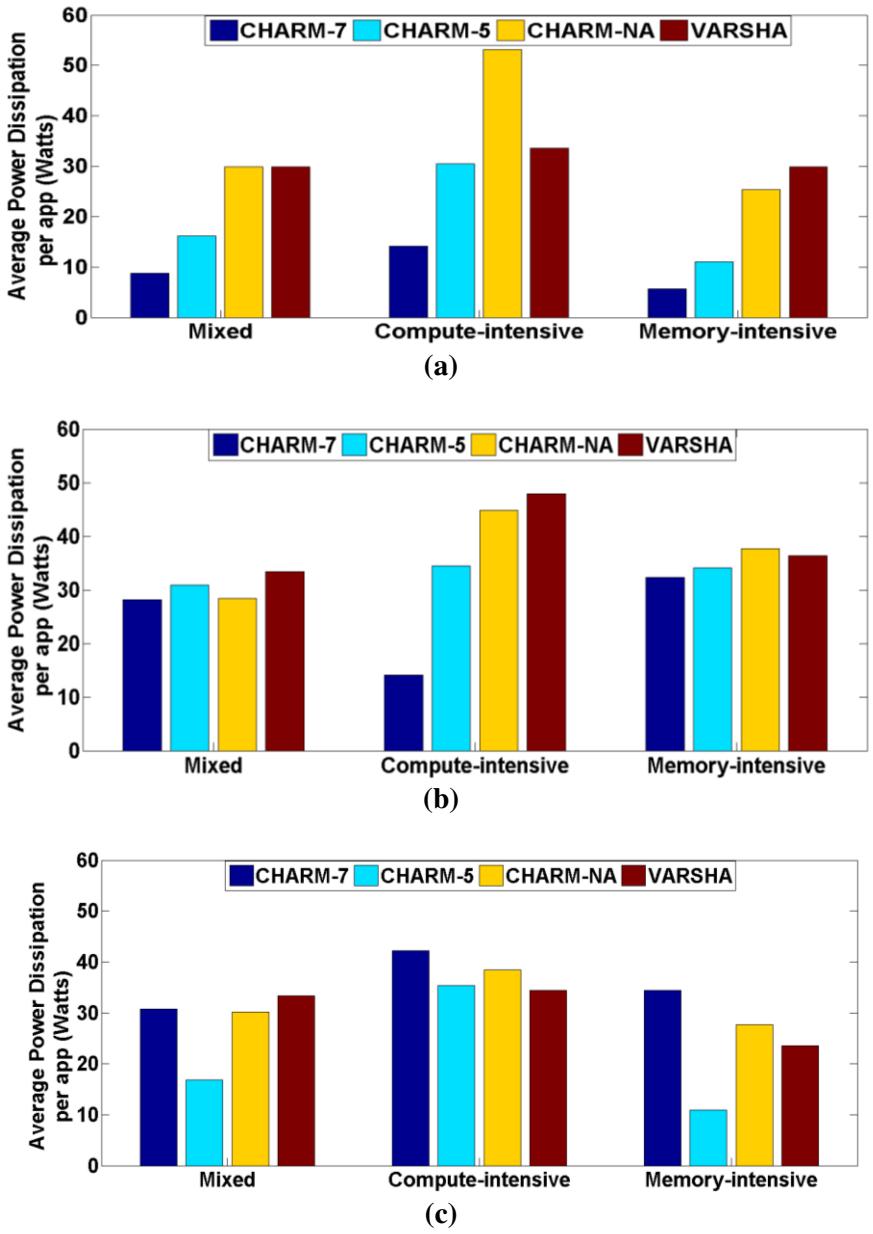


Figure 25 Average power consumed by applications executed on different frameworks across different workloads and arrival rates. (a) Arrival-rate-1, inter-app duration is 1.4s (b) Arrival-rate-2, inter-app duration is 2.8s (c) Arrival-rate-3, inter-app duration is 5s

However, both *CHARM-7* and *CHARM-5* achieve only 2-5% improvement for memory intensive workloads compared to *CHARM-NA* and *VARSHA*. This is because both *CHARM-7* and *CHARM-5* have sufficient power budget and *app-slack* time for these workloads to map many of the applications with higher V_{dd} and DoP, similar to *CHARM-NA* and *VARSHA*. When the arrival

rate is low (Figure 25 (c)), *CHARM-7* dissipates around over 2× more power per application than *CHARM-5* and up to 15% more power than *CHARM-NA*. In order to not significantly exceed the target lifetime constraint of the chip, *CHARM-7* maps applications more aggressively towards the latter part of its lifetime, with higher DoP and V_{dd}. This raises the average power dissipated per application mapped. *CHARM-NA* dissipates higher power than both *CHARM*-5 and *CHARM-7*, because of its aggressive mapping of applications with very high V_{dd} and DoP. *CHARM-NA* and *VARSHA* dissipate power in a similar manner, within 3-8% of their respective powers except at the higher arrival rates of compute intensive workloads where CHARM-NA dissipates the highest power per application.

Table 2 Number of functional cores at the end of the lifetime

	CHARM-7	CHARM-5	CHARM-NA	VARSHA
Mixed	10	15	17	42
Compute	9	14	20	55
Memory	11	17	17	54

Lastly, we analyzed the state of the chip at the end of the lifetime with the four frameworks to understand the implications of their chosen runtime resource management strategies. Table 2 shows the total number of functional cores remaining at the end of the lifetime for different workload types. The results shown are for a single arrival rate (Arrival-rate-1) for brevity. *VARSHA* reaches the end of the lifetime much before all of its cores are degraded. This is because *VARSHA* maps applications at higher V_{dd} and frequency to satisfy reliability constraints. However, the resulting chip-aging-profile with *VARSHA* creates fragmentation, making it impossible to obtain contiguous mapping regions on the chip that meet application deadline constraints, for low or high DoP values. All the *CHARM* frameworks, however, utilize the chip till most of the cores are degraded, thereby utilizing the chip very effectively. On average, *CHARM*-7 has 5×, *CHARM-5*

has 2.5× and *CHARM-NA* has 2× better chip utilization vs. *VARSHA* by the time of the end of the chip lifetime.

3.6 CONCLUSIONS

In this chapter we proposed a novel runtime framework called *CHARM* that aims to maximize the number of applications executed reliably while meeting their performance deadlines without violating the dark-silicon power constraints over a given chip target lifetime. Our experiments show that *CHARM* enables up to 2.5× improvement in the lifetime, up to 6× improvement in number of applications executed and 5× improvement in efficiently using the cores during the lifetime of the chip compared to the state-of-the-art on reliability aware runtime application mapping.

4. PARM: POWER SUPPLY NOISE AWARE RESOURCE MANAGEMENT FOR NOC BASED MULTICORE SYSTEMS IN THE DARK SILICON ERA

Today's processors designed at sub-10nm technology nodes have high device densities and fast switching frequencies that cause fluctuations in supply voltage (V_{dd}) and ground networks. When cores with varying activity profiles switch at the same time, the resulting fluctuation in power supply leads to a reduced V_{dd} at the power supply grid nodes of the cores and network-on-chip (NoC) routers in that region, which can adversely affect the execution of applications running on them. In this chapter, we propose a novel runtime framework to reduce the power supply noise (PSN) in cores and routers at runtime. Experimental results for 7nm FinFET process nodes show that our framework not only achieves up to 4.5× reduction in PSN, and up to 34.3% improvement in application performance, but also manages to map up to 38% more applications when the CMP is oversubscribed, compared to the state-of-the-art.

Figure 26 shows peak PSN in the power delivery wires on the chip due to inter-core interference [4], *which is increasing alarmingly with technology scaling by going beyond the permissible noise margin, making the PSN threat a serious concern for chip designers.* Traditional approaches have addressed the PSN issue at the circuit and micro-architectural levels. Although PSN is most readily observed at the circuit level, the compute intensity and distribution of the workload on the cores decides the magnitude of PSN observed at each cycle. *Hence it is important to address the issue of PSN at a higher level of abstraction than the circuit level.* More recent approaches [95] [96] [97] [98] [99] [100] address PSN at the system/compiler level, and propose PSN aware application mapping, task scheduling, and instruction rescheduling. However, most of these approaches have high overheads when deployed at runtime, are agnostic to the dark silicon

issue, and do not consider workload activity in NoCs in tandem with cores while attempting to overcome the challenges associated with PSN.

In this chapter, for the first time, we propose a novel PSN-aware runtime resource management framework (*PARM*) that employs dynamic voltage scaling (DVS), adaptable application degrees of parallelism (DoP), and an intelligent mapping scheme for a NoC-based CMP that operates at near threshold voltages within dark silicon constraints. *PARM* selects the mapping region, V_{dd}, and DoP for every application that arrives at runtime in such a way that the peak PSN in the mapping region and its vicinity is kept below a threshold, minimizing the number of voltage emergencies. Our key contributions in this chapter are:

- We study the correlation of PSN with application activity, proximity of concurrently executing threads, and core supply voltages;
- We utilize DVS and adaptable application DoP, to reduce the peak PSN and optimally utilize the available dark silicon power slack while maximizing the number of applications serviced at runtime;
- We propose a novel PSN aware application mapping heuristic for emerging sub-10nm fabricated CMPs, to reduce the PSN in a region and minimize communication latency between tasks;
- We devise a novel PSN-aware routing scheme that balances router activity near highly switching cores, and reduces the impact of the NoC traffic on PSN without incurring any additional latency.

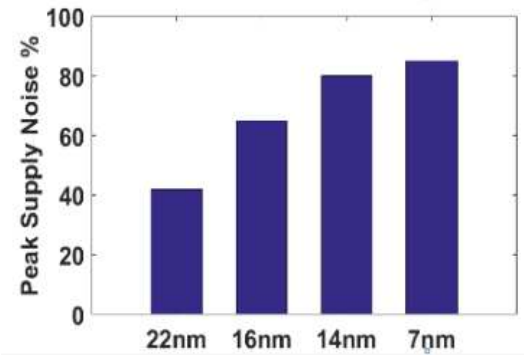


Figure 26 Peak supply noise percentage, relative to the nominal near threshold supply voltage, across fabrication process technology nodes.

4.1 RELATED WORK

Several techniques have been proposed to cope with PSN at the circuit-level. In [101], conservative noise margins are used to ensure safe operation even when worst-case PSN is observed. In [102], decoupling capacitors are used to reduce core-to-core voltage interference. In [103] [104] mechanisms were presented to predict PSN and the occurrence of voltage emergencies. Other efforts have proposed micro-architectural solutions, to reduce inter-core interference, e.g., pipeline throttling, instruction rescheduling, relaxed entry/exit at synchronization barriers, etc. [95] - [100], [105]. But these solutions are primarily designed for single-core or few-core systems. The use of on-die digital sensors was proposed in [106] for the runtime measurement of PSN and to take reactive (corrective) measures post-detection. However, these solutions are not very beneficial in preventing PSN-induced voltage emergencies. Also, the penalty for error correction is expensive when the system is over-subscribed. In [107], Hu et al. proposed a thread mapping and migration scheme for Single Program Multiple Data (SPMD) applications to minimize large voltage fluctuations in CMPs. In [32] [108], reliability aware task mapping and NoC routing schemes have been proposed to minimize the effects of aging and soft errors. In [23] [109], PSN-aware workload

assignment schemes are proposed for 2D and 3D CMPs. The schemes map highly active threads at longer Manhattan distances from each other to minimize PSN in a region. However, these schemes do not consider the impact of activity in NoC routers on PSN. In [110], PSN-aware routing and flow control schemes are proposed, to reduce NoC router activity. However, application mapping plays a crucial role in overall NoC activity as tasks separated by longer distances cause more NoC routers to switch. Also, none of these works consider the low voltage margins imposed by NTC to meet dark silicon constraints.

To the best of our knowledge, this is the first work that addresses PSN due to both core and NoC switching activity in the presence of dark silicon power constraints for CMPs executing multi-application workloads and designed at sub-10nm technology.

4.2 BACKGROUND: MODELS AND ASSUMPTIONS

4.2.1. PROCESSOR MODEL

We assume a CMP with N tiles $T = \{T_1, T_2 \ldots T_N\}$. Each tile T_i, has a processing core, a NoC router, and L1 instruction and data caches as shown in Figure 27. The tiles also have a shared global L2 cache, organized in banks. The tiles are connected by a NoC fabric. The CMP is constrained by a dark silicon power budget (DsPB) which is the thermally safe power limit that the cooling system of the chip can operate effectively within. The chip supports dynamic voltage scaling and can operate at different supply voltages $V = \{V_1, V_2 \ldots V_s\}$.

4.2.2. APPLICATION MODEL

We assume the applications $A = \{A_1, A_2, \ldots A_M\}$ that execute on the CMP to be multithreaded, with each application A_j able to spawn up to K threads $\{T_1, T_2 \ldots T_K\}$. Each thread executes on a dedicated core $C_i \in \mathcal{T}_i$. An application A_j can execute with different thread counts hence allowing for variable degree of parallelism (DoP). Each application has a performance deadline constraint. Application communication requirements are represented by an application graph APG = $G(V, E)$ which is a directed acyclic graph, where each vertex $v_i \in V$ represents a thread and each edge $e_{i,j} \in E$ represents communication volume between thread i and thread j. All cores that execute the threads of an application are supplied with the same V_{dd}. The applications are stored in a service queue upon arrival at runtime and are considered for mapping to the CMP on a first-come-first-serve (FCFS) basis. Applications are mapped on non-overlapping regions on the CMP for inter-application isolation. In the rest of the chapter, the terms *thread* and *task* are used interchangeably.

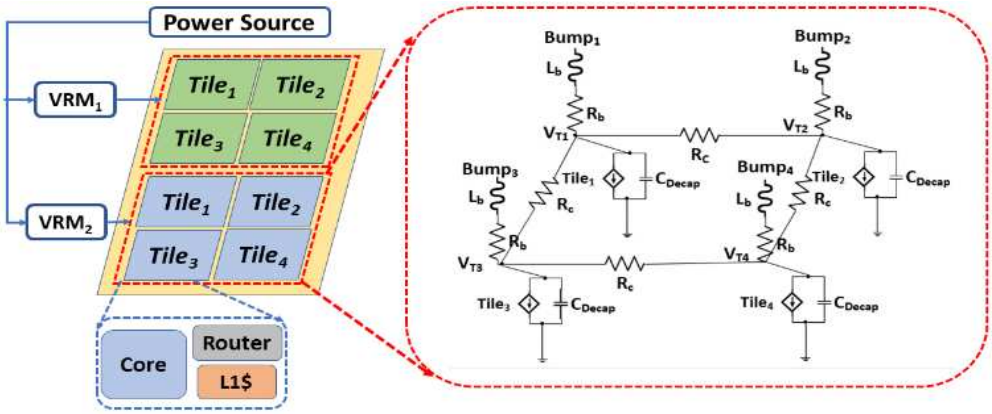


Figure 27 Baseline CMP with power supply domain of four tiles; each tile is powered by a voltage regulator (VRM) connected to a power source.

4.2.3. POWER DELIVERY NETWORK (PDN) IN CMPS

We assume a baseline PDN with multiple independent domains as shown in . A domain is a group of four tiles that has its own voltage regulator module (VRM). These domains are physically

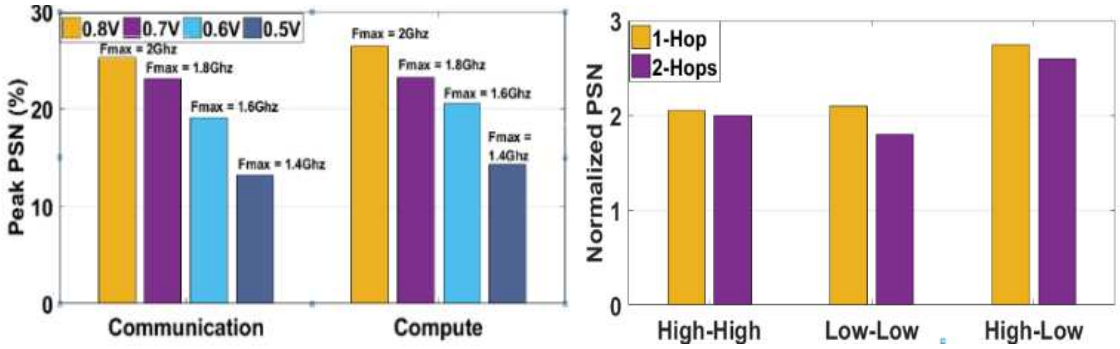
separated so that there is no interference between tiles from different domains. Each domain is powered by an independent source, which enables efficient monitoring of the power consumption on the chip as the core count scales up. All of the tiles in a domain are supplied with the same V_{dd}, although the actual voltage received at the tile varies due to PSN-induced variation. We assume the presence of digital sensors [106] to monitor the runtime PSN levels at cores and NoC routers. We also assume that tasks of different applications are not mapped into a single domain, which is ensured by limiting the DoP values of each application to be multiples of 4.

4.2.4. POWER SUPPLY NOISE (PSN) MODELING AND ESTIMATION

PSN is caused by *(i)* resistive drop of power delivery wires (*IR*), and *(ii)* inductive droop due to wire inductance ($L.\Delta i/\Delta t$). While resistive drop is proportional to current flowing in the wires, inductive droop is proportional to the switching activity of the wires carrying current. We model the PDN as in previous works as shown in Figure 27, where L_b and R_b are inductance and resistance at a bump, R_c is resistance of the PDN wires, and C_{decap} is the decapacitance between cores. The workload on a tile is modeled as a current source, similar to [107] [103] [109], based on power consumption of the core and NoC router in a tile. The dynamic values of PSN observed at each tile is given by:

$$\Delta V = V_{bump} - V_{Ti} \qquad (27)$$

where V_{bump} is the voltage supplied by the source and V_{Ti} is the voltage observed at tile *i* after on-chip parasitic drop. As in [98], we consider a PSN of 5% as a margin for a VE in near threshold voltages that leads to faulty outcomes for a thread executing on the PSN-affected tile.



(a) **(b)**

Figure 28 (a) Peak PSN (as % of supply voltage) observed in a domain for communication- and compute-intensive workloads; (b) Normalized PSN due to interference between pairs of tasks of different switching activity (High or Low) and separated by Manhattan distances of 1 and 2 hops.

4.2.5. IMPACT OF MAPPING DECISIONS AND DVS ON PSN

PSN is significantly impacted by variations in switching activity of transistors that leads to interference between the current flows in wires. Moreover, as shown in Figure 28(a), the peak PSN observed in a domain is also directly proportional to its operating voltage (V_{dd}), which decides the maximum operating frequency (F_{max}) of cores and routers in that domain. The trend exists for both communication-intensive and computation-intensive applications. To reduce PSN, one solution is to reduce V_{dd}. However, this also reduces F_{max}, which diminishes application performance. Dynamic adaptation of application DoP is one way to improve performance while running at a low V_{dd} [83].

The switching characteristics of tasks executing in close proximity to each other on a chip also have a considerable impact on PSN. Figure 28 (b) shows interference effects of different combinations of switching activities for two tasks executing on adjacent cores. We categorize application tasks into two bins, “High” and “Low” active, based on extensive analysis of their

switching activity. The PSN observed due to interference between tasks with High-Low switching activity (one task has high average switching activity and the other has low average switching activity) is up to 35% higher than tasks with High-High and Low-Low switching activity. Cores running low switching activity tasks get affected by the resistive and inductive interference from the power drawn by high switching tasks running on the neighboring cores in the same domain. This behavior is also observed in [111]. Interestingly, Figure 28 (b) indicates that highly interfering tasks mapped at a distance of 2-hops away interfere up to 10% less than tasks mapped at a distance of 1-hop away. None of the prior works have exploited this observation to reduce the negative impacts of PSN.

Given application performance deadlines and the dark silicon power budget (DsPB) for a CMP, our objective is to use the above observations to opportunistically select a combination of V_{dd}, DoP, and mapping region for each application that arrives for execution at runtime, to minimize the PSN observed in power supply domains. The next section discusses our proposed framework to meet this objective.

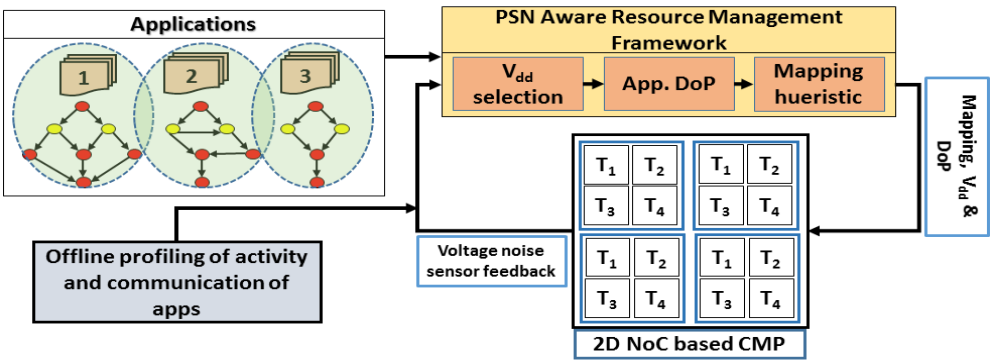


Figure 29 Overview of the proposed PARM framework

4.3 PSN AWARE RESOURCE MANAGEMENT (PARM)

Figure 29 gives an overview of our proposed *PARM* framework. Offline profiling information about applications, and online voltage noise feedback from on-chip voltage noise sensors are inputs to the framework. The application profiling collects statistics on switching activity, power consumption, and NoC communication characteristics for all of the tasks of an application at different V_{dd}'s and DoPs. The output of the framework is a task to core mapping, V_{dd} assignment, and DoP selection for each application that arrives for execution on the NoC-based CMP with independent power domains. V_{dd} and DoP are assigned for an application at the beginning of its execution, and their values do not change till the application completes. Our *PARM* framework first selects an appropriate V_{dd} and DoP for an application to be mapped, based on the application performance and chip-level DsPB constraints. After V_{dd} and DoP selection, *PARM* uses a PSN aware mapping heuristic, to find a mapping in such a way that the total PSN in PDN domains is minimized; and communication distance between tasks is minimized (to improve performance). The following subsections discuss the components of the *PARM* framework.

4.3.1. V_{DD} AND DoP SELECTION

Algorithm 5 presents our V_{dd} and DoP selection method. The inputs are a set of permissible voltages sorted in *increasing order* $V = \{V_1, V_2, \ldots V_S\}$, and the set of applications waiting in the service queue $A = \{A_1, A_2, \ldots A_M\}$. To consume low power (and generate low peak PSN) while ensuring that application deadlines are met, the algorithm starts with the lowest V_{dd} and the highest DoP combination. This is because peak PSN is always low at lower V_{dd} values (Figure 28 (a)). Also, to meet application deadlines, it is intuitive to utilize the available tiles to spawn more number of threads (i.e., use a higher application DoP) without violating the DsPB constraint. To

ensure this, the permitted DoP values of an application are sorted and considered in a decreasing order (line 1). The minimum DoP considered in our work is 4, as single and dual threaded versions of applications that we analyzed had poor performance and missed deadlines.

Our selection algorithm then iteratively searches for a suitable (V_{dd}, DoP) combination. First, the *Worst Case Execution Time* (WCET) of an application for a selected (V_{dd}, DoP) combination is estimated using the *offline application profile data* (line 5) and checked to see if the application deadline constraint will be met (line 6). If the deadline constraint is met, this (V_{dd}, DoP) combination is sent as an input to the *PSN-aware mapping heuristic* (line 7; this heuristic is discussed in the next subsection). If the mapping is successful, the next application in the service queue is processed (line 8). If not, the algorithm waits till the CMP completes a currently executing application (which would free up tiles for mapping), and tries again to find a mapping region (lines 9-11). If the algorithm fails to find a mapping region, it selects the next DoP (lower value) from the list D, and performs the same operations as above (line 12). This can be useful to map an application when there are a lack of sufficient number of tiles, or a limited power budget. Selecting a lower DoP would resolve both of these concerns. However, if the estimated WCET (from line 5) does not meet the application deadline constraint, the algorithm skips iterating through DoPs, as the lower values of DoP cannot satisfy the deadline constraint, and continues searching for a new (V_{dd}, DoP) combination with the next V_{dd} from the list V, that is higher than the current V_{dd} value (line 13). If the deadline constraint is not met or if the mapping region is not found on the CMP after exploring all V_{dd} and DoP combinations, the current application is dropped and the next waiting application is processed, to avoid stagnation in the service queue due to the stalled application.

Algorithm 5: V_{dd}, DoP selection

Inputs: V_{dd} values sorted in increasing order $V = \{V_1, \ldots V_S\}$, applications $A = \{A_1, \ldots A_M\}$

1: for all A_j *in* A **do**
2: $D \leftarrow$ **Sort(**$A_i.\{ D_1, \ldots D_T \}$**)** *//sorted in descending order*
3: for all Vi in V **do**
4: for all D_k in D **do**
5: WCET ← *EstimateExecutionTime*(Vi, D_k, A_j)
6: **if** WCET < *Deadline* (A_j) **then**
7: **if** *PSNAwareMapping* (Vi, D_k, A_j) is successful **then**
8: **goto** line 1 (continue to next app in ***A***)
9: **else** *stall till an app exit event on CMP*
10: **if** *PSNAwareMapping* (Vi, D_k, A_j) is successful **then**
11: **goto** line 1 (map the next app in ***A***)
12: **else goto** line 3 (Try with lower D_k,)
13: **else goto** line 2 (Try with next V_i)

Outputs: V_{dd} , DoP, and a valid region to map the application

4.3.2. PSN AWARE MAPPING HEURISTIC

Given a V_{dd} and DoP that satisfy the deadline constraint for the application to be mapped, we next attempt to find a mapping that fulfils the CMP dark silicon (DsPB) constraint and minimizes PSN in CMP power domains, as well as minimizing the total Manhattan distance of communication between tasks. This can be formulated as multi-objective optimization problem which has been shown to be NP-Hard. Traditional multi-objective optimization methods (e.g., integer programming, genetic algorithms) to solve the problem are too slow for decision making at runtime. Hence, we propose a fast runtime heuristic to select a suitable mapping region and meet all constraints.

Algorithm 6: PSN Aware Mapping

Inputs: V_{dd}, DoP, application A, Sorted APG edges A(E) = {e_{12}, …, e_{nn-1}}

1: if *EstimatedPowerConsumption*(*A*) > DsPB **then**
2: return False //unable to find viable mapping
3: H ← {Ø} **L** ← {Ø} // Set of clusters
4: for all e_i **in** ***A(E)*** **do**
5: for each tasks T_j, connected to e_i **do**
6: if $T_j \notin$ **H or** $T_j \notin$ **L then**
7: **if** T_j .*High* **then** *push_back*(T_j,**H**)
8: else *push_back*(T_j,**L**)
9: *num_cluster* ← *create_clusters*(**H, L**)
10: if *num_available_domains* < *num_cluster* **then**
11: return False //unable to find viable mapping
12: else
13: *task-cluster-to-domain-mapping()*
14: return True

Output: Successful mapping or indication of failed mapping

Algorithm 6 shows our PSN-aware mapping approach. Given V_{dd} and DoP for the application *A* as inputs, the mapping heuristic aims to map all of the tasks of application *A* on to the CMP without violating the DsPB constraint, while minimizing the observed PSN. The algorithm first checks if the power consumption estimated from offline profiling is more than the available DsPB and returns false if the condition is not met without going further (lines 1-2). The tasks are labeled as high switching active or low switching active based on offline profile data (we found that more than two bins for switching activity increased heuristic overhead and also did not lead to notable benefits for runtime mapping). The heuristic utilizes the application graph *APG* of the application to be mapped to extract a sorted list of edges in the decreasing order of edge weights (communication volumes). To reduce PSN due to inter-task interference, the heuristic maps as many tasks with similar switching activity into the same power supply domain as possible. To reduce the NoC traffic, the heuristic also tries to map tasks with the highest communication volumes in the same domain. To achieve this, the heuristic iterates through the sorted edge list and

creates clusters of 4 tasks, corresponding to the power supply domains of 4 cores. As the tasks are categorized into two types (high and low switching), we create two lists corresponding to the two task types (line 3). The algorithm checks if tasks connected to the edge being evaluated are already assigned to a list (line 4); if not, they are pushed to one of the two lists (lines 5-8).

The two lists end up with tasks arranged in the decreasing order of communication volumes, as the edges have been evaluated in the decreasing order of their weights. Each list is then divided into clusters of four tasks in the order in which they are stored in the list (line 9). Any remaining un-clustered tasks from each list (< four; if the list size is not a multiple of four) are grouped into a single cluster. Clustering is done to ensure that (1) all but one of the created clusters will have tasks with similar switching activity, to be mapped in the same domain, (2) tasks with high communication volume between them are not mapped far from each other. If the available domains are less than the number of clusters (line 10) the algorithm returns false (i.e., no mapping found). If there are sufficient number of domains, each task cluster is mapped on to domains (line 13), in a manner that minimizes the hop distance between inter-domain mapped tasks. Further details of this step are omitted due to lack of space.

Figure 30 presents an example of the mapping heuristic for an APG and a sorted edge list. When mapping a task cluster on to a domain, if a cluster has two tasks of each switching activity level, tasks of the same level are mapped adjacent to each other, as shown in Figure 30, to reduce PSN due to inter-task interference (Section 4.2.4). On successful mapping, the heuristic returns true (line 11), indicating a successful mapping). After mapping, the tasks of the mapped application are scheduled using the fast and efficient earliest deadline first (EDF) scheduling scheme. For EDF, each task is assigned a deadline (priority) based on the deadline of the entire application, using a technique proposed in our prior work on task-graph scheduling [112].

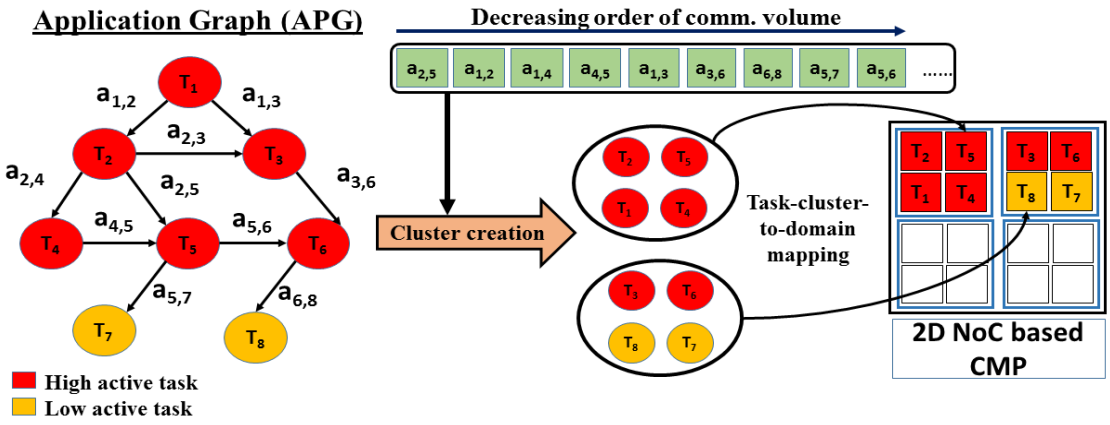


Figure 30 Overview of PSN aware mapping heuristic

4.3.3. TIME COMPLEXITY ANALYSIS OF PARM

The V_{dd} and DoP selection step runs in linear time complexity with respect to the permissible V_{dd} and DoP levels ($|V| = 5, |D| = 4$ in this work). In the mapping step, task clustering runs in linear complexity with respect to the number of edges in the APG. The total number of possible edges in an APG is $T \times (T+1)/2$, where T is the total number of tasks of an application. So, the clustering step has $O(T^2)$ complexity. *Task-cluster-to-domain-mapping()* has linear complexity with respect to number of tiles, hence the mapping step takes $O(\mathcal{T})$, where $\mathcal{T}$ is the total number of tiles in the CMP. The runtime complexity of EDF scheduling scheme, given by $O(T \times \log T)$, where T is the total number of tasks of an application, is masked by running in parallel with the mapping scheme that takes longer. So, *PARM* runs with a complexity of $O(V \times D \times \{max(\mathcal{T}, T^2)\})$ which depends on the number of CMP tiles, or number of tasks of an application, while V and D are small integers.

4.3.4. PSN AND CONGESTION AWARE NOC ROUTING (PANR)

To complement our PSN-aware mapping framework (*PARM*), we propose a *PSN- and congestion-aware NoC routing* scheme (*PANR*). *PANR* builds on and enhances a deadlock-free

turn model based routing scheme called west-first routing [113]. *Algorithm 7* shows the decisions made at each hop in a route. For each header flit in the input channel buffers of a NoC router, the routing scheme first computes the permitted destination hop directions (lines 1-3). It then selects a hop direction, from a set of permitted directions, by considering the voltage noise sensor data and incoming data rate (flits/cycle) from the routers in the tiles that are adjacent to the current tile. If the buffer occupancy of the input channel is beyond a threshold *B*, the output direction with the least incoming data rate is chosen to minimize the congestion (line 5). In the case of lower buffer occupancy than *B*, the output direction with the least observed PSN is chosen (line 6), to reduce the activity in the router in that direction, which in turn minimizes the overall PSN observed in the domain. Once a direction is decided for a header flit, the remaining flits in a packet follow the header flit.

Algorithm 7 PSN and Congestion Aware NoC Routing

Inputs: destination tile coordinates, PSN activity of adjacent tiles, traffic load in adjacent NoC routers

1: **for each** channel **in** Input_channels **do**
2: flit ← channel.packet.header_flit
3: *{permissible directions}* ← *WestFirstRouting* (flit.src, flit.dest)
4: **if** channel.buffer_occupancy > *B* **then**
5 hop ← min_data_rate*{permissible directions}*
6: **else** hop ← min_PSN*{permissible directions}*
7: **return** hop

Output: Next hop direction for packets in input channels

Overhead computation: The overhead of our routing scheme involves registers to store the values of noise and router traffic levels of the adjacent tiles, and additional wires to transmit those values between tiles. In addition, two 64-bit comparators are used per router to find the minimum values of PSN and incoming data rates of adjacent routers. The additional circuitry at each router consumes ~1 mW (3%) power and ~115 μm^2 (0.5%) area overhead over the baseline NoC router,

at the 7nm node. Hop selection takes 1 cycle in a router, at 1 GHz. This latency is masked by executing the selection step in parallel with the route computation step. The overhead of the network of digital PSN sensors used to sense the voltage noise [106], is around 413μm^2 which is negligible compared to the core area which is ~4 mm^2, and the router area of ~71300 μm^2, at a 7nm FinFET node. Results are calculated based on a previous work [114] that proposed a 7nm cell library for circuit and architectural simulation of NoCs in DSM technology nodes.

4.3.5. FAULT DETECTION AND CORRECTION

Our proposed PSN aware mapping and routing minimizes voltage fluctuations due to PSN. However, there may still be some cases when inter-core interferences lead to PSN above a certain threshold which leads to voltage emergencies (VE). VEs have the potential to cause errors in the functionality of logic devices and faulty application execution. To prevent such scenarios, applications are checkpointed at periodic intervals [91]. When a VE is detected using on-chip sensors in a tile that runs an application, it is rolled back to its last saved checkpoint and begins execution from there.

4.4 EXPERIMENTS

4.4.1. SIMULATION SETUP

We conducted experiments on 13 different parallel applications from the SPLASH-2 [77] and PARSEC [78] benchmark suites. We used the GEM5 [115] multicore simulator to generate the offline profile data for applications. We modeled the PDN as discussed in section 4.2.3 using

the SPICE simulator. We estimated the power consumption of different applications at various V_{dd}, clock frequency values, and app-DoP, at a 7nm FinFET technology node, using data from McPAT [94] and ITRS [116]. The DoP values used range from 4 to 32 (in multiples of 4, beyond which most of the applications were observed to have lower performance due to communication (synchronization) overheads. The *PARM* framework is implemented in C++, and the models of PDN, and checkpointing-and-rollback are integrated into it. We conducted trace driven simulations and assigned application execution characteristics to the tiles that are executing it, to speed up analyses. The power and communication traces are captured at 0.01ms interval, while applications execute for 400-700ms.

The switching activity of the core and the NoC router in a tile was observed to be directly proportional to its power consumption. A task running on a core that consumes more than 1.5W of power, for more than 150 captured intervals, on a tile that is executing it is assumed to be a high switching activity task, otherwise it is considered a low switching activity task. We also sample PSN values of the tiles at periodic intervals, and when a new application begins or a current one ends execution on the CMP, using the PDN SPICE model. The buffer occupancy threshold B for hop selection in *PANR* is set to 50% after analyzing the effects of different occupancy levels on router throughput, with a cycle-accurate NoC simulator.

We categorized 13 benchmarks into two groups: (i) communication-intensive benchmarks: *{cholesky, fft, radix, raytrace, dedup, canneal, vips}*; and (ii) compute-intensive benchmarks: *{swaptions, fluid-animate, streamcluster, blackscholes, radix, bodytrack, radiosity}*. As *radix* has properties of both, we use it in both groups. We employed three sequences of application with up to 20 applications picked randomly from each of the groups mentioned above. The sequences are categorized as compute-intensive, communication-intensive, and mixed, according to the type of

applications used in each sequence. We also experimented with three different inter-application arrival rates of 0.2s, 0.1s, and 0.05s to test the efficiency of our framework for different CMP utilization (workload subscription) scenarios.

We consider a 60 core 2D NoC-based CMP designed at 7nm FinFET technology node for our studies. The tiles are arranged in a 10 ×6 mesh layout. Each tile has an ARM Cortex A-73 low power mobile core, a NoC router, and a private L1 cache. We assume that *PARM* is part of the OS or middleware that assigns V_{dd}, DoP, and task-to-core mapping regions to each application according to the availability of DsPB and idle cores at runtime. The V_{dd} values supported by each tile (core + router) are between 0.4V (NTC) to 0.8V in steps of 0.1V. We assume that all of the cores on which an application is mapped to run at the same V_{dd}. We assume a dark silicon power budget (DsPB) of 65W. We treat PSN above 5% as a voltage emergency, similar to prior work [98]. We assume an overhead of ~256 cycles for periodic checkpointing with a 1ms checkpoint period, and ~10000 cycles of overhead for rolling back (restart) to an earlier state, after an error.

4.4.2. SIMULATION RESULTS

We compare our *PARM* framework against a prior work [109] that tries to minimize PSN using a harmonic mapping scheme, where tasks with high activity are mapped far away from each other. We refer to the scheme as *HM*. To show the impact of NoC routing on PSN, we compare our proposed *PANR* routing scheme with an *XY* routing scheme, and a scheme from prior work *ICON* [110] that minimizes PSN in NoCs, but is agnostic of application mapping. Overall, we evaluate six combinations of comparison works, *HM+XY, HM+ICON, HM+PANR, PARM+XY, PARM+ICON, PARM+PANR.* The prior works *HM* and *ICON* do not consider fault detection and correction in the event of failures due to VEs. For a fair comparison, we assume the presence of

fault detection and correction mechanisms in these works, as in our proposed work. Not having such mechanisms would result in unpredictable outcomes for applications, which is not desirable.

We simulated and analyzed total execution time (Figure 31), peak and average PSN (Figure 32), and total number of applications executed successfully (Figure 33), for the six frameworks mentioned above, across different types of workloads.

Compute intensive workload: As shown in Figure 31 and Fig.7, *PARM+PANR* shows up to 25.4% improvement in execution time and 4.15× improvement in peak PSN observed compared to *HM+XY*, whereas *PARM+ICON* and *PARM+XY* show 23.3% (3.82×) and 20.3% (3.81×) improvements in execution time (peak PSN) respectively. The *HM* framework does not consider the negative effect of mapping tasks with varying switching intensities in close proximity on PSN, as shown in Figure 28(b) in section 4.2. This leads to high PSN and hence poor performance of applications with *HM*. *PANR* routes flits through less congested and low switching paths to minimize PSN in the tiles that execute tasks with high switching activity, leading to lower PSN (Figure 32). *ICON* and *XY* routing schemes are unaware of the core activity and create a higher PSN (> 5%) (Figure 32) and hence have (a few) more number of VEs than *PANR*. Hence, *HM+PANR* has lower execution time over *HM+XY* and *HM+ICON*, due to its PSN aware routing scheme that minimizes overall PSN of cores and routers, and avoids VEs. Thus *for compute intensive workloads, intelligent task mapping and packet routing is crucial to minimize PSN.*

Communication intensive workload: NoC components consume 18-20% of the chip power, and are in the critical path of the application performance, when running communication intensive workloads. Hence it is important to consider the effects of NoC routers on PSN, along with cores. Figure 31 and Fig.7 show that *PARM+PANR* gives 34.3% improvement in execution time and 4.5× improvement in observed PSN compared to *HM+XY*. This is because in communication

intensive workloads, PSN-aware mapping and PSN-aware NoC routing work in tandem to reduce overall packet latency while keeping the router activity low around highly active cores. *ICON* only considers router activity and ignores the activity in the cores while making decisions. Hence PSN is higher in *ICON* compared to *XY* and *PANR* in both *HM* and *PARM* mapping schemes, as shown in Figure 32. This increases the tile activity and results in more VEs and worse application execution time in both *HM+ICON* and *PARM+ICON*. *PARM+XY* shows 23.3% performance improvement compared to *HM+XY* due to the PSN-aware mapping heuristic. Even though PSN observed with *PARM+XY* is much lower than *HM+XY* (Figure 32), there is an increased NoC traffic and higher packet latency which slows down the application when the application DoP is high with *PARM*. This case is handled better by the *PANR* routing scheme when compared to *XY*. *HM+PANR* is aware of the tile PSN and hence balances the activity with its intelligent routing scheme. This leads to lower PSN than *HM+ICON* and *HM+XY* (Figure 32), as well as 20% improvement in execution time compared to *HM+XY* (Figure 31). This proves that *PSN aware NoC routing schemes provide better results when they are combined with a PSN aware mapping scheme.*

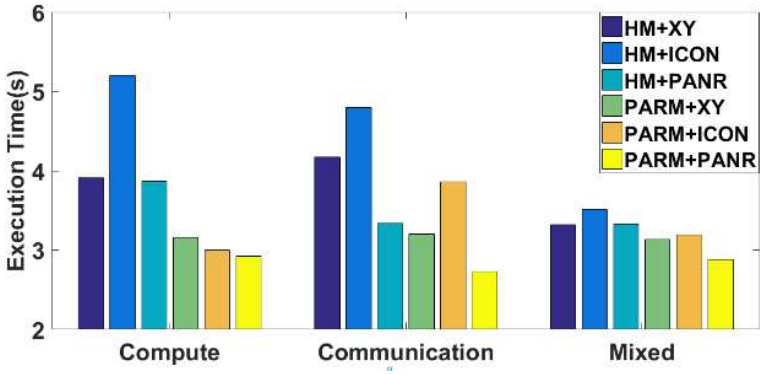


Figure 31 Total time taken to execute 20 applications with different frameworks across different types of workloads.

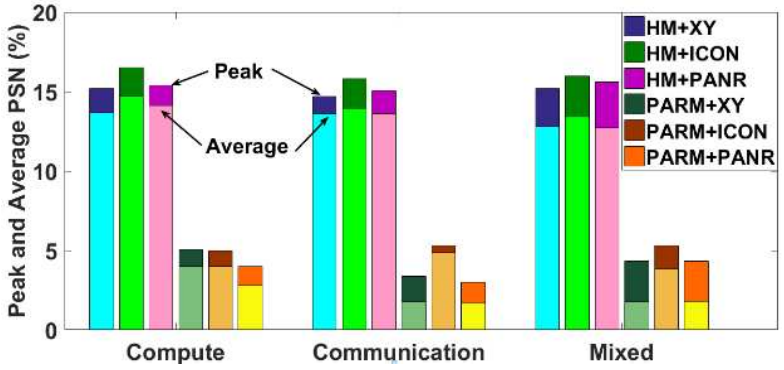


Figure 32 Peak and average PSN (as % of voltage supply) observed with different frameworks across different types of workloads

Mixed workload: When both types of applications are executed in a sequence, *PARM+PANR* give improvements of ~13.1%, which is better than *PARM+XY* (5.7%), *PARM+ICON (6.5%)* compared to *HM+XY*.

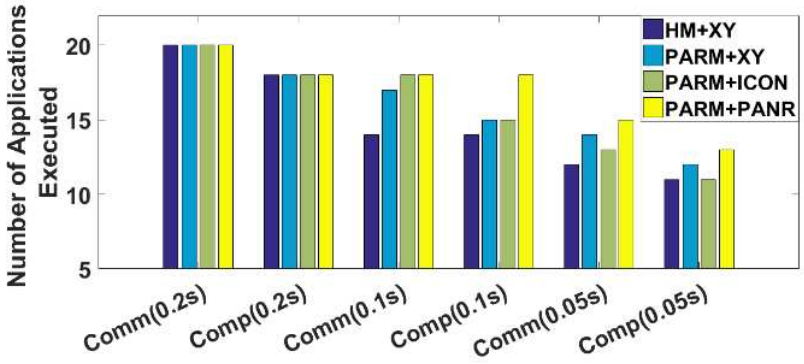


Figure 33 Total number of application successfully completed across different workload types and arrival rates for different frameworks.

Lastly, we analyze the efficiency of various mapping frameworks for different application arrival rates at runtime. We compare *HM+XY* with *PARM+XY, PARM+ICON,* and *PARM+PANR* across three different inter-application arrival rates and two workload types (with 20 applications each, as in earlier experiments). From Figure 33, it can be observed that all of the frameworks perform similarly at the 0.2s arrival rate since the applications arrive at a relatively slower rate. This slow

arrival rate allows more applications in the queue to be mapped within the given DsPB and deadline constraints. For communication intensive workloads, *PARM+PANR* maps 38(25)% more applications than *HM+XY,* and 17(9)% more application when the arrival rate is 0.1s(0.05s), compared to *PARM+XY*. For compute intensive workload *PARM+PANR* maps 38(18)% more applications than *HM+XY,* and 29(11)% more application when the arrival rate is 0.1s(0.05s) compared to *PARM+XY* and *PARM+ICON*. *HM* maps tasks in non-contiguous regions unlike *PARM* and scatters them across the CMP. This gives an opportunity to map more number of applications in *HM*. However, *HM* fails to do so without violating the DsPB constraint because of its increased power consumption (due to high V_{dd}). As *PARM* adaptively reduces V_{dd} and increases DoP to reduce PSN, it fits more number of applications within a given DsPB.

4.5 CONCLUSION

In this chapter, we proposed a novel runtime framework called *PARM* that minimizes PSN in near threshold voltages while meeting application performance deadlines without violating the DsPB constraint. We also proposed a PSN aware NoC routing scheme called *PANR* which routes flits through tiles with low switching activity to reduce the overall PSN in the CMP. Our experiments show that *PARM* enables up to 4.5× reduction in peak and average PSN observed, and up to 34.3% improvement in application execution times compared to the state-of-the art in PSN-aware runtime application mapping. *PARM* can be used to minimize the hardware overhead due to costly guardbanding techniques and decapacitance circuits to reduce the effect of interferences between cores, and minimize the software overhead due to schemes such as thread migration employed to keep the tile switching activity in check. Our experimental results also

show that reducing PSN in both cores and NoC routers is crucial for improvement in application performance.

5. RAPID: MEMORY-AWARE NOC FOR LATENCY OPTIMIZED GPGPU ARCHITECTURES

In this chapter, we propose a novel memory-aware NoC that has two (request and reply) planes tailored to exploit the traffic characteristics in GPGPUs. The request layer consists of low power, and low latency routers that are optimized for the many-to-few traffic pattern. In the reply layer, flits are sent on fast overlay circuits to reach their destinations in just 3 cycles (at 1GHz). In addition, as traditional memory controllers are not aware of the application memory intensity that leads to higher waiting time for applications on the shader cores, we propose an enhanced memory controller that prioritizes burst packets to improve application performance on GPGPUs. Experimental results indicate that our framework yields an improvement of 4-10× in NoC latency, up to 63% in execution time, and up to 4× in total energy consumption compared to the state-of-the-art.

General Purpose Graphics Processor Units (GPGPU) are becoming popular platforms for efficiently executing parallel applications. GPGPUs have been used in several data driven applications in cutting edge technologies such as artificial intelligence, deep learning, high performance computing in data centers, autonomous machines, and self-driving cars [117] [118] [119]. Recent research on GPGPUs has led to the optimization of thread level parallelism and maximizing the execution of cooperative thread arrays [120] [121] [122]. This has made GPGPUs more viable for high performance computation. Frameworks such as CUDA [4] and OpenCL [5] have provided programmers with a diversity of tools to parallelize their applications and leverage the computing capabilities of GPGPUs. But highly parallel applications running on GPGPUs generate huge volumes of communication traffic between cores and memory controllers (*MCs*).

Minimizing the latency of the network-on-chip (NoC) in GPGPUs is crucial to sustain high application performance.

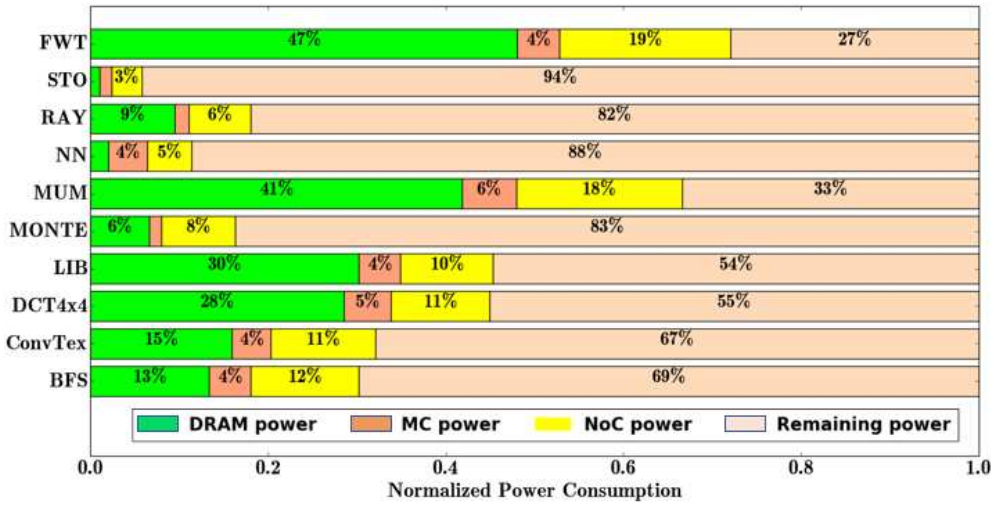


Figure 34 percentage of power consumed by major components in a 16 core GPGPU across various parallel CUDA applications.

The traffic pattern in GPGPUs is primarily many-to-few and few-to-many, with high volumes of traffic skewed towards memory replies [10] [26] [123]. Traditional mesh-based NoC topologies used in CMPs are not capable of handling such skewed traffic effectively, leading to underutilized resources, which increases NoC latency, and congestion at MCs. This impacts the performance of applications running on GPGPUs. In addition to the above challenges, the NoC accounts for up to 20% of total GPGPU power consumption in memory intensive applications (that also have higher DRAM power) at the 22nm technology node, as shown in Figure 34. *There is thus a critical need to enhance the NoC fabric to better suit GPGPU traffic scenarios without increasing the area and energy footprint.*

The main contributors to NoC latency are route computation, arbitration, and switch allocation at each hop. Thus, minimizing NoC latency requires optimizing some or all of these

steps. Also, the main contributor to NoC router area and power consumption are the input port buffers. In this article, based on an understanding of the traffic characteristics in GPGPUs, we propose a novel network architecture with low packet latency using memory-aware overlay circuits (described in Section 5.3.2) and with properties close to that of an ideal NoC which has all-to-all connections between cores and MCs. Our proposed NoC has a multi-plane, deadlock-free physical architecture, with memory-centric enhancements for the reply traffic as well as the request traffic. We adapt and enhance the ideas from [124] to enable single-cycle multi-hop traversal in this NoC. We also propose an enhanced memory controller (MC) that prioritizes *burst packets* to reduce the memory bottleneck and network backpressure issues that lead to increased wait time for applications executing on GPGPUs. Overall, our novel contributions in this chapter can be summarized as follows:

- We propose a novel NoC router architecture to efficiently cope with the many-to-few traffic pattern in the request plane, to minimize power consumption and network latency.
- We also introduce a new router architecture in the reply plane called *hinge router* to support single cycle overlay circuits where flits travel towards their destination along each network dimension in a cycle, stopping only at the turns.
- We propose a *global overlay manager* (*GOM*) that manages the time windows of each overlay circuit.
- We further propose enhancements to pipelined transmission at the MCs and enable simultaneous circuit overlay establishment, to improve utilization of NoC resources and minimize NoC latency as well as application execution time.
- We present an enhanced MC that prioritizes servicing *burst packets* from the output buffer of the MC.

Our rigorous experimentation with various CUDA benchmarks for 16-core, 64-core, and 144-core GPGPU platforms shows an improvement of 4-10× in NoC latency, 63% in application execution time, and up to 4× improvement in overall energy consumption, with up to 50% improvements in NoC area, compared to the best known prior works on NoC design for GPGPU architectures.

5.1 RELATED WORK

A few recent works have explored NoC architectures for GPGPUs. Kim et al. [10] propose a multiplane NoC called DA2 to improve the bandwidth utilization in links. They partition the network into request and reply planes, further partitioning the reply plane into multiple slices that are shared by MCs. With each MC possessing multiple slices instead of a single slice, the overall bandwidth utilization is improved. Also, as the routers in the reply plane slices do not have input buffers and complex arbitration, they can support higher frequency operation. However, DA2 has limitations due to serialization overhead and the reduced bandwidth of reply network slices, which effects the scalability of their model. Jang et al [26] propose an MC placement approach and a virtual channel partitioning scheme in mesh-based GPGPU NoCs. In their work, MCs are placed along the x-axis of the mesh and flits use XY routing for requests and YX routing for replies, to reduce congestion at MCs and competition for links along the Y-axis. However, the reply packets still compete with request packets for links at the center of the NoC. The work also requires a rigid MC placement, and consumes more area and power compared to a baseline NoC owing to additional virtual channels and arbitration requirements. *Unlike these works, our proposed approach in this chapter does not incur any serialization latency.*

Another recent work [125] considers many-to-few traffic pattern in the request plane, and uses a constrained MC placement rule to propose a new low-cost conflict free NoC to reduce the

energy consumption of the NoC. The request plane of the NoC is partitioned into slices called *enetworks* that are shared between columns of cores for rapid transmission of requests to the MCs. A token-based mechanism is also proposed to ensure a conflict free sharing of *enetworks* by different cores that send requests to the same MC. However, when the GPGPU has over a hundred cores and tens of memory controllers, the channel width of the *enetwork* diminishes drastically and the token sharing network adds to the latency of the NoC. *Our approach proposes a low cost and low power NoC in the request plane, and a fast overlay network in the reply plane that minimizes NoC latency, without incurring energy consumption and area overheads.*

A few recent works [126] [127] [124] also attempt to minimize NoC latency by using bypassing. For example, Chen et al. [124] propose single cycle multiple hop techniques that enable flits to bypass the router arbitration and traverse the NoC in a few cycles to their destinations. *We adapt and enhance this approach with awareness of traffic characteristics in GPGPUs and the integration of smart circuit reservations, to more aggressively minimize communication conflicts, reduce overall NoC latency, and improve bandwidth utilization.*

Several other prior efforts attempt to improve latency for NoCs with awareness of traffic pattern characteristics. For example, Cong et al [123] propose hybridized circuit switching with circuit reservation that is done in advance by a global manager based on traffic pattern awareness. Abousamra et al. [128] [129] [130] explain the benefits of circuit switching for different traffic scenarios, including many-to-few and few-to-many. However, in all of these works that involve circuit switching, there is a notable circuit setup and teardown overhead. The effort of establishing a circuit is wasted when the flits do not arrive at the estimated times. Network prioritization and packet scheduling techniques are proposed in [131], [132] to improve NoC throughput and latency.

Unlike these efforts, our approach that opportunistically utilizes circuits does not incur significant circuit setup and teardown overheads that impact NoC latency.

More recent work, e.g., [133], has proposed a hybrid NoC architecture with wired and wireless routers for improving the overall network latency. However, such wireless on-chip NoC components have high fabrication overheads and may also not scale well for hundreds of GPGPU cores. In [134] [135] [136], software and middleware level techniques such as data prefetching or intelligent warp scheduling are proposed to mitigate the memory bottleneck that is caused by high volume of read reply packets from MCs to cores. In [137] a priority NoC is proposed to resolve bottlenecks created by cache coherence traffic in CPU based many-core NUCA architectures. We propose an enhanced MC that prioritizes *burst packets* received from DRAM in response to the read request packets that are sent over a short duration. *Our work is a hardware approach that is complementary to the higher level aforementioned solutions that alleviate the bottleneck at MCs.*

5.2 BACKGROUND AND MOTIVATION

5.2.1. BASELINE ARCHITECTURE CONFIGURATION

We consider a manycore GPGPU based accelerator similar to [138] as the baseline platform for our work. An accelerator typically consists of a traditional x86 or ARM based core, and a grid of shader cores with private L1 caches (instruction cache, data cache, and texture cache), to drive the data parallel multi-threaded workloads. A shader core consists of parallel integrated pipelines with a common instruction fetch unit that executes a single instruction on multiple data (SIMD) simultaneously. Each integrated pipeline has an integer arithmetic logic unit and a floating point unit. A shader core also has several load store units that fetch data from a private L1 cache or from

the main memory. A GPGPU based accelerator has a shared L2 cache bank located at the MCs that caches data coming from the main memory. All the shader cores and MCs are connected to an interconnection network. Whenever shader cores need to communicate with each other, this communication occurs via the MCs (with data stored either in L2 banks at the MCs or DRAM) where one shader core writes to the memory and the other reads from it. There is no direct packet transfer between shader cores over the NoC. The communication between CPU and GPU cores takes place through main memory. The shader cores send read/write requests to MCs over a NoC. A memory reply takes several cycles based on the location of and availability of data (either at L2 or DRAM). The baseline NoC architecture between the shader cores and the MCs has a channel width of 128-bits and consists of 4-stage routers (stage 1: buffer write; stage 2: route computation, stage 3: virtual-channel/switch allocation; stage 4: switch/link traversal) with 5 virtual channels (VCs) per input port and 4 flit buffers for each VC, connected to each shader core. Flits are routed along the XY path from source to destination. In this work we only focus on optimizing the NoC that connects shader-cores and memory controllers. Henceforth, the term *cores* implies *shaders cores* for the remainder this article.

5.2.2. OPPORTUNITIES FOR NOC OPTIMIZATIONS IN GPGPUS

In this section we explain the motivation for our work. In highly multi-threaded applications, the reply arrival rate from MCs to shader cores is very high. For example, applications such as *Transpose* and *Convolutions* from the CUDA SDK have a reply arrival rate of 1.5-3 flits per cycle [123]. Such traffic needs a high throughput on-chip network fabric. But traditional homogeneous mesh-based NoCs are incapable of efficiently handling high volumes of reply traffic, causing packets to be stored in the buffers at the MCs for several clock cycles. This in turn creates

congestion or traffic hotspots around MCs that leads to poor overall application performance. Under ideal conditions we would like to have a NoC with all-to-all connections between MCs and shader cores to satisfy latency and throughput requirements. But such a network is impractical due to its high power and area overheads. Thus, *there is a critical need for customized and low-overhead NoCs for GPGPUs, to handle their unique traffic characteristics*.

In this chapter, observations regarding the traffic pattern of the applications executed on GPGPU are exploited to innovate and enhance the NoC architecture. Firstly, the traffic pattern of memory requests to the NoC, is uniquely *many-to-few*, i.e., memory read and write requests are sent from many cores to a few MCs. In traditional manycore processors with n cores each running a thread, the upper bound on the total number of communication flows is $O(n^2 + n \times m)$, where m is the total number of MCs, and $m \prec\prec n$. However, in a GPGPU, the upper bound on the total number of communication flows (each flow represents a stream of requests sent from shader cores to MCs) is $O(n \times m)$. In such a scenario, the number of VCs per input port of a NoC router (primarily used for sharing a physical channel between different communication flows) can be reduced. Secondly, in a *many-to-few* traffic scenario, the total number of possible output directions that an incoming flit to a NoC router takes could be restricted based on the location of the router. As shown in Figure 35, if the packets use an XY routing scheme, based on the location of the router relative to the MC, flits traverse in one of the following directions ($N \rightarrow S$, $S \rightarrow N$, $S/N \rightarrow L$) on the Y axis. This in turn reduces the total number of logical comparisons done in the route computation step compared to the baseline NoC router (in-depth explanation is presented in Section 5.3.1). Given the above two observations, we propose a new router architecture that reduces the power consumption and latency of the NoC in the request plane.

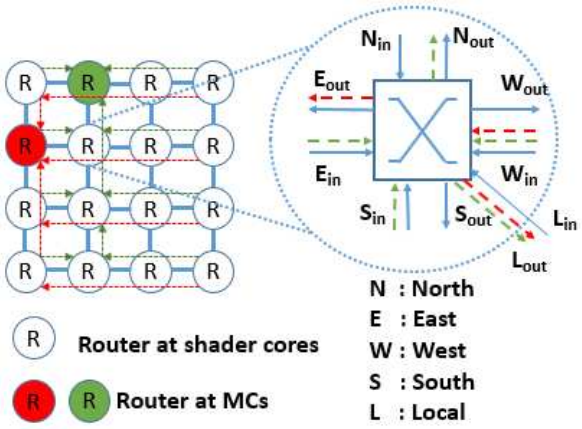


Figure 35 Demonstration of routes (represented by dashed lines) taken by the request packets from different cores to memory controllers in a 4×4 NoC with XY routing scheme.

Figure 36 shows the traffic pattern from MCs to individual cores for a 4×4 NoC. The circles colored red, blue, green, and yellow denote routers at MCs. The colored arrows in (a), (b), (c), and (d) of Figure 36 denote the XY routes taken by the reply flits from each of the four MCs in the design. As there is no inter-core traffic in the reply plane in the GPGPU, it is possible to reserve circuits along the path shown in Figure 36 (a)-(d) for a time window, during which the flits from the corresponding MCs travel without having to go through the route computation stage at each hop. A circuit (e.g., in Figure 36 (a)) should be reserved only for a limited time window, to allow for circuits from other MCs to also be established (e.g., in Figure 36 (b)) as needed, depending on the application data requirements.

The above process is analogous to a circuit switched network where routers reserve paths for a particular flow and the reservation is voided once the tail flit enters a router [128] [129] [130]. However in our case circuits are reserved for MC traffic only along the reply plane. We call these *memory-aware overlay circuits* because they form a new topology on top of an existing (e.g., 2D mesh) topology. While flits traversing these circuits would not encounter congestion delays, they will need to stop at every router for the crossbar and link traversal stages. However, by using

asynchronous links with repeaters and latches in routers, it is possible to allow flits to bypass the buffer write, route compute, and switch allocation stages and travel across multiple routers along the X or Y directions in a single cycle. Prior work [124] indicates that such asynchronous links need one voltage locked repeater per hop distance (~2mm) at 1 GHz clock frequency. The total peak power dissipated by repeaters per link is around 64µW at the 22nm technology node. In a 12×12 NoC, there are 24 asynchronous links. The overall power overhead of asynchronous links is ~1.5mW, which is negligible compared to the power dissipated by cores in a 12×12 GPGPU platform. Similarly, the area overhead of voltage locked repeaters on asynchronous links in a 12×12 platform is ~1.2%, since each repeater consists of 2 inverters and a feedback loop to know the signal swing. The authors in [124] also demonstrate that asynchronous links can traverse up to 16 hops (router crossbar + Mux) at 1GHz frequency in 45nm technology node. Hence, in our work we have conservatively considered the maximum link traversal of 12 hops in each direction in the reply NoC that is designed at the 22nm technology node.

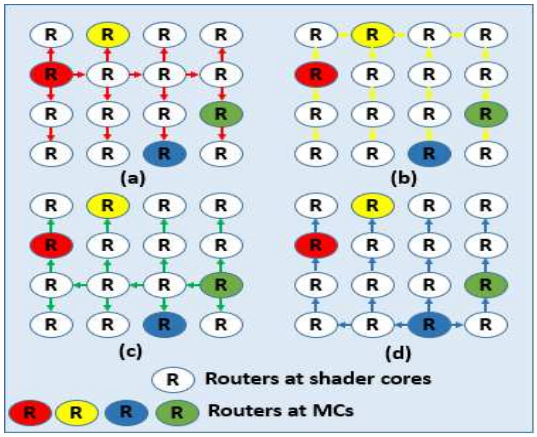


Figure 36 A 4×4 NoC showing dedicated overlay circuits for each MC in a few-to-many reply traffic scenario

Lastly, Figure 37 shows the number of *burst requests* in the total number of memory requests sent from shader cores to MCs when an application is executed on a GPGPU. We have identified

the pattern of burst requests from the traces obtained by executing CUDA benchmark applications on the GPGPU-simulator. When a core sends two or more requests within a short time duration, those requests can be labeled as *burst requests*. The *burst requests* are typically sent when the warps being executed on a shader core have less data locality, or because of the private L1 cache pollution due to excessive number of warps scheduled on the core. Burst requests generally are the major cause of bottlenecks at the MC, which increases warp waiting time at a shader core, and delays the servicing of warps that are waiting in the queue to be scheduled on the same shader core. *To avoid this problem, the responses to burst requests, called burst packets, should be given a higher priority over normal packets for transmission from MC output queue.*

All of the above enhancements overcome key bottlenecks to enable low-overhead, low-latency NoC data transfers in GPGPUs. In the next section, we explain our architecture and approach for GPGPU NoC optimization in detail.

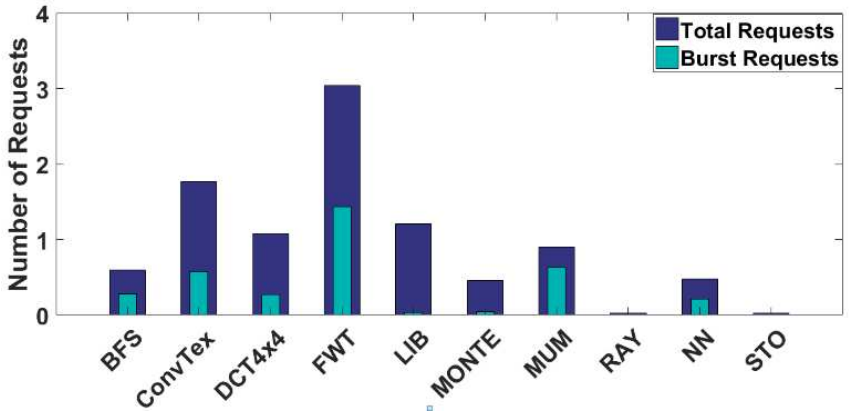


Figure 37 Total number of memory requests and burst requests sent across various parallel CUDA benchmark applications

5.3 RAPID MEMORY-AWARE NOC: OVERVIEW

Our RAPID memory-aware NoC architecture employs a 2D mesh topology to connect shader

cores and MCs. We utilize XY routing in the request plane, which ensures freedom from routing deadlocks. To further avoid request-reply protocol deadlock, the network is physically partitioned into a request plane and a reply plane each having a channel width of 64 bits. Memory requests from shader cores to MCs are sent on the request plane, whereas replies from the L2 and DRAM that arrive at MCs are sent to the cores on the reply plane. *The primary constraint we consider for MC placement is that no two MCs can have the same X or Y coordinates.* This is to ensure that each column and each row of the 2D-mesh NoC consists of only one memory controller. This constraint is crucial in establishing conflict free overlay circuits in the reply plane for high speed packet transfers.

To achieve low power and low latency communication in GPGPU architectures, RAPID employs: (1) A request plane NoC that minimizes the energy required to deliver memory requests from shader cores to MCs using a modified router that is less complex than the baseline router, and suits the many-to-few traffic pattern; (2) A Memory Aware CiRcuit Overlay (*MACRO*) reply plane, that minimizes flit latency by establishing fast overlay circuits that change over time, and by utilizing single cycle multi-hop flit traversal; (3) An enhanced MC that prioritizes burst response packets in its buffers over the *MACRO reply plane* to reduce the bottleneck that adversely affects NoC latency. We explain the details of each contribution in the following sections.

5.3.1. REQUEST PLANE NOC

As explained in Section 5.2.2 there is a scope for trimming down the router complexity in the request plane NoC. Figure 38 shows the baseline router with four pipeline stages. In a baseline router, each channel has 5 VCs. However, in a GPGPU, the total number of communication flows in the request plane is significantly less compared to traditional CPU traffic as explained in section

5.2.2, which results in unused VCs at the input ports. The unused VCs in each router can be trimmed down at each input port. Trimmed VCs reduces the number of flit buffers, and the size of the look-up table that stores the flow control credit information in a router. We propose to reduce the number of VCs to 2 per channel to minimize the router power consumption, and reduce the VC allocation computation time.

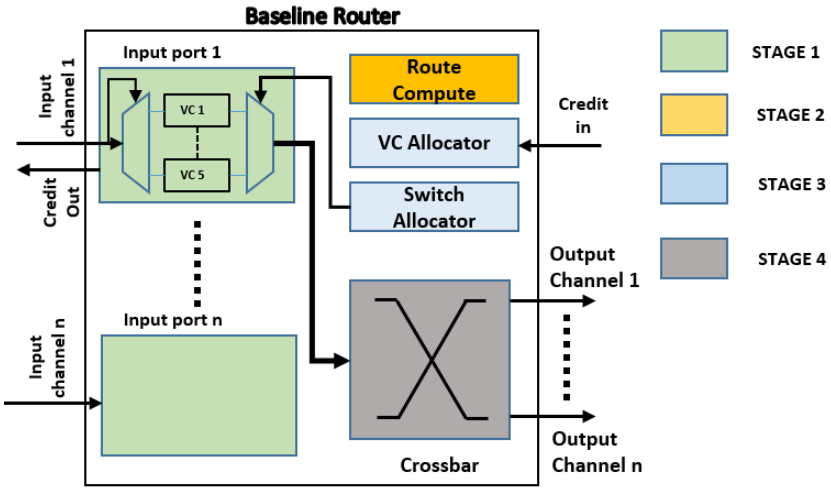


Figure 38 Baseline router with 4 stage pipeline

Further, the route computation logic of the request plane routers can be simplified based on the location of a router. As explained in Section 5.2.2, a many-to-few traffic pattern with XY NoC routing restricts the flit traversal in the routers along the Y-axis. In Figure 39, the routers that are located north of an MC in any column do not route the incoming flits further along the North direction, as there is no destination MC located further North. Similarly, the routers that are located south of an MC in a column do not route the incoming flits further along the South direction. These restrictions on flit routing are because of the constraint that each column can only have one MC. We use this observation to reduce the number of comparison operations in the route compute stage.

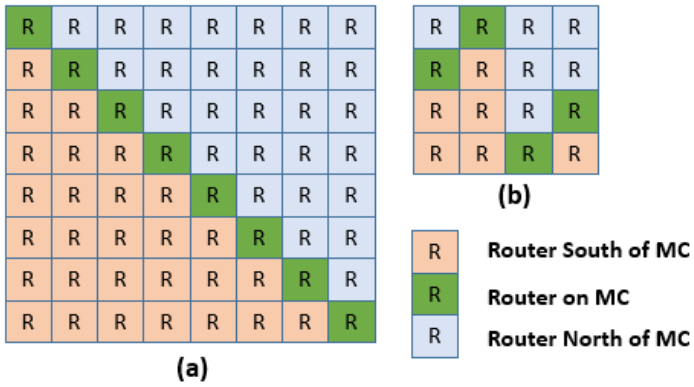


Figure 39 Location based routers in the request plane of 2D-mesh NoC with different MC placements for (a) 64-core GPGPU and (b) 16-core GPGPU

Algorithm 8 : Original XY Routing scheme

```
1:  for flit in each input port do
2:    if ( flit.src.x > flit.dest.x ) then
3:      out_direction = west
4:    else if (flit.src.x < flit.dest.x) then
5:      out_direction = east
6:    else if (flit.src.x == flit.dest.x) then
7:      if (flit.src.y > flit.dest.y) then
8:        out_direction = south
9:      else if (flit.src.y < flit.dest.y) then
10:       out_direction = north
11:     else if (flit.src.y == flit.dest.y) then
12:       out_direction = local
13:     end if
14:   end if
15: end for
Output: out_direction
```

Algorithm 8 shows the pseudo code of the original XY routing scheme. For a head flit in each input direction, the algorithm first compares the X-coordinates of source and destination, and chooses east or west as output directions (lines 1-5). If the X-coordinates of source and destination match, it compares Y-coordinates to choose north, south, or local as output directions (lines 6-15). Overall, there are six comparison operations (lines 2, 4, 6, 7, 9, 11) used to compute the output direction for head flit waiting in the virtual channel buffer of each input port. This amounts to six 3-bit comparators. *Algorithm 9* shows the pseudo code of the modified XY routing scheme used in NoC routers located north-of/south-of/on-the the MCs. The incoming flits whose X-coordinates

match (line 6) are directly sent to south/north/local ports (line 7) based on the location of the router. Hence, the modified XY routing needs only three comparison operations.

Algorithm 9: Modified XY Routing scheme

```
1:  for flit in each input port do
2:    if ( flit.src.x > flit.dest.x ) then
3:      out_direction = west
4:    else if (flit.src.x < flit.dest.x) then
5:      out_direction = east
6:    else if (flit.src.x == flit.dest.x) then
7       out_direction =north/south/local
8:    end if
9:  end for
Output: out_direction
```

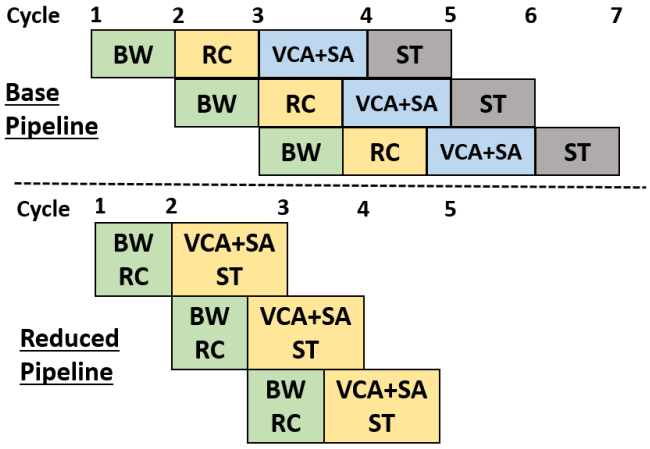


Figure 40 Pipeline stages of a baseline router and a modified router in the request plane of memory aware NoC

To design a functional NoC using the modified router architecture, the request plane needs three types of routers as shown in Figure 39 Location based routers in the request plane of 2D-mesh NoC with different MC placements for (a) 64-core GPGPU and (b) 16-core GPGPU. For example, routers north (south/local) of an MC are configured with a modified XY routing scheme with output direction as south (north/local) in line 7 of ***Algorithm 9***. This reduces the route compute time. Using the reduced VC per channel and modified XY routing scheme, we reduce the

pipeline stages of the modified router to two stages from four stages in the baseline router. Using the reduced pipeline router, as shown in Figure 40, the flit passes through a router 2 cycles faster. Figure 41 shows the power consumption of the modified request plane router compared to the baseline router. Due to the reduced VCs size, the buffer power consumption, which constitutes more than 60% of the router power, and virtual channel allocation and switch allocation (VCA+SA) power consumption are significantly lowered. Thus, having a modified router consumes lower power and increases the speed of NoC in the request plane.

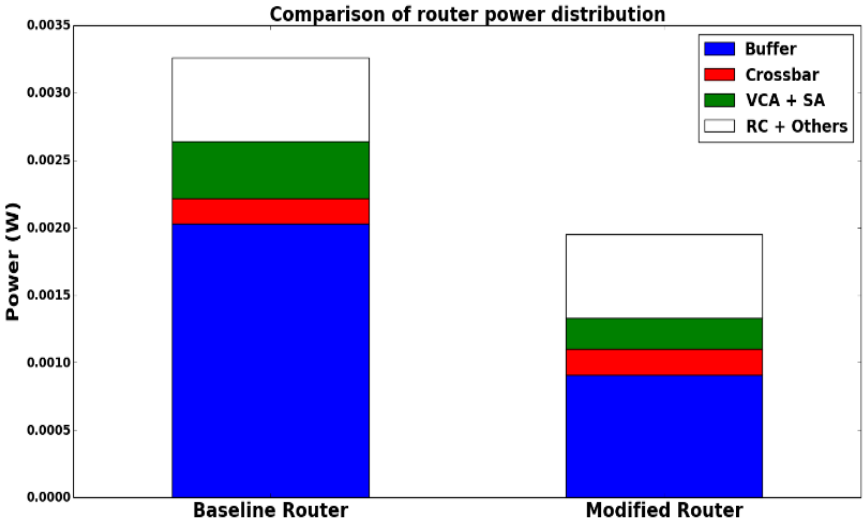


Figure 41 Power consumption of NoC router components for baseline and modified routers.

5.3.2. MACRO REPLY PLANE NOC

To realize the *MACRO* reply plane, we require: (*i*) a global monitoring mechanism that intelligently computes time windows for each overlay circuit, and (*ii*) a new router architecture that lets the flits bypass its stages while routing them to their destination. These key components of the *MACRO* reply plane are discussed in the rest of this section.

5.3.2.1. GLOBAL OVERLAY MANAGER

In our proposed approach, execution time is divided into epochs. Each epoch is divided further into smaller time windows. At the beginning of an epoch, a Global Overlay Manager (GOM) allocates an overlay circuit for each time window in the epoch. This is akin to time division multiplexing (TDM) of overlay circuits. Every overlay circuit is associated with a unique MC. The duration of a time window allocated to an overlay circuit is proportional to *(i)* the number of outstanding packets in the output queues of the MC associated with the circuit, and *(ii)* the packet arrival rate at the MC from L2 and DRAM. The GOM utilizes the values of these two parameters at the end of an epoch to make decisions about time window durations for the next epoch. Each MC sends its average reply arrival rate (from L2 and DRAM), and buffer occupancy to the GOM at the end of an epoch. The GOM uses that information to compute a weighted function ξ(m):

$$\boldsymbol{\xi(m) = \alpha.A(m) + \gamma.B(m)} \qquad (28)$$

where A(m) is the reply arrival rate and B(m) is the average buffer occupancy at the MC m in the previous epoch. α and γ are coefficients of the weight function. The GOM compares ξ's of each MC and computes time window durations T_1,T_2,T_3..T_m for the next epoch as:

$$\boldsymbol{Ti = K * \xi(i)/[\xi(1) + \xi(2) + \xi(3) + \cdots + \xi(m)]} \qquad (29)$$

where T_i is the time window of the i^{th} MC overlay and $\xi(i)$ is its weight function. The ratio of the weight functions is then multiplied by a constant K which is equal to the periodicity of the time windows in an epoch. The time windows repeat periodically for E/K iterations in an epoch, where E is the epoch interval duration and K is the periodicity of the time window set. By having the time windows repeat and overlays switch multiple times in an epoch, MCs send flits in multiple bursts across an epoch. Figure 42 shows an example of time windows across two epochs. The GOM broadcasts time window durations at the start of an epoch to all routers. The routers store this

information inside special buffers and subsequently establish (and then tear down) overlay circuits that adhere to the allocation decisions made by the GOM.

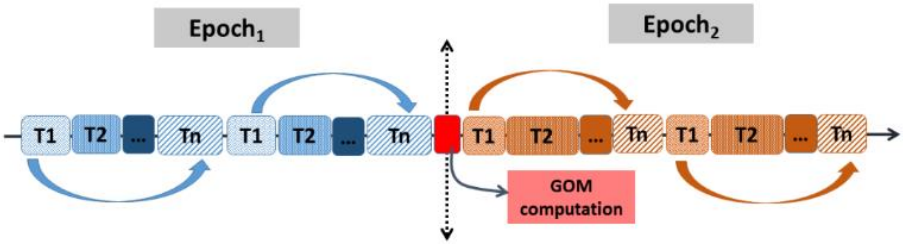


Figure 42 Time windows of overlay circuits. Each window is repeated periodically (E/K times) till the end of an epoch.

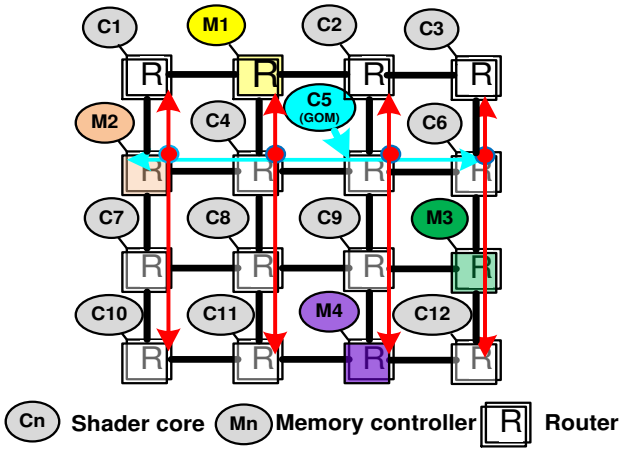


Figure 43 Overlay circuit used for broadcasting time window information to each router. GOM is located at core5. Broadcast over fast circuit in X direction is denoted by light blue line and in Y direction by red lines.

One may ask two important questions about GOM operation:

(i) What is the overhead involved in computing and broadcasting time window information? MCs send their status information to the GOM at the end of their last time window in an epoch, before their overlay circuit is torn down. The GOM is located near the center of the NoC, so it takes up to three cycles for this information to travel on overlay circuits through bypass links to reach the

GOM. The GOM takes up to 21 cycles to process the information and compute time windows for each MC (as per Eqs. (28), (29)), for the next time epoch. The generated time window information is broadcast to the routers on a separate overlay circuit, as shown in Figure 43. As a time window is always < 1000 cycles in our framework, at most 10 bits are required to transmit the time window duration for the overlay circuit of an MC. Thus, for a 4×4 NoC with 4 MCs, only at most 40 bits (1 flit) must be broadcast by the GOM at the start of an epoch. Similarly, for 8×8 and 12×12 NoCs, the broadcast consists of 80 and 120 bits respectively, that can be accommodated in 2 flits. This circuit is established while the GOM computation is taking place at the beginning of each epoch. The broadcast operation takes three cycles over an overlay circuit. So, the entire operation takes less than 30 cycles (irrespective of NoC size) which is negligible compared to a typical epoch interval that lasts for thousands of cycles.

(ii) Sometimes, the L2 and DRAM arrival rates at an MC vary a lot from one epoch to the next. How are such variations in reply arrival rate and buffer occupancies at MCs handled in an epoch? Such variations can lead to congestion at MCs or an underutilization of overlay circuits. To handle these variations, we allow time window durations to be changed across epochs, as discussed earlier in the section and shown in Figure 42 for T_2. It is also possible to adjust the epoch time such that GOM updates time windows frequently and adjusts to the traffic conditions. The process of fixing the epoch duration happens at design time, by testing for the lowest average latency on different benchmarks with different epoch intervals.

5.3.2.2. ROUTER ARCHITECTURE

Along the fast overlay circuits, flits travel in the X and Y directions in one cycle, bypassing routers and being stopped (latched) only at turns. These turns are called *hinges*. To realize this

behavior, we propose a novel *hinge router* architecture, as shown in Figure 44. All routers in *RAPID*'s reply plane are hinge routers. Each port of a hinge router supports a bypass path made of asynchronous repeaters and links, which is shown in the figure as a green line. When not in bypass mode (e.g., when a flit needs to turn), the router *hinges* the flit in a latch. The local port receives flits from the core or MC interface FIFO buffer, while the other ports receive flits from other routers. The three main components of a hinge router are: *(a)* Overlay controller, *(b)* Route lookup table, and *(c)* Input selection. These are discussed in the rest of this subsection.

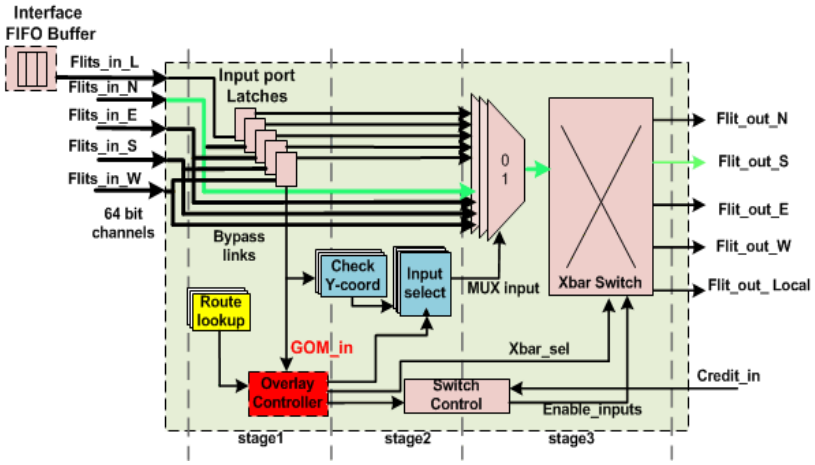


Figure 44 Architecture of a hinge router on reply plane. The figure shows one asynchronous bypass connection between North_in and South_out ports only, although bypass paths exist for all ports (except local port).

Overlay controller

Algorithm 10 shows the pseudo code of the overlay controller in a hinge router. The router has a local counter for tracking time window duration and a global counter for tracking the epoch duration. At the beginning of the first epoch, the time windows are set to default values (line 2; all the time windows are equal). At every cycle (line 3), the controller checks if the global and local counters have reached their thresholds (lines 4-5). The set of time window thresholds of overlay circuits (T) are received from the GOM and stored in a buffer from where they are accessed and

checked against counters periodically. If a local counter reaches a time window threshold, the controller sends the overlay number for the next time window as an input to a *route lookup table* to get the *crossbar configuration* of the upcoming overlay circuit (line 7). The crossbar configuration specifies the mapping of the switch between input and output ports. The controller sends that information to another important module called *Input selection unit* (line 8) that decides the input source for the configuration (either latches or bypass links). It then adds the local counter value to the global counter (line 9), resets the local counter (line 10), and begins tracking the new time window. In this manner, at the end of a time window, all the hinge routers change their configurations collectively to form a different overlay circuit.

Algorithm 10: Pseudo code for overlay controller operation

```
Inputs: GOM_input, epoch_duration, def_ win_values; N (no. of MCs)
Variables: local_counter, global_counter, i, set(T)
1:  local_counter = 0; global_counter = 0; i = 0 // reset counter values
2:  set(T) = def_ win_values // initialize time window durations
3:  for every cycle do
4:    if (global_counter < epoch_duration) then
5:      if (local_counter == Ti) then
6:        i = (i+1) % N // move to next time window
7:        xbar_config = get_next_crossbar_config(overlayi)
8:        input_select(xbar_config)
9:        global_counter += local_counter
10:       local_counter = 1
11:     else if (local_counter < Ti) then
12:       local_counter++
13:     end if
14:   else if (global_counter == epoch_duration) then
15:     set(T) = read_values(GOM_input) // save time windows
16:     global_counter = 0; local_counter = 0; i = 0 // reset counters
17:   end if
18: end for
Output: Crossbar configuration for each new time window
```

This entire operation takes two cycles at the beginning of each time window T_i, which is negligible when compared to the duration of the time window (~few hundred cycles). Once all the N time windows in the set have been serviced, the process of servicing time windows begins again at the first time window (line 6) and this continues till the end of the epoch (Figure 42). If the

global counter reaches the epoch duration value, the controller saves the GOM input it receives at the beginning of an epoch (line 15) in its buffers and resets the local and global counters (line 16) before starting to service the new epoch.

Route lookup table

As mentioned above, the overlay controller sends a request to the route lookup table for the crossbar configuration of an overlay circuit at the beginning of each time window. Each router has a different configuration for its lookup table based on its location on the 2D mesh. For example, Figure 36 showed a 4×4 architecture with *m*=4 MCs. Figure 36 (a)- Figure 36 (d) highlight 4 overlay circuits with each router having different connections for different overlays depending on its location on the mesh. Hence, a route lookup table in a router has *m* rows for *m* overlay circuits. The lookup table is configured as a read-only-memory at design time in each router.

	Router$_n$				
Overlay$_1$	E–» ϕ	W–» ϕ	N–» ϕ	S–» N	L–»Lat(N)
Overlay$_2$	E–» ϕ	W–» ϕ	N–» ϕ	S–» N	L–»Lat(N)
Overlay$_3$	E–» ϕ	W–» L	N–» Lat(L)	S–» Lat(L)	L–»Lat(L)
Overlay$_4$	E–» ϕ	W–» ϕ	N–» S	S–» ϕ	L–»Lat(S)

Figure 45 Route look-up table for green router from Figure 36 (mapping of output to input ports). L is local port. Number of rows = number of overlay circuits (number of MCs); each column represents an output port.

Figure 45 shows an example of a route lookup table in a hinge router in a 4×4 NoC with four MCs. The table has mappings of output to input ports (ϕ indicates no mapping, Lat(x) indicates mapping between an output port and the input direction x's latch). There are 10 possible input sources (bypass links and latches) for each of the 5 output ports per overlay. It takes 6 bits to represent all the 50 possible mappings per overlay ($\log_2 50$). Hence, in a 16-core system with 4

MCs and 4 overlay circuits, the size of the table is 120 bits per hinge router. In an 8×8 64-core architecture with 8 MCs, the routing table also has a nominal footprint of only 240 bits. In a 12×12 144-core architecture with 12 MCs, the size of the routing table is only 480 bits.

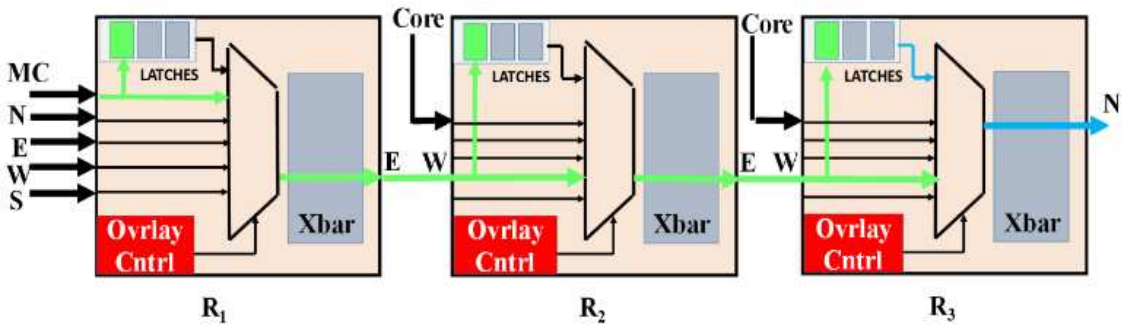


Figure 46 Flit propagation on bypass links of hinge routers

Input selection

All the connections of an overlay circuit are established at the beginning of a time window. A flit enters the reply plane from a local port of an MC and travels across the bypass links in the X direction in a single cycle and gets latched (hinged) along all the routers in that direction. The hinge router whose Y direction matches with that of the destination then transmits that flit across the bypass links in that specific Y direction. The remaining routers along the X direction whose Y directions do not match the flit destination do not transmit along the Y direction, and simply drop their hinged flits. Figure 46 illustrates this process. At router R1 a flit starts traversal from a local port that is connected to an MC. This flit travels through R1, R2 and R3 on the bypass links along the X-axis in a single cycle. The horizontal green lines show the bypass path. At each router, the flit is also hinged in the latch. Then in the next cycle, latched entries are matched for a turn and if a match is found (e.g., in R3) the flit is sent on the bypass path along the Y direction (North in this example), as shown in Figure 46 with blue lines at R3.

Algorithm 11: Pseudo code of Input selection unit

Inputs: controller_input(xbar_config), local_coord

```
1:  for each input d in xbar_config do
2:    if ( d == latch ) then
3:     if (Input latch[d] contains a flit) then
4:       if (flit.dest_coord.x == local_coord.x) then
5:         if (flit.dest_coord.y > local_coord.y) then
6:           north_mux_select(Input_latch[d])
7:         else if (flit.dest_coord.y < local_coord.y) then
8:           south_mux_select(Input_latch[d])
9:         else if (flit.dest_coord.y == local_coord.y) then
10:            local_mux_select(Input_latch[d])
11:        end if
12:      else if flit.dest_coord.x ≠ local_coord.x
13:        drop(flit)
14:      end if
15:    end if
16:   end if
17: end for
```

Output: Input select for the multiplexers

Algorithm 11 shows the pseudo code of the input selection unit. For each input of the crossbar configuration (line 1), if the input source is a latch (line 2) it is checked for a flit (line 3). If a flit is present, it is checked for its destination X coordinate (line 4). A match signifies either a destination or a turn. If the destination Y coordinate is also the same as the flit's Y coordinate, the latched flit is sent to the local port (line 10), else the hinged flit is sent along the north or south ports (lines 6, 8) for transmission along the Y direction. If there is no match, then the hinged flit is dropped (line 13). *Algorithm 11* can easily be implemented to complete in one cycle. Traversal along the X and Y directions takes one cycle each. *Hence, the entire traversal along an overlay circuit takes a maximum of 3 cycles.*

5.3.2.3. OVERLAY CIRCUIT ENHANCEMENTS

In the *MACRO* reply plane architecture discussed so far, MCs have to wait for 3 cycles to

inject successive flits even though it takes only 3 cycles to reach the destination once the flit is injected. Also, when an overlay circuit is established it is used only by a single MC, preventing the remaining MCs from returning data. To enhance the utilization and performance with overlay circuits, we propose two enhancements:

(i) Overlay pipelining: If the process of sending flits is pipelined, an MC can inject a flit every two cycles into the network. Figure 47 explains how the pipelining is done. In the figure, R_M is a router at the MC, R_H is a router where the flit is hinged, and R_D is a router at the destination core. A flit is first transmitted from R_M at cycle 1. R_H which is along the same X axis as R_M receives it in the same cycle. In cycle 2, R_H checks the Y-coordinates of the flit. In cycle 3, R_H sends the flit in the Y direction to R_D. R_M can use this cycle (cycle-3) to send a second flit in the X direction, instead of being idle. This method of pipelining the output traffic at R_M sends a flit into the NoC every two cycles, thus improving the overall network latency.

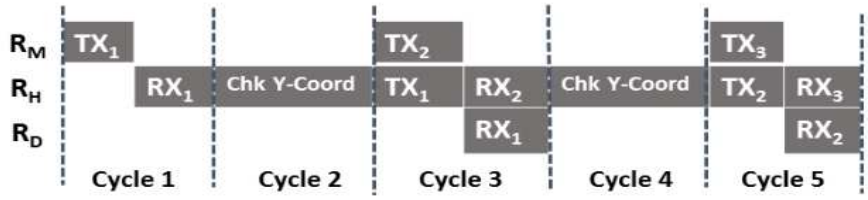


Figure 47 Pipelining the flow at an MC on an overlay circuit

(ii) Overlay multiplexing: While an MC uses its overlay circuit to send flits, some links may remain unused till the next window begins for the next overlay circuit. To increase the utilization of links, we can overlay two overlay circuits in the same time window, when the flits take XY and YX paths on the multiplexed overlay circuits. This can be achieved with a slight modification to the input selection unit and crossbar configuration sent by the GOM at the end of a time window. *This is also feasible only when there are no conflicts in port reservations between the multiplexed*

overlay configurations. There are no conflicts when the 2D mesh NoC has only one MC per row and per column, as shown in Figure 49 (a) and Figure 49 (b). Thus, the *MACRO* reply plane supports overlay multiplexing for two MCs.

5.3.3. MEMORY CONTROLLER (MC) DESIGN

We propose an enhanced MC design that prioritizes servicing responses to burst requests (or *burst packets*). *We define a request as a part of the burst when it is sent by a core within the 'burst duration' after the previous request.* A request is labelled as a burst request when it is sent within a burst duration after a previous request. So, when two or more requests are sent within a burst duration, they are labelled as burst packets by the network interface after being received from the core and before injecting into the NoC. We evaluated five different values of burst duration in our experiments (Section 5.4.1) and arrived on a burst duration value that gave the best results.

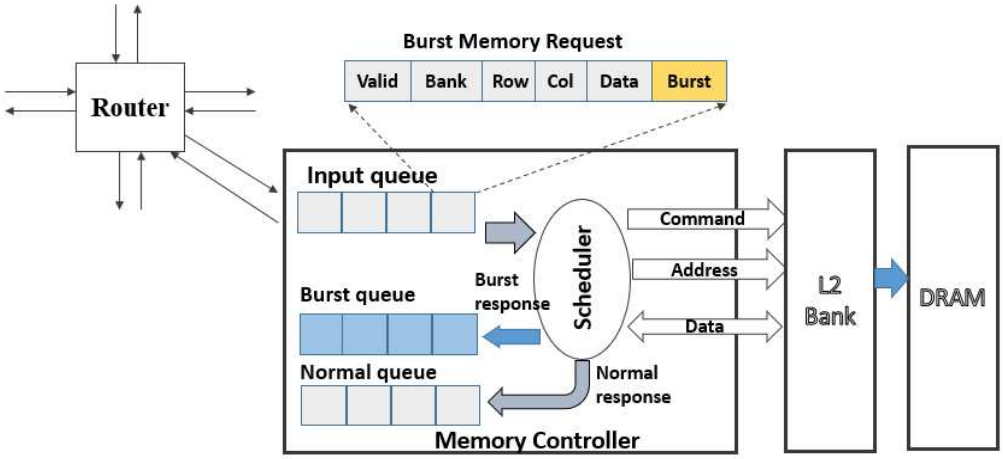


Figure 48 Memory controller (MC) with separate queues for burst packet responses and normal packet responses.

Figure 48 shows the MC architecture for achieving the proposed burst response prioritization. We have considered a baseline MC from **[139]** with an open page policy and first

ready- first come first serve (FR-FCFS) based scheduling policy for incoming requests. Our proposed MC is designed on top of the baseline MC from **[139]**. There is an abundance of literature on intelligent memory access scheduling in accelerators such as **[137]**. Our burst prioritization scheme (discussed next), if implemented on such MCs, works orthogonal to intelligent schedulers in reducing access latency of memory.

When a request that is labelled as a *burst request* reaches an MC, it adds a burst flag to the request that is sent to the scheduler. The scheduler then maps the data responses that arrive from L2 or DRAM to the corresponding burst request, called as *burst response*. The burst responses are then stored in a separate queue called *burst queue.* While servicing the reply packets, an MC first prioritizes sending the packets waiting in the *burst queue* to ensure that warps that issued the corresponding burst memory requests receive their responses first. Typically, warps are scheduled in such a way that the memory access latency is hidden, by assuming the access latency of each warp. If the memory access latency of a warp is unusually high, the wait time cannot be hidden by scheduling. By prioritizing the *burst packets*, some warps that issued burst requests do not wait unusually long for the data which makes warp scheduling easier and predictable. However, to avoid starvation in the normal output queue, the MC services one packet from the normal queue, for every 3 packets from the *burst queue*. If the *burst queue* is empty, it services the normal queue first. Generally, an MC services the output queues in a first-come-first-serve order. This leads to output queues getting fully occupied due to *burst packets*. In such cases, the MC does not accept new requests, creating a *back-pressure* on the request plane of the NoC. However, by having two separate queues and more rapidly servicing burst packets, our MC prevents back-pressure on the request plane and improves overall application performance.

5.4 EXPERIMENTAL RESULTS

5.4.1. EXPERIMENTAL SETUP

Table 3 GPGPU-Sim parameters used for evaluation

Parameters	Value
Shader Cores/MCs	132 / 12 (144-core), 56 / 8 (64-core), 12 / 4 (16-core)
Shader core pipeline	1536 Threads, warp size = 32
Shader registers	32768 per core
Constant / Texture Cache	8KB / 8KB per core
L1, L2 cache	16KB L1 per core, 128KB L2 per MC
NoC Topology	4×4 and 8×8 2D mesh, XY Routing
Channel width	128 bits
Baseline router	4-stage router, 5 VC/port, 4 buffers/VC

We target a 16-core, a 64-core and a 144-core GPGPU for our evaluation studies, to test the performance, energy dissipation and scalability of the proposed *RAPID* NoC in comparison to other state-of-the art approaches. We consider a NoC fabric that is clocked at 1GHz. Table 3 shows the platform configurations used for our evaluation.

We used GPGPU-Sim [139] to collect detailed application traces and simulated the network and memory traffic on a customized Noxim NoC simulator [140] that integrates our *RAPID* architecture model. We use trace driven simulation, as it is fairly accurate for architectural analysis [141], and as full system simulation is highly time consuming for design space exploration. We obtained traces for 10 CUDA benchmarks [4] each with a different number of kernels and levels of memory intensity: *Breadth First Search, ConvolutionsTexture, Discrete Cosine Transform 4x4, LIBOR, Monte Carlo, MUMmer GPU*, *Neural Networks, Ray trace, Storage GPU,* and *Fast Walsh*

Transform.

We compared our architecture with four prior works that also propose NoC architectures for GPGPUs: [10] [26] [125] and [35] (all are discussed in Section 5.1). The architecture discussed in [10] is called Direct all-to-all (DA2), while that from [26] is called XYYX. The architecture discussed in [125], called CE-NOC, is a cost-efficient and conflict-free NoC that minimizes the power cost for communication in GPGPUs. We also compare our work with our prior work MACRO [35] that only optimizes latency on the reply network. In contrast, *RAPID* incorporates a low power request plane NoC with a low-latency MACRO reply plane and an enhanced MC architecture. Lastly, we also compare against a baseline NoC that uses the configuration presented in Table 3 and Section 5.2.1 for both the request and reply planes. Figure 49 shows the MC placement we used in our 16-core, 64-core and 144-core platforms. We have assumed a plastic ball grid array (BGA) [142] packaging scheme for easier I/O pin connection for MCs that are placed in the interior of the chip.

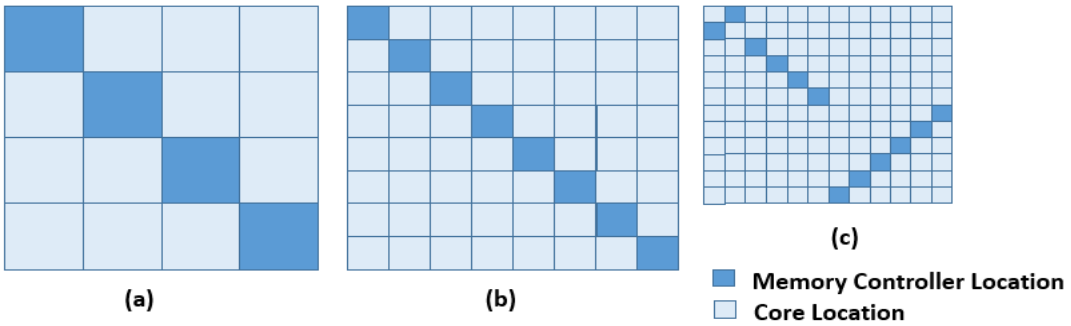


Figure 49 MC placement used in evaluation (darker cells indicate MC) (a) 4×4 mesh (b) 8×8 mesh (c) 12×12 mesh

For *RAPID*, we experimented with five different burst durations of 2, 4, 8, 10, and 13 cycles. The results across five different applications are shown in Figure 50. All five durations give almost similar values, with a burst duration of 8 cycles giving the minimum application execution time. With burst durations of greater than or less than 8 cycles the burst queue is either under-occupied

or over-occupied. The size of the reply queue at each MC is 132 packets, as used in [139], for all the comparison works. In *RAPID*, the reply queue is split between the burst queue and normal queue (66 packets each). We evaluated network latency, total application execution time, and total energy consumption for all NoC architectures. We also explored area overheads, and the impact of platform scaling across the six different NoC architectures. For MACRO and *RAPID*, in the reply network, we set epoch duration as 10,000 cycles. We set α, γ coefficients of the weight functions from equation $\boldsymbol{\xi(m) = \alpha.A(m) + \gamma.B(m)}$ (**28**) to 0.6, 0.4 and K = 1000.

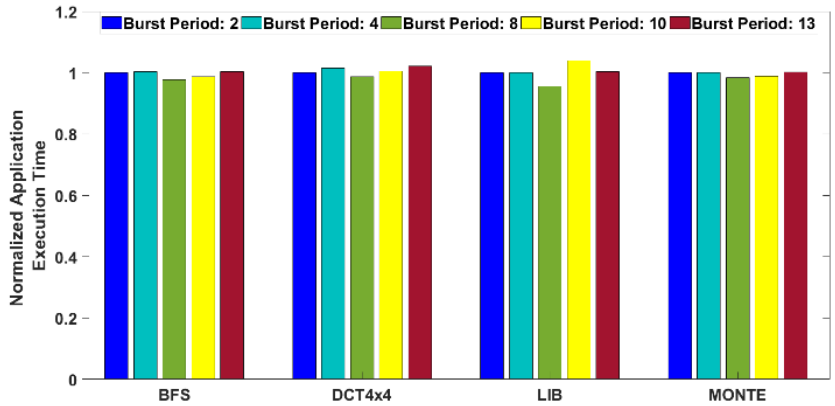


Figure 50 Sensitivity analysis of burst duration on a 64-core GPGPU with *RAPID* across different benchmark applications

5.4.2. NETWORK LATENCY

Figure 51 (a), (b), and (c) compare the network latency of *RAPID* with prior works for the 16-core, 64-core and 144-core GPGPUs, respectively. We compare the latencies of both request and reply networks. Request network latency includes the waiting time of the flit in the queues of the source routers due to back-pressure from MCs, and the time taken by a request flit to reach its corresponding MC from a core. Reply network latency includes waiting time of the flit at the output buffers of MCs, and time taken by the flit to reach to a core from an MC.

From Figure 51 (a), it is evident that there is an improvement in latency of up to 4× in the request and 10× in the reply networks with *RAPID* when compared to the baseline in a 16-core platform. MACRO gives up to 1.5× and 9× improvement in the request and reply network latencies compared to the baseline. The improvement in request network latency with *RAPID* is due to the utilization of low latency router in its request plane. The reply network improvement in MACRO and *RAPID* is due to the single cycle fast overlay circuit implementation with intelligent adaptation and allocation of time window durations by the GOM which results in less waiting time at the MCs and lower congestion. Along with that, *RAPID* shows further improvements because of the improved MC architecture which prioritizes *burst packets*. XYYX shows slight improvements in request network latency due to the virtual channel separation that allocates more bandwidth to the request packets. However, the reply network latency suffers due to the insufficient bandwidth for reply packets in the NoC and the *back-pressure* from MCs. Also, the XY and YX paths taken by request and reply packets in XYYX create traffic hotspots in the center of the mesh. Table 4 gives the channel widths of request and reply networks for all of the comparison works.

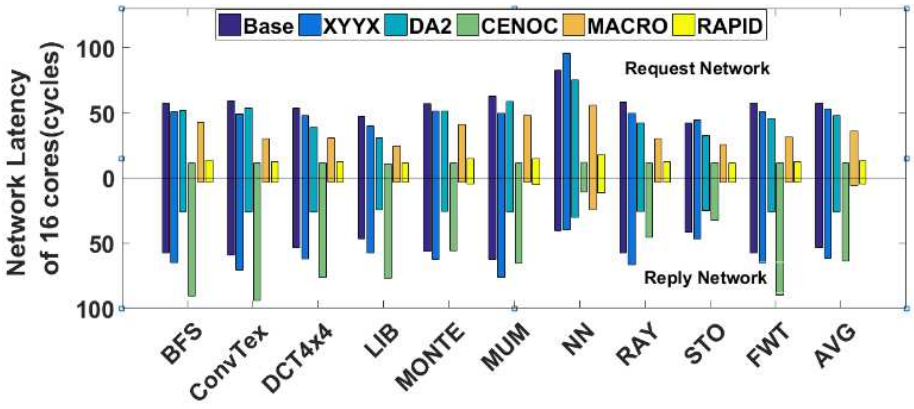
DA2 has a 64-bit channel width for the request network and 16-bit channel width for each slice in the reply network in a 16-core platform. DA2 shows a slight improvement compared to the baseline, of up to 10% in the request network and 50% improvement in the reply network. The improvements achieved because of the sliced reply network are less than the overlay-circuit based reply network in MACRO and *RAPID* due to a relatively slower DA2 reply router and serialization overhead. CENOC has 16-bit channel width for each of the slices in the request network of a 16-core platform, and 64-bit channel width for the reply network. It shows up to 4× improvement in the request network due to its conflict free request network topology. But the reply network latency is worse than the baseline. The Baseline NoC is not strictly partitioned, and hence the average

reply network bandwidth is higher than that of CENOC. This leads to worse reply network latency for CENOC.

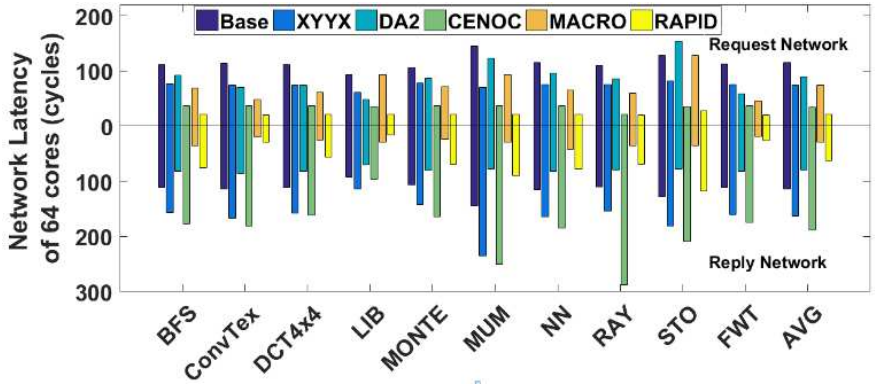
Table 4 NoC channel width (bits) for each plane

	16 cores		**64 cores**		**144 cores**	
	Req	**Rep**	**Req**	**Rep**	**Req**	**Rep**
Baseline	128		128		128	
XYYX [26] **(plane)**	64	64	64	64	64	64
DA2 [10] **(slices)**	64	16(4)	64	8(8)	32	8(12)
CENOC [125]**(slices)**	16(4)	64	8(8)	64	8(12)	32
MACRO [35] **(plane)**	64	64	64	64	64	64
***RAPID* (plane)**	64	64	64	64	64	64

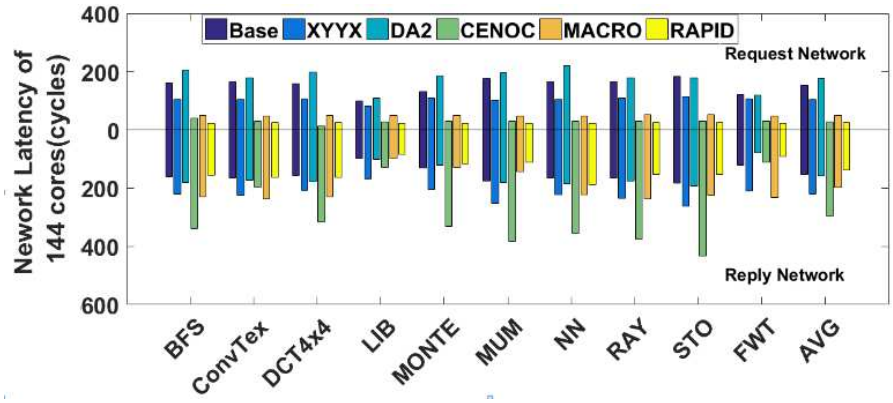
Figure 51 (b) shows the network latency in a 64-core GPGPU. *RAPID* shows up to 4× improvement in the request network and up to 60% improvement in the reply network compared to the baseline NoC. MACRO shows up to 70% improvement in request network and up to 3× improvement in the reply network. MACRO gives better reply network latency than *RAPID*. In a 64-core platform, *RAPID* transmits requests faster than MACRO which leads to more requests sent to DRAM and replies coming back to MCs. This increases the queuing time for packets at the output buffers of MCs. In a 16-core platform, *RAPID* MCs are able to handle reply packets without significant queuing time. In a 144-core platform, the request latency of MACRO and *RAPID* are almost similar, leading to similar queuing time at MCs (but a burst prioritized reply network in *RAPID* reduces its latency).



(a)



(b)



(c)

Figure 51 Comparison of network latency between Baseline, XYYX, DA2, CENOC, MACRO, and *RAPID* models for (a) 16-core and (b) 64-core and (c) 144-core GPGPU. The last set of bars (AVG) represents the average of all results.

In general, the combined (request and reply) network latency of *RAPID* is lower than that of MACRO across all platforms. XYYX gives up to 10% improvement in request network latency but up to 50% higher reply network latency compared to the baseline. The reason for poor network latency with XYYX is the same as in the 16-core scenario. DA2 gives diminished improvements (9% request network latency and 33% reply network latency) compared to its improvements for the 16-core scenario due to the increased number of MCs in the 64-core GPGPU that leads to an increased number of slices, which reduces the bandwidth of each slice. As a result, DA2 suffers with increased serialization overhead as the MC count increases. CENOC shows up to 2.5× improvement in the request network over the baseline but performs worse in the reply network that has half the bandwidth of the baseline.

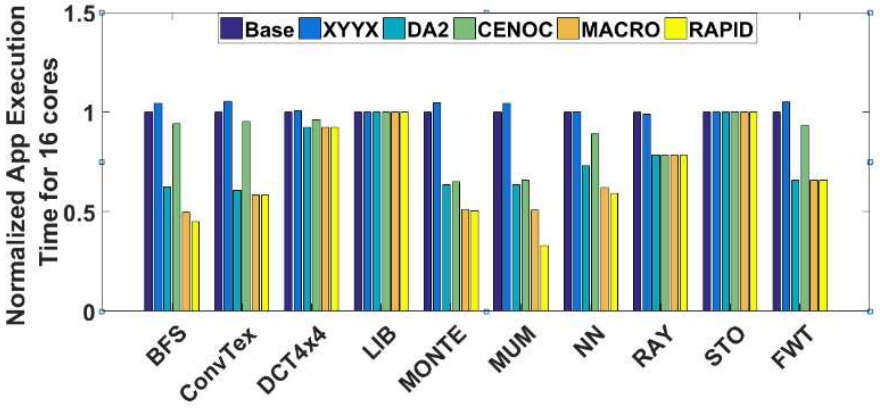
Figure 51 (c) shows the network latency of a 144-core GPGPU with 12 memory controllers. This result shows the scalability of our *RAPID* architecture compared to prior works. *RAPID* shows up to 4.5× improvement in request network and up to 10% improvement in the reply network. MACRO gives up to 2× improvement in the request network and up to 7% improvement in reply network. XYYX gives an improvement of 10-35% in the request network due to the separate conflict free virtual channels for request and reply packets. But, the reply network performs worse than the baseline by up to 25% for the same reasons as explained earlier. The NoC configuration used for DA2 for the 144-core case is different from the 16 and 64-core cases (Table 4). To have an even partition, we allocated 8-bit channel width to each of the 12 slices of the reply network. This reduces the request network bandwidth to one-fourth of the baseline bandwidth. Hence, the request network of DA2 is 20% slower than the baseline NoC in the request plane because of the reduced bandwidth. DA2 gets 1-9% improvement in the reply network compared to the baseline due to the increased bandwidth of 96 bits combining across all the slices in the reply network.

Hence, DA2 shows latency that is almost equal to *RAPID* in the reply network. However, *RAPID* has a faster request network and an enhanced MC that benefits application performance as discussed in the next section. An important trend to observe is that in NoC architectures that have a faster request network (CENOC, MACRO, and *RAPID*), the improvement in reply network latency diminishes with an increase in NoC size. This is because the request packets in these architectures arrive much faster to the MC compared to the baseline, DA2, and XYYX, and the corresponding response packets also arrive earlier and get queued at the MC to be delivered to the cores. As the network latency includes queuing time at the MC, this increases the reply network latency. However, the overall network latency is impacted significantly.

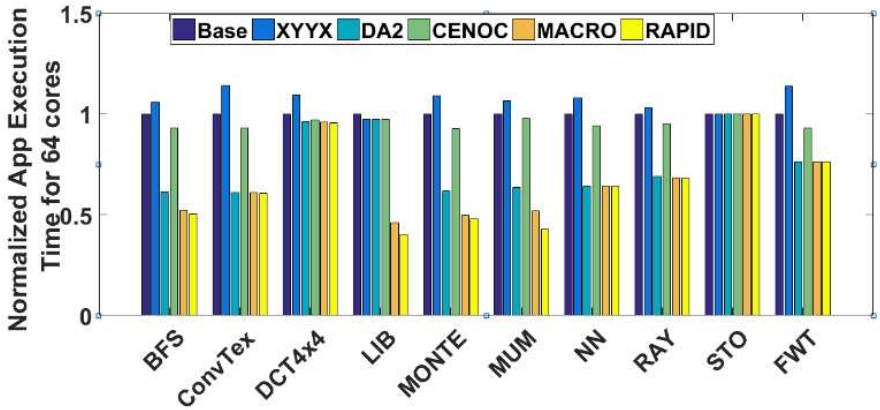
5.4.3. APPLICATION EXECUTION TIME

Figure 52 shows the normalized application execution times across different benchmark applications for 16, 64, and 144-core GPGPUs. In a 16-core platform, *RAPID* shows 10-63% improvement in application execution time compared to the baseline as shown in Figure 52 (a). MACRO shows 9-65.5% improvement compared to the baseline. This is primarily due to the low latency request network, and the fast overlay circuits in the reply plane for both architectures. *RAPID* shows better performance than MACRO due to the presence of the enhanced MC that services burst packets more rapidly. XYYX shows slightly worse performance than baseline due to the poor network latency as explained in Section 5.4.2. DA2 shows up to 34% and CENOC shows up to 10% improvement in the application performance due to their low latency conflict free routers in the request and reply networks respectively. However, their performance is limited

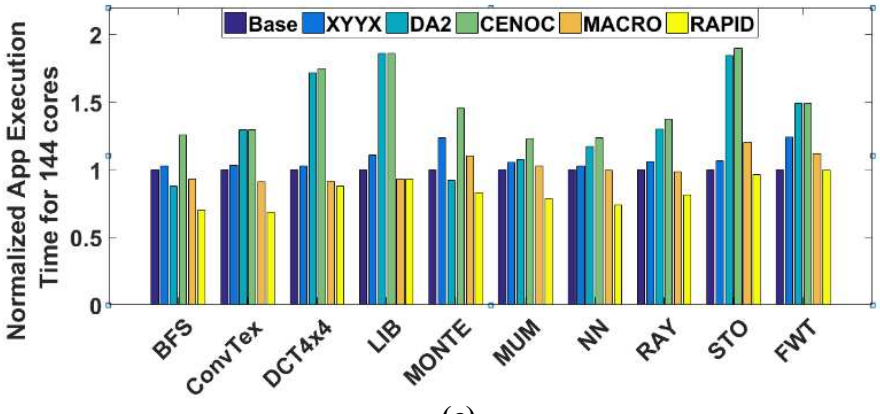
by the back-pressure caused by *burst packets*, which is seen in applications such as BFS, MUM, and NN.



(a)



(b)



(c)

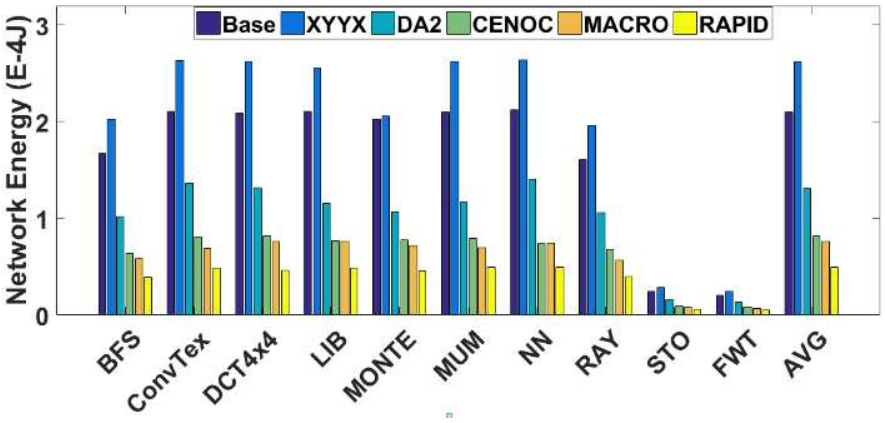
Figure 52 Comparison of normalized application execution times across the different comparison works for (a) 16-core (b) 64-core and (c) 144-core GPGPU platform

Some applications such as LIB, RAY and STO give negligible performance improvement for the 16-core platform as the memory bottleneck does not impact their computation time. These applications are either very small in terms of input data and execution time (RAY, STO) or they have relatively high number of kernels to be executed (LIB), which allows for masking the network latency by scheduling more warps. Figure 52 (b) shows the application execution times of different platforms for a 64-core GPGPU. *RAPID* and MACRO achieve 21-62% and 19-57% improvements in the application execution time compared to the baseline, respectively. XYYX performs worse than baseline in all cases due to the poor network latency observed in 64-core platform because of traffic hotspots, as discussed earlier. DA2 achieves up to 33% improvement in the application execution time due to improvement in the reply network latency as observed in Figure 51 (b). CENOC, gives around 10% application execution time due to its low latency NoC in the request network. However, reducing reply network latency is more critical to reduce the application execution time, which is not addressed effectively in CENOC.

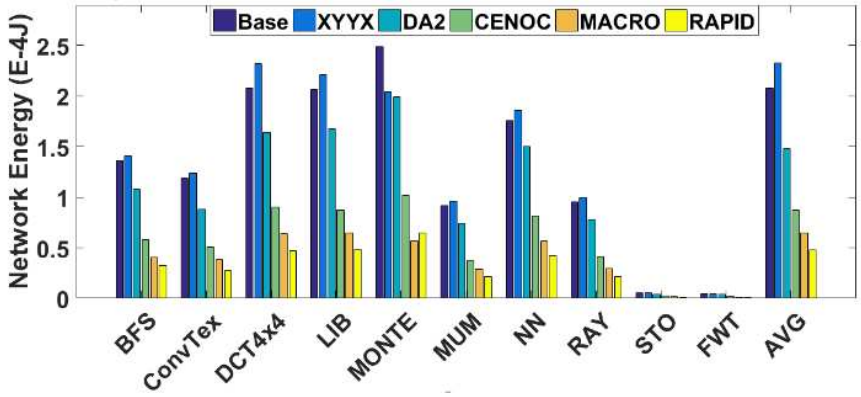
Figure 52 (c) shows the execution times for the 144-core GPGPU across different platforms. The figure shows that *RAPID* scales well compared to other works when the core count increases. XYYX, DA2, and CENOC all perform worse compared to the baseline due to their high request or reply network latencies. Also, when the core count is increased, more number of warps are scheduled in parallel creating higher number of burst packets. This is well addressed by *RAPID*. *RAPID* gives up to 19% improvement in application execution time compared to the baseline. MACRO also scales well with increased core count with an improvement of up to 7%, but less so than *RAPID* due to the lack of an enhanced MC and low latency request network in MACRO.

5.4.4. ENERGY CONSUMPTION

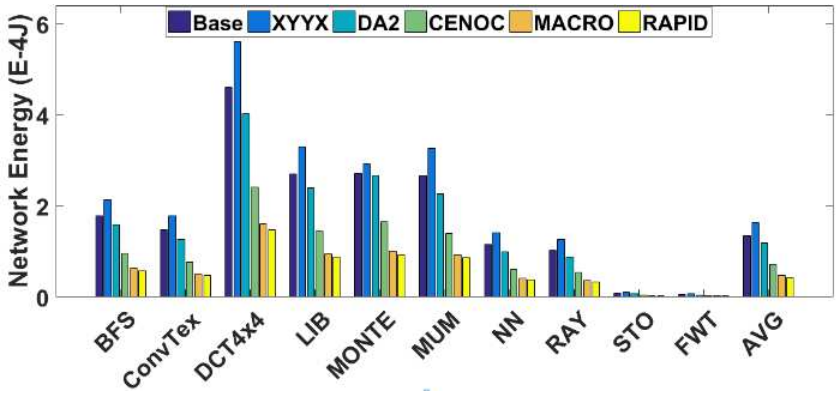
Figure 53 shows a comparison of energy consumed by different NoC architectures for the 16-core, 64-core and 144-core platforms across different applications. NoC power values at 22nm node are obtained using DSENT [143] simulations. As shown in Figure 53 (a)-(c), *RAPID* is the least energy consuming among all the compared architectures across all the platforms. *RAPID* achieves up to 4× and MACRO achieves up to 2.5× improvement in energy consumption across platforms. Speedup in application execution together with the low power consumption of the request plane routers in *RAPID* and the hinge router in the reply plane of both MACRO and *RAPID* enables lower energy consumption. XYYX consumes higher energy than the baseline because of the additional buffers used in its router architecture together with the longer duration of application execution compared to the baseline platform. DA2 achieves up to 43% reduction in energy consumption for the 16-core platform, 19% reduction in the 64-core platform, and 9% reduction in the 144-core platform compared to the baseline. The reason for DA2's lower energy consumption is that DA2 achieves speedup in application performance and its reply slices contain routers with only one virtual channel per input resulting in low power consumption. CENOC achieves up to 1.5× reduction in energy consumption for the 16-core platform, up to 67% reduction in the 64-core platform, and 42% reduction in the 144-core platforms respectively. CENOC uses a cost-efficient router in the request network that lowers energy consumption. However, the improvements of DA2 and CENOC are lesser than that of *RAPID* and MACRO. DA2 and CENOC reduces power dissipation in either the reply or the request networks. In contrast, *RAPID* reduces power dissipation in both the request and reply networks, by using buffer-less hinge routers in the reply network and low-power, low-latency routers in the request network.



(a)



(b)



(c)

Figure 53 Comparison of energy consumption across (a) 16-core, (b) 64-core, and (c) 144-core systems. The last set of bars (AVG) represents the average of all results.

5.4.5. ROUTER AREA OVERHEAD

We compared the combined area of routers of all the planes for different platforms, across different platform sizes. Figure 54 shows the percentage of chip area consumed by the NoC routers across the comparison works. The values of NoC router areas are obtained using the DSENT tool and gate-level analysis at the 22 nm technology node. *RAPID* observes an improvement of around 30.3% in 16-core platform, 51.9% in 64-core platform, and 54.2% in 144-core platform compared to that of the baseline due to the modified, trimmed-down router architecture in the request plane and low overhead hinge routers in the reply plane. Input buffers account for more than 50% of the router area in the baseline NoC. The routers used in both planes of *RAPID* have less number of buffers and hence take less area than the baseline. MACRO has slightly higher area (~10%) than the baseline NoC, as the request plane network is still essentially a baseline network with reduced channel width. Additional hardware required for the routing table, overlay management, input selection unit, and latching the incoming flits accounts for the overhead of around 6% for both MACRO and *RAPID*. In DA2 and CENOC, each tile has 5 routers. Each tile takes 53.4% more area than the baseline even though individual slice routers take less area than the baseline router. DA2 and CENOC consume similar area as they both employ sliced networks. CENOC has slightly higher area due to the presence of a token sharing network. Hence, DA2 and CENOC consume 34%, 45.7%, and 51.9% more area than the baseline. The router area of XYYX on the other hand consumes more area than that of the baseline due to the presence of additional buffers that support multiple virtual channels to ensure deadlock free XY and YX routing. Hence, XYYX has an area overhead of around 143% compared to that of the baseline across all platform sizes.

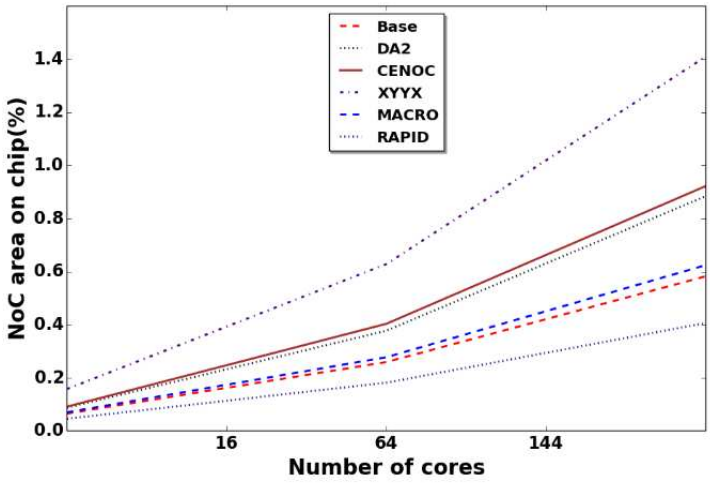


Figure 54 NoC area (as % of total chip area) across different comparison works for different platform sizes

5.5 CONCLUSION

In this chapter we introduced a novel NoC architecture called *RAPID* that is customized for many-to-few and few-to-many traffic patterns in GPGPUs. *RAPID* utilizes a low-power and low-latency router architecture in the request plane, and overlay circuits to deliver flits to their destinations within 3 cycles in the reply plane. We proposed customized hinge routers to accomplish the establishment of overlay circuits in the request plane, and a global overlay manager that monitors and allocates overlay circuits for memory controllers (MCs) to reduce latency for reply traffic to the shader cores. We also proposed an enhanced MC architecture that prioritizes sending burst packets to overcome bottlenecks. Experimental results on 16-core, 64-core, and 144-core platforms show an improvement of up to 4-10× in network latency, up to 67% in application execution time, up to 4× saving in energy, and around 50% improvement in area footprint compared to the baseline NoC architecture. Our experiments also show that *RAPID* outperforms several state-of-the-art architectures and is more scalable with increasing core counts.

6. APPROXIMATE NOC AND MEMORY CONTROLLER ARCHITECTURES FOR GPGPU ACCELERATORS

High interconnect bandwidth is crucial for achieving better performance in many-core GPGPU architectures that execute highly data parallel applications. This leads to scenarios with rapid arrival of an even larger volume of reply data from the DRAM, which creates a bottleneck at memory controllers (MCs) that send reply packets back to the requesting cores over the network-on-chip (NoC). Coping with such high volumes of data requires intelligent memory scheduling and innovative NoC architectures. To mitigate memory bottlenecks in GPGPUs, we first propose a novel approximate memory controller architecture (*AMC*) that reduces the DRAM latency. To further realize high throughput and low energy communication in GPGPUs, we propose a low power, approximate NoC architecture (*Dapper*) that increases the utilization of the available network bandwidth by using single cycle overlay circuits for the reply traffic between MCs and shader cores. Experimental results show that *Dapper* and *AMC* together increase NoC throughput by up to 21%; and reduce NoC latency by up to 45.5% and energy consumed by the NoC and MC by up to 38.3%, with minimal impact on output accuracy, compared to state-of-the-art approximate NoC/MC architectures.

For today's high-performance computing workloads, general purpose graphics processing units (GPGPUs) have become highly popular due to their support for massive thread and data level parallelism. However, parallel applications often generate large volumes of memory requests to memory controllers (MCs), resulting in a rapid influx of reply data from DRAM to be transmitted from MCs to SMs, which creates memory bottlenecks. Traditional MCs use first-ready first-come-first-serve (FR-FCFS) that is not designed to handle such requests that may have high row buffer

locality (RBL) and bank level parallelism (BLP) simultaneously. Also, traditional mesh based NoC architectures are not designed to handle the high volumes of reply traffic between MCs and cores. To support the high traffic rates of GPGPUs, NoC channel widths should be increased multifold compared to conventional NoCs, but this leads to a significant increase in GPGPU power dissipation. Figure 55 gives a breakdown of power consumed by various components of a GPGPU when executing data parallel CUDA applications from the CUDA SDK sample code suite. For memory intensive applications that generate high volumes of memory requests, the NoC and MC together dissipate up to 30% of the overall chip power. *Thus, there is a need for innovative NoC and MC architectures that can support the high volumes of data generated and consumed in GPGPUs, while dissipating low power (and energy), and without sacrificing application performance.*

Recent works [144] [145] have demonstrated the impact of a new paradigm called *approximate computing* that trades-off computation accuracy for savings in energy consumption. Many emerging applications in the domains of machine learning, image processing, and pattern recognition are today exploring approximate computing techniques to save energy and also improve application performance while tolerating a small range of output errors. *One of the main goals of this chapter is to exploit data approximation intelligently to increase the throughput of data movement between MCs and shader cores (SMs) to speed up application execution and also minimize the energy consumed during application execution on GPGPU platforms.*

In a typical many-core GPGPU platform, the NoC traffic consists of load/store (LD/ST) data with LD replies forming a majority of the traffic that causes MC bottlenecks [10]. Hardware designers have come up with high radix NoCs with intelligent routing schemes [26], or complex

warp scheduling schemes [121] to minimize the MC bottleneck. However, these techniques incur high power/area overheads.

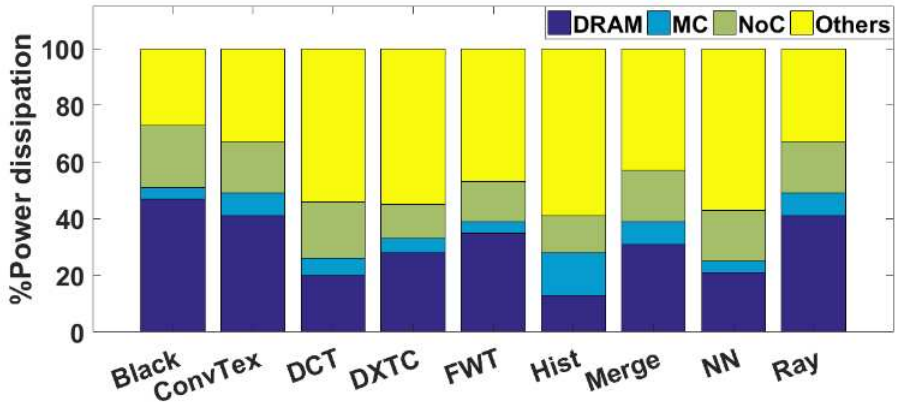


Figure 55 Breakdown of power dissipated by different components of a GPGPU when executing various parallel workloads

In this chapter, we conjecture that *several applications that use GPGPUs generate high volumes of data with redundant or similar values that can be approximated to reduce the number of packets transmitted from MCs to cores.* We leverage this observation and propose an approximate MC architecture and a high-speed circuit overlay based NoC architecture that together overcome the MC bottleneck issue and minimize energy consumption more aggressively than state-of-the-art techniques, with minimal application accuracy degradation. Our novel contributions are summarized as follows:

- We introduce a novel approximate memory controller architecture (*AMC)* that incorporates approximate data-aware memory scheduling in the request channel, to increase MC throughput by intelligently leveraging row buffer locality and bank level parallelism of DRAM requests; we extend the *AMC* with support for flagging read reply data arriving from DRAM for potential approximate transmission over the NoC;

- We introduce a data-aware approximate NoC architecture (*Dapper*) that utilizes asynchronous fast overlay circuits for transmitting both general and approximate data between MCs and cores in 3 cycles; we also design a novel NoC router architecture, and an arbitration mechanism that uses a global overlay manager (GOM) to share the overlay circuits between different MCs and cores;
- We conduct rigorous simulation-based experimentation of our proposed *Dapper* NoC and *AMC* architectures for various CUDA applications to compare the performance and energy consumption against the state-of-the art.

6.1 RELATED WORK

Several prior works have addressed the issue of NoC throughput and energy consumption for traditional many-core processors. In [146], the authors propose a small world NoC that utilizes machine learning to establish fast connections between cores that generate high volumes of data. In [132], the authors propose an application criticality-aware packet routing scheme that prioritizes memory requests of time-critical applications. In [147]- [148] complex-network based application scheduling and mapping mechanisms have been proposed to minimize inter-cluster communication in heterogeneous platforms. These works are orthogonal to the proposed approximate NoC and MC architectures. *Further, these techniques perform well in CPU based platforms under lighter traffic conditions, but they cannot be used to minimize the bottleneck caused at MCs in GPGPU platforms.*

In [149], [150] heterogeneous NoC architectures have been proposed to address many-to-few traffic patterns in CPU-GPU heterogeneous architectures, but, they have not addressed the bottleneck caused by few-to-many traffic between the last level caches (LLCs) near MCs and the cores. Further, [151] and [152]

discuss the benefits of mesh based NoC topologies such as scalability, simplicity and ease of implementation. Hence, in this work we customize the mesh based NoC topology for few-to-many traffic pattern in GPGPU processors. The memory bottleneck issue in many-core CPUs and GPUs has been addressed in a few prior works. In [153] an application-aware staged memory scheduling mechanism is proposed that creates batches of requests to maximize row buffer hits and bank level parallelism (BLP) in the DRAM, and to minimize the inter-application bandwidth interference. In [154], an MC that uses BLP-aware prefetching is proposed, to increase MC and DRAM throughput. But these works ignore the adverse impact of NoC latency on the memory bottleneck issue. *In our work, we address the memory bottleneck issue from a perspective of both intelligent memory scheduling and minimizing NoC latency*.

In [155] a high performance wireless NoC architecture with short ranged wired links and long-range wireless links is proposed to address NoC congestion due to multicast packets arising from cache coherence traffic. However, this work cannot be directly adapted to GPGPU traffic conditions where the reply path has much longer packets (64 B each) compared to cache coherence traffic. In [156] an encoding scheme along with a deadlock free corridor routing and adaptive flit dropping mechanisms are proposed to improve the throughput and latency of NoC under high multicast traffic conditions. In this work as well, the size of the packets is smaller with lower injection rates compared to GPGPU traffic between memory controllers and cores. This technique has the potential to give some improvement in GPGPU NoC throughput in the reply network with applications of lower memory intensity; however, it incurs high power and area overheads at MCs and NoC routers. *Unlike the above two works, DAPPER+AMC is more lightweight and is designed to improve the performance of GPGPUs specifically in the commonly observed many-to-few and few-to-many GPGPU traffic conditions.*

Approximate (or inexact) computing has been studied by several researchers to save energy by sacrificing the correctness of the output to an acceptable extent. A few works [157] [158] have demonstrated the use of compiler support to label and treat approximate variables differently from other variables and enabling inexact computing on those variables at the hardware level. Other works have used approximate computing at the circuit level [159] [160] to trade-off accuracy of the logic circuits for shorter critical paths and low operating voltages that save power and area footprint. A few other works have introduced approximate last level caches [161] [162]. In these works, the authors reduce the amount of data stored in the caches, and increase the cache hit rate by associating tags of multiple approximately similar blocks to the same data line. In [163], the authors use approximate computing with spatial and temporal locality to increase the benefit of instruction reuse in GPGPUs. *None of these techniques can be applied for resolving the problems arising in MCs and NoC due to the heavy influx of memory requests that cause MC bottlenecks in GPGPUs.*

In [164], the authors have proposed a NoC centric approach to minimize the latency caused by congestion in many-core CPUs by introducing approximate compression and decompression of packets at network interfaces, to reduce the number of flits entering the NoC. This work utilizes a dictionary-based compression/decompression technique that consumes high energy and leads to performance overheads when the network traffic is high. However, this work ignores the challenges with MC scheduling to minimize the memory access latency, in GPGPUs. *In contrast to these works, in this article, we provide a holistic solution to the memory bottleneck problem in GPGPUs, with a multi-pronged solution that uses approximate computing design principles with smart resource adaptation at both the MC and NoC architecture levels.*

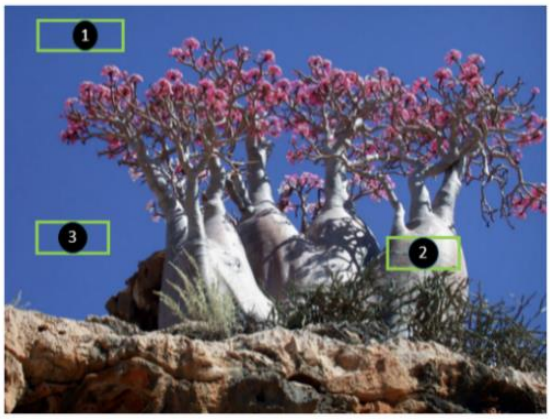
6.2 BACKGROUND AND MOTIVATION

6.2.1. BASELINE CONFIGURATION

We consider a heterogeneous accelerator-based system with an x86 CPU and a grid of shader-cores of GPGPUs that have private L1 caches (instruction and data) to support data parallel multithreaded application execution. A shader core consists of parallel integrated pipelines with a common instruction fetch unit that executes a single instruction on multiple data (SIMD) simultaneously. Each integrated pipeline has an integer arithmetic logic unit and a floating-point unit. A shader core also has several load store units that fetch data from a private L1 cache or from the main memory. A GPGPU based accelerator has an L2 cache that is shared across the shader cores. Each shader core also has a texture cache and a scratchpad memory. All shader cores and MCs are connected to an on-chip interconnection network.

Typically, there is little to no direct communication between the shader cores on the chip, as there is no data shared between shader cores that are executing thread blocks. Hence, the L2 is used to cache the contents of private memory of each shader core. The communication between CPU and GPU cores takes place through main memory. The shader cores send read/write requests (via LD/ST instructions) to MCs over the NoC. A memory reply takes several cycles based on the location of and availability of data in the L2 or DRAM. The baseline NoC architecture between the shader cores and the MCs has a channel width of 128-bits, twice the size of 64 bit NoC channels in traditional CPU based platforms, and consists of 4-stage routers (stage 1: buffer write; stage 2: route computation, stage 3: virtual-channel/switch allocation; stage 4: switch/link traversal). There are 5 virtual channels (VCs) per input port and 4 flit buffers for each VC. Flits are routed along the XY path from source to destination. In this chapter we focus on optimizing the MC as well as

the interconnect between shader-cores and MCs. Henceforth, the term *cores* implies *shader-cores* for the remainder this article.



(a)

	R	G	B	R	G	B
1	91	125	193	91	122	193
2	123	155	183	131	145	183
3	90	128	191	92	123	191

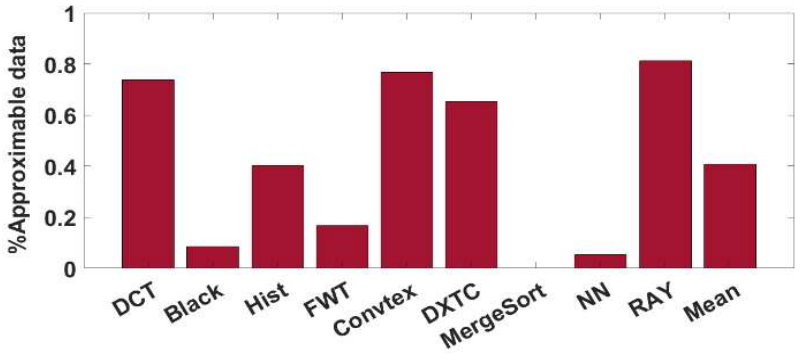
(b)

Figure 56 (a) Example image showing similar data values in pixels; (b) RGB values of the marked locations as stored in texture memory

6.2.2. DATA VALUE APPROXIMABILITY

Several types of data parallel applications are typically executed on GPGPU platforms. Many of them belong to the domain of image processing, signal processing, pattern recognition, and machine learning. There also exist other scientific and financial applications that use GPGPUs that operate on large input data sets. The data used for executing image and signal processing applications in many cases is highly approximable. For example, as shown in Figure 56 (a), the areas in boxes contain pixels that are very similar to their adjacent pixels. The RGB values of two

pixels for each box are shown in Figure 56 (b). These values (for each box) are quite similar and it is not energy efficient to save and transmit cache lines containing similar RGB values separately from DRAM to the cores. Instead, if cache lines with same (or similar) data can be identified at the MCs, we can avoid their repeated transmissions to the cores, to save energy. *Dapper* and *AMC* are designed to realize this idea.



(a)

```
void LoadBmpAsGray(char *FileName, int Stride,
                   ROI ImSize, byte *Img)
{
    BMPFileHeader FileHeader;
    BMPInfoHeader InfoHeader;
    FILE *fh;
    fh = fopen(FileName, "rb");

    fread(&FileHeader, sizeof(BMPFileHeader), 1, fh);
    fread(&InfoHeader, sizeof(BMPInfoHeader), 1, fh);

    for (int i=ImSize.height-1; i>=0; i--)
    {
        for (int j=0; j<ImSize.width; j++)
        {
            @approx int r=0;
            @approx int g=0;
            @approx int b=0;
            fread(&b, 1, 1, fh);
            fread(&g, 1, 1, fh);
            fread(&r, 1, 1, fh);
            @approx int val = ( 313524*r + 615514*g +
                                119537*b + 524288) >> 20 ;
            Img[i*Stride+j] = (byte)clamp_0_255(val);
        }
    }
```

(b)

Figure 57 (a) Normalized percentage of approximable data transmitted from main memory to cores in different CUDA applications; (b) Example of marking approximable variables using EnerJ [157]

The next logical question one may ask is: how approximable are typical applications that run on GPGPUs? Figure 57 (a) shows the percentage of approximable data in different parallel CUDA applications, while still generating acceptable results. Applications such as *Discrete Cosine*

Transformation, *Convolution Texture, and Raytrace* can have up to 78% of approximable input data that can be exploited to mitigate the memory bottleneck issue and save NoC energy. By making use of the programming paradigm proposed in [157] , it is possible to specify approximable variables using the *@approx* keyword as highlighted in green in Figure 57 (b). Such approximable variables will have a distinct set of instructions for load and store that are used when compiling the C++ code to machine code. These instructions can be used by cores and network interfaces to identify approximable data and accept inexact values for them from main memory.

6.2.3. MEMORY SCHEDULING IN GPGPUS

DRAM is organized into a hierarchy of modules as follows: each MC has a dedicated channel to access DRAM *DIMMs* that each consist of two ranks. Each rank typically has 8 or 16 DRAM chips depending on the data bus width (8 or 16 bytes). Each chip has up to 32 banks that activate rows of data to be accessed, which are then buffered in each bank. An MC sends requests to access specific rows and columns within those rows across banks to read/write data. Once accessed, a row is either kept open (to enable fast accesses for future requests to the same row) or closed (written back to its respective bank before a new request arrives). Memory accesses that hit the rows that are already buffered are said to have row buffer locality (RBL). The performance of DRAM is enhanced by scheduling requests intelligently to increase the utilization of rows by maximizing RBL. If the subsequent accesses are not in the same row, they have to wait until the next row is buffered.

To minimize access latency, DRAMs today also support bank level parallelism (BLP) where accesses to multiple banks are scheduled in parallel. Figure 58 (a) shows the average RBL of memory requests waiting in the MC input queue at any instance, and Figure 58 (b) shows the

average BLP in memory requests waiting in the MC input queue, for different CUDA applications. Most of the applications have high spatial locality leading to high RBL of around 3. CUDA applications also have a high BLP of above 3 in some applications due to their inherent data parallelism. However, prior work [153] shows that RBL and BLP is not easy to harness at the same time, making designers choose among the two options to increase DRAM performance. In contrast, in *AMC* we propose a novel credit-based request scheduling mechanism that leverages both RBL and BLP based on the runtime application characteristics, to increase NoC throughput and leverage the spatial locality of the approximable DRAM requests.

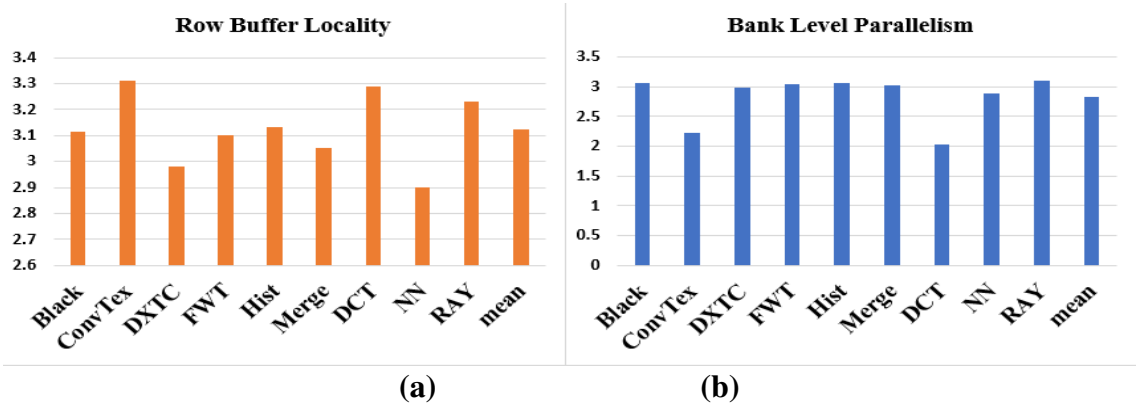


Figure 58 (a) Row buffer locality in memory requests and (b) Bank level parallelism for memory requests across CUDA applications

6.2.4. OVERLAY CIRCUITS FOR LOW LATENCY TRAVERSAL

As discussed earlier, in GPGPUs, read reply data is the main source of MC bottlenecks. Minimizing read reply latency is crucial for application performance. Also, read reply data comprises most of the traffic from MCs to cores in a few-to-many traffic pattern. To ensure that MCs do not get clogged by a heavy influx of reply data, it is intuitive to design a NoC with all-to-all connections. However, power and wire routability constraints in modern chips limit such architectures. Hence, researchers have come up with smart solutions in [10], [26] where they

propose intelligent NoC routing schemes in tandem with MC placement to create conflict free paths in NoCs for packets to traverse from MCs to cores. However, the complex router design in such architectures adds to NoC power overheads.

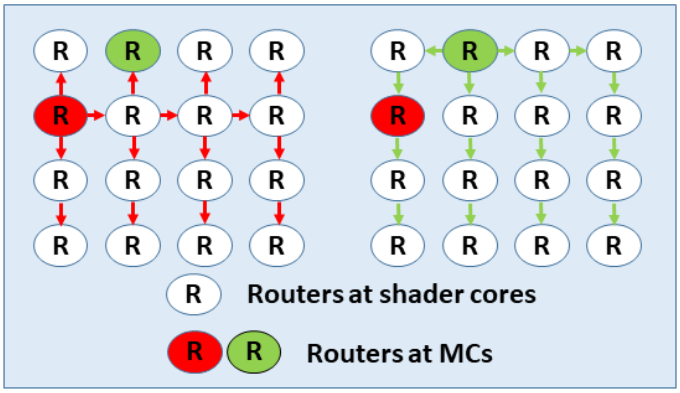


Figure 59 A 4x4 NoC showing overlay circuits for two MC's read reply paths, in a few-to-many traffic scenario

In this chapter, we make use of the few-to-many traffic pattern of the reply packets and utilize an overlay circuit topology that forms dedicated circuits for each MC. An overlay circuit connects an MC to all the cores, forming return paths for read reply packets. Figure 59 shows how the circuits are established from two MCs (represented as colored circles) to cores. These overlay circuits are established from each MC to all the cores, on a 2D mesh-based NoC topology. An overlay circuit formed between an MC to the cores stays for a fixed time window during which it transmits the reply packets of that MC, before switching to the next overlay circuit (for another MC). On overlay circuits (shown with red and green colored arrows in the figure) each MC transmits flits of packets waiting in its queues that reach their destination cores in 3 cycles (or less) using asynchronous links and repeaters that bypass one or more NoC routers. Flits traverse in X and Y directions in one cycle each, stopping only at the turns. Hence, flits traversing over overlay circuits do not need switch arbitration and route computation at every hop. This leads to a low

energy consumption NoC that establishes high-throughput, congestion-free paths for read reply packets. More details about how overlay circuits are established and used for data traversal are discussed in the following sections.

6.3 OVERVIEW OF AMC AND DAPPER

In our approximate memory controller (*AMC*), we enhance GPGPU MC throughput by exploiting the inherent parallelism in request packets, and the approximability of the read reply data. In *Dapper* we reduce the latency of on-chip transfers by optimizing for the few-to-many pattern of the approximable read reply data packets between *AMC*s and cores.

The *AMC* consists of two key components: (1) Approximate data-aware memory scheduling in the request channel that intelligently switches between BLP or RBL with a credit-based throttling mechanism, and (2) Data approximation support in the reply channel of the memory controller, that identifies approximable data waiting in its output queues and marks them for *coalescence.* The overlay circuits of *Dapper* then broadcast the flits of *coalesced* packets to their destinations over fast overlay circuits that are established for each *AMC*. We define *coalescence* as a group of DRAM reply data whose read reply data from *AMC*s are approximable and within an *error_threshold*. These replies are then *coalesced* into a single packet, to reduce the number of packets transmitted into the NoC for delivery to multiple destinations.

Dapper employs a 2D-mesh based NoC topology to create overlay circuits between cores and *AMC*s. The network is divided into two planes (request and reply) each with 64-bit channel width, to avoid protocol deadlock. The NoC routes packets with an XY turn-based routing scheme to avoid routing deadlock. The request packets are transmitted on a request plane and the reply packets are transmitted via fast overlay circuits on the reply plane. A majority of the reply plane

traffic consists of read reply packets. Hence, *Dapper performs packet coalescing using approximation only on read reply data that is received from DRAM*. The reply plane of the NoC contains a novel router architecture that enables establishing and tearing down of the overlay circuits. More details of these modules are discussed in the following sub-sections.

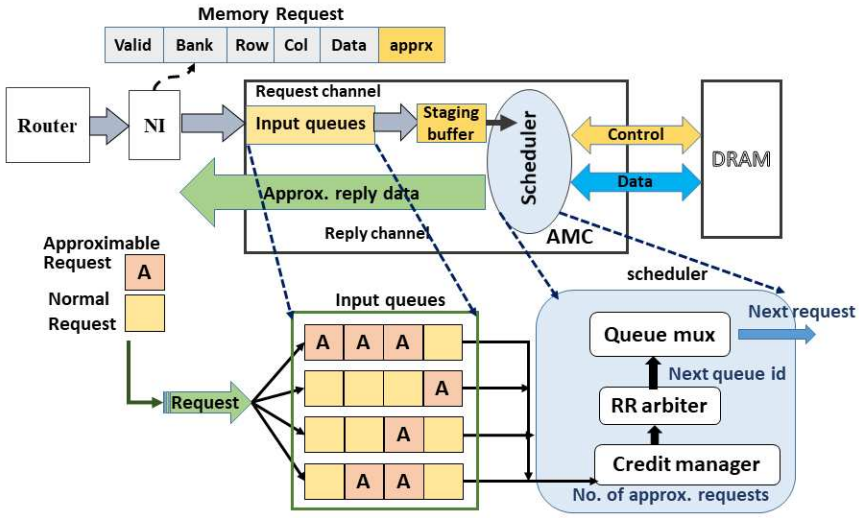


Figure 60 Overview of approximate data-aware memory scheduling in approximate memory controller (*AMC*)

6.3.1. APPROXIMATE MEMORY CONTROLLER (AMC)

*AMC*s are connected to the NoC through a network interface (NI) and a router. An *AMC* receives memory requests from cores and creates commands to send to DRAM. Before sending a request to an *AMC*, a core sets an *approximable flag* for the memory request, if the load operation is on an approximable variable. The NI that connects the core to the router generates flits from the memory request packet and adds an *approximable flag* to the header flit. The receiver NI connected to the *AMC* generates a memory request from these flits and sets an *approximable flag* for the memory request before sending to the memory subsystem, as shown in Figure 60.

6.3.1.1. APPROXIMATE DATA-AWARE MEMORY REQUEST SCHEDULING

On the request channel, the *AMC* has a novel scheduler to intelligently leverage RBL and BLP of the arriving approximable requests. An important observation is that in some applications

such as *DCT, ConvTex,* and *RAY*, most of the approximable data is located in nearby memory addresses leading to a higher row buffer locality, as shown in Figure 58 (a). For these applications, it is preferable to send those requests in a *burst* in consecutive clock cycles so that the replies arrive in consecutive DRAM cycles, and can be stored in adjacent output queues in the *AMC,* which facilitates higher *packet coalescing* in the output queue. However, it is also preferred to support bank level parallelism to reduce the waiting time of the high volumes of parallel memory requests in GPGPU applications. To leverage both RBL and BLP, we propose a credit-based scheme that prioritizes RBL burst commands when there is a high influx of approximable requests into the *AMC*'s input queue, and BLP otherwise. To facilitate this, we divide the input queue of an *AMC* into *n* parallel queues as shown in Figure 60.

The incoming requests are stored in one of the *n* parallel queues based on the *bank number* of the memory request. An input queue is allocated to each memory request by performing a modulo operation on its bank number with the total number of input queues (*queue id = bank number % total queues*). For each input queue, the scheduler assigns a fixed and equal number of credits at boot up time. For each incoming request processed from a queue, a credit is deducted. The queues are processed only when their available credits are > 0. The total number of credits is equal to the total available input command queue depth at the interface between the *AMC* and DRAM. The *AMC* receives a signal from the DRAM after a command is processed internally to replenish the credits back to the queues. Figure 60 shows an overview of the approximate data-aware scheduler in *AMC*. The input queues are processed by the credit manager that decides if RBL or BLP should be selected for the next DRAM command and controls the round-robin (RR) arbiter to choose the next queue. The RR arbiter then signals a queue mux that selects a queue to fetch the memory request from and forward the next command to DRAM.

Algorithm 12 : Approximate data-aware scheduling

Inputs: *num_approx_reqs (for each queue), check_depth*

```
1:  next_queue_id ← 0
2:  burst_count ← 0
3:  for each cycle do
4:    if next_queue_id→num_approx_reqs < Threshold then  //BLP block
5:      if next_queue_id→credits > 0 then
6:        queue_mux_sel ← next_queue_id
7:        next_queue_id→credits - -
8:      end if
9:       next_queue_id → next_queue_id + 1 mod total_num_queues // RR
10:    else   // RBL block
11:      if burst_count < check_depth then
12:       queue_mux_sel ← next_queue_id
13:       burst_count ++
14:     else
15:       next_queue_id→credits = credits - check_depth
16:       burst_count ← 0
17:       next_queue_id → next_queue_id + 1 mod total_num_queues //RR
18:     end if
19:   end if
20:   increment queue credits by 1 in a RR scheme if notified by DRAM
21: end for
```

Algorithm 12 describes the functionality of the scheduler. The algorithm takes number of approximable requests waiting in each queue (*num_approx_reqs*), and *check_depth* as inputs. At every cycle, the scheduler selects a queue and checks if the total number of waiting approximable requests is below a threshold (line 4). It then checks if that queue has at least one credit to process a request (line 5). If both conditions are true, then the input queue does not contain sufficient requests to hit the same row buffer, thus the credit manager selects the BLP scheme for scheduling to leverage parallelism in DRAM commands. In the BLP scheme, an input queue is serviced in a round robin manner in each cycle to schedule the next command (lines 5-9). The RR arbiter sends the queue id as input to the queue mux to select a single request from this queue to create the next DRAM command (line 6), and a credit is deducted (line 7). The RR arbiter then points to the next queue in a round robin manner for the credit manager to process in the next cycle (line 8). If the

queue does not have at least one credit, the credit manager invariably selects the next input queue in an RR manner for processing in the next cycle (line 9). However, if the queue has more approximable requests than the threshold (line 10), the credit manager selects the RBL scheme for the next *check_depth* DRAM commands (lines 11-17). In RBL, the DRAM commands are sent to hit the same DRAM row buffer to get the maximum approximability in the read reply data. Hence, in each cycle the credit manager processes the same input queue till it reaches *check_depth* to make sure that all the subsequent commands are sent in a burst (lines 11-13). The credit manager notifies the RR arbiter to select the same queue until the burst mode is completed. As a result, the queue mux selects requests from the same queue for *check_depth* number of times. The queue's credits are also deducted by *check_depth* (line 15). This can lead to negative credits for the queue. The queue is not eligible to send any requests until it replenishes its credits back to at least 1. After sending commands in a bursty manner from the same queue, the command manager sends an input to the RR arbiter to select the next queue for processing in the next cycle (line 17). The credits are replenished in a round robin manner when DRAM notifies the *AMC* that it has completed processing a command (line 20). The credit replenishment stops if a queue receives its quota of credits that are assigned at boot up time. The *AMC* has an internal staging buffer as shown in figure 6 that stores the requests before processing them to DRAM commands. When this queue is full, the scheduler stops processing any memory requests until it is cleared.

6.3.1.2. DATA APPROXIMATION IN REPLY CHANNEL

Whenever a reply is received from the DRAM, it is matched with the corresponding request and saved in the output queue. In the reply channel, the *AMC* is equipped with a *data approximator* which parses through each of the read reply data units waiting at the output queue of the *AMC*, finds the approximable data among the waiting data, and checks if any data can be coalesced before

sending to the NI for delivery to the destination cores. The granularity of approximable data waiting in the output queue is an L1 cache line because a read reply from *AMC* contains a cache line with the address that led to a cache miss. *The data approximator tries to coalesce cache lines waiting in queue entries only if each of the variables contained in the cache lines is approximable and contains similar data values in all the cache line entries.* Figure 61 shows an overview of the *data approximator*. If the output queue has some approximable reply data, the *data approximator coalesces* them into a single read reply data unit based on their data values to minimize the number of replies injected by the NI into the NoC.

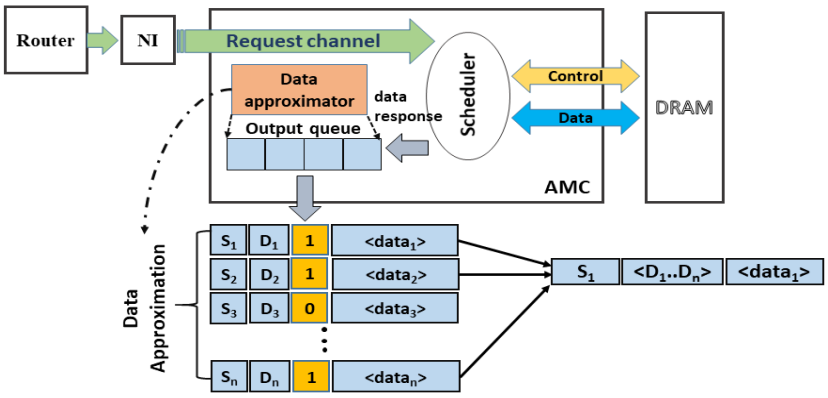


Figure 61 Overview of approximation done within the approximate memory controller (*AMC*) in the reply channel

Algorithm 13 describes the steps involved in the data approximation phase in the reply channel. The algorithm takes two parameters (*error_threshold*, *check_depth*) as inputs. The *check_depth* is the same as *check_depth* in *Algorithm 12*, because it signifies the number of requests processed in the input queue for sending an approximable *burst* of commands to DRAM. The data approximator iterates over each entry in the output queue and checks if the data is approximable (line 5). If the data is not approximable, it sends the data unaltered to the NI for delivery (line 6). If the data is approximable, the algorithm iterates through subsequent data entries to find other approximable data (lines 8-9).

Algorithm 13: Data Approximator

```
Inputs: error_threshold, check_depth
1:  while Output_queue.size( ) > 0 do
2:    ð ← 1
3:    dest_list ← Ø
4:    reply_data ← Output_queue. front( )
5:    if reply_data.approximable == 0 then
6:      send_to_NI(reply_data)
7:    else
8:      while ð < check_depth do
9:        nxt_data ← Output_queue.get(ð )
10:       if nxt_data.approximable == 1 then  // coalesce the data
11:         if (reply_data – nxt_data)/reply_data < error_threshold then
12:           dest_list. add(nxt_data.dest)
13:           Output_queue.erase(ð)
14:         end if
15:       end if
16:       ð++
17:     end
18:     reply_data.add_dest(dest_list)
19:   end if
20:   send_to_NI(reply_data)
21:   Output_queue.pop( )
22: end while
```

If an approximable data is found, the difference between the first approximable data and the found data is calculated (lines 10-11). If the difference is within the *error_threshold*, the found data entry is erased from the output queue, and its destination address is saved (lines 12-13). This iteration process stops once it reaches the *check_depth,* which also signifies the maximum number of packets that are coalesced if their data values are within an *error_threshold.* The list of destination addresses along with the original address is appended to the first reply (line 18). The approximated data and the list of addresses are then sent to the NI (line 20) to generate a packet that is broadcast to the list of destination nodes on the overlay circuits as explained in the following sections. The data types considered for approximation include both integer and floating-point types. For floating point data, only the mantissa is approximated for a faster computation.

6.3.1.3. OVERHEADS

The power and area overheads involved in approximate data-aware scheduling in the request plane, and data approximation in the reply plane are minimal compared the power consumed by the computing shader-cores. *Algorithm 12* needs *n* additional counters to keep track of the credits, where *n* = total number of input queues, an additional comparator, and queue mux. The credit manager operation takes place in 1 cycle, which is pipelined with the other stages of the *AMC* to reduce the overall latency. *Algorithm 13* takes *check_depth* number of cycles for execution, and requires logic for division, a comparator, and registers to store the parameters *check_depth* and *error_threshold*. The NI connected to an *AMC* is often fully occupied at run-time, resulting in a wait time for data at *AMC*s in most cases, before they are sent to the NI. This wait time helps to mask the overhead involved in data approximation. A packet is only sent to the NI after it passes the data approximation stage. We consider all of these performance and power overheads when modeling the *AMC* for our experiments (Section 6.4). Note also that the parameters used in *AMC,* such as *check_depth* and *error_threshold,* can be updated by GPGPU firmware according to the needs of the application.

6.3.2. DATA AWARE APPROXIMATE NOC (DAPPER)

The request plane of *Dapper* is a conventional 2D mesh based NoC with XY routing as mentioned in Section 6.2.1. In the reply plane, *Dapper* establishes dedicated overlay circuits from *AMC*s to cores. Each *AMC*'s overlay circuit is a predefined mapping of input and output ports in the reply plane NoC routers, during which flits traverse from source (*AMC*) to destinations (cores) in 3 cycles or less. An overlay circuit assigned to an *AMC* lasts for a time duration determined at

run-time. To establish overlay circuits, the reply plane NoC is equipped with *(i)* a global overlay controller that decides the time duration for which an overlay circuit is established, and *(ii)* modified routers called *bypass routers* through which flits traverse in X or Y axes in a single cycle, stopping only at a turn.

6.3.2.1. GLOBAL OVERLAY CONTROLLER (GOC)

The primary function of the global overlay controller (GOC) is to determine and assign time durations for overlay circuits, for each *AMC*. The execution time is divided into epochs, and each epoch is further divided into time windows which are computed at the beginning of every epoch by the GOC. A time window for an *AMC* is determined at run-time based on the number of reply packets waiting at the *AMC* output queues, and the reply arrival rate at the *AMC* from the previous epoch. Each *AMC* sends the stats collected during an epoch to the GOC at the end of that epoch, using the overlay circuits. The GOC uses this received information to compute a weight function as shown in equation (30):

$$\boldsymbol{\xi(m) = \alpha.A(m) + \gamma.B(m)} \qquad (30)$$

where *A(m)* is the reply arrival rate and *B(m)* is the average queue occupancy at the m^{th} *AMC* in the previous epoch. α and γ are coefficients of the weight function. The GOC compares ξ's of each *AMC* and computes time window durations $T_1, T_2, T_3 \ldots, T_m$ for the next epoch as:

$$\boldsymbol{Ti = K * \xi(i)/[\xi(1) + \xi(2) + \xi(3) + \cdots + \xi(m)]} \qquad (31)$$

where T_i is the time window of the i^{th} *AMC* overlay and $\xi(i)$ is its weight function. The ratio of the weight functions is then multiplied by a constant *K* which is equal to the periodicity of the

time windows in an epoch. The time windows repeat periodically for *E/K* iterations in an epoch, where *E* is the epoch interval duration and *K* is the periodicity of the time window set. By having the time windows repeat and overlay circuits switch multiple times in an epoch, *AMC*s send flits in multiple bursts across an epoch. The time window durations are broadcast by the GOC at the beginning of an epoch, which is saved by the reply plane routers in dedicated registers, and then used for the setup and tear down of circuits. At the end of each epoch, the arrival rate *A(m)* and buffer occupancy *B(m)* of each AMC are sent to GOC using the corresponding overlay circuits. This requires 2 additional cycles at the end of each epoch (which is 10000 cycles in duration).

6.3.2.2. BYPASS ROUTER ARCHITECTURE

The reply plane NoC is made up of bypass routers that support flits passing through them without stopping at each hop for arbitration or route computation. Figure 62 shows an overview of a bypass router architecture. A bypass router has asynchronous bypass links connecting output ports to input ports and latches via a *crossbar*. Each input port is connected to a dedicated *latch* to save the flit that is coming over the X axis. The router also saves time windows of each overlay circuit received from the GOC in a local *overlay controller* as shown in Figure 62. The crossbar is configured at the beginning of each time window and reset at the end of the window to establish a different overlay configuration. The crossbar configuration of a router is based on its location on the 2D NoC and is selected by the *selection unit* based on the current overlay circuit. The output ports of a bypass router are connected to either input ports or input latches to enable flit transmission. In the first cycle, a flit traverses the bypass routers along the X axis through asynchronous links and gets latched at the input latches. In the second cycle, when the output ports

are connected to input latches, the flit traverses along the bypass routers in the Y direction and gets latched again at the input latches. In the third cycle, the flit is sent from an input latch to the local (core) port. Thus, flit transmission can take 1 to 3 cycles. Two key components of a bypass router are the *overlay controller* and *selection unit*. The *overlay controller* keeps track of the time windows and calls the *selection unit* to dynamically configure the crossbar to establish the overlay circuits. The *crossbar* dynamically makes connections between input latches to output ports, and turns flits from X axis to Y axis, and from Y axis to local out ports as explained in the subsequent sections.

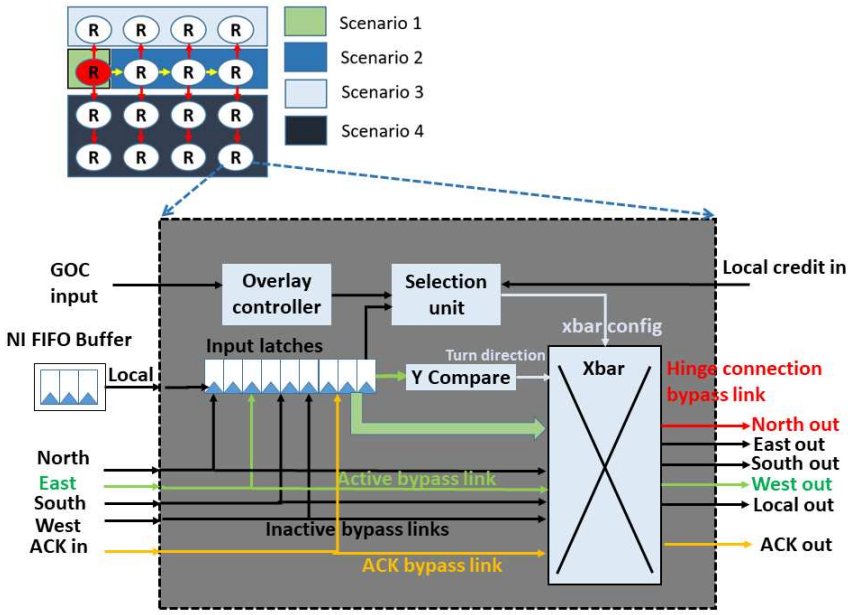


Figure 62 Overview of bypass router architecture

Algorithm 14 gives an overview of the overlay controller operation. The controller gets a list of time window durations and epoch duration as inputs from the GOC. It sends the list of time windows to the *selection unit* as input (line 3). At every cycle, it increments a local counter to keep track of the epoch duration (lines 5-6). At the end of an epoch duration, updated time window

values are received from the GOC which are sent to the *selection unit,* and the local counter is reset (lines 8-10).

Algorithm 14: Overlay controller operation

```
Inputs: {time windows}, epoch_duration
1:  e_counter ← 0; // reset counter value
2:  {T1...Tk} ← {time windows} // initialize time window durations
3:  selection unit ({T1...Tk})
4:  for every cycle do
5:    if e _counter < epoch_duration then
6:      e _counter ++;
7:    else if e _counter == epoch_duration then
8:      {T1...Tk} ← read_values(GOC_input) // save time windows
9:      selection unit({T1...Tk})
10:     e _counter ← 0; // reset counters
11:   end if
12: end for
```

Algorithm 15 gives an overview of the selection unit operation. The inputs to the selection unit are the time window durations $\{T_1 \ldots T_k\}$ for k overlay circuits (corresponding to the k *AMC*s). In *Algorithm 15*, {*N, E, W, S, L}* means North, East, West, South, and local port. Also, *La* is the latch. The selection unit possesses knowledge of the 2D coordinates of the *AMC* that utilizes the overlay circuit on the NoC along with its own coordinates. With the given inputs, the selection unit creates connections for an overlay circuit between input ports, input latches and output ports. At the beginning of each time window, the selection unit compares the coordinates of the current router and the *AMC* for which the overlay is being established.

Based on the location of the router, there are 4 possible scenarios for each *AMC* as shown in Figure 62. *Scenario 1:* If the router and *AMC* are at the same NoC coordinate, the local input port is connected to the east and west output ports and local latch (line 6). *Scenario 2:* If the router and the *AMC* are along the same X axis, the east and west inputs are connected to the west and east

output ports, and to their corresponding latches (line 8). *Scenario 3:* If the router and *AMC* are not along the same X axis and the router's Y coordinate is greater than the *AMC'*s Y coordinate, the north input is connected to the south output and north latch (line 10). *Scenario 4:* If the router and *AMC* are not along the same X axis and the router's Y coordinate is less than the *AMC'*s Y coordinate, the south input is connected to the north output and south latch (line 12). At the end of a time window, the selection unit implements connections for the overlay circuit of the next *AMC* and its corresponding time duration. This entire operation takes 2 cycles at the beginning of each epoch. Figure 62 shows an example of such a configuration where green lines represent the configuration made by the selection unit to connect the east input to east latch and west output.

Algorithm 15: Selection unit operation

Inputs: $\{T_1 \ldots T_k\}$, $\{MC_1 \ldots MC_k\}$

1: $t \leftarrow 0$, $i \leftarrow 1$
2: **for** *each cycle* **do**
3: **if** $t == T_i$ **then**
4: $i \leftarrow (i+1)$ **mod** k
5: **if** $local.(X,Y) == MC_i.(X,Y)$ **then** *// scenario 1*
6: $L_{in} \rightarrow E_{out}, W_{out}, La(L)$
7: **else if** $local.(Y) == MC_i.(Y)$ **then** *// scenario 2*
8: $E_{in} \rightarrow W_{out}$, $La(E)$ **and** $W_{in} \rightarrow E_{out}$, $La(W)$
9: **else if** $local.(Y) > MC_i.(Y)$ **then**
10: $N_{in} \rightarrow S_{out}$, $La(N)$ *// scenario 3*
11: **else if** $local.(Y) < MC_i.(Y)$ **then** *// scenario 4*
12: $S_{in} \rightarrow N_{out}$, $La(S)$
13: **end if**
14: **else**
15: t ++
16: **end if**
17: **end for**

<u>*Routing*</u>*:* For conflict free routing using bypass routers, each row in the 2D NoC should only have one *AMC*. A packet that is approximable might have more than one destination based on the *check_depth* parameter of the *AMC*. To accommodate more than one destination, the header flit of

the packet has more than one destination fields, and a destination length field. When a packet is ready to be transmitted at the NI of an MC, it is sent to the NI interface buffer based on the availability of the buffer space. At each cycle, the router transmits flits via the asynchronous links along the X direction and saves them at the corresponding input latches of each bypass router (shown via the green line in Figure 62). The Y-compare unit (Figure 62) then compares the destination Y coordinates of the flits with its own coordinates and sends a signal to the crossbar to establish *hinge connections* between input latches, and the north/ south/local output ports based on the Y coordinates of the current router at which the flit is latched and the destination router. Once the *hinge connections* between latches and output ports are established, the flits traverse in the Y direction (shown via the red line in Figure 62) and reach the destination router where they are sent to the local output port. When the tail flit passes the bypass router, the *hinge connections* are reset to tear down the connections between latches and output ports.

Flow control: Since *Dapper* has lower latency than traditional NoCs, it fills up the receiving cores' queues much more rapidly than a traditional, 4-stage hop-by-hop, router. Hence, it is crucial that a robust flow control mechanism is integrated to ensure that the receiving queues of cores do not fill up quickly and start creating backpressure. Unlike the wormhole switching scheme where the intermediate routers at each hop save flits for re-transmission, in *Dapper* the flits are only saved at the receiving core. As a result, if there is any data loss on the asynchronous links, they may be undetected at the receiving core. Hence, there is also a need for a mechanism to ensure lossless transmission of packets traversing on overlay circuits.

To meet these goals, we use an acknowledgement-based flow control mechanism where the receiving cores send ACK signals back to the *AMC* when the packet is completely received. We establish ACK circuit using dedicated links of 7 bits width to send ACK signal back to AMCs.

Bypass routers establish an ACK circuit using ACK links for each overlay circuit. These ACK circuits are not shared with data packets. A core does not send an ACK signal when its input queues are already full or when there is data loss in the packet. To facilitate fast transmission of the ACK signal from the cores to an *AMC,* each bypass router has an ACK in and ACK out port (Figure 62), which are connected using asynchronous links, switch, and a *selection-unit* similar to the fast overlay circuit to relay the ACK message back to the *AMC* from the cores in 1-3 cycles. The *AMC*, upon receiving the ACK signals from the destination cores releases its output queue for the read reply data coming from DRAM. If the *AMC* does not receive an ACK signal, it services the next waiting packet in the output queue out-of-order. Based on the request type (read/write) the addresses of the missed request and the subsequent request are compared before the next packet is selected for transmission into the NoC to avoid read-after-write (RAW) or write-after-read (WAR) errors. Further, to avoid starvation of long waiting packets, the output queues in the *AMC* are designed as cyclic buffers so that all the waiting packets are processed in a round robin manner. Thus, this flow control mechanism ensures that packets at the head of the queue at the *AMC* do not block the subsequent packets that can use the available overlay circuit. To facilitate data loss detection, we assume the presence of a simple *XOR*-based parity bit error detection mechanism.

6.3.2.3. OVERHEADS

The overheads involved with implementing the overlay controller, selection unit, and Y-compare are minimal. The overlay controller uses a counter, and a register to store the time window values received from the GOC, while the selection unit uses a counter, a cyclic register, and five 3-bit comparator circuits. The bypass router also has an additional Y-comparator, and logic to establish *hinge connections*. All of these components together take up a very small fraction of the

power and area of a router. The bypass routers also do not have input buffers to save incoming flits, VCA, SA, and route computation logic. Hence, a bypass router consumes up to 50% less power compared to a traditional router. However, we increase the output buffer capacity of *AMC* by 50% as it takes at least 6 cycles for the complete read-reply transaction. All of these performance and power overheads are considered in our experimental analysis, presented next.

6.4 EXPERIMENTS

6.4.1. EXPERIMENTAL SETUP

We target a baseline 16-core GPGPU based accelerator to test the performance, energy consumption, network latency, and output error of *Dapper+AMC* compared to the state-of-the-art. We also test the scalability of our proposed architectures on a 64-core GPGPU. Table 5 lists the platform configurations. We used GPGPU-Sim [139] to collect detailed application traces and simulated the network and memory traffic on a customized Noxim NoC simulator [140] that integrates our *Dapper+AMC* architecture model. We obtained traces for 9 CUDA-based applications [4]. Table 6 gives an overview of each application and the approximable regions of their working data sets.

Table 5 GPGPU-Sim Parameters

Parameters	Value
Shader Cores/AMCs	12 / 4 (16-core), 56 / 8 (64-core)
Shader core pipeline	1536 threads/core, warp size = 32
Shader registers	32768 per core
Constant / Texture Cache	8KB / 8KB per core
L1, L2 cache	16KB L1 per core, 128KB L2 per MC
NoC Topology	4×4 (16-core), 8×8 (64-core), XY Routing
Channel width	128 bits
Base case router architecture	4-stage router, 5 VC/port, 4 buffers/VC

Table 6 Approximable cuda benchmark applications

Application	Domain	Approximable data
Black Scholes	Stock price evolution predictor	Stock price, CDF data
Convolution Texture	Image texture denoising filter using convolution	512 × 512 image
Discrete Cosine Transform 4 × 4	Image compression technique used in JPEG codec	256 × 256 image
DXTC	Image texture compression technique	256 × 256 image
Fast Warshal Transform	Faster version of discrete FFT	1024 × 1024 matrix
Histogram	Image color segregation into bins	512 × 512 image
Merge Sort	Highly parallel sorting algorithm	1MB input data
Neural Network	An advanced machine learning technique	Training and testing data sets
Raytrace	Image 3D rendering algorithm	512 × 512 image

As mentioned earlier, we use the programming paradigm proposed in EnerJ [157] to specify the variables in these applications that are potentially approximable. We set the epoch duration in *Dapper+AMC* as 10,000 cycles. We performed experimental sensitivity analysis for parameter values and accordingly set the values of α, γ coefficients of the weight function from equation (30) to 0.6, 0.4, and K = 1000 in equation (31) as they give the best NoC throughput for the applications we considered, in both 16 and 64 core platforms. We have also considered that the *Threshold* (line 4) in Algorithm 12 is equal to half of the *check_depth* parameter value because having a threshold larger than *check_depth*/2 leads to poor approximability as the request scheduler prioritizes BLP over RBL even when half of the requests are approximable. Conversely, a smaller threshold might favor RBL over BLP even when the ratio of packets that could be coalesced is lower which

adversely affects the DRAM utilization. The power, performance, and area values for our NoC architecture, modified MCs, and cores at the 22nm node are obtained using the open source tools DSENT [143] and GPUWATTCH [165], and gate-level analysis.

We compare our proposed *Dapper+AMC* architecture with several alternatives, as shown in Table 7. The baseline NoC has a channel width of 128-bits which is the size of the L1 cache line, to provide high throughput to MCs. We also compare with the prior work that utilizes dictionary-based approximate compression and decompression at each network interface, called *Approx-NoC* [164]. *Approx-NoC* uses dictionary-based inexact compression using approximation at the sending network interface to reduce the number of flits that are injected into the NoC and save the overall energy consumed. We additionally compare our work with the memory-aware-circuit-overlay-NoC, *MACRO* [35]. *MACRO* has a fast overlay circuit network in the reply plane from MCs to cores similar to that of the current work but lacks packet broadcast capability, as well as fast broadcast ACK system for fault tolerance, and the data approximator in its MC. The Baseline NoC, *Approx-NoC*, and *MACRO* have first row first come first serve (FR-FCFS) memory scheduling scheme in the request channel of their MCs. To analyze the drawbacks of MC scheduling without NoC latency optimization, we compare our work with staged memory scheduling (*SMS*) [153] that proposes an application aware bank level parallelism to minimize the wait time of packets in the MC input queue. However, [153] does not have a latency optimized NoC like *Dapper+AMC*.

Table 7 summarizes the NoC and MC configuration of all five comparison works. The size of the *AMC* output buffer is set to accommodate 66 data packets, as used in GPGPU-Sim, for all the comparison works. We also consider 4 parallel input queues in the *AMC* as the average bank level parallelism of the requests is around 3.5 as shown in Figure 58 (b).

Table 7 Configuration of comparison works

	MC configuration	NoC configuration
Baseline	FR-FCFS scheduler in request channel. FCFS in reply channel.	1-plane, 128-bit wide channel.
Baseline + SMS [153]	Staged memory scheduling in request channel. FCFS in reply channel.	1-plane, 128-bit wide channel.
Approx-NoC [164]	FR-FCFS scheduler in request channel. FCFS in reply channel.	1-plane, 128-bit wide channel. Approx -NI
MACRO [35]	FR-FCFS scheduler in request channel. FCFS in reply channel.	2 planes, 64 bits each. Overlay circuits in reply plane.
Dapper+AMC	Approximate data-aware scheduler in request channel. Data approximator in reply channel.	2-plane, 64 bits wide. Fast overlay NoC in reply plane with packet broadcast, and fast ACK mechanism

6.4.2. SENSITIVITY ANALYSIS

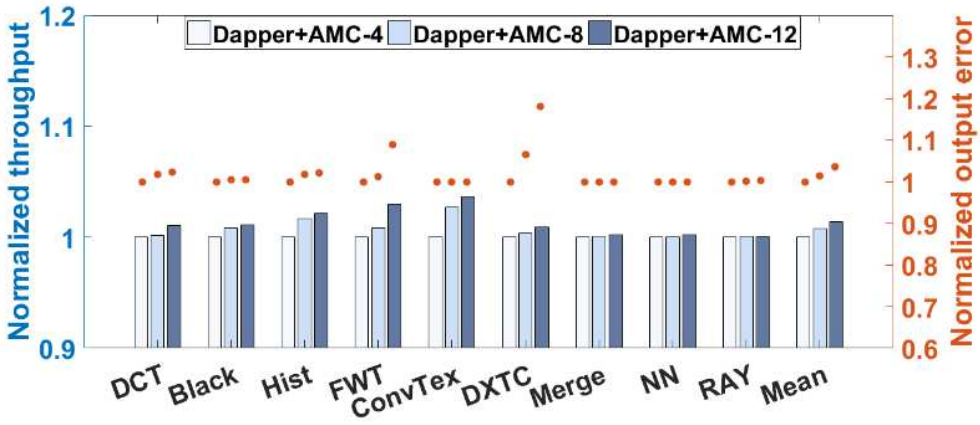
We first conduct experiments to determine the best values of buffer *check_depth* and *error_threshold* parameters used in our data approximation stage in the AMCs in *Algorithm 12* and *Algorithm 13*. We compare *Dapper+AMC* with different *check_depths* and *error_thresholds* to analyze the tradeoffs between NoC throughput improvements and the output error observed at the end of the application execution. Output error is computed using equations (32) and (33):

$$\boldsymbol{e = (V - V')/V} \qquad (32)$$

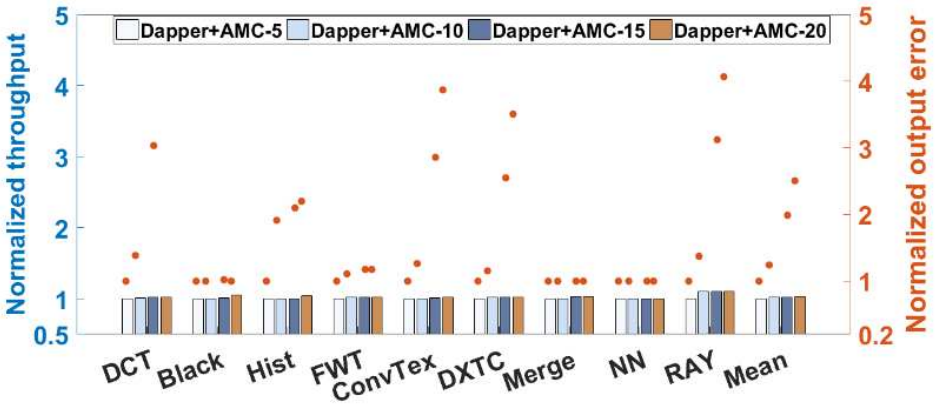
$$\boldsymbol{error = \frac{1}{M} * \sum_{i=0}^{M} |e_i|} \qquad (33)$$

where for each point in the output that comprises of images or matrices, *V* and *V'* are the actual and inexact points and *M* is the total number of output points. Equation (33) gives the value of output error obtained at the end of the simulation using approximation.

Figure 63 (a) shows the plots of normalized NoC throughput and normalized output error observed with *Dapper+AMC* across different values of output buffer *check_depths* (4, 8, 12). The bars represent normalized average NoC throughput observed, and dots represent the normalized output error across different benchmark applications. *Check_depth* is defined to be the maximum number of input queue entries that are processed by the *approximate-data aware* scheduler of *AMC* in the RBL scheme. *Check_depth* is also defined as the maximum number of reply data units that can be *coalesced* in the reply channel of *AMC*. In this manner, the approximation knob in the request and the reply channels are controlled by a single parameter. All the results are normalized to the result for the *check_depth* = 4 configuration of *Dapper+AMC*. From Figure 63(a), on average, a *check_depth* of 12 gives up to 3.5% higher throughput in *Dapper+AMC* with minimum increase in output error compared to *check_depths* of 4 and 8. *Dapper+AMC* leverages the *check_depth* parameter for opportunistic burst of approximable RBL requests that increases the number of *coalescable* packets. The benefits of increased *check_depth* can be seen better in applications that are both memory intensive and contain higher ratio of approximable input data (e.g., *ConvTex*, *Hist*). *Dapper+AMC* also shows throughput improvement at higher *check_depths* in applications with lower ratio of approximable data (*BlackScholes*, *FWT, MergeSort*) due to its intelligent memory scheduling.



(a)



(b)

Figure 63 Comparison of average NoC throughput and output error observed with Dapper+AMC across (a) check_depths, and (b) error_thresholds

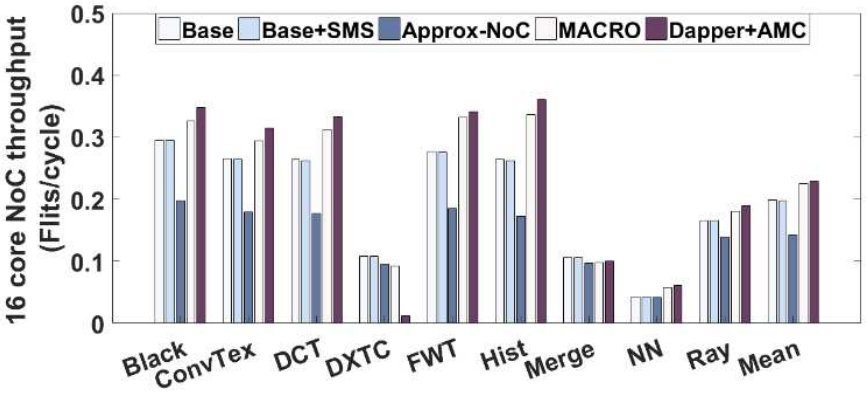
For the applications *DCT, DXTC, RAY* and *NN*, NoC throughput does not increase rapidly with higher *check_depth*, and the output error also remains constant across different *check_depths*. This is because these applications are either compute intensive or execute in a manner that does not fully congest the NoC. Hence, for the rest of our experimental analysis, we use *check_depth* = 12. If the *AMC* has less than 12 buffers filled at any point, it uses the maximum buffer depth available. Also, *check_depth* > 12 leads to a higher output error in applications such as *FWT* and

DXTC along with additional processing time latency in *AMC* for both request scheduling and data coalescing which adds to the *AMC* bottleneck.

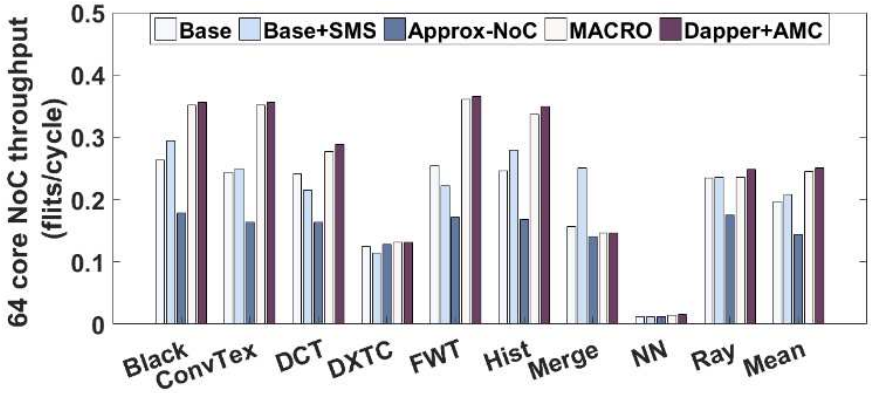
Figure 63 (b) shows the plots of normalized average throughput and output error values observed *Dapper+AMC* across different *error_thresholds* (5%, 10%, 15%, 20%). All results are normalized to the result for the 5% *error_threshold* configuration. The applications that are compute intensive and highly approximable, such as *DCT*, *DXTC*, *NN,* and *RAY*, show a slight increase in NoC throughput at higher *error_thresholds*. However, for *DCT, and RAY* the increase in output error is high at higher *error_thresholds* due to the high volume of approximable variables present in the memory reply data.

For memory intensive approximable applications such as *ConvTex* and *Hist*, NoC throughput increases by around 10% if the *error_threshold* is increased from 10 to 15%. For memory intensive applications that do not have highly approximable data such as *BlackScholes, FWT,* and *MergeSort,* the NoC throughput remains the same even with increase in *error_threshold*. On an average an *error_threshold* of 10% leads to ~15% increase in output error and ~9% increase in throughput compared to an *error_threshold* of 5%. For *error_thresholds* of 15% and 20% the output error observed is 2.3× and 2.7× higher compared to *error_threshold* of 5%. This eventually leads to poor application output quality.

We have also observed that on average, an *error_threshold* of 5% gives no performance gains compared to an error-free execution. For this reason, for the rest of our experimental analysis we use the *error_threshold* of 10% with *Dapper+AMC* when comparing with other architectures as it gives an acceptable application output error with better throughput than *error_threshold* of 5%.



(a)



(b)

Figure 64 NoC throughput analysis of *Dapper+AMC* with Baseline NoC, Baseline NoC with SMS memory scheduling [153], Approx-NoC [164], MACRO NoC [35] for (a) 16 core and (b) 64 core GPGPU accelerator, across various CUDA applications.

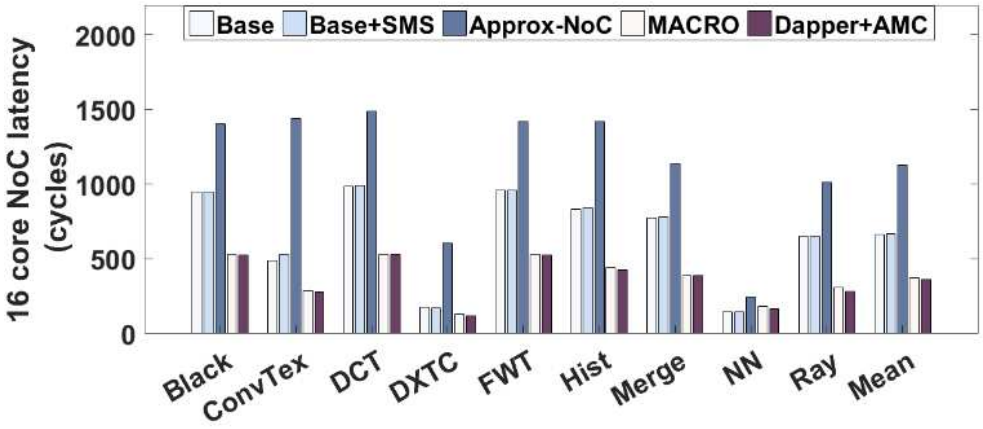
6.4.3. NOC THROUGHPUT ANALYSIS

Figure 64 (a) shows the average NoC throughput observed across different NoC and MC architectures in a 16-core accelerator. On average *Dapper+AMC* shows 21% improvement compared to baseline, and 21.5% improvement compared to *base+SMS*. Although *base+SMS* tries to maximize the number of RBL and BLP hits by forming memory request batches, it does not dynamically modify the RBL and BLP intensity of the commands like our scheduling scheme does. Secondly, it does not have a fast overlay NoC. *Dapper+AMC* outperforms *Approx-NoC* that

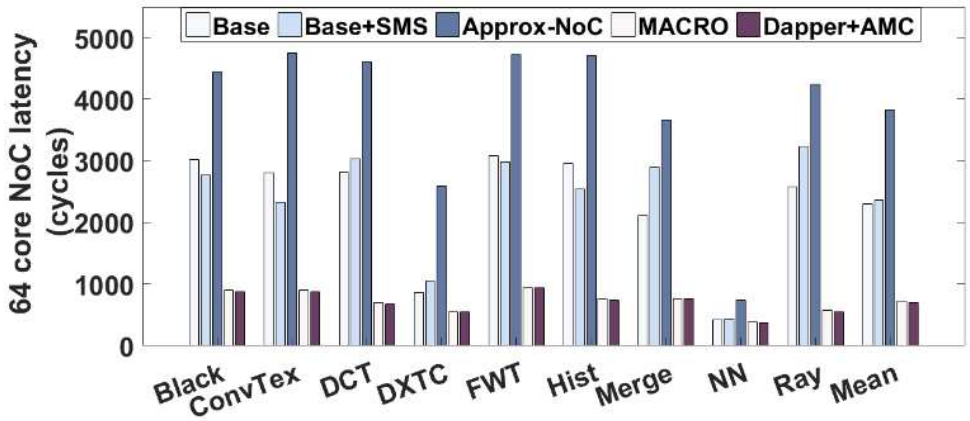
uses dictionary-based encoding and decoding at NIs by 68%, as the encoding and decoding process takes an additional 2-3 cycles at both the sending and receiving nodes of a NoC for all packets. *Dapper+AMC* outperforms *MACRO* by around 7%. These benefits are due to the joint optimization of request scheduling and reply packet coalescing in *AMC* that is absent in *MACRO*. For memory intensive and high approximable workloads such as *ConvTex*, *DCT*, and *HIST* the throughput improvement of *Dapper+AMC* reaches up to 40% compared to the baseline. For minimal approximable benchmarks like *MergeSort*, the throughput benefits of *Dapper+AMC* are diminished, however it achieves same throughput as the baseline. Figure 64 (b) shows that *Dapper+AMC* is highly scalable with average throughput improvement of up to 15.7% compared to the baseline and up to 15.34% compared to *base+SMS*. The improvements are slightly diminished in *Dapper+AMC* in the 64-core platform compared to the 16-core platform due to the higher number of cores sending more requests at the MC which increases DRAM latency for memory requests that worsens the MC bottleneck issue. Unlike *Dapper+AMC*, *Approx-NoC* loses up to 50% of its throughput in this configuration. *Dapper+AMC* has up to 5.35% better throughput than *MACRO*. This shows the importance of *approximate data-aware* request scheduling and data *coalescing* with the *AMC*.

6.4.4. NoC LATENCY ANALYSIS

Figure 65 (a) shows the average NoC latency observed from the source to destination in a 16-core accelerator. *Dapper+AMC* is much faster than baseline and *base+SMS* by up to 45.5% and 46%, respectively. This is one of the key contributors of increase in throughput.



(a)



(b)

Figure 65 NoC latency analysis of *Dapper+AMC* with Baseline NoC, Baseline NoC with SMS memory scheduling, Approx-NoC, MACRO NoC for (a) 16 core and (b) 64 core GPGPU accelerator, across CUDA applications.

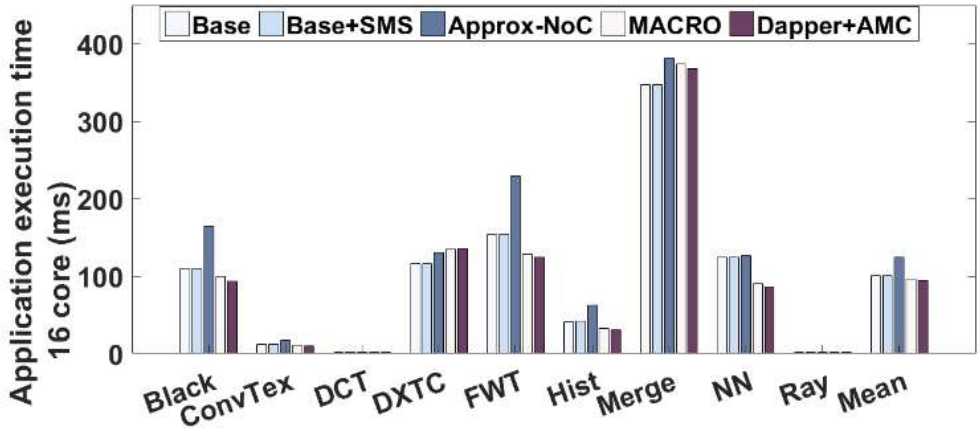
The *AMC*'s *approximate data-aware* scheduling scheme is responsible for *Dapper+AMC'*s 2.5% improvement compared to *MACRO* due to the increased approximable and *coalesced* read reply data that reduces the MC bottleneck and NoC congestion. Also, in *Dapper+AMC,* the ACK based flow-control mechanism together with the cyclic output buffer ensures that MCs utilize their share of overlay circuits efficiently. *Approx-NoC* has a higher latency compared to the baseline due to its slower encoding and decoding steps at the NIs.

Figure 65 (b) shows the comparison of NoC latency in 64-core accelerators. The baseline NoC latency in 64-core platforms is up to 3× higher than the 16-core platforms because of the increased NoC traffic due to the higher number of cores in the platform. *Dapper+AMC* shows up to 78.7% improvement compared to the baseline, and 85% improvement compared to *base+SMS*. The NoC latency in *base+SMS* is higher than baseline due to the waiting time (at the incoming buffers of the MC) needed to create batches of requests according to the SMS scheduler. This impact is higher in a 64-core platform than a 16-core platform. In *Approx-NoC*, the latency degradation is reduced from 50% to around 22% compared to the baseline. As the ratio of MCs to cores decreases from 16-cores to 64-cores, the overhead of encoding/decoding is hidden in the packet traversal latency, allowing the approach to recover some of its lost latency. *Dapper+AMC* performs better than *MACRO* by 4.8% in 64-core platform which is better than improvements of ~3% in 16-core platform due to the higher ratio of approximable packets *in Dapper+AMC* in a 64-core platform.

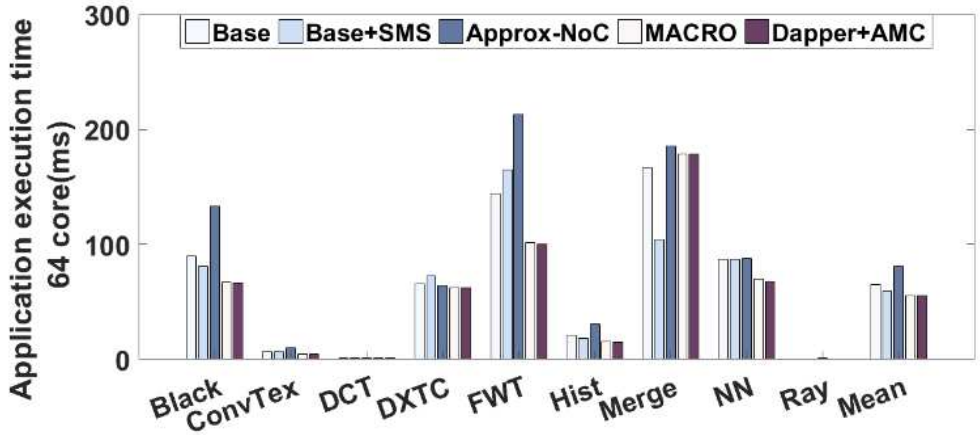
6.4.5. APPLICATION EXECUTION TIME ANALYSIS

Figure 66 (a) shows the application execution times observed in 16-core GPGPU accelerators for the various architectures. The throughput and latency improvements observed in the previous sections are translated into shorter application execution time for *Dapper+AMC* compared to baseline, and *base+SMS*. In 16-core platforms, *Dapper+AMC* achieves around 9.5% improvement compared to both baseline and *base+SMS*. The improvements for applications such as *DXTC, and MergeSort* the application execution time are not proportional to the NoC throughput and latency. These applications are also compute intensive where they have enough kernels to execute while waiting for data. The faster NoC latency can reduce the number of kernels concurrently scheduled on the cores that could potentially increase the execution time in these applications. In these

applications, *Approx-NoC* performs similar to *Dapper+AMC*. The same trends are observed in the 64-core platform as shown in Figure 66 (b). On average *Dapper+AMC* shows 5.4% improvement compared to baseline, and 4.5% improvement compared to *base+SMS*. In the *MergeSort* application, *base+SMS* gives 35% improvement compared to baseline in a 64-core platform due to the batch-based memory scheduling that is optimized for highly data parallel applications. However, most of the contemporary applications that are highly approximable perform better with *Dapper+AMC*.



(a)

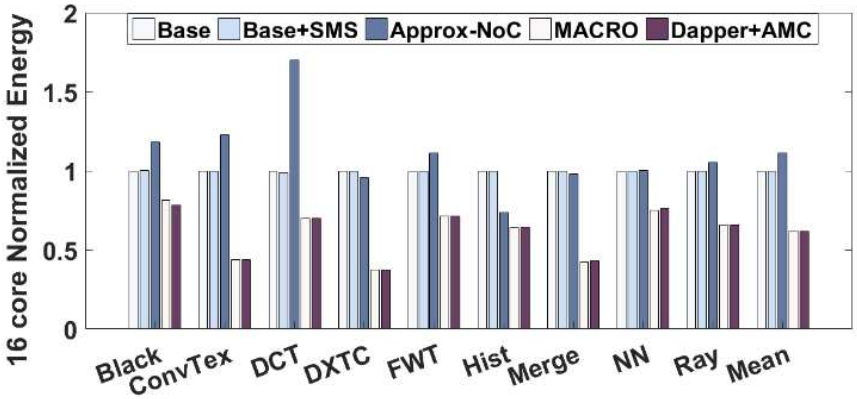


(b)

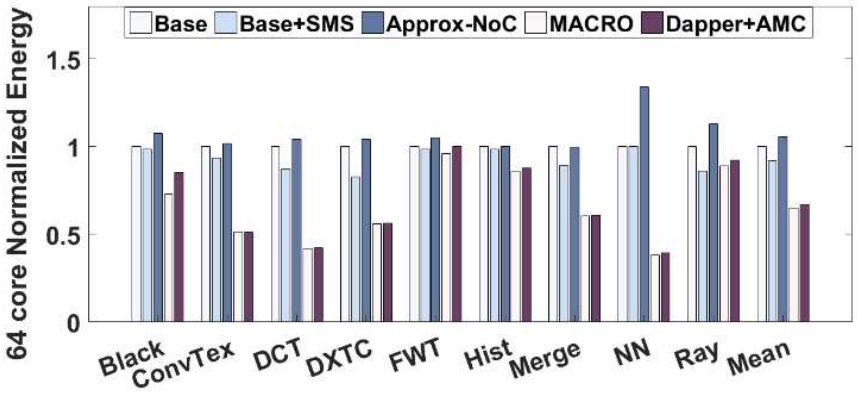
Figure 66 Comparison of application execution times for baseline NoC, baseline NoC with SMS memory scheduling, Approx-NoC, MACRO NoC and *Dapper+AMC* for (a) 16 core and (b) 64 core GPGPU accelerator, across CUDA applications.

6.4.6. ENERGY CONSUMPTION ANALYSIS

Figure 67 (a) shows the energy consumption of *Dapper+AMC* compared to the prior works across different benchmarks in a 16-core platform. On average, *Dapper+AMC* consumes up to 38.3% less energy compared to the baseline and 38.1% less energy compared to *base+SMS* architectures. *MACRO* and *Dapper+AMC* consume lower energy due to their low power router architectures in the reply plane. Even though *Dapper+AMC* has energy overhead involved in the *AMC* architecture, the overall reduction in the number of packets injected into the NoC reduces the total energy consumption, compared to the baseline. *Approx-NoC* on the other hand consumes higher energy due to the higher power dissipated in its routers that need additional logic and content addressable storage (CAS) based registers for saving the most used values for compression and decompression. In *Dapper+AMC,* energy savings are observed for all the applications including those that have lower to medium ratio of approximable data such as *MergeSort*, *FWT*, and *Blackscholes,* due to the low power NoC used in the reply plane and faster execution times. However, the energy savings are higher when applications have a higher ratio of approximable packets, such as in *ConvTex, DXTC* and *DCT*.



(a)



(b)

Figure 67Comparison of energy consumption between baseline NoC, baseline NoC with SMS memory scheduling, Approx-NoC, MACRO NoC and *Dapper+AMC* for (a) 16 core and (b) 64 core GPGPU accelerator, across CUDA applications.

Figure 67 (b) shows the energy consumption of different architectures for a 64-core platform. In the 64-core platform, the energy savings follow the same trend as in the 16-core platform. On average, *Dapper+AMC* consumes up to 30% lower energy compared to the baseline, and 27.5% lower energy compared to *base+SMS*. However, *Dapper+AMC* consumes slightly higher energy while scheduling requests and coalescing reply packets than *MACRO* by around 1.5%. But this is not a major drawback given the improvements in NoC throughput and latency given by the addition of *AMC* in *Dapper+AMC*.

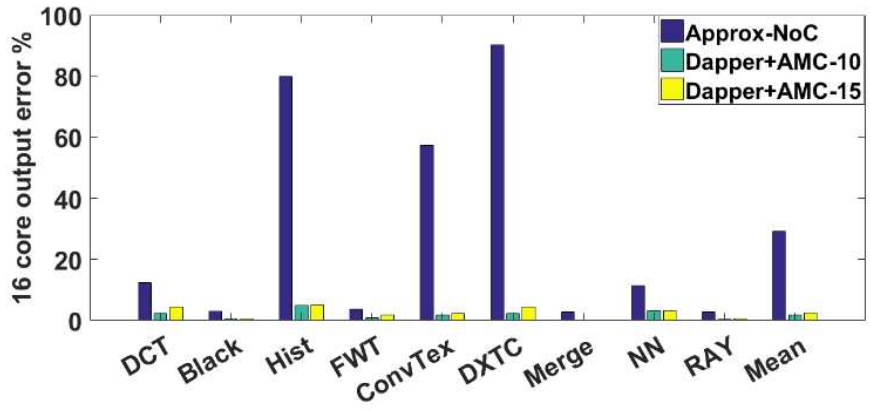
6.4.7. OUTPUT ERROR PERCENTAGE ANALYSIS

Obviously, it is important to analyze the error in applications that are subjected to approximation. Figure 68(a) shows the comparison of output error % for *Approx-NoC* at 10% *error_threshold* and *Dapper+AMC* at 10% and 15% *error_threshold*. Note that the baseline, *base+SMS*, and *MACRO* NoC are not shown, as they have no output error. The error percentage is computed as $100 \times error$ from equation (33). *Approx-NoC* shows high percentage of error in its

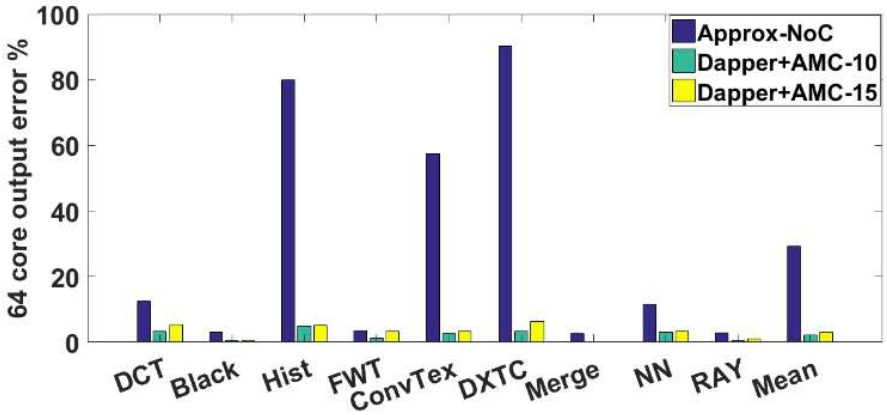
output due to a flaw (right shift division) in its *error_threshold* computation which is used for identifying the approximable data before compression. This results in a very high error percentage when data values are incorrectly marked for approximation. *Dapper+AMC* on the other hand incurs 1 to 4% output error in the application. On average, *Dapper+AMC* gives around 2.3% error in application output in the 16-core platform.

The same trend is shown in Figure 68 (b) for a 64-core platform. Although it seems intuitive that a 64-core platform may *coalesce* more packets than a 16-core platform, the ratio of approximable packets is dependent on the application's input and not the platform size. Hence, the output error percent has no discernable change as we move from a 16-core to a 64-core platform. *Thus, Dapper+AMC represents a promising NoC-and-MC-centric solution to improve performance and save energy consumption in GPGPUs for applications that have a potential for being approximated (i.e., applications that can tolerate some output error).*

As an illustrative example, Figure 69 shows the *DCT* application output under no error (when using the baseline NoC), and when using the *Dapper+AMC* in a 16-core platform. It can be observed that the output for the configuration of *Dapper+AMC* with 10% *error_threshold* is virtually indistinguishable from the no error case, while saving almost 38% energy (Figure 67 (a)) compared to the baseline NoC. This example highlights the exciting potential of *Dapper+AMC* to save energy in GPGPUs while increasing overall performance.



(a)



(b)

Figure 68 Comparison of output error values between Approx-NoC [164] and *Dapper+AMC* for (a) 16 core and (b) 64 core GPGPU accelerator, across various CUDA applications.

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

Figure 69 DCT output: (a) original (no error), (b) with 10% error_threshold (c) 15% error_threshold in *Dapper+AMC*

6.5 CONCLUSIONS

In this chapter, we propose a novel data-aware approximate NoC *Dapper*, and an approximate memory controller *AMC*, for GPGPU accelerators. Our *Dapper+AMC* architecture uses intelligent memory scheduling in the MC to increase throughput and also identifies the approximable data waiting in the output buffers of the MC and coalesces them to reduce the number of packets injected into the NoC. Also, we advocate for a fast overlay circuit based NoC architecture in the reply plane of the NoC for the reply packets to reach their destinations in 3 cycles or less. Experimental results show that on average *Dapper+AMC* increase the NoC throughput by 21% in 16-core platforms and up to 7% in 64-core platforms while consuming up to 38% less energy compared to baseline NoC, with up to 9.5% lower application execution time and around 2.3% error in the application output compared to the baseline. Thus, for the class of emerging applications that have the ability to tolerate a small amount of error in their outputs, *Dapper+AMC* can provide significant savings at the NoC and MC level. These savings are orthogonal to (and can be combined with) further approximation strategies at the computation components (e.g., using approximate adders) to further expand the design space of trade-offs between application output error, energy costs, and execution time.

7. LIGHTWEIGHT MITIGATION OF HARDWARE TROJAN ATTACKS IN NOC-BASED MANYCORE COMPUTING

Data-snooping is a serious security threat in NoC fabrics that can lead to theft of sensitive information from applications executing on manycore processors. Hardware Trojans (HTs) covertly embedded in NoC components can carry out such snooping attacks. In this chapter, we first describe a low-overhead snooping invalidation module (*SIM*) to prevent malicious data replication by HTs in NoCs. We then devise a snooping detection module (*THANOS*) to also detect malicious applications that utilize such HTs. Experimental analysis shows that unlike state-of-the-art mechanisms, *SIM* and *THANOS* not only mitigate snooping attacks but also improve NoC performance by 48.4% in the presence of these attacks, with a minimal ~2.15% area and ~5.5% power overhead. With the rise in number of processing cores and growing parallelism in applications, the communication traffic in a manycore processor has been increasing. Chip designers and manufacturers are moving towards network-on-chip (NoC) as their de-facto intra-chip communication fabric [34] [37]. Typically, emerging manycore processors have tens to hundreds of components that are designed either by in-house engineers or obtained from third-party vendors (3PIP), and then finally integrated together in a single global facility. With the growing complexity in NoC design, designers are opting for third-party NoC IPs, e.g., [27], to connect the components in their processors. This global trend of distributed design, validation, and fabrication has led to major challenges in ensuring secure execution of applications on manycore platforms, in the presence of potentially untrusted hardware and software components.

Much work has been done to mitigate side-channel attacks on shared resources and to detect counterfeit ICs that compromise manycore chip performance [166] [167]. This work focuses on

an orthogonal attack scenario where an adversary can insert a hardware Trojan (HT) into the RTL or the netlist of a manycore processor to disrupt or alter the integrity of its behavior without being detected at the post silicon verification stage. HTs can be inserted by an intellectual property (IP) vendor, untrusted CAD tool/designer, or at the foundry via reverse engineering [168]. We focus on one such attack called a data-snooping attack where a malicious software and an HT work together to steal information from applications executing on manycore processors.

NoCs are ideal candidates for such attacks as they have a complex design that can be used to hide an HT which cannot be easily detected via functional verification. HTs can be placed in NoC links, routers, or network interfaces (NIs) to secretly snoop on the data or corrupt data passing through them. Typically, in data-snooping attacks HTs create duplicate packets with modified headers and send them into the NoC for an accomplice thread to receive them [28]. Several works propose packet encoding/error correction mechanisms such as parity bits and ECC in NoC packets to detect faulty data packets at the receiver [169] [170]. Other works such as [171] [172] [28] [173] have also proposed data protection mechanisms in the presence of an HT in NoC components. However, there are three major shortcomings with the state-of-the-art: (1) these works assume the presence of HTs in NoC routers or links which can be detected by physical inspection or functional verification, without employing costly security mechanisms; (2) the mechanisms proposed in prior works protect application data from snooping attacks but do not detect the attack and mitigate future attacks; and (3) most of the security enhancement mechanisms are costly to implement and increase NoC latency and power consumption which worsens the overall performance. *It is important to design and deploy lightweight mechanisms that can detect the operation of malicious HTs embedded in NoCs and accomplice threads, and secure against their data-snooping attacks in emerging manycore processors.*

In this chapter we focus on security enhancement that do not notably increase performance and power overheads. We provide robust yet low-power mechanisms to detect the source of the attacks by utilizing controlled aging in circuits at runtime, which is not easy to obfuscate or tamper with in the design and fabrication process. Our novel contributions in this work are as follows:

- We first design and demonstrate a data-snooping attack using an HT in the NoC interface that duplicates packets and injects them into the NoC with a minimal area and power footprint, making it difficult to detect by traditional functional verification mechanisms;
- We then protect against such data-snooping attacks by proposing a novel snooping invalidation module (*SIM*) that uses an encoding-based duplicate packet detection mechanism;
- We further propose a novel data-snooping detection circuit called *THANOS* that uses threshold voltage degradation as a means to detect an on-going attack at runtime and blacklist the malicious software task that initiated the attack;
- Experimental analysis shows that *SIM* with *THANOS* provides security against HTs with minimal area and power overhead.

7.1. RELATED WORK

Significant research has been done to increase robustness against attacks by HTs in NoCs by assuming that an HT tampers or snoops data passing through it. In [171], bit shuffling and Hamming ECC are used to reduce the effectiveness of HTs that corrupt data. In [172], security zones managed by a centralized security manager are proposed to protect sensitive information from being accessed by malicious agents. In [28] data scrambling, packet authentication, and node obfuscation are proposed to prevent data stealing by a compromised NoC. Data scrambling, and packet-authentication mechanisms use a one-time pad XOR cipher that can be broken by the

malicious tasks when enough encrypted packets are accumulated. In [173], CRC and algebraic manipulation detection (AMD) are used to encode packet headers to safeguard from faults and snooping attacks. In [174], a novel wave-based scheduling mechanism for NoCs is proposed that eliminates the need for TDMA-based NoC resource sharing, hence providing non-interference between different domains of applications. In [175], a process variation-based packet encoding and decoding mechanism is proposed to prevent data-snooping in silicon photonic NoCs. *Most of these schemes that protect application data from NoC security attacks lack an efficient and low-power attack detection mechanism which makes them incomplete in providing security.*

A few works address HT detection in NoC components at design-time and runtime. At design time, techniques such as physical inspection [176], functional testing [177], and side channel analysis [178] have been proposed. But testing for HTs at design time is still in infancy, and the growing complexity of NoC components make this even more difficult. Hence, designers are now exploring runtime detection methods. A key logic built-in self-test (LBIST) was proposed in [179] that uses test vectors generated by programmable keys to detect Trojans. However, LBIST requires that the chip operation should be paused while testing at regular and frequent intervals, which is not suitable for NoCs that should function seamlessly. A few other works such as [180] [181] propose in-situ HT detection modules that rely on verification units placed in NoC components to detect HTs. There are two limitations with all of these works: (1) the verification units used to detect HTs can also be reverse-engineered and tampered, (2) these mechanisms are used to detect only HT induced data-corruption attacks. Data-snooping attacks unlike data-corruption attacks attempt to leak critical application data to malicious software tasks. *None of the prior works have addressed the problem of detecting the software task that initiates data-snooping attacks to blacklist and prevent future attacks.*

In [182], a run-time technique called NoCAlert is proposed to detect failures in the control logic of NoC components. This technique is further enhanced by [183] that proposes modules which alert the host system if the control logic in NoC routers detects invariance violations caused by HTs placed in its control-path, e.g., logic for route computation (RC) or virtual channel allocation (VCA). However, these techniques focus on NoC components that have substantial control logic, such as routers. They ignore the network interface (NI) which prevents easy placement of model checkers to detect packet duplication. In this chapter we propose a novel snooping invalidation module (*SIM*) in the NI that can mitigate snooping attacks. We then propose low-overhead techniques to detect the source of data-snooping attacks in NoCs. *To the best of our knowledge, this is the first work that mitigates snooping attacks in NoCs with minimal performance and power overheads, while also detecting the source of snooping attack to protect against future attacks.*

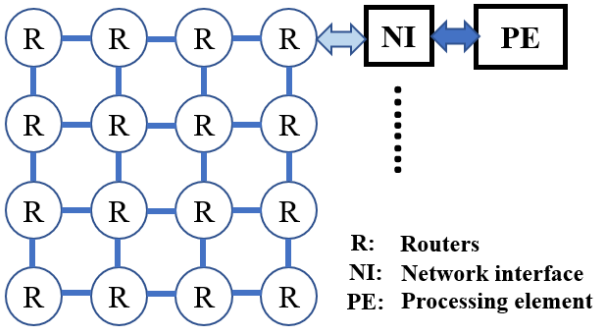


Figure 70 Baseline NoC architecture with example routers, a PE and an NI

7.2. BACKGROUND AND ATTACK MODEL

7.2.1. BACKGROUND

In this section we discuss our assumed baseline NoC design. We consider a traditional 2D mesh based NoC with processing elements (PE) connected to the NoC via a network interface

(NI). The packets entering the NoC are routed towards their destination by routers that use a hop-by-hop, turn-based distributed deadlock free XY routing algorithm. Figure 70 shows the schematic of the baseline 2D mesh NoC with an NI and PE connected to routers. We use traditional 3-stage {*buffer write, RC+VCA+SA, LT*} pipelined routers in the NoC with wormhole switching and 4-VC buffers at each input port. PEs communicate using messages that are passed to the NI which packetizes them before sending them to the NoC. The packets received by the NI from the NoC are de-packetized and sent to the connected PE. We consider ARM Cortex-A73 cores in our PEs that use the AXI interface for communication. Each PE has a private L1 cache and a shared distributed L2 cache that uses a scalable directory-based cache coherence protocol to send messages in the form of NoC packets.

7.2.2. ATTACK MODEL

Prior works [171] [172] [28] [173] assumed data-snooping attacks to be carried out by HTs embedded in NoC routers or by compromised links that enable HTs to modify the packet headers. These HTs, once activated by a flit with a special activation sequence, make copies of packets passing through the router and transmit them to the PE that has a malicious accomplice task running on it. Once an HT is activated in a router, it generates new packets, or diverts an existing packet to the PE running the accomplice task. This type of HT, that has a high 4% area overhead [28], may be noticed by testers while conducting physical inspection or side channel analysis. Moreover, this type of attack can lead to illegal utilization of router resources such as buffers, VCs, and switch allocators, which cause control logic violations that can be detected by secure model checkers [183]. We thus focus on a harder-to-detect attack with an HT embedded in the NI where

packets are generated, and hence packets can be duplicated with relatively simpler logic without interfering with the basic NI functionality.

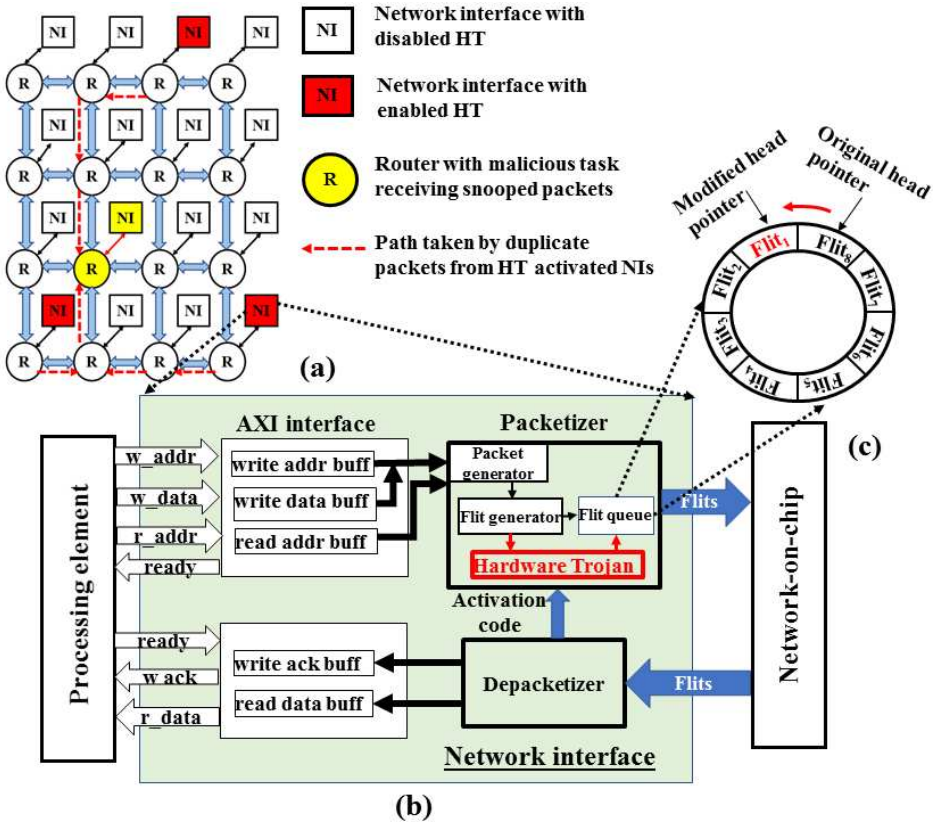


Figure 71 (a) Overview of attack model on a NoC with a malicious software task coordinating the data-snooping attack, (b) microarchitecture of network interface (NI) with a hardware Trojan embedded in packetizer module, (c) FIFO queue modification by hardware Trojan.

Figure 71 (a) shows an overview of an on-going data-snooping attack taking place in a 2D NoC based manycore processor with multiple HTs activated in the NoC NI modules shown in red, and a malicious task running on a PE connected to the yellow router and NI. The HTs make duplicate copies of packets in NIs that are sent to the malicious task.

7.2.3. DESIGN DETAILS: NETWORK INTERFACE WITH A HARDWARE TROJAN

Figure 71 (b) shows the microarchitecture of an NI with an embedded HT in the packetizer module. The NI receives messages from the PE via the AXI interface that are then stored in its buffers. The messages usually are read/write commands with address and data fields. The packetizer module appends source ID, destination ID, and virtual channel ID information to the commands and creates packets. A packet is further divided into flits, with the header flit containing the NoC routing related information. The packet flits are then injected into the circular flit queue that is accessed via head and tail pointers. After the packetizer injects a flit, the tail pointer of the queue is incremented. After a flit is transmitted to a router, the head pointer is incremented to transmit the next flit.

An HT can potentially tamper with the pointer values to re-send duplicate packets intelligently. Once a flit has been transmitted from NI to the router, it stays in the cyclic queue until a new flit is overwritten on that location. The HT can keep track of these locations to read a header flit that has already been transmitted to the router, append it with a duplicate destination ID of the malicious node, and *update the head-pointer*. Figure 71 (c) shows how the HT modifies the head pointer. By moving the head pointer at regular intervals, the HT can send duplicate packet flits without having to store them externally. The duplicate packet is re-sent to the router for transmission. If the flit queue is full (head pointer = tail pointer), both the HT and packetizer do not inject new flits into the queue, and do not accept any more incoming data from the PE until the outstanding flits are transmitted. The HT does not interfere with the control logic which is mostly present in the AXI interface, and an attacker can snoop on data using this HT in NIs between two PEs, or between a PE and a memory controller that is connected to main memory channels.

We now perform an overhead analysis of this HT. The proposed HT requires an internal memory to save the head flit to modify its destination ID, save the header pointer of the queue, and save the current state of the HT (~72 bits). We designed the NI shown in Figure 71 (b) by modifying the CONNECT open-source NoC model [184] and used Xilinx's Vivado HLS [185] tool to analyze the overheads. Table 8 shows the clock cycle period, number of flipflops and LUTs used for an FPGA implementation of the packetizer. The optimized design indicates that the NI with an HT requires an additional ~5% FFs and ~1% LUTs (1.3% area overhead) without incurring additional timing latency. This low overhead HT can be inserted at the RTL level, or by reverse engineering and changing the netlist at the place and routing stage [168] [176]. The small size of the HT makes it hard to detect by physical inspection or by side-channel analysis. Also, the run-time secure model checkers from [182] [183] are not able to check the validity of flits in the NI as it does not interfere with the control logic. *Hence, there is a need to design a low-overhead flit validation module in NIs to check flit validity before injecting them into the NoC.*

Table 8 FPGA implementation of NI packetizer with and without hardware Trojan (HT)

	Timing (ns)	**Number of FFs**	**Number of LUTs**
NI without HT	3.45	258	535
NI with HT	3.45	273	549

7.3. MITIGATION OF NOC SNOOPING ATTACKS

We propose a novel framework that integrates two mechanisms to mitigate data snooping attacks from taking place, as well as to detect the source of an on-going attack, and protect against future snooping attacks. We rule-out data corruption attacks as they can be detected and corrected using ECC codes such as in [173]. Our proposed framework consists of two security mechanisms,

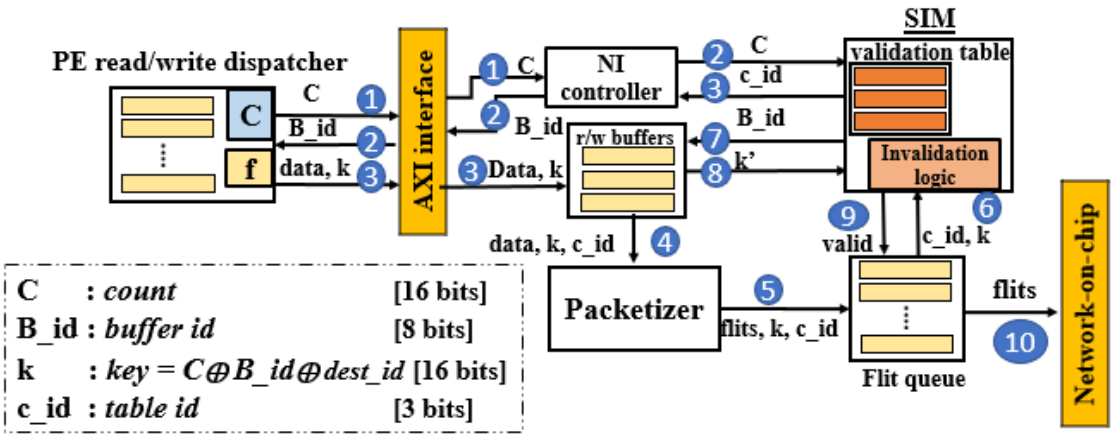
(1) a snooping invalidator module at the NI output queue to discard duplicate packets, (2) detection of data-snooping attacks at the PE where an accomplice thread is executing. This comprehensive protection framework ensures that we proactively mitigate future attacks and safeguard the application data for the entire lifetime of the processor. We have designed our security mechanism to be hard to be tampered by adversaries that use reverse engineering techniques to insert HTs in the netlist. Our approach also works irrespective of the HT triggering process to start snooping attacks such as special flit data, circuit aging, or temperature [186]. The following sections discuss our two security mechanisms.

7.3.1. SECURITY ENHANCED NI: PREVENTING DATA-SNOOPING ATTACK

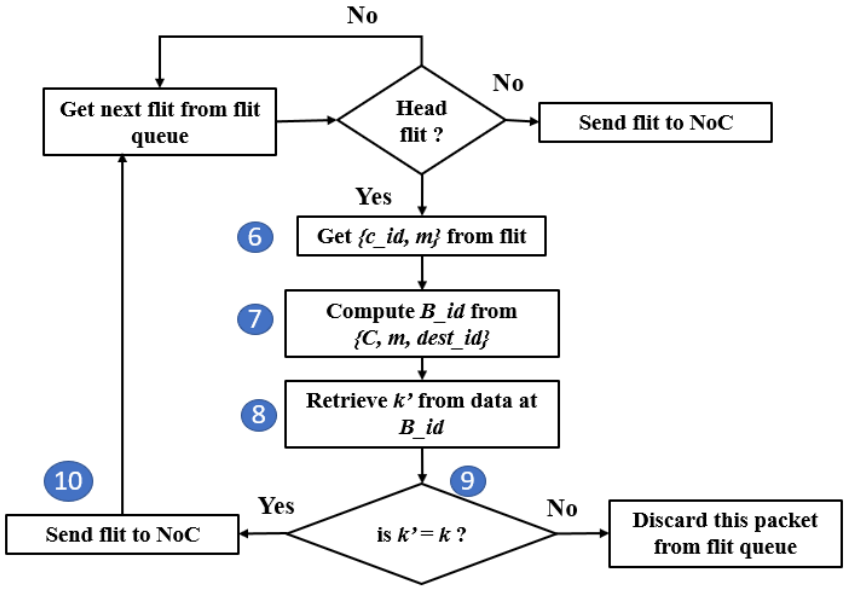
The first security enhancement mechanism is to prevent a snooping attack with the help of a snooping invalidator module (*SIM*) at the NI. Using *SIM*, we aim to discard packets with invalid header flits from being injected into the NoC. Unlike traditional ECC-based security enhancement mechanisms, *SIM* incurs low-power and low latency overheads because of its lightweight computations that are designed solely to mitigate snooping attacks. Figure 72 (a) shows an overview of the security enhancement in NI using *SIM*.

We divide the implementation of *SIM* across the PE and NI to prohibit 3PIP NoC designers/testers to reverse engineer or tamper with the secret encoding/decoding information at runtime. The PE and NI communicate using the standard AXI hand-shake protocol (ready, valid, and valid ready signals). A typical NI receives messages from the PE to be packetized and sent to the NoC and vice-versa. In the security enhanced NI, additional encoding information (*key*) is attached with the data received from the PE to validate the uniqueness of data packets. The

numbered sequence of steps shown in Figure 72 describe how a packet is validated using *SIM*. These steps are discussed next.



(a)



(b)

Figure 72 (a) Security enhanced NI using SIM (b) Flowchart of snooping invalidation mechanism in NI

In step 1, the PE data dispatcher sends a *count* (*C*: increments with each outgoing data) value to the NI controller along with the AXI ready signal. In step 2, the NI controller sends a buffer id

(*B_id*) that is reserved to store the incoming data along with the AXI valid signal to the PE data dispatcher. The NI controller simultaneously sends *C* to *SIM* that stores it in a "validation table". The PE uses an XOR function *f* to generate an encoded key *k* as a function of *C*, *B_id*, and destination ID (*dest_id*) of the packet, as shown in equation (34) below. In step 3, the PE sends a message *{data, k}* combination to the data buffers and toggles the valid ready signal to high. At the same time, the validation table sends a *c_id* (location where *count* is stored in the table) to the NI controller. This is stored with the message sent by the PE. The *{data, k, c_id}* combination is stored as a unit in read/write buffers till the packet is sent out of the NI. While the size of payload *data* varies from 8B to 128B depending on message type, the values of *k, C* and *c_id* require only few bits of storage (see legend of Figure 72 (a)).

$$\boldsymbol{k = C \oplus B\_id \oplus dest\_id} \qquad (34)$$

$$\boldsymbol{B\_id = C \oplus k \oplus dest\_id} \qquad (35)$$

In step 4, the *{data, k, c_id}* combination is sent to the packetizer to generate packets. In step 5, flits are generated with *k* and *c_id* copied into the header flit. We save *k* and *c_id* in the 24-bits reserved in the header flit to store destinations of source-routing path [187] which are unused as we adopt distributed routing for our NoC. The flits are then saved in the output flit queue. *Steps 6-9 are part of snooping invalidation flow explained in more detail in Figure 72 (b). SIM* tries to retrieve the encoded key *k* from the *C* entry in the validation table as a part of packet validation. In step 6, *SIM* reads *dest_id*, *k, and c_id* bits of the header flit, and performs a decoding operation shown in equation (35) to obtain *B_id* of the buffers that stored the corresponding packet data, and *k* sent by the PE (step 7). In steps 8 and 9, *SIM* retrieves the value of *k'* stored in the buffer located at *B_id* and compares with *k* that is read from the header flit. If *k=k', SIM* sends a valid signal that the header flit is valid, and it is injected into the NoC. If *SIM* sends an invalid signal, the flit queue

discards all the flits corresponding to the duplicate packet. *SIM* efficiently detects duplicate packets because, if the value of *dest_id* is modified by the HT, equation (35) leads to an incorrect value of *B_id* that does not retrieve the *k* value corresponding to the data of packet sent in step 3. Note that for broadcast/multicast packets, multiple keys are generated for each *dest_id* value and key verification steps 8 and 9 are performed on each of them separately. After a packet is sent out, the corresponding read/write data buffer and validation table entries are reused for new data. *This low-overhead SIM module with minor modifications can also be used to curb potential data duplication at router-link interfaces or within a router.*

7.3.2. OVERHEAD ANALYSIS

Several steps in the *SIM* module can be performed in parallel. The existing communication data channel between the PE and the NI that is established by AXI interface is used to communicate both packet data and *SIM* metadata (*C, k, B_id* in steps 1 and 2). Hence, no additional wires are needed to transmit *SIM* metadata. Steps 1 and 2 are performed in parallel with AXI interface's ready and valid signal exchange to minimize the latency overhead. Also, there is no additional overhead involved in steps 3 to 5. Steps 6 to 9 take one cycle in a NoC that is clocked at 1GHz frequency which was verified via FPGA synthesis of the modified NI [185]. This increases the number of pipeline stages of the NI microarchitecture. *SIM* takes additional memory to maintain a validation table, and additional logic to perform XOR and comparison operations. *SIM* incurs ~5.5% more power and ~2.15% more area overhead compared to the baseline NI with a buffer capacity of 16 packets, at the 22nm technology node.

7.4. DETECTING THE SOURCE OF A DATA-SNOOPING ATTACK

Using our security enhanced NI with the integrated *SIM*, we can curb packet duplication at the NI. However, the malicious task that is the source of the attack is still not detected that could initiate attacks from compromised routers or links [173]. In this section, we propose a module called *THANOS,* a novel threshold activated snooping attack detector that is implemented at the interface between an NI and a PE, as shown in Figure 73 (a) to detect the source of the snooping attack.

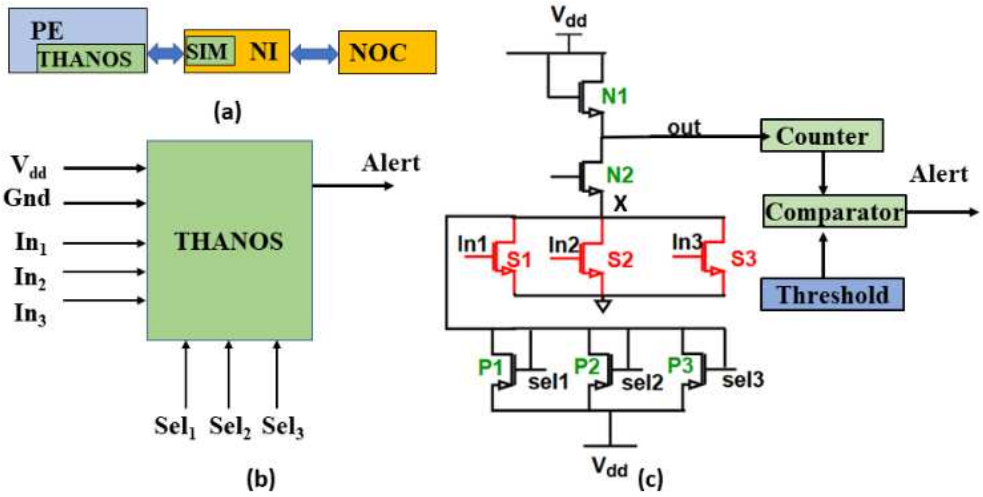


Figure 73 (a) Overview of THANOS (b) block diagram of THANOS showing inputs and outputs (c) snooping detecting circuit used in THANOS

A PE sends and receives various types of messages into the NoC that can be broadly classified into two types: (1) direct messages between cores for inter-core communication, and (2) cache-coherence messages between a PE and directory table. Figure 74 (blue bars) shows the average incoming-outgoing message ratio sent over 64 cores in a NoC by different PARSECv2.1 [188] benchmark applications with 64 tasks each. The error-bars in Figure 74 represent variance across NoC nodes. Figure 74 shows that the ratio is less than 1 over all the benchmarks with each

node receiving a smaller number of messages than the messages it sends out (number of "packets" in a "message" can vary based on message type). Another important observation from Figure 74 is that the incoming-outgoing message ratio is much greater than 1 (red points) when a data-snooping attack takes place, because a PE receives significantly higher number of messages (and packets) than it sends out. This phenomenon can be easily detected in the short term by placing a counter in the NI and observing the number of incoming and outgoing messages over an epoch of time. However, observing messages in the short term can lead to false positives, e.g., due to periodic bursts of messages from a task that requires higher volumes of input data. Also, a message counter is not the most secure way to detect a snooping attack, given the reverse engineering techniques available to tamper digital logic [176].

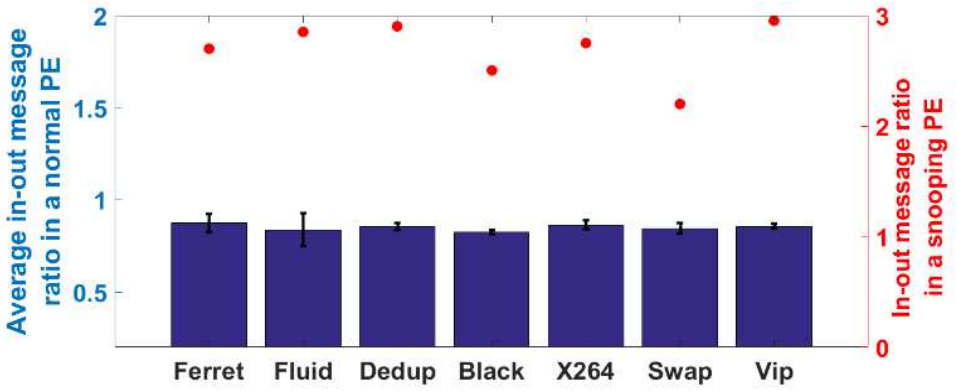


Figure 74 Average incoming-outgoing message ratio at normal PE (left), and at snooping PE (right) across different applications

In *THANOS* we devise a mechanism that observes the ratio of incoming and outgoing messages over a period of few hours and identifies the source of a snooping attack. *THANOS* is designed using a combination of analog and digital logic to detect if a PE is snooping on messages over a duration of time. As *THANOS* is not entirely a digital logic implementation, it is hard to reverse engineer or tamper with, and can be used as a final frontier to mitigate data-snooping

attacks. *THANOS* receives inputs from the PE and sends a security alert signal to the PE as shown in Figure 73 (b). The PE then identifies the source of data-snooping attacks and takes preventive steps to mitigate future attacks. *THANOS* is designed as a standalone module that can also be used with prior data protection schemes [171] [172] [28] [173] to detect the source of attack.

7.4.1. OVERVIEW OF SNOOPING DETECTION CIRCUIT

We take inspiration from a controlled aging module [167] that uses threshold degradation of NMOS transistors due to aging phenomenon such as bias temperature instability (BTI) and hot carrier injection (HCI) to detect chip usage, which helps identifying counterfeit ICs. In *THANOS* we use NMOS threshold voltage degradation to detect a PE that is receiving duplicate packets injected by multiple HT activated NIs in the NoC. NMOS transistors undergo stress-recovery periods in their ON and OFF operations that leads to threshold voltage (V_{th}) degradation [189]. Figure 75 shows the V_{th} degradation observed across different ratios of stress and recovery in an NMOS transistor at 22nm using the long-term aging model proposed in [189]. At 100% stress (no recovery) the V_{th} of a transistor increases by ~ 100mV in about two hours duration. We use this phenomenon to detect snooping attacks.

7.4.1.1. OPERATION OF SNOOPING DETECTION CIRCUIT

In our snooping detecting circuit shown in Figure 73 (c), there are 2 transistors; N_1 that acts as a diode connected load and N_2 that acts as gate-source voltage (V_{gs}) sensor. P_1, P_2, P_3 are diode-connected PMOS transistors that pull the drain voltages of S_1, S_2, S_3 to high. Transistors S_1, S_2, S_3 are driven using low over-drive voltages In_1, In_2, In_3 that barely switch them ON. *We artificially induce stress in a selected transistor among $S_1/S_2/S_3$ when a message is received and induce*

recovery when a message is sent out. Hence, we call them stress-transistors. At any point only one of S_1,S_2,S_3 are connected to the circuit (using *In* and *sel* signals). When $S_1/S_2/S_3$ is turned ON, the source (V_x) of N_2 is pulled low, which turns ON N_2, leading to a "low" *out* state. But, when a stressed transistor ($S_1/S_2/S_3$) undergoes V_{th} degradation, its over-drive voltage ($In = V_{gs}-V_{th}$) is not high enough to turn ON the stress-transistor and hence drives it into the triode region. When $S_1/S_2/S_3$ is in the triode-region, the source voltage (V_x) of N_1 is not pulled low and the *out* signal is set to "high".

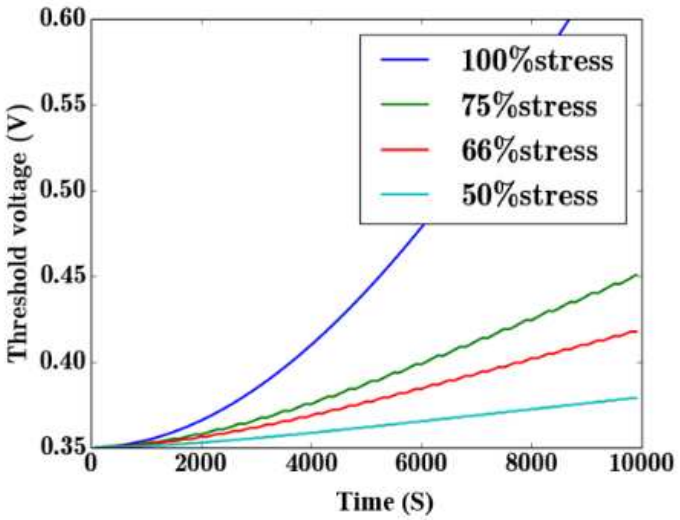


Figure 75 Threshold voltage degradation observed across different stress-recovery ratios in a NMOS transistor at 22nm technology node

Table 9 State transition of snooping detection circuit

Stress-transistors ($S_1/S_2/S_3$)	**V_x**	**N_2**	**out**
Saturated	Low	ON	Low
Triode	High	OFF	High

Table 9 gives the states of different transistors and the corresponding changes in *out* signal state. When a PE is not receiving snooped packets, its incoming-outgoing message ratio is less than 1 as shown in Figure 74. Hence, for normal NoC traffic the stress-recovery ratio of stress-

transistors ($S_1/S_2/S_3$) is less than 40%. Generally, BTI and HCI are slow wear-out phenomenon in logic circuits. But, we input low over-drive (V_{gs}-V_{th}) voltage of ~100mV to the stress-transistors through input signals $In_1/In_2/In_3$. Hence, the circuit would set the *out* signal to high state in a duration of 2-3 days. However, when a malicious task on a PE is snooping with up to four HT activated in NIs, its incoming-outgoing message ratio is 3× the average ratio (shown in Figure 74). As a result, the stress-transistors in *THANOS* undergo 80-90% more stress than recovery when there is a snooping attack. From Figure 75, when a stress-transistor receives ~90% stress, its threshold voltage increases over a shorter duration (~3-4 hours). Hence the snooping detection circuit toggles the *out* signal to "high" state quicker when PE receives snooped packets.

In *THANOS*, we use a counter to track the time taken for the *out* signal to change its state and compare it with a threshold time that is configured by a trusted PE firmware, as shown in Figure 73 (c). *THANOS* sends an ALERT signal when the time taken by the *out* signal to switch the state is less than the threshold. *Overall, THANOS sends a notification about a potential malicious task anywhere from ~2 hours to ~2 days based on the number of HTs that are active.* The trusted PE firmware then alerts the OS about the malicious application task executing on the PE, so that preventive measures can be taken.

The snooping detection circuit should last for the lifetime of the processor to detect snooping attacks. However, due to artificially induced stress and recovery cycles, the stress-transistors (S_1, S_2, S_3) wear-out much more rapidly than the rest of the chip. To increase the lifetime of *THANOS* we take two measures: (1) We input low over-drive voltage (In-$V_{th} \approx 100$mV) and high V_{dd} using separate power lines for stress-transistors; after every state change of the *out* signal, we increment the *In* signal by ~100mV until we satisfy the MOS saturation condition (In-$V_{th} < V_{dd}$); (2) The stress-transistors are over-provisioned; we use only one stress-transistor at any time to detect an

attack and when an *In* voltage of a stress-transistor can no longer be incremented without violating the saturation condition, *THANOS* switches to the next stress-transistor using the *sel* signal. Using three stress-transistors and V_{dd} = 3V, *THANOS* can seamlessly detect snooping attacks for up to 1.5 years. The number of stress-transistors in *THANOS* is hence left to the decision of the designer. The overhead of *THANOS* is negligible in power (~50 μW) and area (~0.9 μm^2) compared to the PE (~1W, ~318mm^2) at 22nm technology node as it requires just 8 MOSFETs, a counter, a comparator, and a simple control logic block to send input signals.

7.5. EXPERIMENTS

We target a 64-core manycore chip with low power ARM cortex-A73 cores and a 2D mesh NoC with 8×8 dimension to test the performance, latency, energy, and area overheads of the proposed lightweight snooping invalidation module *(SIM)* and snooping detection circuit *(THANOS)* compared to the state-of-the-art. For simulations, we modeled the behavior of *SIM* and *THANOS* as part of the cycle-accurate NoC simulator Noxim [140]. We obtained the power and area overheads of *SIM* and *THANOS* modules from post-synthesis vectorless estimation in Vivado [185], and Cadence Virtuoso [190], at 22nm. We integrate the latency and energy overheads of *SIM* and *THANOS* with Noxim for our simulations. We tested our framework using PARSECv2.1 benchmark NoC traces generated by netrace [191] to capture the request-response dependencies to accurately simulate parallel application performance.

We compare our work with a baseline NoC (with a configuration that is described in section 7.2.1) with no security mechanism employed, and with two prior works, FortNoCs [28], and P-Sec [173]. In [28], only data obfuscation and data scrambling techniques are implemented for a fair comparison. In [173] end-to-end algebraic manipulation detection (AMD) and cyclic

redundancy codes (CRC) are appended to the header flit for reliability against faults and HT attacks. We set the threshold time in the snooping detection circuit of *THANOS* as ~2.5 days to get a security violation alert. We first present results of application performance, NoC latency and NoC energy consumption for 4 actively snooping HTs that are randomly placed in NoC. Subsequently we present results for scenarios with 1 and 2 HTs operating in the NoC.

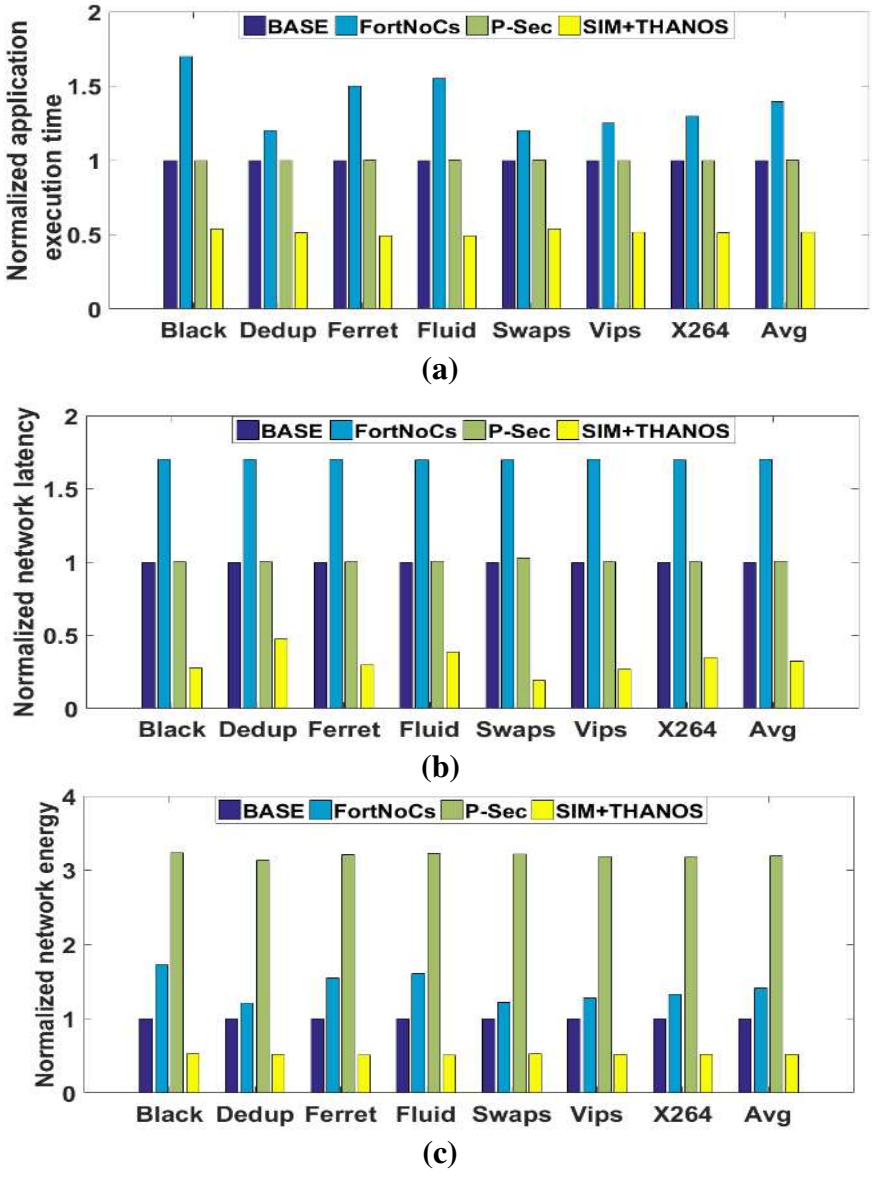


Figure 76 (a) Normalized application execution time, (b) normalized network latency, (c) normalized NoC energy consumption, across NoCs with different security mechanisms in the presence of 4 active HTs attempting to inject duplicate packets to an accomplice thread.

Figure 76 (a) shows the comparison of application execution time across different NoC security mechanisms. P-Sec and FortNoCs cannot prevent the injection of duplicate packets at the NI, and only discard faulty packets at the receiver, which leads to higher NoC traffic. Moreover, P-Sec takes two extra cycles for CRC+AMD encoding/decoding, and FortNoCs takes at least four extra cycles at the NI for node obfuscation and data scrambling techniques on the entire packet. This leads to poor application performance with P-Sec and FortNoCs compared to the baseline. *SIM* mitigates duplicate data packets near the source, resulting in less NoC congestion, thereby actually achieving 48.4% average improvement in application execution time, compared to the baseline.

A similar trend is observed for network latency, shown in Figure 76 (b). The network latency of FortNoCs is higher due to the packet scrambling mechanism that encrypts/decrypts the entire packet using XOR operation, which is time consuming for packets with high payload size. FortNoCs incurs additional overhead due to packet authentication as an additional security mechanism. *SIM*, in comparison, takes one cycle only at the sending NI to detect duplicate packets, and *THANOS* has no latency overhead. In the absence of duplicate packets in the NoC, *SIM* has the lowest NoC latency, and achieves an average of 67.8%, 77.3% and 68.1% latency reduction compared to the baseline, FortNoCs, and P-Sec in the presence of active data-snooping HTs.

Next, we analyze NoC energy consumption. Although *SIM+THANOS* consumes ~5.5% additional NI power, its energy consumption is 47.8% lower compared to baseline on average due to the lower application execution time as shown in Figure 76 (c). FortNoCs consumes around 41.8% additional energy compared to the baseline due to increased execution time and the overheads incurred to employ XOR encryption/ decryption logic in the NI. P-Sec consumes up to

200% more energy compared to the baseline due to its costly AMD, and CRC codec engines present in NIs and NoC routers. P-Sec is thus much more expensive, although it provides combined safety against faults and snooping attacks.

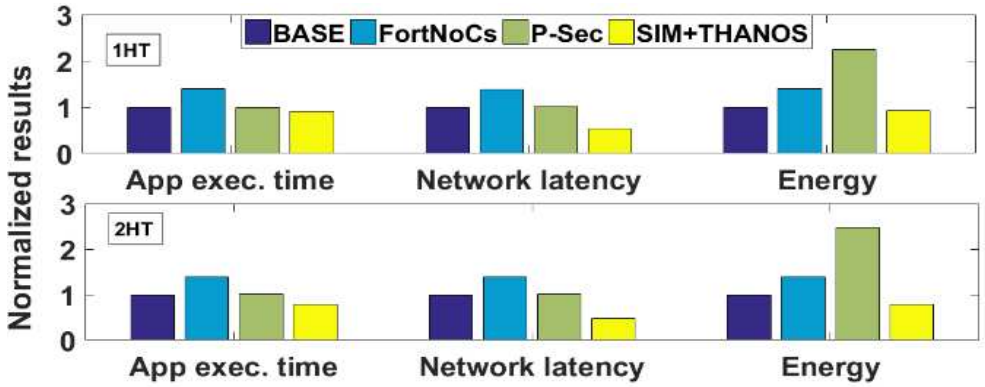


Figure 77 Normalized average values of application execution time, network latency, and NoC energy consumption across different security mechanisms with 1 HT (top), 2 HTs (bottom).

We observe similar trends in application execution time, NoC energy, and latency even when fewer number of HTs are active as shown in Figure 77, with *SIM+THANOS* performing better than the baseline unlike FortNoCs and P-Sec. *This shows that our proposed snooping invalidation and snooping detection mechanisms, SIM+THANOS, does not trade-off NoC performance and NoC energy consumption to provide security.* Lastly, we compare area footprint of *SIM+THANOS* with other schemes. As shown in Table 10, *SIM+THANOS* has the lowest area footprint amongst the three security mechanisms. *SIM+THANOS* mechanism consumes only 2.15% additional area in the NI to implement the packet validation mechanism.

Table 10 Area footprint of different noc security enhancement mechanisms

SIM+THANOS	FortNoCs	P-Sec
2.2 μm^2	4.9 μm^2	500μm^2

7.6. CONCLUSIONS

In this chapter we proposed a low-overhead mechanism called *SIM* to prevent data-snooping attacks that are initiated by HTs embedded in NoC network interfaces. We also proposed a lightweight standalone snooping-attack detection mechanism called *THANOS* that uses controlled circuit aging to detect the source of attacks that can help processors take preventive steps to mitigate future attacks. In FortNoCs and P-Sec it is impossible to detect the source of the attack, which can be addressed by using *SIM+THANOS*. Experimental results show that *SIM+THANOS* reduces application execution time by 62.9% and 48.3% and energy consumption by 63.5% and 83.7% compared to FortNoCs and P-Sec. *SIM+THANOS* incurs a minimal additional 5.5% power and 2.15% area overhead, compared to the baseline, much lower than the overhead for FortNoCs and P-Sec. Thus *SIM+THANOS* represents a promising solution to enhance NoC security in manycore processors.

8. CONCLUSION AND FUTURE WORK SUGGESTIONS

8.1. RESEARCH CONCLUSION

In this dissertation, we addressed significant challenges related to reliability, security, and performance bottlenecks faced by modern CMP designs. Today's CMPs are designed to trade-off chip performance for improving reliability or vice-versa. We have demonstrated that using intelligent resource management schemes and on-chip architectural enhancements in NoC and MC architectures, we can accomplish the goals of improved reliability and security without sacrificing CMP performance. We have proposed a cross-layer framework called *RELAX* that uses: (i) resource management techniques such as application mapping, DVS scheduling and checkpointing interval scheduling at the OS/system level, and (ii) novel NoC and MC architectures at the architectural level, that work in tandem with the feedback received from on-chip sensors at the circuit level. *RELAX* enhances the reliability of CMPs in the presence of device level phenomena such as NBTI, HCI, EM, along with transient faults due to single and multiple bit flips in CMPs while adhering to platform constraints such as the dark silicon power budget, chip target lifetime, and application deadlines. Apart from that, *RELAX* resolves performance bottlenecks in CMPs with integrated manycore accelerators that use NoC fabrics for communication between cores and DRAM. Lastly, *RELAX* integrates lightweight security enhancement mechanisms to protect the CMPs from hardware Trojan attacks. Experimental results of our proposed cross-layered framework validate the benefits of the holistic solution that jointly enhances reliability, performance, and security of CMP design.

ARTEMIS is the first component of *RELAX*. *ARTEMIS* is designed to extract useful chip performance from a given 3D CMP and extend its lifetime by intelligently mapping applications

on 2D/3D regions of the CMP and by dynamically scheduling supply voltage (V_{dd}) of the tiles with different aging profiles. Unlike the state-of-the-art, we consider both V_T-degradation in cores and IR-drop in PDN while choosing cores for mapping applications and DVS scheduling. We also propose a symmetric aging-enabled routing scheme (SAR) to balance the core and NoC routing within a tile which prevents the cores from being disconnected from the NoC due to extremely aged routers. Compared to the state-of-the-art prior work, *ARTEMIS* enables execution of 25% more applications over the 3D CMP's lifetime without sacrificing application throughput.

Our next contribution is *CHARM,* a checkpointing based runtime resource management framework that improves the application execution rate on CMPs in the presence of soft errors with a target lifetime. The underlying mechanisms for minimizing soft error rate and maximizing CMP lifetime are counter-productive to each other, making it difficult to find a joint solution. In *CHARM,* we have designed a heuristic that leverages the *application slack time* and utilizes the V_{dd} and DoP parameters to intelligently trade-off between CMP aging with the checkpointing-and-rollback overhead of soft errors for applications with relaxed deadlines. Using *CHARM,* CMPs can meet up to 6× higher number of application deadlines in a target CMP lifetime compared to the state-of-the-art frameworks that are designed either to optimize for soft error reliability or lifetime enhancement of CMPs.

Next, we have proposed *PARM,* a power supply noise (PSN) aware resource management framework that employs a novel heuristic to minimize the adverse effect of voltage emergencies on assign application DoP, V_{dd}, and mapping region for every application that arrives in the runtime to minimize the peak power supply noise observed in the CMP that is operating in the *near threshold region*. The application mapping heuristic minimizes the variation in switching activity between different on-chip components, which is the root cause of PSN in manycore CMPs.

Unlike prior works that considered NoC and core switching activity separately in their optimization methods, the resource management heuristic in *PARM* jointly balances the switching activity in cores and NoC elements to reduce the overall switching activity of tiles that are grouped into voltage islands. This PSN aware application mapping, and NoC routing minimizes the average PSN observed in *near threshold* CMPs by up to 4.5× and improves the application performance by up to 38% when the CMP is oversubscribed, compared to the state-of-the-art.

We then propose architectural innovations for resolving memory bottleneck problem in manycore GPGPU accelerators. We proposed *RAPID,* a latency optimized NoC for GPGPU accelerators. In *RAPID*, the NoC is customized for many-to-few and few-to-many traffic patterns in manycore GPGPUs. In the request plane, we propose a low power router that uses fewer buffers than traditional NoC router. In the reply plane, we propose a fast overlay circuit mechanism that transmits packets from MC to cores within 3 cycles. This reduces the wait time to read reply data in MCs which is one of the major causes of the memory bottleneck issue. We also propose an enhanced MC architecture that prioritizes sending burst packets that are generated as a result of burst requests from adjacent memory locations. Our experimental results show that by using the enhanced MC architecture in tandem with the *RAPID* NoC architecture, 64-core and 144-core platforms improve their NoC latency by up to 4-10× and application execution time by up to 67% while saving up to 4× NoC energy compared to the baseline NoC and MC architectures.

Our next contribution is a data-aware approximate NoC architecture called *DAPPER*. We also propose an approximate memory controller architecture called *AMC* to jointly maximize the DRAM utilization and minimize the memory bottleneck issue using the approximate computing paradigm. Our *DAPPER+AMC* architecture uses intelligent memory request scheduling in the MC to increase the throughput of DRAM requests serviced and then coalesces the approximable data

waiting in the *AMC* output queue to minimize the NoC traffic in the reply plane. Also, we advocate for a fast overlay circuit to transmit the reply packets back to the cores in less than 3 cycles. We utilize a global overlay manager (GOM) and an ACK-based flow control mechanism to ensure the overlay circuits are shared fairly between different *AMC*s. Experimental analysis shows that *DAPPER+AMC* increases the NoC throughput by up to 21% in 16-core platforms and up to 7% in 64-core platforms while consuming 38% less energy compared to the baseline NoC and around 2.5% error in application execution time.

Lastly, we propose lightweight modules called *SIM* and *THANOS* to mitigate snooping attacks and detect the source of such attacks in 3PIP NoCs with hardware Trojans embedded in their network interfaces (NIs). *SIM* uses a key-based duplicate packet detection technique in the network interface that generates a key in trusted cores and validates it in the untrusted NIs. *SIM* consumes only 5.5% power, 2.15% area, and 1 cycle latency overhead. *THANOS* utilizes threshold degradation of stress transistors located between cores and NIs to detect the malicious application which is receiving the duplicate packets generated by hardware Trojans embedded in 3PIP NoC components. *THANOS* is hard to tamper with and is orthogonal to various other attack mitigation mechanisms that prevent different types of hardware Trojan attacks initialized from any of the NoC components. Using *SIM+THANOS,* the overall application execution time is reduced by up to 62.9% while consuming up to 48.3% less energy compared to the state-of-the-art snooping attack mitigation mechanisms.

8.2. SUGGESTIONS FOR FUTURE WORK

With the rapid growth and advancements in CMOS fabrication and integration methodologies, CMPs are embracing a higher number of cores that execute at higher clock speeds to increase their compute performance. However, a slower adaptation of memory bandwidth to CMP clock speeds is a significant bottleneck to realizing the full potential of CMPs. Hence, CMP designers are working towards reducing the distance between data and compute elements using several techniques such as near-memory computing, 3D integration of memory, and phase change material (PCM) based main memory that can store a higher volume of data with lower access latencies. With rapid integration methodologies, CMPs with integrated memory are going to face renewed threats of reliability and security. Hence, we envision the following as likely directions of our future work:

- *Reliability of CMPs with emerging memory technologies*: CMPs that integrate PCM based memory in their design are bound to face reliability challenges due to the rapid wear-out phenomenon that is common in phase change materials during write cycles. Hence, to increase the lifetime of a CMP, it is also crucial to minimize the wear out of the PCM based memory used in the CMPs using application mapping heuristics along with data mapping techniques that minimize the wear out in PCM based memory DIMMs.
- *Security enhancement in 3D CMPs with integrated memory:* The threat of data loss of sensitive application data is very high in CMPs that integrate processing logic with memory banks in 3D layers. Without robust security enforcing mechanism, the shared resources such as TSVs can be used by malicious software tasks to corrupt the data traversing from memory to cores, or snoop the sensitive information such as encryption/decryption keys. There is a need for

security enhancement in 3D ICs with integrated memory to efficiently reap other benefits of performance and power consumption with 3D integration.

- *Application-specific CMP reliability:* With the advent of AI and machine learning, designers are optimizing CMPs to meet the performance requirements of the emerging applications. These applications might not require a similar level of reliability as the general-purpose CMPs. In such scenarios, there is a need to study the opportunities provided by emerging applications to improve CMP and memory reliability by leveraging the inherent fault tolerance of these applications.
- *Resiliency to soft errors and voltage emergencies in TPUs:* Google's tensor processing units (TPUs) are the accelerators that operate 15× faster than general-purpose GPUs on matrix multiplication operations. However, the rapid growth in deep neural network (DNN) workloads, increase the energy consumption of these processors. Hence, the TPUs are designed to run at a very low supply voltage (V_{dd}) called *near-threshold voltage.* In such configuration, the control logic, and memory elements are prone to higher single event upsets (SEUs) due to alpha particle strikes. Together with that, timing errors caused by power supply noise (PSN) also reduces the performance and reliability of the TPUs. A cross-layer solution that comprises of an application mapping and task migration mechanisms utilizing runtime feedback from on-chip load monitoring system can complement the existing hardware techniques designed to combat soft errors in TPUs, without further increasing the overhead of the systolic arrays.
- *A holistic framework for runtime approximation selection:* The approximate computing paradigm has demonstrated that trading off accuracy in the applications can be converted to savings in energy without losing CMP performance. However, several frameworks that are designed to leverage applications' tolerance for inaccuracy are configured at design time to

utilize a constant rate of approximation throughout the application execution. However, the approximability of each application varies with time. Hence, there is a need for a runtime framework that encapsulates the variability of applications' tolerance for inaccuracy using a feedback mechanism. This framework should also comprise compiler and OS-level knobs along with hardware modifications to accommodate the runtime selection of the rate of approximation.

- *Network on chip for neuromorphic computing*: Spiking neural networks (SNNs) have been recognized as efficient computing models for spatio-temporal pattern recognition on resource and power constrained platforms. SNNs, also called as neuromorphic processors comprise of crossbars with input-output neurons with fully connected synapses. Several bus-based and time-multiplexed interconnect architectures have been proposed for connecting the communication spike between crossbar synapses. However, due to the sharing of interconnect between different types of synapses, the NoC becomes the bottleneck for SNNs with a large number of neurons. Hence, there is a need to design interconnects that suit the SNN spike traffic pattern and mitigate the performance bottleneck in SNNs with a large number of neurons.

BIBLIOGRAPHY

[1] M. Bohr, "Moores law leadership," 2017. [Online]. Available: https://newsroom.intel.com/newsroom/wp-content/uploads/sites/11/2017/03/Mark-Bohr-2017-Moores-Law.pdf.

[2] S. Borkar, "Thousand core chips: a technology perspective," in *IEEE/ACM Design Automation Conference*, 2007.

[3] "IBM truenorth," IBM computers, [Online]. Available: http://www.research.ibm.com/articles/brain-chip.shtml.

[4] "CUDA SDK toolkit," [Online]. Available: https://developer.nvidia.com/cudadownloads.

[5] "OpenCL Toolkit," OpenCL, [Online]. Available: https://www.khronos.org/opencl/.

[6] "OpenMP," OpenMP, [Online]. Available: https://www.openmp.org/.

[7] "IBM TrueNorth, Brain Inspired Chips," IBM , 2016. [Online]. Available: http://www.research.ibm.com/articles/brain-chip.shtml.

[8] J. Mottin, M. Cartron, and G. Urlini, The STHORM Platform, Springer, 2007.

[9] Y. Hoskote, S. Vangal, A. Singh, N. Borkar, and S. Borkar, "A 5-GHz Mesh Interconnect for a Teraflops Processor," in *IEEE Micro*, 2007.

[10] H. Kim, J. Kim, W. Seo, Y. Cho, and S. Ryu, "Providing cost-effective on-chip network bandwidth in GPGPUs," in *IEEE International Conference on Computer Design*, 2012.

[11] V. B. Kleeberger, M. Barke, C. Werner, D. Schmitt-Landsiedel, and U. Schlichtmann, "A compact model for NBTI degradation and recovery under use-profile variations and

its application to aging analysis of digital integrated circuits," *Microelectronics Reliability,* vol. 54, no. 6, pp. 1083-1089, 2014.

[12] S. Sapatnekar, "What happens when the circuits grow old: Aging issues in CMOS design," in *IEEE International Symposium on VLSI Technology, Systems and Application* , 2013.

[13] B. Amelifard and M. Pedram, "Optimal design of the power delivery networks for multiple voltage-island system-on-chips," *IEEE Transactions on Computer Aided Design and Integrations of Circuits and Systems,* vol. 28, no. 6, pp. 888-900, 2009.

[14] H. Kim, A. Vitkovskiy, P. V. Gratz, and V. Soteriou, "Use It Or Lose It: Wear-out and Lifetime in Future Chip Multiprocessors," in *IEEE/ACM Symposium on Microarchitecture*, 2013.

[15] X. Huang, Tan Yu, Valeriy Sukharev, and Sheldon X-D. Tan, "Physics based electromigration assessment for power grid networks," in *IEEE/ACM Design Automation Conference*, 2014.

[16] T. Okumura, F. Minami, K. Shimazaki, K. Kuwada, and M. Hashimoto, "Gate delay estimation in STA under dynamic power supply noise," in *IEEE Asia and South Pacific Design Automation Conference*, 2010.

[17] B. Geden, "Understand and avoid electromigration (em) and ir-drop in custom ip blocks," 2011. [Online]. Available: http://www.synopsys.com/Tools/Verification/CapsuleModule/CustomSim-RA-wp.pdf.

[18] S.I. Abe, Y. Watanabe , N. Shibano, N. Sano, H. Furuta, M. Tsutsui, T. Uemura, and T. Arakawa, "Neutron-induced soft error analysis in MOSFETs from a 65 to a 25 nm

design rule using multi-scale monte-carlo simulation method," in *IEEE International Reliability Physics Symposium*, 2012.

[19] C.H. Chen, P. Knag, and Z. Zhang, "Characterization of Heavy-Ion-Induced Single-Event Effects in 65nm Bulk CMOS ASIC Test Chips," *IEEE Transactions on Nuclear science,* vol. 61, no. 5, pp. 2694-2702, 2014.

[20] A. H. Johnston, "The Effect of Device Scaling on Single-Event Effects in Advanced CMOS Devices," California Institute of Technology, 2005.

[21] Bailey Steven, and Borivoje Nikolic, "Modeling Radiation-Induced Soft Errors in Logic and the Overhead of Resiliency Techniques," university of berkeley, 2014.

[22] M. Gupta, Jarod L. Oatley, Russ Joseph, Gu-Yeon Wei, and David M. Brooks, "Understanding voltage variations in chip multiprocessors," in *IEEE Design, Automation and Test in Europe* , 2007.

[23] Y. Cheng, Aida Todri-Sanial, Alberto Bosio, Luigi Dilillo, Patrick Girard, and Arnaud Virazel, "Power supply noise-aware workload assignments for," in *IEEE Asia and South Pacific Design Automation Conference*, 2014.

[24] Rahmani, Amir, The Dark Side of Silicon, Springer, 2016.

[25] A. Bhakoda, J. Kim, and T.M. Aamodt, "Throughput-Effective On-Chip Networks for Manycore Accelerators," in *IEEE International Symposium on Microarchitecture*, 2010.

[26] H. Jang, J Kim, P Gratz, KH Yum, and EJ Kim, "Bandwidth-Efficient On-Chip Interconnect Designs for GPGPUs," in *IEEE/ACM Design Automation Conference*, 2015.

[27] "Arteris," Arteris - IP the NoC interconnect company , [Online]. Available: www.arteris.com.

[28] D. Ancajas, K Chakraborty, and S Roy, "Fort-nocs: Mitigating the threat of a compromised noc," in *ACM Design Automation Conference*, 2014.

[29] Stephen Chen, "Could Huawei be Using Hardware Trojan Circuits," Yahoo, 2019. [Online]. Available: https://finance.yahoo.com/news/could-huawei-using-trojan-circuits-093000480.html.

[30] M. Blaze, J. Ioannidis, and A. Keromytis, "DSA and RSA key and signature encoding for the KeyNote trust management system," 2000.

[31] V. Y. Raparti, N. Kapadia, and S. Pasricha, "ARTEMIS: An Aging-Aware Runtime Application Mapping Framework for 3D NoC-based Chip Multiprocessors," *IEEE Transaction on Multi Scale Computing Systems,* vol. 3, no. 2, pp. 72-85, 2017.

[32] V.Y. Raparti and S. Pasricha, "CHARM: A checkpoint-based resource management framework for reliable multicore computing in the dark silicon era," in *IEEE 34th International Conference on Computer Design* , 2018.

[33] V.Y. Raparti and S. Pasricha, "PARM: power supply noise aware resource management for NoC based multicore systems in the dark silicon era," in *IEEE/ACM/ESDA Design Automation Conference*, 2018.

[34] V.Y. Raparti and S. Pasricha, "RAPID: Memory-Aware NoC for latency Optimized GPGPU Architectures," *IEEE Transaction on Multi Scale Computing Systems,* vol. 4, no. 4, pp. 874-887, 2017.

[35] V.Y. Raparti, S. Pasricha , "Memory-aware circuit overlay NoCs for latency optimized GPGPU architectures," in *IEEE International Symposium on Quality Electronic Design*, 2016.

[36] N. Kapadia, V.Y. Raparti, and S. Pasricha, "ARTEMIS: An Aging-Aware Runtime Application Mapping Framework for 3D NoC-based Chip Multiprocessors," in *ACM International Symposium on Networks-on-Chip*, 2015.

[37] V.Y. Raparti and S. Pasricha, " DAPPER: data aware approximate NoC for GPGPU architectures," in *IEEE/ACM International Symposium on Networks-on-Chip*, 2018.

[38] V. Y. Raparti and S. Pasricha, "Lightweight Mitigation of Hardware Trojan Attacks in NoC-based Manycore Computing," in *IEEE/ACM Design Automation Conference* , 2019.

[39] V.Y. Raparti, S. Pasricha, "Approximate NoC and Memory Controller Architectures for," *IEEE Transactions on Parallel and Distributed Systems (under review),* 2019.

[40] Jeffrey Hicks, Daniel Bergstrom, Mike Hattendorf, Jason Jopling, Jose Maiz, Sangwoo Pae, Chetan Prasad, and Jami Wiedemer, "Intel's 45 nm CMOS technology," *Intel techology journal,* vol. 12, no. 2, pp. 131-144, 2008.

[41] E. Mintarno, Joëlle Skaf, Rui Zheng, Jyothi Bhaskar Velamala, Yu Cao, Stephen Boyd, Robert W. Dutton, and Subhasish Mitra, "Self-tuning for maximized lifetime energy-efficiency in the presence of circuit aging," *IEEE Transactions on Computer Aided Design,* vol. 30, no. 5, pp. 760-773, 2011.

[42] N. H. Khan, S. M. Alam, and S. Hassoun, "System-level comparison of power delivery design for 2D and 3D ICs," in *IEEE international conference on 3D system integration*, 2009.

[43] A. Tiwari, J. Torrellas, "Facelift: Hiding and slowing down aging in multicores," in *IEEE/ACM International Symposium on Microarchitecture*, 2008.

[44] W. Chan, A. B. Kahng, and S. Nath, "Methodology for electromigration signoff in the presence of adaptive voltage scaling," in *IEEE International workshop on system level internconnect prediction*, 2014.

[45] J. Lee and N. S. Kim, "Optimizing total power of many-core processors considering voltage scaling limit and process variations," in *international symposium on low power electronics*, 2009.

[46] S. Borkar, "Design perspectives on 22nm CMOS and beyond," in *IEEE Design Automation Conference*, 2009.

[47] J. Allred, S. Roy, and K. Chakraborty, "Designing for dark silicon: A methodological perspective on energy efficient systems," in *IEEE International Symposium on Low Power Electronics and Design* , 2012.

[48] B. Raghunathan, Y. Turakhia, S. Garg, and D. Marculescu, "Cherry-picking: Exploiting process variations in dark-silicon homogeneous chip multi-processors," in *IEEE Design Automation and Test in Europe*, 2013.

[49] S. Feng, S. Gupta, A. Ansari, and S. Mahlke, "Maestro: Orchestrating lifetime reliability in chip multiprocessors," in *IEEE International conference on high performance embedded architectures and compilers*, 2010.

[50] F. Paterna, A. Acquaviva, and L. Benini, "Aging-aware energy-efficient workload allocation for mobile multimedia platforms," *IEEE Transactions on Parallel Distributed Systems,* vol. 24, no. 8, pp. 1489-1499, 2013.

[51] P. Mercati, A. Bartolini, F. Paterna, T. S. Rosing, and L. Benini, "Workload and user experience-aware dynamic reliability management in multicore processors," in *IEEE Design Automation Conference*, 2013.

[52] M. H. Haghbayan, A. Miele, A. M. Rahmani, P. Liljeberg, and H. Tenhunen, "A lifetime-aware runtime mapping approach for many-core systems in the dark silicon era," in *IEEE Design Automation and Test in Europe*, 2016.

[53] D. Gnad, M. Shafique, F. Kriebel, S. Rehman, D. Sun, and J. Henkel, "Hayat: Harnessing dark silicon and variability for aging deceleration and balancing," in *IEEE Design Automation Conference*, 2015.

[54] A. K. Singh, M. Shafique, A. Kumar, and J. Henkel, "Analysis and mapping for thermal and energy efficiency of 3-D video processing on 3-D multicore processors," *IEEE Transactions on very large scale integrated circuits,* vol. 24, no. 8, pp. 2745-2758, 2016.

[55] S. Pasricha and Y. Zou, "A low overhead fault tolerant routing scheme for 3D Networks-on-Chip," in *IEEE International Symposium on Qualitative Electronic Design*, 2011.

[56] S. Pasricha and Y. Zou, "NS-FTR: A fault tolerant routing scheme for networks on chip with permanent and runtime intermittent faults," in *IEEE/ACM Asia and South Pacific Design Automation Conference*, 2011.

[57] S. Rehman, F. Kriebel, D. Sun, M. Shafique, and J. Henkel, "dTune: Leveraging reliable code generation for adaptive dependability tuning under process variation and aging-induced effects," in *IEEE/ACM Design Automation Conference*, 2014.

[58] A. Das, A. Kumar, and B. Veeravalli, "Reliability and energy-aware mapping and scheduling of multimedia applications on multiprocessor systems," *IEEE Transactions on Parallel and Distributed Systems,* vol. 27, no. 3, pp. 869-884, 2016.

[59] A. Das, A. K. Singh, and A. Kumar, "Execution trace–driven energy-reliability optimization for multimedia MPSoCs," *ACM. Transactions on Reconfigurable Systems,* vol. 8, no. 3, 2015.

[60] S.S. Sahoo, A. Kumar, and B. Veeravalli, "Design and evaluation of reliability-oriented task re-mapping in MPSoCs using time-series analysis of intermittent faults," in *IEEE Design Automation and Test in Europe*, 2016.

[61] K. Bhardwaj, K. Chakraborty, and S. Roy, "Towards graceful aging degradation in NoCs through an adaptive routing algorithm," in *IEEE Design, Automation and Test in Europe* , 2012.

[62] S. R. Sarangi, B. Greskamp, R. Teodorescu, J. Nakano, A. Tiwari, and J. Torrellas, "VARIUS: A model of process variation and resulting timing errors for microarchitects," *IEEE Trans. Semiconductor Manufacturing,* vol. 21, no. 1, pp. 3-13, 2008.

[63] J. B. Velamala, K. Sutaria, H. Shimizu, H. Awano, T. Sato, and Y. Cao, "Statistical aging under dynamic voltage scaling: A logarithmic model approach," in *IEEE Custom integrated circuits Conference*, 2012.

[64] J B Velamala, K Sutaria, T Sato, and Y Cao, "Physics matters: statistical aging prediction under trapping/detrapping," in *IEEE Design Automation Conference*, 2012.

[65] J. B. Velamala, K.B. Sutaria, H. Shimizu, H. Awano, T. Sato, G. Wirth, and Y. Cao,, "Compact modeling of statistical BTI under trapping/detrapping," *IEEE Transactions on Electronic Devices,* vol. 60, no. 11, pp. 3645-3654, 2013.

[66] V. Mishra, and S. S. Sapatnekar, "The impact of electromigration in copper interconnects on power grid integrity," in *IEEE/ACM Design Automation Conference*, 2013.

[67] M. Fattah, M. Daneshtalab, P. Liljeberg, and J. Plosila, "Smart hill climbing for agile dynamic mapping in many-core systems," in *IEEE/ACM Design Automation Conference*, 2013.

[68] S. Dighe, S.R. Vangal, P. Aseron, S. Kumar, T. Jacob, K.A. Bowman, J. Howard, J. Tschanz, V. Erraguntla, N. Borkar, and V.K. De, "Within-die variation-aware dynamic-voltage-frequency-scaling with optimal core allocation and thread hopping for the 80-core teraflops processor," *IEEE Journal of solid-state circuits,* vol. 46, no. 1, pp. 184-193, 2011.

[69] P. Singh, E. Karl ,D. Sylvester, and D. Blaauw, "Dynamic nbti management using a 45 nm multi-degradation sensor," *IEEE Transactions of Circuits and Systems Integration,* vol. 58, no. 9, pp. 2026-2037, 2011.

[70] A. Sassone, M. Petricca, M. Poncino, and E. Macii, "A fully standard-cell delay measurement circuit for timing variability detection," in *International workshop on Power Timing Model*, 2013.

[71] S.W. Chen, M.H. Chang, W.C. Hsieh, and W. Hwang, "Fully on-chip temperature process and voltage sensors," in *IEEE International symposium on circuits and systems*, 2010.

[72] P. Singh, E. Karl, and D. Blaauw, "Compact degradation sensors for monitoring NBTI and Oxide degradation," *IEEE Transactions on VLSI,* vol. 20, no. 9, pp. 1645-1655, 2012.

[73] M. Arjomand, and H. Sarbazi-Azad, "Voltage-frequency planning for thermal-aware low-power design of regular 3-D NoCs," in *IEEE International Conference on VLSI Desig*, 2010.

[74] W. Jang, D. Ding, and D. Z. Pan, "A voltage-frequency island aware energy optimization framework for networks-on-chip," in *IEEE/ACM International Conference on Computer Aided Design*, 2008.

[75] "3D-ICE open source tool," [Online]. Available: http://esl.epfl.ch/ 3d-ice.html.

[76] "LP solve 5.5.2.0," [Online]. Available: http://lpsolve.sourceforge. net/5.5/ .

[77] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The SPLASH-2 programs: Characterization and methodological considerations," *ACM Special Interest Group on computer architecture news,* vol. 23, no. 2, pp. 24-36, 1995.

[78] C. Bienia, S. Kumar, J. P. Singh, and K. Li, "The PARSEC benchmark suite: Characterization and architectural implications," in *ACM International Conference on Parallel Architectures and Compilation Techniques*, 2008.

[79] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: Exploring the level of abstraction for scalable and accurate parallel multicore simulation," in *IEEE International Symposium on High-Performance Computer Architecture* , 2011.

[80] S. Li, J. H. Ahn, R. D. Strong, J. B. Brockman, D. M. Tullsen, and N. P. Jouppi, "McPAT: An integrated power, area, and timing modeling framework for multicore and manycore architectures," in *IEEE/ACM Symposium on Microarchitecture*, 2009.

[81] N. H. Khan, S. M. Alam, and S. Hassoun, "Power delivery design for 3-D ICs using different through-silicon via (TSV) technologies," *IEEE Transactions on VLSI Systems,* vol. 19, no. 4, pp. 647-658, 2011.

[82] A. Kaouache, F. Wrobel, F. Saigne, A. D. Touboul, and R. D. Schrimpf. , "Analytical method to evaluate soft error rate due to alpha contamination," *IEEE Transactions on Nuclear Science,* vol. 60, no. 6, 2013.

[83] N. Kapadia, S. Pasricha, "VARSHA: Variation and reliability-aware application scheduling with adaptive parallelism in the dark-silicon era," in *ACM/IEEE Design, Automation and Test in Europe* , 2015.

[84] S. Hari, M. Li, P. Ramachandran, B. Choi, and S. V. Adve, "mSWAT: Low-Cost Hardware Fault Detection and Diagnosis for Multicore Systems," in *IEEE MICRO*, 2009.

[85] A. Das, A. Kumar, B. Veeravalli, C. Bolchini, and A. Miele, "Combined DVFS and mapping exploration for lifetime and soft-error susceptibility improvement in MPSoCs," in *ACM/IEEE Desgin Automation and Test in Europe*, 2014.

[86] M. Haque, H. Aydin, D. Zhu, "Energy-aware task replication to manage realiability for periodic real-time applications on multicore platforms," in *IEEE International Green Computing Conference* , 2013.

[87] Y. Guo, D. Zhu, H. Aydin, "Generalized standby-sparing techniques for energy-efficient fault tolerance in multiprocessor real-time systems," in *IEEE International Conference on Embedded and Real-Time Computing Systems and Applications*, 2013.

[88] A. Bravaix, C. Guerin, V. Huard, D. Roy, J. M. Roux, and E. Vincent, "Hot-carrier acceleration factors for low power management in DC-AC stressed 40nm NMOS node at high temperature," in *Reliability Physics Symposium*, 2009.

[89] D. Ancajas, James McCabe Nickerson, Koushik Chakraborty, and Sanghamitra Roy, "HCI-tolerant NoC router microarchitecture," in *IEEE/ACM Desing Automation Conference*, 2013.

[90] D. Zhu, R Melhem, D Mossé, "The effects of energy management on reliability in realtime embedded systems," in *IEEE International Conference On Computer Aided Design*, 2004.

[91] Q. Han, Ming Fan, Linwei Niu, and Gang Quan, "Energy minimization for fault tolerant scheduling of periodic fixed-priority applications on multiprocessor platforms," in IEEE Design Automation and Test in Europe, 2015.

[92] G. M. Chiu , "The odd-even turn model for adaptive routing," *IEEE Transactions on Parallel and Distributed Systems,* vol. 11, no. 7, pp. 729-738, 2000.

[93] S.V. Woo, Moriyoshi Ohara, Evan Torrie, Jaswinder Pal Singh, and Anoop Gupta, "The SPLASH-2 programs: characterization and methodological characterization," in *IEEE International Symposium on Computer Architecture*, 1995.

[94] L. Sheng, Jung Ho Ahn, Richard D. Strong, Jay B. Brockman, Dean M. Tullsen, and Norman P. Jouppi. , "McPAT: An integrated power area and timing modeling framework for multicore and manycore architectures," in *IEEE Symposium on Microarchitecture*, 2009.

[95] Michael D. Powell , and T. N. Vijaykumar, "Pipeline damping: a microarchitectural technique to reduce inductive noise in supply voltage," in *ACM International Symposium on Computer Architecture*, 2003.

[96] Timothy N. Miller , Renji Thomas , Xiang Pan , and Radu Teodorescu, "VRSync: Characterizing and eliminating synchronization -induced voltage emergencies in many-core processors," in *ACM Special Interest Group on computer architecture*, 2012.

[97] A. Paul, Matt Amrein, Saket Gupta, Arvind Vinod, Abhishek Arun, Sachin Sapatnekar, and Chris H. Kim, "Staggered Core Activation: A Circuit/architectural approach for mitigating resonant supply noise issues in multi-core multi-power domain processors," in *IEEE Custom Integrated Circuits Conference*, 2012.

[98] V. J. Reddi, Simone Campanoni, Meeta S. Gupta, Michael D. Smith, Gu-Yeon Wei, David Brooks, and Kim Hazelwood, "Eliminating voltage emergencies via software-guided code transformations," in *ACM Transactions on Architecture and Code Optimization* , 2012.

[99] M.S. Gupta, V.J. Reddi, G. Holloway, G.Y Wei, and D.M. Brooks, "An event-guided approach to reducing voltage noise in processors," in *ACM Design Automation and Test in Europe*, 2009.

[100] M. Healey, F. Mohamood, H.H.S Lee, and S.K. Lim, "A unified methodology for power supply noise reduction in modern microarchitecture design," in *IEEE Asia and South Pacific Design Automation Conference*, 2008.

[101] N. James, Phillip Restle, Joshua Friedrich, Bill Huott, and Bradley McCredie, "Comparison of split-versus connected-core supplies in the power6 microprocessor," in *International Solid-State Circuits Conference*, 2007.

[102] J. Gu, Hanyong Eom, and Chris H. Kim, "On-chip supply noise regulation using a low-power digital switched decoupling capacitor circuit," in *IEEE Journal of Solid-State Circuits*, 2009.

[103] Y. Chen, M. Shintani, T. Sato, Y. Shi, and S.C Chang, "Pattern based runtime voltage emergency prediction: An instruction-aware block sparse compressed sensing approach," in *IEEE Asia South Pacific Design Automation Conference*, 2017.

[104] V. J. Reddi, Meeta S. Gupta, Glenn Holloway, Gu-Yeon Wei, Michael D. Smith, and David Brooks, "Voltage emergency prediction: Using signatures to reduce operating margins," in *IEEE International Symposium on High-Performance Computer Architecture* , 2009.

[105] L. Zheng, Y Zhang, and MS Bakir, "Full-chip power supply noise time-domain numerical modeling and analysis for single and stacked ICs," *IEEE Transactions on Electron Devices,* vol. 63, no. 3, pp. 1225-1231, 2016.

[106] M. Sadi and M. Tehranipoor, "Design of a network of digital sensor macros for extracting power supply noise profile in SoCs," *IEEE Transactions on Very Large Scale Intergration Systems,* vol. 24, no. 5, 2016.

[107] X. Hu, G Yan, Y Hu, and X Li, "Orchestrator: a low-cost solution to reduce voltage emergencies for multi-threaded application," in *IEEE Design, Automation and Test in Europe*, 2013.

[108] N. Kapadia and S. Pasricha, "VISION: A Framework for Voltage Island Aware Synthesis of Interconnection Networks-on-Chip," in *ACM Great Lakes Symposium on Very Large Scale Integrated circuits*, 2011.

[109] N. Dahir, T Mak, F Xia, and A Yakovlev, "Minimizing power supply noise through harmonic mappings in networks-on-chip," in *IEEE/ACM International Conference on Hardware/Software Codesign*, 2012.

[110] P. Basu, R.J Shridevi, K. Chakraborty, and S. Roy, "IcoNoClast: Tackling Voltage Noise in` the NoC Power Supply Through Flow-Control and Routing Algorithms," *IEEE Transactions on Very Large Scale Integration Systems,* vol. 25, no. 7, pp. 1-10, 2017.

[111] A. Todri, Malgorzata Marek-Sadowska, and Joseph Kozhaya, "Power supply noise aware workload assignment for multi- core systems," in *IEEE/ACM International Conference On Computer Aided Design*, 2008.

[112] Y. Xiang and S. Pasricha, "Soft and Hard Reliability-Aware Scheduling for Multicore Embedded Systems with Energy Harvesting," *IEEE Transactions on Multi Scale Computing Systems,* vol. 1, no. 4, pp. 220-235, 2015.

[113] J. C. Glass and LM Ni, "The turn model for adaptive routing," in *ACM Special Interest Group on computer architecture*, 1992.

[114] A. Bejestan, Y. Wang, S. Ramadurgam, Y. Xue, P. Bogdan, and M Pedram, "Analyzing the dark silicon phenomenon in a many-core chip multi-processor under deeply-scaled process technologies," in *ACM Great Lakes Symposium on VLSI*, 2015.

[115] N. Binkert, Bradford Beckmann, Gabriel Black, Steven K. Reinhardt, Ali Saidi, Arkaprava Basu, and Joel Hestness, "The gem5 simulator," in *ACM Special Interest Group on computer architecture*, 2011.

[116] "International Tech Roadmap for Semiconductors," [Online]. Available: http://www.itrs2.net.

[117] R. Wu, B. Zhang, and M. Hsu, "Clustering billions of data points Using GPGPUs," in *ACM Combined Workshops Un-Conventional High-Perform. Comput. Workshop Plus Memory Access Workshop*, 2009.

[118] M. Imani, D. Peroni, Y. Kim, A. Rahimi, and T. Rosing, "Efficient neural network acceleration on gpgpu using content addressable memory," in *IEEE Design Automation and Test in Europe*, 2017.

[119] S. Kato, E. Takeuchi, Y. Ishiguro, Y. Ninomiya, K. Takeda, and T. Hamada, "An open approach to autonomous vehicles," in *IEEE MICRO*.

[120] Y. Yang and H. Zhou, "CUDA-NP: Realizing nested thread-level parallelism in GPGPU applications," *ACM Special Interest Group on Programming Languages Notices,* vol. 49, no. 8, pp. 93-106, 2014.

[121] Y. Yu, W. Xiao, X. He, H. Guo, Y. Wang, X. Chen, "A stall aware warp scheduling for dynamically optimizing thread-level parallelism in GPGPUs," in *ACM International Conference on Supercomputing*, 2015.

[122] O. Kayiran, A. Jog, M. T. Kandemir, and C. R. Das, "Neither more nor less: Optimizing thread-level parallelism for GPGPUs," in *IEEE International Conference on Parallel Architectures and Compilation*, 2013.

[123] J. Cong, M. Gill, Y. Hao, G. Reinman, and B. Yuan, "On-chip interconnection network for accelerator-rich architectures," in *ACM Design Automation Conference*, 2015.

[124] C. Chen, S. Park, T. Krishna, S. Subramanian, A. P. Chandrakasan, and L. S. Peh, "SMART: A single-cycle reconfigurable NoC for SoC applications," in *IEEE/ACM Design, Automation and Test in Europe* , 2013.

[125] X. Zhao, S. Ma, Y. Liu, L. Eeckhout, and Z. Wang, "A low-cost conflict-free NoC for GPGPUs," in *ACM Design Automation Conference*, 2016.

[126] T. Krishna, A. Kumar, P. Chiang, M. Erez, and L. S. Peh, "NoC with near-ideal express virtual channels using global-line communication," in *IEEE High Performance Interconnects*, 2008.

[127] T. Krishna, C. H. O. Chen, W. C. Kwon, and L. S. Peh, "Breaking the on-chip latency barrier using SMART," in *IEEE International Symposium on High-Performance Computer Architecture*, 2013.

[128] A. Abousamra, A. Jones, and R. Melhem, "Proactive circuit allocation in multiplane NoCs," in *ACM Design Automation Conference*, 2013.

[129] A. Abousamra, R. Melhem, and A. Jones, "Deja vu switching for multiplane nocs," in *IEEE/ACM International Symposium on Network on Chips*, 2012.

[130] A. Abousamra, R. Melhem, and A. Jones, "Winning with pinning in NoCs," in *IEEE High Performance Interconnects*, 2009.

[131] T. Pimpalkhute and S. Pasricha, "NoC scheduling for improved application-aware and memory-aware transfers in multi-core systems," in *IEEE International Conference on VLSI Design*, 2014.

[132] T. Pimpalkhute and S. Pasricha, "An application-aware heterogeneous prioritization framework for NoC based chip multiprocessors," in *IEEE International symposium on Quality Electronic Design*, 2014.

[133] W. Choi, K. Duraisamy, R. G. Kim, J. R. Doppa, P. P. Pande, R. Marculescu, and D. Marculescu, "Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms," in *IEEE Compilers and Architecture Synthesis of Embedded Systems*, 2016.

[134] A. Jog, O. Kayiran, A. K. Mishra, M. T. Kandemir, O. Mutlu, R. Iyer, and C. R. Das, "Orchestrated scheduling and prefetching for GPGPUs," *ACM Special Interest Group on computer architecture,* vol. 41, no. 3, pp. 332-343, 2013.

[135] Y. Oh, K. Kim, M. K. Yoon, J. H. Park, Y. Park, W. W. Ro, and M. Annavaram, "APRES: Improving cache efficiency by exploiting load characteristics on GPUs," *ACM Special Interest Group on Computer Architecture,* vol. 44, no. 3, pp. 191-203, 2016.

[136] A. Li, G. J. Van den Braak, A. Kumar, and H. Corporaal, "Adaptive and transparent cache bypassing for GPUs," in *ACM International Symposium on High-Performance Computer Architecture*, 2015.

[137] E. Bolotin, Z. Guz, I. Cidon, R. Ginosar, and A. Kolodny, "The power of priority: NoC based distributed cache coherency," in *IEEE NOCS*, 2007.

[138] A. Heinecke, M. Klemm, and H. J. Bungartz, "From gpgpu to many-core: Nvidia fermi and intel many integrated core architecture," *Computer Science Engineering,* vol. 14, no. 2, pp. 78-83, 2012.

[139] A. Bakhoda, G. L. Yuan, W. W. Fung, H. Wong, and T. M. Aamodt, "Analyzing CUDA workloads using a detailed GPU simulator," in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2009.

[140] V. Catania, A. Mineo, S. Monteleone, M. Palesi, and D. Patti, "Noxim: An open extensible and cycle-accurate network on chip simulator," in *IEEE Application-specific Systems, Architectures and Processors*, 2015.

[141] R. M. Li, C. T. King, and B. Das, "Extending Gem5-garnet for efficient and accurate trace-driven NoC simulation," in *ACM International Workshop on Network on Chip Architecture*, 2016.

[142] "Intel Packaging Databooks," [Online]. Available: https://ww.intel.com/content/dam/www/public/us/en/documents/packaging-databooks/packaging-chapter-14-databook.pdf.

[143] C. Sun, C.H.O. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.S. Peh, and V. Stojanovic, "DSENT-a tool connecting emerging photonics with electronics for opto-

electronic networks-on-chip modeling," in *IEEE/ACM International Symposium on Networks-on-Chip*, 2012.

[144] K. Palem, and A. Lingamneni, "Ten years of building broken chips: The physics and engineering of inexact computing," *ACM Transactions on Embedded Computing Systems,* vol. 12, no. 2, p. 87, 2013.

[145] S. Venkataramani, S. T. Chakradhar, K. Roy, and A. Raghunathan, "Approximate computing and the quest for computing efficiency," in *ACM Design Automation Conference*, 2015.

[146] S. Das, J. R. Doppa, P. P. Pande, and K. Chakrabarty, "Design-Space Exploration and Optimization of an Energy-Efficient and Reliable 3-D Small-World Network-on-Chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 36, no. 5, pp. 719-732, 2017.

[147] Y. Xiao, Y. Xue, S. Nazarian, and P. Bogdan, "A load balancing inspired optimization framework for exascale multicore systems: A complex networks approach," in *IEEE International Conference On Computer Aided Design*, 2017.

[148] S. N. a. P. B. Y. Xiao, "Self-Optimizing and Self- Programming Computing Systems: A Combined Compiler, Complex Networks, and Machine Learning Approach," *IEEE Transactions on Very Large Scale Integration Systems,* vol. 27, no. 6, pp. 1416-1427, 2019.

[149] B. K. Joardar, R.G. Kim, J.R. Doppa, and P.P.Pande, "Design and Optimization of Heterogeneous Manycore Systems enabled by Emerging Interconnect Technologies: Promises and Challenges," in *IEEE Design Automation and Test in Europe*, 2019.

[150] J. Lee, S. Li, H. Kim, and S. Yalamanchili, "Design space exploration of on-chip ring interconnection for a CPU–GPU heterogeneous architecture," *Journal of Parallel and Distributed Computing,* vol. 73, no. 12, pp. 1525-1538, 2012.

[151] Z. Xu, X. Zhao, Z. Wang, and C. Yang, "Application-aware NoC Management in GPUs multitasking," *Journal of Supercomputing,* vol. 75, no. 8, pp. 4710-4730, 2019.

[152] X. Cheng, Y. Zhao, H. Zhao, and Y. Xie, "Packet pump: overcoming network bottleneck in on-chip interconnects for GPGPUs," in *IEEE/ACM Design Automation Conference*, 2018.

[153] R. Ausavarungnirun, K.K.W. Chang, L. Subramanian, G.H. Loh and O. Mutlu, "Staged memory scheduling: Achieving high performance and scalability in heterogeneous systems," *ACM Special Interest Group on computer architecture,* vol. 40, no. 3, pp. 416-427, 2012.

[154] C.J. Lee, V. Narasiman, O. Mutlu, and Y. N. Patt, "Improving memory bank-level parallelism in the presence of prefetching," *IEEE/ACM MICRO,* pp. 327-336, 2009.

[155] K. Duraisamy, Y. Xue, P. Bogdan, and P. P. Pande, "Multicast-aware high-performance wireless network-on-chip architectures," *IEEE Transactions on Very Large Scale Integration Systems,* vol. 25, no. 3, pp. 1126-1139, 2016.

[156] Y. Xue, and P. Bogdan, "User cooperation network coding approach for NoC performance improvement," in *ACM International Symposium on Network on Chips (NOCS)*, 2017.

[157] A. Sampson, W. Dietl, E. Fortuna, D. Gnanapragasam, L. Ceze, and D. Grossman, "EnerJ: Approximate data types for safe and general lowpower computation," *ACM Special Interest Group on Programming Languages,* vol. 46, no. 6, pp. 164-174, 2011.

[158] H. Esmaeilzadeh, A. Sampson, L. Ceze, and D. Burger, "Neural acceleration for general-purpose approximate programs," *IEEE/ACM International Symposium on Microarchitectures,* pp. 449-460, 2012.

[159] V. Gupta, D. Mohapatra, S.P. Park, A. Raghunathan. and K. Roy, "IMPACT: imprecise adders for low-power approximate computing," in *IEEE International Symposium on Low Power Electronics and Design*, 2011.

[160] M. Shafique, W. Ahmad, R. Hafiz, J. Henkel, "A low latency generic accuracy configurable adder," in *IEEE/ACM Design Automation Conference*, 2015.

[161] J.S. Miguel, J. Albericio, A. Moshovos, and N.E. Jerger, "Doppelgänger: a cache for approximate computing," in *IEEE/ACM MICRO*, 2015.

[162] J.S Miguel, J. Albericio, N.E. Jerger, A. Jaleel, "The bunker cache for spatio-value approximation," in *IEEE/ACM International Symposium on Microarchitecture*, 2016.

[163] A. Rahimi, L. Benini, and R.K. Gupta, "CIRCA-GPUs: increasing instruction reuse through inexact computing in GP-GPUs," *IEEE Design and Test,* vol. 33, no. 6, pp. 85-92, 2016.

[164] R. Boyapati, J. Huang, P. Majumder, K.H. Yum, and E.J. Kim, "APPROX-NoC: A Data Approximation Framework for Network On-Chip Architectures," in *ACM International Symposium on Computer Architecture*, 2017.

[165] J. Leng, T. Hetherington, A. ElTantawy, S. Gilani, N. S. Kim, T. M. Aamodt, and V. J. Reddi, "GPUWattch: enabling energy optimizations in GPGPUs," *ACM Special Interest Group on computer architecture,* vol. 41, no. 3, pp. 487-498, 2013.

[166] Y Yarom and K Falkner, "FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX*, 2014.

[167] N.E.C. Akkaya, "Secure chip odometers using intentional controlled aging," in *IEEE Hardware Oriented Security and Trust* , 2018.

[168] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware Trojan attacks: Threat analysis and countermeasures," in *Proceedings of the IEEE*, 2014.

[169] D. Park, C. Nicopoulos, J. Kim, N. Vijaykrishnan, and C. R. Das, "Exploring fault-tolerant network-on-chip architectures," in *International conference on Dependable Systems and Networks*, 2006.

[170] S. Shamshiri, A. Ghofrani, and K. Cheng, "End-to-end error correction and online diagnosis for on-chip networks," in *Proceedings of International Test Conference*, 2011.

[171] J.Y.V.M. Kumar, A.K. Swain, S. Kumar, S. R. Sahoo, and K. Mahapatra, "Run Time Mitigation of Performance Degradation Hardware Trojan Attacks in Network on Chip," in *IEEE Symposium on VLSI* , 2018.

[172] J. Sepúlveda, D. Flórez, and G. Gogniat, "Reconfigurable security architecture for disrupted protection zones in NoC-based MPSoCs," in *IEEE International Symposium on Reconfigurable Communication-centric Systems-on-Chip*, 2015.

[173] T. Boraten, A.K Kodi, "Packet security with path sensitization for NoCs," in *IEEE Design, Automation and Test in Europe* , 2016.

[174] H.M. Wassel, Y. Gao, J. K. Oberg, T. Huffmire, R. Kastner, F. T. Chong, and T. Sherwood, "SurfNoC: a low latency and provably non-interfering approach to secure networks-on-chip," in *ACM Special Interest Group on computer architecture News*, 2013.

[175] S.V.R Chittamuru, I. Thakkar, S. Pasricha, "SOTERIA: exploiting process variations to enhance hardware security with photonic NoC architectures," in *IEEE/ACM Design Automation Conference*, 2018.

[176] S.Skorobogatov, "Physical attacks and tamper resistance," in *Introduction to Hardware Security and Trust*, Springer, 2011.

[177] E. Dubrova, M. Näslund, and G. Selander, "Secure and efficient LBIST for feedback shift register-based cryptographic systems," in *IEEE International Test Conference*, 2014.

[178] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," Springer-Verlag, 1999.

[179] E. Dubrova, M. Näslund, G. Carlsson, and B. Smeets, "Keyed logic BIST for Trojan detection in SoC," in *IEEE International System-on-Chip Conference* , 2014.

[180] M. Oya, "In-situ Trojan authentication for invalidating hardware-Trojan functions," in *IEEE International Symposium on Quality Electronic Design*, 2016.

[181] M. Hussain, A. Malekpour, H. Guo, and S. Parameswaran, "EETD: An Energy Efficient Design for Runtime Hardware Trojan Detection in Untrusted Network-on-Chip," in *IEEE International Symposium on VLSI*, 2018.

[182] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, "Nocalert: An on-line and real-time fault detection mechanism for network-on-chip architectures," in *IEEE International Symposium on Microarchitecture*, 2012.

[183] T. Boraten, D. DiTomaso, and A. Kodi, "Secure model checkers for Network-on-Chip (NoC) architectures," in *IEEE Great Lakes Symposium on VLSI* , 2016.

[184] M.K. Papamichael and J. C. Hoe, "CONNECT: re-examining conventional wisdom for designing nocs in the context of FPGAs," in *IEEE FPGA*, 2012.

[185] "Vivado HLS Tool," Xilinx, [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/integration/esl-design.html.

[186] S.F. Mossa, SR Hasan, O Elkeelany, "Self-triggering hardware Trojan: Due to NBTI related aging in 3-D ICs," *Integration, the VLSI Journal,* vol. 58, pp. 116-124, 2016.

[187] S.R. Vangal, J. Howard, G. Ruhl, S. Dighe,H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain, and V. Erraguntla, "An 80-tile sub-100-w teraflops processor in 65-nm cmos," *IEEE Journal of Solid State Circuits,* vol. 43, no. 1, pp. 29-41, 2008.

[188] M. Gebhart, J. Hestness, E. Fatehi, P. Gratz, and S. W. Keckler, "Running PARSEC 2.1 on M5, Technical Report TR-09-32," UT Austin Department of Computer Science, 2009.

[189] S. Bhardwaj, W. Wang, R. Vattikonda, Y. Cao, and S. Vrudhula, "Predictive modeling of the NBTI effect for reliable design," in *IEEE Custom Integrated Circuits Conference*, 2006.

[190] "Cadence Virtuoso," Cadence, [Online]. Available: https://www.cadence.com/content/cadence-www/tools/customic-ic-analog-rf-design/layout-design/virtuoso-layout-suite.html.

[191] J. Hestness, B. Grot, and S. W. Keckler, "Netrace: dependency-driven trace-based network-on-chip-simulation," in *ACM International Workshop on NoC Architectures*, 2010.

[192] Y. Zou and S. Pasricha, "Reliability-Aware and Energy-Efficient Synthesis of NoC based MPSoCs," in *IEEE International Symposium on Quality Electronic Design*, 2013.