UNDERSTANDING SECURITY THREATS OF EMERGING COMPUTING ARCHITECTURES AND MITIGATING PERFORMANCE BOTTLENECKS OF ON-CHIP INTERCONNECTS IN MANYCORE NTC SYSTEM

by

Chidhambaranathan Rajamanikkam

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

of

DOCTOR OF PHILOSOPHY

in

Electrical Engineering

Approved:

Sanghamitra Roy, Ph.D. Major Professor	Koushik Chakraborty, Ph.D. Committee Member
Reyhan Baktur, Ph.D. Committee Member	Zhen Zhang, Ph.D. Committee Member
David Geller, Ph.D. Committee Member	Richard S. Inouye, Ph.D. Vice Provost for Graduate Studies

UTAH STATE UNIVERSITY
Logan, Utah

2019

ABSTRACT

Understanding Security Threats of Emerging Computing Architectures and Mitigating Performance Bottlenecks of On-Chip Interconnects in Manycore NTC System

by

Chidhambaranathan Rajamanikkam, Doctor of Philosophy
Utah State University, 2019

Major Professor: Sanghamitra Roy, Ph.D.
Department: Electrical and Computer Engineering

Neuromorphic computing marks the beginning of a new era in computing system design, owing to the introduction of complex and unorthodox non-von neumann architectures, in conjunction with new post-CMOS nano-ionic devices. These neuromorphic chips are projected to soon become a mainstay platform for a diverse set of applications, ranging from day to day rudimentary decision making, to safety critical defense and healthcare management. In this environment, hardware security can no longer take an auxiliary role in system design. Similarly, Trustworthiness is an emerging prominent concern in the development of Multi-Processor System-on-Chip (MPSoC) systems, which often include Third-Party Intellectual Property (3PIP) cores. Since many 3PIPs are protected from introspection, malicious hardware trojans in these 3PIP components can create a plethora of system vulnerabilities.

Near threshold computing has unraveled a promising design space for energy efficient computing. However, it is still plagued by sub-optimal system performance. Application characteristics and hardware non-idealities of conventional architectures (those optimized for nominal voltage) prevent us from fully leveraging the potential of NTC systems. *Increasing the computational core count* still forms the bedrock of a multitude of contemporary works that address the problem of performance degradation in NTC systems. However, these works do not categorically address the shortcomings of the conventional on-chip interconnect fabric in a many core environment.

In this dissertation, First, we lay a solid foundation of understanding the broad security implications of emerging nanoionic device based neuromorphic computing paradigm. We investigate various security loopholes arising from an untrustworthy neuromorphic design environment. We propose, examine and evaluate the security threats arising from—*(a)* covert hardware Trojans and *(b)* external attacks exploiting implementation specific vulnerabilities. Subsequently, we discuss three specific attacks targeting crucial components of the neuromorphic design, and demonstrate how the security breaches in the neuromorphic systems can directly impact the end-user experience. Uncovering these security vulnerabilities in the emerging neuromorphic computing paradigm can be instrumental in shaping our design practices. Second, we explore an imminent threat that originates from a 3PIP core: architectural state preserving trojans (TASP). We illustrate that a malicious TASP can affect the performance of a core without altering its architectural state, thereby disrupting the availability of on-chip resources in a MPSoC system. We propose three complementary techniques to detect active TASPs in a 3PIP core with overheads of less than 5%. Third, we quantitatively demonstrate the performance bottleneck created by a conventional NTC architecture in many-core NTC systems. To reclaim the performance lost due to a sub-optimal NoC in many-core NTC systems, we propose *BoostNoC*—a power efficient, multi-layered network-on-chip architecture. *BoostNoC* improves the system performance by nearly 2× over a conventional NTC system, while largely sustaining its energy benefits. Further, capitalizing on the application characteristics, we propose two *BoostNoC* derivative designs: (i) *PG BoostNoC*; and (ii) *Drowsy BoostNoC*; to further improve the energy efficiency over conventional NTC system.

(110 pages)

PUBLIC ABSTRACT

Understanding Security Threats of Emerging Computing Architectures and Mitigating Performance Bottlenecks of On-Chip Interconnects in Manycore NTC System

Chidhambaranathan Rajamanikkam

Emerging computing architectures such as, neuromorphic computing and third party intellectual property (3PIP) cores, have attracted significant attention in the recent past. Neuromorphic Computing introduces an unorthodox non-von neumann architecture that mimics the abstract behavior of neuron activity of the human brain. They can execute more complex applications, such as image processing, object recognition, more efficiently in terms of performance and energy than the traditional microprocessors. However, focus on the hardware security aspects of the neuromorphic computing at its nascent stage. 3PIP core, on the other hand, have covertly inserted malicious functional behavior that can inflict range of harms at the system/application levels. This dissertation examines the impact of various threat models that emerges from neuromorphic architectures and 3PIP cores.

Near-Threshold Computing (NTC) serves as an energy-efficient paradigm by aggressively operating all computing resources with a supply voltage closer to its threshold voltage at the cost of performance. Therefore, STC system is scaled to many-core NTC system to reclaim the lost performance. However, the interconnect performance in many-core NTC system pose significant bottleneck that hinders the performance of many-core NTC system. This dissertation analyzes the interconnect performance, and further, propose a novel technique to boost the interconnect performance of many-core NTC system.

To my Mother, Father and my Sister...

ACKNOWLEDGMENTS

I am seated in everyones heart, and from Me come remembrance, knowledge and forgetfulness. - Sri Krishna - Bhagavad Gita Chapter 15 Verse 15

I would like to pay my humble obeisances unto the lotus feet of The Supreme Personality of Godhead, Lord Sri Krishna, without whom nothing is possible.

I would like to express my sincere gratitude to all the persons who had helped me during my PhD graduate program. First and foremost, I would like to thank my major adviser, Dr. Sanghamitra Roy and co-adviser, Dr. Koushik Chakraborty, for their immense support, valuable inputs, feedbacks, critiques, and guidance throughout my PhD program. They have provided me excellent opportunities to work on cutting-edge research topics and publishing them in top-tier conferences and journals. Further, working with them, I have learnt many things that had a huge influence in both my professional and personnal life.

Secondly, I would like to thank my committee members: Dr. Reyhan Bhaktur, Dr. Zhen Zhang and Dr. David Geller for their continuous support, valuable comments and insights during my research. I also would like to thank them for their immense support and contributions in obtaining summer internships during my third and fourth year of PhD.

I would like to thank my fellow lab mates in USU BRIDGE lab for numerous stimulating research topic discussions, their precious time and hard work before the deadlines, and all the fun, parties we had during the last four and half years. My sincere thanks to Rajesh JS, who collaborated in most of my research, for his valuable research contributions during my PhD studies. We had exchanged more heated research discussions, came up with exciting research ideas and the sleepless nights we had spent before the deadlines. I also would like to thank Kurt Brenning and Andrew Deiken for their continuous support during my first two years of my PhD study. They had provided significant contributions to my initial research in 3PIP Hardware Security. I extend my thanks to fellow lab mates Prabal Basu, Shamik Saha, Sourav Sanyal, Atif Yasin, Pramesh Pandey, Aatreyi Bal, Asmita Pal, Tahmourous

Sabastian, Harshitha Pulla, Dieudonne Manzi, Dean Michael Ancajas, and Hu Chen for their help and their support during my PhD program.

Next, I would like to extend thanks to my roommates: Prabal Basu, Shamik Saha, Tarak Saha and Sourav Sanyal for their friendship and more interesting non-technical late night discussions. They had helped me in different ways and in many difficult situations during the last four and half years.

I would like to appreciate the help and support provided by the all the staff members of ECE Department for providing me an opportunity to pursue my PhD Graduate program at USU. I am very thankful to MaryLee Anderson, Tricia Brandenburg, Kathy Phippen and Diane Buist for their invaluable help in supporting my PhD program in many ways. I would like to specifically thank Tricia Brandenburg for helping me out with hassle-free paperwork processing for the summer internships.

I also would like to extend my appreciation to Dr. Ravi Gupta and his family. Friday congregations in his house have given me great spiritual strength. They are very kind to me and always made me feel like a home away from home. Last, but not least, I would like to thank my parents: Rajamanikkam and Jayalakshmi, and sister: Padmapriya, for constantly encouraging and motivating me throughout my career. Without their love and encouragement, it wouldn't have been possible to complete my PhD program. Finally, I would like to thank my friends: Abhishek Manjunath, Abhijit Kulkarni, Praveen Medisetti and Swathi Ramanathan, for making my journey fun-filled and memorable.

Chidhambaranathan Rajamanikkam

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACRONYMS

IC	Integrated Circuit
NTC	Near-Threshold Computing
STC	Super Threshold Computing
AI	Artificial Intellegence
CMOS	Complementary Metal-Oxide-Semiconductor
SNN	Spiking Neural Network
IBM	International Business Machines (IBM) Corporation
ReRAM	Resistive Random-Access Memory
SoC	System-on-Chip
CAD	Computer Aided Design
CIA	Confidentiality, Integrity, Availability
PCM	Phase Change Memory
NS	Neurosynaptic Cores
LIF	Leaky Integrate and Fire
STDP	Spike Timing Dependent Plasticity
LTP	Long Term Potentiation
LTD	Long Term Depression
AGC	Automatic Gain Control
NoC	Network-on-Chip
DoS	Denial of Service
MNIST	Modified National Institute of Standards and Technology
MPSoC	Multi-Processor System-on-Chip
IP	Intellectual Property
3PIP	Third Party Intellectual Property
TASP	Architectural State Preserving Trojans
SPAR	Specification-centric Performance-aware Analytical Representation
CMPS	Covert Performance Monitoring System

ESTD	Employing Side-channel analysis for TASP Detection
ISA	Instruction Set Architecture
RTL	Register-Transfer Level
DUT	Design Under Test
BPU	Branch Prediction Unit
TMU	TASP Monitoring Unit
TSMC	Taiwan Semiconductor Manufacturing Company
CPI	Cycles Per Instruction
IMC	Intensive Monitoring Core
VC	Validation Core
PDF	Probability Density Function
HDL	Hardware Description Language
FP	False Positives
FN	False Negatives
DRAM	Dynamic Random-Access Memory
SRAM	Static Random-Access Memory
FPGA	Field-Programmable Gate Array
SECRET	Smart Employment of Circuit Redundancy to Effectively counter Trojans
PG	Power-Gated
PV	Process Variation
BoPeL	Boost Performance Layer
FruPUL	Frugal Power Usage Layer
NoRD	Node and Router Decoupling
CMP	Chip Multi-Processor
LC	Layer Controller
RC	Router Controller
NTV	Near-Threshold Voltage
FO4	Fanout of 4
DSENT	Design Space Exploration for Network Tool

CHAPTER 1
INTRODUCTION

The phenomenal growth and integration of transistor devices has reshaped our notion of a reliable and trustworthy hardware layer—the execution platform of all complex software applications. Several emerging trends in hardware integration and user computing practices are exposing new challenges for secure hardware design. First, with the popularity of cloud computing and artificial intelligence, emerging computing architectures such as, neuromorphic computing and third-party intellectual property (3PIP) cores, are poised to take over general purpose processors in cloud computing hosting [1] and for complex diverse applications fueled by AI's digital evolution [2–5]. Second, emerging neuromorphic platforms fabricated using novel nano-ionic devices are expected to offer significant performance and power efficiency benefits over the conventional computing architectures [6]. However, the potential ramifications of security vulnerabilities arising from following the same precedent of evolution can prove catastrophic. Third, the Muti-Processor System-on-Chips (MPSoCs) promote unprecedented integration of Third Party Intellectual Property (3PIP) components within a single hardware platform, for reducing cost and design complexity [7]. Further, MPSoC integrators are reluctant to re-verify the procured 3PIP, due to sky-rocketing verification costs and aggressive time-to-market schedules. Such practices create a potent threat in hardware security as it is impossible to ensure full trustworthiness of every 3PIP [3, 8]. This dissertation: (i) uncovers various security vulnerabilities originating from the emerging neuromorphic computing paradigm; and (ii) uncovers imminent threat model, architectural state preserving Trojans (TASP), stemming from a 3PIP core. Further, this dissertation evaluates and analyze the impact of the security vulnerabilities in both computing paradigm [9].

Near-Threshold Computing (NTC), on the other hand, has emerged as one of the promising directions in the pursuit of improving the energy-efficiency of computer designs

[10–12]—a growing concern in the current geopolitical landscape. A device operating at NTC sets its supply voltage close to its threshold voltage, while still maintaining a positive difference between them. Consequently, the energy consumption is dramatically reduced, both due to the quadratic impact of supply voltage reduction, as well as, a linear reduction from the operating frequency. However, NTC circuits also suffer a substantial performance degradation, for example, a 10-100X processor frequency reduction, restricting their domain to low power applications [10, 11]. This dissertation addresses a key research question: *can an NTC system approach the performance of a conventional super-threshold computing (STC) system, while still retaining the energy efficiency advantage of NTC? [13]*

1.1 Contribution of This Dissertation

The research works pertaining with this dissertation have been partially published and under various stages of review process in conferences and journals, including 2016 IEEE/ACM International Conference on Computer Aided Design (ICCAD), 2018 Elsevier Science Publishers Journal in Microprocessors and Microsystems (MICPRO), and IEEE Transactions on Very Large Scale Integration Systems (TVLSI) (two research works are under various stages of review process). Publications stemming from this dissertation are listed as follows:

1.1.1 Conference Papers

- **Chidhambaranathan Rajamanikkam**, Rajesh JS, Koushik Chakraborty and Sanghamitra Roy, *"BoostNoC: Power efficient network-on-chip architecture for near threshold computing"*, Proceedings of the 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, 2016, pp. 1-8.

1.1.2 Journal Papers

- **Chidhambaranathan Rajamanikkam**, Kurt Brenning, Andrew Deakin, Sanghamitra Roy and Koushik Chakraborty, *"TASPDetect: Reviving Trust in 3PIP By*

Detecting TASP Trojans", Proceedings of the 2018 Elsevier Science Publishers Journal in Microprocessors and Microsystems (MICPRO), vol. 56, pp. 76-83.

- **Chidhambaranathan Rajamanikkam**, Rajesh JS, Koushik Chakraborty and Sanghamitra Roy, *"Energy Efficient Network-on-Chip Architectures for Many-core Near-Threshold Computing System"*, Proceeding of the Journal of Low Power Electronics (JOLPE) (Under Review).

- **Chidhambaranathan Rajamanikkam**, Rajesh JS, Sanghamitra Roy and Koushik Chakraborty, *"Understanding Security Threats in Emerging Neuromorphic Computing Architecture"*, Proceedings of the Journal of Hardware and Systems Security (HaSS) (Under Review).

CHAPTER 2
LITERATURE REVIEW

This chapter discusses the comprehensive literature study on recent works closely associated to security concerns in emerging computing architecture and performance bottlenecks of on-chip interconnects in many-cores low power system, to build a strong foundation to this dissertation. Security concerns in emerging computing architecture covers state-of-the-art works in Neuromorphic computing and 3PIP cores. Further, this study investigates the performance bottlenecks in NoC architectures in many-core NTC system. This chapter can be further divided into few sections: Section 2.1 investigates the relevant security related works in Neuromorphic computing architectures. Section 2.2 reviews the existing works related to 3PIP hardware security. Section 2.3 studies the existing related works in NTC, NoC and manycore systems. Finally, Section 2.4 details the significance and importance of this dissertation.

2.1 State-of-the-Art Works in Neuromorphic Computing Security

The existing works in Neuromorphic computing security can further classified into four categories: various neuromorphic architectures, use of nano-ionic devices as synapses, adversarial machine learning and security of neuromorphic computing.

2.1.1 Various Neuromorphic Architectures

Neuromorphic computing in the form of hybrid digital-analog SNNs has attracted significant attention [14, 15]. For example, IBM has developed TrueNorth chip specifically to perform neuromorphic applications [6].

- **TrueNorth [6]:** Akopyan et al. had developed a neurosynaptic processor, TrueNorth, which implements non-von Neumann, power-efficient, scalable and parallel architecture. TrueNorth chip consists of 4096 neurosynaptic cores, with 1 million digital neurons and 256 million synapses. Further, the authors had developed a novel design methodology that uses mixed asynchronous–synchronous circuits and an entire tool flow for building an event-driven, power-efficient neurosynaptic chip. The connectivity and neural parameters are fully configurable to enable custom configurations for a varied range of cognitive and sensory perception applications.

- **SpiNNaker [15]:** Painkras et al. had evaluated Spiking Neural Network Architecture, SpiNNaker, a massively parallel computer system developed to provide cost effective and more flexible simulator for neuroscience experiments. The objective of SpinNNaker is to model large systems of spiking neurons with computationally demanding applications. SpiNNaker can model billion neurons and trillion synapses at real-time. The building block of SpinNNaker is based on SpiNNaker Chip Multiprocessor (CMP), which is a custom-designed globally asynchronous locally synchronous (GALS) system with 18 ARM968 processor nodes. The authors had evaluated the system requirements based on the high demanding applications. Further, they also evaluated peak power consumption and performance requirements.

- **Neurogrid [14]:** Benjamin et al. had presented the design of Neurogrid, a neuromorphic system simulator for large-scale neural models. The authors have emulated three primary neural elements: axonal arbor, synapse and dentritic tree. The fourth element, soma, is used with shared electronic circuit to allow maximum number of synaptic connections. The simulators energy-efficiency is maximized by analog axonal arbor and throughput is maximized by tree-style interconnected neural arrays. However, the architecture has configurability and scaling limitations to larger neuron models.

2.1.2 Use of Nano-ionic Devices as Synapses

Plethora of works capitalize the benefits of nano-ionic devices, such as memristors and ReRAM, for managing the synaptic weights [16, 17]. Memristor, popular among these devices, is depicted as a promising candidate for synapses as it portrays identical characteristics of synaptic weight tuning. However, emerging neuromorphic architectures and circuit properties of nano-ionic devices has inherent security loopholes that are susceptible to various attacks. Therefore, securing the neuromorphic architectures and nano-ionic circuit properties are the need of the hour.

- **ReRAM-based Synapse [17]:** Park et al. had showcased an advanced ReRAM based artificial synapse for neuromorphic systems. ReRAM based synapse portrayed promising approach in improving the performance and reliability of the neuromorphic systems. In this work, they demonstrated an auditory recall on artificial neural networks using EEG experiment.
- **STT Magnetic Tunneling Junction [16]:** Sengupta et al. has introduced Magnetic Tunneling Junction based nano device to perform the thresholding behavior of the biological neuron. They also proposed the low power neuromorphic computing architecture utilizing emerging sprintronic and memristive devices for synapses. The excitatory/inhibitory charging current are generated by the MTJ based crossbar array that consists of programmable resistive synapses. The strength of the magnetization represents the strength of the synapse connections between two *STT-Neuron*. The magnetization is manipulated by spin transfer torque generated net synaptic current.

2.1.3 Adversarial Machine Learning

Attacks against machine learning systems can be classified based on three properties described by Huang et al. [18]: (a) *influence of attack* - altering training process vs model/data discovery by probing, (b) *security violation* - confidentiality, integrity and availability, and (c) *attack specificity* - targeted vs indiscriminate. Learning models have previously been shown to be vulnerable to adversarial inputs [19]. Similarly, SNN trained as classifiers are also vulnerable to such adversarial perturbations, where spikes may be added or removed to cause adversarial behavior [20].

Sensitivity studies and research on training SNNs towards adversarial perturbations is still at a very nascent stage, with researchers considering small networks, and datasets with small set of features. *Adversarial training still suffers from the curse of dimensionality in complex dataset, where the test input can reside far from the distribution of adversarial training dataset, thus limiting the impact of adversarial training.* The introduction of novel next generation unorthodox architectures and novel devices into this volatile environment aggravates the problem of trust and security further.

- **Adversarial Machine Learning [18]:** Huang et al. had showed two machine learning methods for anomaly detection that are vulnerable to causative attacks and explained how the applications, features and data restricts the adversary's actions. They also presented models that corrupts the input data, output class limitations, and their awareness of learning algorithm and feature space exploration. Further, they also provided the defenses and countermeasures against such attacks.

- **Adversarial SNN Training [20]:** Bagheri et al. had examined the sensitivity of the probabilistic 2-layer SNN under various adversarial perturbations. They have taken rate and time encoding schemes, and, rate and first-to-spike decoding schemes to build adversarial inputs. Further, they employed robust training method to boost the resilience of the SNN.

- **Transferability in Machine Learning [19]:** Papernot et al. introduced transferability attacks—an attacker can substitute their own model, train with adversarial samples and transfer the model into a victim model. They experimented the attacks on two commercial machine learning classification (Amazon and Google) and showcased that existing machine learning algorithms are vulnerable to black-box attacks.

2.1.4 Security of Neuromorphic Computing

Contemporary research works on security of neuromorphic computing explored the challenges and solutions associated to the software related security aspects of neuromorphic

Techniques	**Trojan Location**	**Harm Semantics**	**Time of Detection**
SECRET [23]	Any	Integrity	runtime
Fort-NoCs [24]	NoC	Confidentiality	runtime
FANCI [25]	Any	Confidentiality	pre-silicon
Blockage [26]	Any	Confidentiality	runtime
MORPH [27]	Any	Confidentiality	runtime
SPAR, ESTD	BPU, cache	Availability	post-silicon
CPMS	μP	Availability	runtime

Table 2.1: Comparison of existing 3PIP trojan detection techniques with SPAR, CPMS, and ESTD.

computing [21, 22]. While these works highlight critical security vulnerability, they do not consider the exploitation of the security aspects of neuromorphic hardware.

- **Challenges and Solutions [21]:** Liu et al. emphasized the security issues associated with AI based applications and further proposed secured neuromorphic system design that prevents the malicious adversary from duplicating the learned model. The prevention mechanism leverages the drifting effect of the memristor device to rapidly degrade the computational accuracy of the learned model if user failed to provide correct authentication.

- **Thwarting Learning Attacks [22]:** Yang et al. proposed a mechanism to prevent a malicious adversary with physical access from learning the proprietary algorithms implemented by neuromorphic hardware. They exploit the obsolescence effect in memristor device to carefully reduce the accuracy of learned model if any unauthorized user. The obsolescence effect is carefully regulated for the authorized user to control the accuracy of the learned model.

2.2 Existing Hardware Security Related Works in 3PIP Cores

Existing hardware trojan detection techniques can be split into three categories: pre-silicon verification, runtime solutions, and post-silicon physical testing. Many of these techniques are limited due to two challenges imposed in a 3PIP: a) lack of a golden checkpoint; and b) limited visibility of the RTL. Pre-silicon verification techniques typically focus on formal verification [26,28–30] or functional analysis [25,31]. Runtime solutions are hardware or software techniques that operate during the lifetime of the chip. Ancajas et al. propose a technique called Fort-NoCs that introduces three layers of security mechanisms in the SoC firmware [24]. Bloom et al. propose using a reconfigurable FPGA architecture with additional security mechanisms to protect sensitive information [27]. Farag et al. propose a technique called SECRET that employs two identical instances of a 3PIP core to detect hardware trojans [23]. Physical testing can detect hardware trojans during the post-silicon phase by looking for anomalies in the power, area, performance, and thermal characteristics of the chip [32–34]. Table 2.1 summarizes several existing and our proposed 3PIP trojan detection techniques. Of the listed techniques, only SPAR, CPMS, and ESTD can detect TASPs.

Recent techniques integrate a given functional component from multiple IP providers to detect hardware trojans in one of them [3]. Our detection techniques are distinct from these as: (a) we are able detect TASPs that cause harm without disturbing the functionally correct instruction execution; and (b) we do not require IP instances from multiple vendors.

2.2.1 Pre-Silicon Verification

Numerous works exist on protecting and detecting Hardware Trojans in 3PIPs. Few of them from Pre-Silicon verification domain are:

- **HT Protection for 3PIPs [26]:** Al Anwar et al. presented a novel method for protecting the system against Hardware Trojans that alleviate the need for a golden chip. The protection mechanisms against Hardware Trojans utilizes a simple blockage method. This method operates at runtime and targets protection from Hardware Trojans that are embedded in 3PIPs. The simple blockage method uses standard cryptographic functions to obfuscate the output of suspected IPs to hide any information leakage.

- **Overcoming an Untrusted Computing Base [28]:** Hicks et al. presented a hybrid approach of Hardware and Software to defend against malicious Hardware.

They proposed BlueChip—defense strategy that detours the signals effectively around the suspicious Hardware. BlueChip utilizes unused circuit identification technique to flag any suspicious circuit (which is unused in the circuits or activated only during the verification tests). BlueChip replaces the suspicious circuits with exception generation hardware, thereby protecting the IPs from suspicious circuits.

- **Detecting Malicious Modification in 3PIPs [30]:** Rajendran et al. proposed a technique that formally verify 3PIPs against unauthorized corruption of critical data. The proposed approach learns properties to identify corruption of critical registers. They also explained two attacks that performs computations on corrupted data while not corrupting the critical registers. The proposed technique detects DeTrust Trojans which bypass FANCI [25] and VeriTrust [35]. The developed properties detect the Trojans and detect security vulnerabilities caused by design bugs and for security validation. However, the limitations of proposed technique are one-way functions and scalability.

- **Formal Methods for HT Detection [29]:** Rathmair et al. addresses critical problem which contains malicious hardware modifications obscured in the integrated chip. This particular work concentrates on formal verification methods to detect Hardware Trojans at design phase. The proposed methods cannot guarantee the inexistence of a malicious Hardware. This proposed method increases the Trojan Assurance Level of the integrated chip.

2.2.2 Pre-Silicon Functional Analysis

The existing works in 3PIP Hardware Security related to Pre-silicon functional analysis are listed below:

- **Trusted RTL [31]:** Mainak et al. proposed a four-step approach to identify, filter and locate malicious circuitry implanted in a 3PIPs. First step removes easy-to-detect signals which activates and propagates easily in the circuit. Second step, using N-detect full-scan ATPG tool to determine the hard-to-excite and/or propagate signals in the circuit. Third step, suspected signals are verified against spec by a suspect-signal-guided equivalence checking. Final step, region isolations applied on the filtered signal to identify the clusters of untestable gates in the circuit.

- **FANCI** [25]**:** Waksman et al. addresses critical problems in Hardware Security that determines the built-in backdoors that compromises the security after deployment. The Hardware backdoors incorporate logic that is almost nearly-unused logic (stealthy) in the circuit. These unused-logic never influences the outputs of the circuits. They proposed FANCI—tool that flags suspicious logic that have malicious intentions in the circuits. FANCI is scalable, approximate, Boolean functional analysis to detect the unused logic.

2.2.3 Runtime HT Detection Solutions

Runtime detection solutions identify the malicious behavior of the circuit dynamically during runtime. Few existing works that are related to this domain are listed below:

- **Fort-Nocs** [24]**:** Dean et al. introduced a novel and imminent threat emerging from MPSoC that has built-in 3PIP NoCs. The compromised NoC enables a varied security attacks with an accomplice software component. They proposed Fort-NoCs—a series of techniques work together for protecting compromised NoC in an MPSoC. First, Data Scrambling that allows stiff barrier for Hardware Trojan activation in NoC. Second, Packet Certification that isolates the communication link between the untrusted NoC and its accomplice thread and third, Node Obfuscation that decouples the source and destination nodes of a communication.

- **FPGA SoC Architectures** [27]**:** Bloom et al. proposed a novel technique for an FPGA SoC to ensure confidentiality of trusted software despite Hardware Trojan attacks. The proposed Morph Onion-encryption Replication technique, a defensive approach that feature morphing on-chip resources for moving target defense against fabrication time Trojans, encryption for confidential information, and replicate the functional equivalent PEs with arbitrated voting for resilience to design time Trojans.

- **SECRET [23]:** Farag et al. presented SECRET (Smart Employment of Circuit Redundancy to Effectively Counter Trojans), a high-level architecture, proof-of-concept application to a 3PIP crypto core containing a Hardware Trojan Horses. Two identical instances of protected IP core are used for monitoring and operating purposes and a time shift is created between two cores. Trojan detection circuit monitors the observation core at runtime. If the Trojan is detected, operating core with delayed input is suspended or identified triggering inputs are isolated for specific period to bypass Trojan activating trigger.

2.2.4 Post-Silicon Physical Testing

Post-silicon physical testing is widely used to identify any anomalies by monitoring area, power and energy. Several existing works already explored the Trojan detection mechanism during post-silicon testing. Few among them are explained below:

- **Multimodel Characterization [32]:** Kangqiao et al. proposed a multimodal characterization framework to detect and locate hardware Trojans using post-silicon thermal maps and power maps. They consider 2-dimensional principal component analysis (2DPCA) framework for detecting Trojans even with smaller size. The proposed approach estimates the detailed post-silicon power consumption using thermal maps and applies 2DPCA to extract spatial power consumption, and use statistical methods to detect hardware Trojans.

- **Path Delay Fingerprint [34]:** Jin et al. proposed a behavior-oriented category method that divides Trojans into two categories: explicit payload Trojan and implicit payload Trojan. Path delays of chips are obtained to build a series of fingerprints, each fingerprint representing one aspect of total characteristics of a benign design. These fingerprints are compared with delay parameters for chip validation. Therefore, path delays comparison flags a small Trojans circuits significant from a delay point of view.

- **BISA [33]:** Xiao et al. proposed a novel technique, BISA (Built-In Self-Authentication) that fills unused spaces in a circuit layout by functional filler cells instead of non-functional filler cells. All functional filler cells are tested by BISA and a digital signature is generated. Any malicious modifications in a circuit results in different signatures that can be identified by BISA. BISA prevents Trojan insertion and it can be applied to single-module or bottom-up hierarchical design.

2.3 On-Chip Interconnects for NTC Systems

Existing works related to the performance bottlenecks of on-chip interconnects in a manycore low power system can be classified into works explored in four related domains: (i) Near-Threshold Computing; (ii) Process Variation at NTC; (iii) Performance loss concern in NTC; and (iv) NoC Topologies and Architecture; This section provides an overview of existing works that closely related to the work presented in this dissertation.

2.3.1 Near-Threshold Computing Fundamentals

Over the last decade, near threshold computing has been studied extensively, and a major share of these works focuses on developing circuits and optimizing the computation and memory for a many core NTC system.

- **Near-Threshold Computing [10]:** Dreslinski et al. investigated near-threshold computing design, where the supply voltage is slightly greater than the threshold voltage of the transistors. Near threshold region retains much of energy savings with more favorable performance and variability characteristics. This work examined the barriers of employing NTC and provided solutions to overcome the obstacles associated with NTC. The most prominent barriers of NTC are performance loss, increased performance variation, and functional failures. This work also presents several solutions to address these barriers.

- **Voltage Islands at NTC [36]:** Silvano et al. proposed dark silicon mitigation technique by leveraging the benefits of NTC to overcome the manycore power-wall.

This work investigates the performance sustainability when moving towards many-cores NTC through variability-aware voltage and frequency tuning strategies. This work presented three such strategies highlighting that Super-Threshold Computing performance can efficiently sustain or even optimized at NTC regime.

2.3.2 Process Variation

To fully understand the impact of PV and capture the increased sensitivity to PV at NTC, researchers have developed microarchitectural PV models [37]. Further, several innovative solutions, such as the use of PV tolerant memory structures [38], use of multiple voltage-frequency domains [36] and computational core pipeline weaving [39], among others, have been proposed to tackle the challenges arising due to PV at NTC.

- **Micro-Architectural PV Models [37]:** Karpuzcu et al. presented the first PV micro-architectural model, VARIUS-NTV, for NTC circuits. The major contributions of the proposed model includes gate delay, SRAM memory, memory mode failures and impact of leakage. This model accounts the variation that affects the frequency and power consumed by cores and memories at NTC manycore and timing faults in SRAM memories at NTC.

- **PV Tolerant Memories [38]:** Mukhopadhyay et al. proposed two post-silicon tuning mechanisms that can reduce the leakage spread and improve parametric variations in memories simultaneously. The proposed self-repair mechanism utilizes body bias to reduce the parametric failures and leakage spread in memories. The proposed self-adaptive mechanism reduces the standby power dissipation in memories and at the same time keeping the failures under control.

- **Multiple V/F Island Management in NTC [36]:** Silvano et al. investigated the performance guarantees can be assured when scaling from STC to manycore NTC through multiple voltage and frequency allocation schemes. They proposed three voltage tuning and allocation schemes for NTC and demonstrated that STC performance can be efficiently sustained at NTC region depending on the underlying application characteristics.

- **Core Pipeline Weaving [39]:** Krimer et al. proposed a near-energy optimal stream processor which supports massively parallel, near-threshold circuits, that can tolerate large delay variations. They proposed two techniques, Decoupled SIMD Parallel Pipelining for dynamic variation handling and Pipeline Weaving for static variations. By combining with course grained approaches, the overall architecture provides unparalleled efficiency while the performance requirements of the low power applications are fulfilled.

2.3.3 Performance Loss Concern in NTC

To reclaim the lost performance caused by the reduction in operating frequency, contemporary works have proposed circuit-architectural solutions, such as device optimization by improving channel doping profile [10] and clustered architecture [10, 36]. But the most intuitive approach has been to increase the number of computational cores to exploit application parallelism [11, 36]. A direct consequence of this approach has been the tremendous increase in the on-chip communication demand.

- **Circuit-Architectural Solutions [10]:** Dreslinski et al. explored near threshold computing which provides much of energy savings of sub-threshold region along with favorable performance and variability characteristics. This work discusses the barriers to the adoption of NTC such as, Performance loss, Increased performance variation and Increased functional failures. Further, this work describes existing work aimed at overcoming these obstacles.

- **Performance Limits of Parallelized NTC [11]:** Pinckney et al. investigated the limit of voltage scaling along with task paralellization in order to maintain task completion latency, when slower clock is compensated by task parallelization across multiple cores. By operating at near threshold voltages, minimum task energy is obtained.

2.3.4 Existing NoC Topology and Architectures

Contemporary research on NoC topology and architectures such as clustered NoC [40], hierarchical NoC [41] and tile based NoC [36,42], have aimed to reduce the inter-core packet hop distance. While these works are an important step forward, they do not adequately address the challenges of reduced operational frequency and PV induced performance variation posed by the NTC regime.

- **Clustered NoC Architecture [40]:** Seifi et al. proposed a novel NoC architecture, Clustered-NoC (C-NoC) to improve the performance in group communication. C-NoC is enhanced with 8 bi-directional ports that are connected to 4 adjacent switches and 4 local ports. In the group communication, C-NoC reduces the average delay time of a packet and number of clock pulses significantly. Such a high performance is due to the efficient internal switching of C-NoC. Further, in corner-to-corner communication, C-NoC improves the maximum hops and average delay time of a packet significantly, compared to existing NoC architectures.

- **Hierarchical NoC Architecture [41]:** Lankes et al. proposed hierarchical NoCs (HNoCs) topologies to improve access type of shared resources such as, off-chip memory controllers, data I/O interfaces. HNoCs connects tiles, that are frequently accessed by other tiles in the network, to the higher hierarchy levels that connects sub-networks. Thus, it reduces the communication costs significantly. HNoCs are built from various sub-network topologies including, mesh, ring, crossbars and buses. Further, this work explored various hierarchical NoC architectures and highlighted the benefits in terms of hop count, latency and network throughput of the proposed HNoCs.

- **Tile-Based NoC Architecture [42]:** Janidarmian et al. introduced a new mapping algorithm, Onyx, that maps the IP cores on the mesh tiled-based NoC. Onyx maps the cores with less complexity and provides reduced hop count between the cores and thereby improving the energy consumption and other performance parameters such as latency and throughput.

2.4 Significance and Importance of this dissertation

This dissertation consists of three research works, including understanding security threats in emerging neuromorphic computing architecture, uncovering an imminent threat, architectural state preserving trojans (TASP) and its impact on the MPSoC system, and evaluating performance bottleneck of NoC in many-core NTC system. Our first work aims to etch security and trust into the design practices of next generation neuromorphic hardware platforms. To oversee secure hardware design and methodically eradicate security threats, we first analyze the emerging neuromorphic architectures and identify vulnerabilities arising due to unorthodox architectures. Subsequently, we aim to uncover circuit properties of memristor that could hinder secure designs. In summary, this work investigates the security threats arising from emerging neuromorphic architectures, as well as, memristor circuit properties, and showcase how these threats can thwart the performance of neuromorphic systems. Our second work uncovers an imminent threat, TASP that affect the performance of a core without altering its architected state, originating from the pervasive use of 3PIP core. We demonstrate that active TASP in 3PIP core affects the performance of the system significantly. Furthermore, we present three TASP detection techniques to detect always-on and intermittent TASP in a 3PIP core with marginal overheads. Our third work advances the research in low power computing domain, and focuses on optimizing the on-chip communication in many-core NTC systems. In this work, we clearly demonstrate the performance bottleneck created by sub-optimal NoC architectures in many-core NTC systems. We then propose and evaluate a multilayered NoC architecture, *BoostNoC*, to boost the performance, while largely sustaining the power and energy efficiency benefits in a many-core NTC system. Further, we develop optimal variants of *BoostNoC*, by meticulously analyzing the application characteristics and the NoC network traffic in many-core NTC system.

CHAPTER 3
UNDERSTANDING SECURITY THREATS IN EMERGING NEUROMORPHIC COMPUTING ARCHITECTURE

3.1 Background and Contributions of This Work

"*Artificial Intelligence* (A.I)"—the simulation of natural human intelligence in computing systems, has been a disruptive innovation with far-reaching consequences in the digital world. Algorithms and software that learn, adapt and update themselves, when exposed to real time data are galvanizing rapid growth in all end-use segments that require data monitoring or manipulation. Figure 3.1 encapsulates the diverse applications fueled by AI's digital revolution. As a testament to this rapid adoption, the global AI market ($642 million in 2016) is expected to reach $35 billion by the year 2025, growing at an unprecedented annual rate of 57.2% [43]. Owing to the collapse of Dennard scaling, the impending end of Moore's law, and the Von-Neumann memory bottleneck, existing systems cannot cater to the performance demands of modern AI applications [44, 45]. Hence, a new breed of brain-inspired **neuromorphic chips**, fabricated using emerging nano-ionic devices are coming forth to power the next wave of AI applications.

Heretofore, CMOS devices, circuits and architectures, have experienced an accelerated technological evolution, with hardware security policies struggling to catch up to the advances. Hence, time and again, traditional computing platforms have fallen prey to security threats [46]. If the evolution of emerging neuromorphic hardware platforms follow the same precedent, the potential ramifications of security threats can prove catastrophic, as neuromorphic systems are expected to form the backbone of various life-critical systems in healthcare, military and automotive industries. The amalgamation of novel non-von neumann architectures, and new post-CMOS nano-ionic devices, truly marks the beginning of a new era in computing system design. This genesis of neuromorphic computing presents

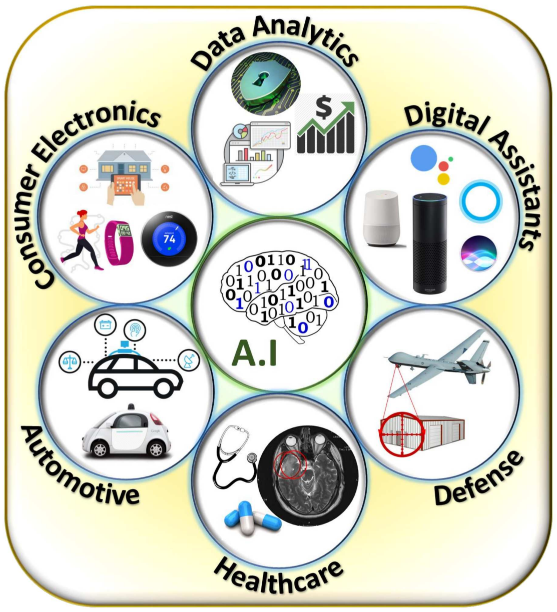


Fig. 3.1: Diverse applications due to AI's digital revolution.

us with an unique opportunity—***Incorporate security as a forethought to stay ahead of the threat curve in the emerging neuromorphic computing paradigm.***

Uncovering security vulnerabilities in the emerging neuromorphic computing paradigm will be instrumental in reshaping our design practices. Our work lays a strong foundation to understanding the broad security implications of emerging nano-ionic device based neuromorphic hardware. *To the best of our knowledge, this is the first work on neuromorphic hardware security to comprehensively analyze potential threat models, identify multiple attack vectors and examine their impact on applications.* We summarize the novel contributions of our work, next.

- We identify and discuss the circumstances under which the neuromorphic hardware can become a breeding ground for security vulnerabilities, and propose two lateral threat models (Section 3.2).

- We examine different exploitable attack vectors, including hardware Trojan insertion, and external attacks stemming from inherent device, circuit and implementation specific vulnerabilities (Section 3.3).

- We propose multiple attack, model their behavior and evaluate the impact of security threats on neuromorphic hardware (Section 3.4).

3.2 Threat Models

To understand the security threats in emerging neuromorphic computing, we examine two lateral threat models, classified based on the source of security vulnerability.

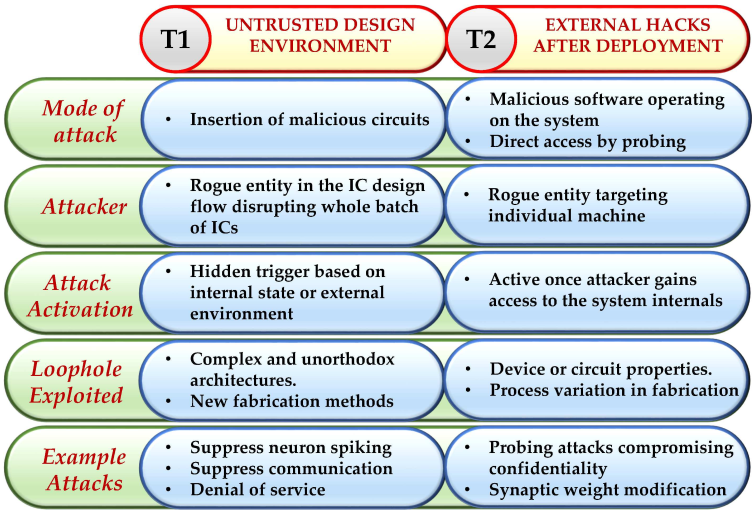
- *Untrustworthy design environment* : Insertion of hardware Trojans.
- *External hacks after deployment* : Exploiting implementation-specific vulnerabilities.

Figure 3.2 outlines the two threat models, revealing key parameters of discussion. In Section 3.2.1 and Section 3.2.2, we highlight the relevance of these threat models in the neuromorphic computing paradigm, and briefly examine their potential ramifications to motivate our work (Section 3.2.3).

3.2.1 Untrustworthy Design Environment

Neuromorphic computing inherits the system-on-chip's (SoC) flawed and untrustworthy design space [2]. Rising design and verification costs, unreasonable time-to-market, and complex designs, made the SoC design flow and supply chain a breeding ground for malicious Trojans and covert backdoors [47]. At this juncture, the introduction of a new class of nano-ionic devices, in succession to novel, unfamiliar and intricate brain-inspired architectures can push security into a deep abyss.

In this threat model, one or more components of the neuromorphic design flow, encompassing third party intellectual property vendors, CAD tool providers and fabrication houses are considered to be untrustworthy. Malicious circuits, hardware Trojans, and back-doors are inserted in the design flow, with their functionality discreetly hidden within the complex architectures using innovative rare-event based triggers. Figure 3.3 highlights the various opportunities present in the design flow to covertly insert a hardware Trojan into the complex architectures. Once active, the hardware Trojans are designed to inflict a



	T1 UNTRUSTED DESIGN ENVIRONMENT	T2 EXTERNAL HACKS AFTER DEPLOYMENT
Mode of attack	• Insertion of malicious circuits	• Malicious software operating on the system • Direct access by probing
Attacker	• Rogue entity in the IC design flow disrupting whole batch of ICs	• Rogue entity targeting individual machine
Attack Activation	• Hidden trigger based on internal state or external environment	• Active once attacker gains access to the system internals
Loophole Exploited	• Complex and unorthodox architectures. • New fabrication methods	• Device or circuit properties. • Process variation in fabrication
Example Attacks	• Suppress neuron spiking • Suppress communication • Denial of service	• Probing attacks compromising confidentiality • Synaptic weight modification

Fig. 3.2: Comparison of key attributes of the two proposed lateral threat models.

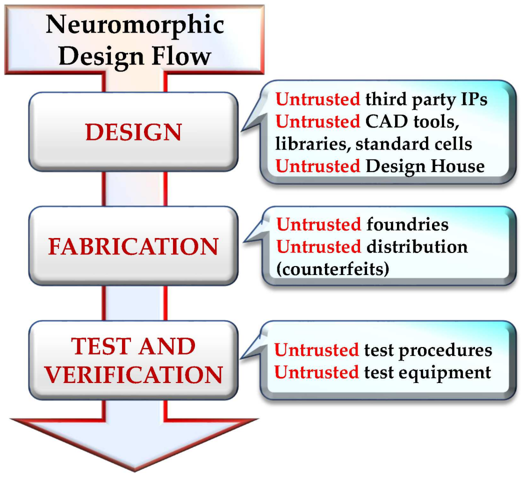

Neuromorphic Design Flow
DESIGN
Untrusted third party IPs
Untrusted CAD tools, libraries, standard cells
Untrusted Design House
FABRICATION
Untrusted foundries
Untrusted distribution (counterfeits)
TEST AND VERIFICATION
Untrusted test procedures
Untrusted test equipment

Fig. 3.3: Neuromorphic design flow illustrating opportunities for hardware Trojan insertion.

multitude of attacks, disrupting the *CIA* triad (confidentiality, integrity and availability) of neuromorphic systems.

3.2.2 External Hacks after Deployment

The development of novel functional materials and devices, such as ReRAM, PCM, and spintronic devices, have incited a quantum leap in computing, by making the implementation of brain-inspired neuromorphic hardware possible. These fundamental devices, based on new material physics, are sought after for the properties that mimic the biological

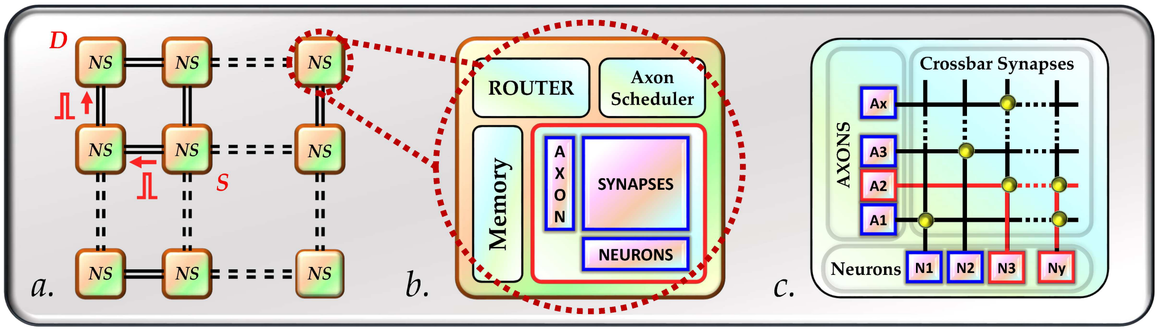


Fig. 3.4: A typical neuromorphic architecture. Figure 3.4(a) shows the system level architecture, where many neurosynaptic cores (NS) are connected by a mesh interconnect fabric. It also shows the movement of spike packets from source (S) to destination (D). Figure 3.4(b) shows the block diagram of a NS. Figure 3.4(c) presents the crossbar synapses, axons and neurons. When an axon A_2 receives an input spike, the current flows through the highlighted path to the neurons N_3 and N_y. If the membrane potential of either neuron is breached, a spike is generated.

brain's synapses. The use of these materials in neuromorphic applications requires extensive knowledge of properties such as multistate behavior, sensitivity to external stimuli, fault tolerance, temperature window, ability to synthesize, electromagnetic interaction, among others. *Due to the broad range of desirable requirements, most chosen materials often have inherent properties that can expose security vulnerabilities when employed in circuits.*

In this threat model (Figure 3.2), the design environment is considered trustworthy, i.e., the neuromorphic hardware is not intentionally compromised during the design and fabrication stages. However, post-deployment, an external rogue agent targets an individual system, by gaining direct (probing the hardware) or indirect (running malicious software) access to the neuromorphic hardware. The hacker needs to have sufficient information regarding the hardware implementation or design internals. On gaining access to the system, the attacker can exploit circuit characteristics, device properties or even process variations, that occur during the fabrication, to inflict damage to the system.

3.2.3 Security Implications

Neuromorphic hardware is expected to rapidly gain inroads into diverse applications, to cater to the computing demands of AI applications. To understand the security implica-

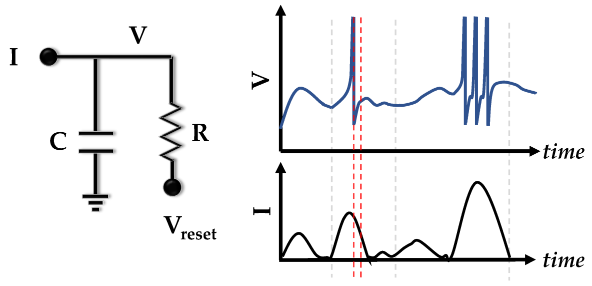


Fig. 3.5: Leaky integrate and fire (LIF) neuron model representation and simulation illustrating the neuron spiking.

tions, and identify concerns around applications, we examine a few cases of untrustworthy neuromorphic systems.

- The military application of neuromorphic hardware includes, target identification in autonomous lethal weapons. Unexpected system behavior, such as mistaken targets, or even faulty trigger, can cause unwarranted tensions between nations [48].
- In the automotive industry, inaccurate pattern/object recognition can lead to fatal accidents in self driving cars [49]. In addition, external hacks to the autonomous vehicles can give terrorists new weapons for inciting violence [49].
- In the healthcare industry, progress made in health risk assessment, and disease diagnosis, can be nullified due to incorrect cataloging [48, 49].
- In government and other social applications, security vulnerabilities can be exploited to discriminate against groups, and effectively target political sentiments [48].

The above assortment of grim events, underlines the importance of secure neuromorphic hardware design.

3.3 Neuromorphic Architecture

In this section, we briefly review the operation of a typical neuromorphic architecture (Section 3.3.1), and examine properties of memristor based circuits, that can be exploited to harm the system (Section 3.3.2). Figure 3.4 illustrates a typical spiking neural network

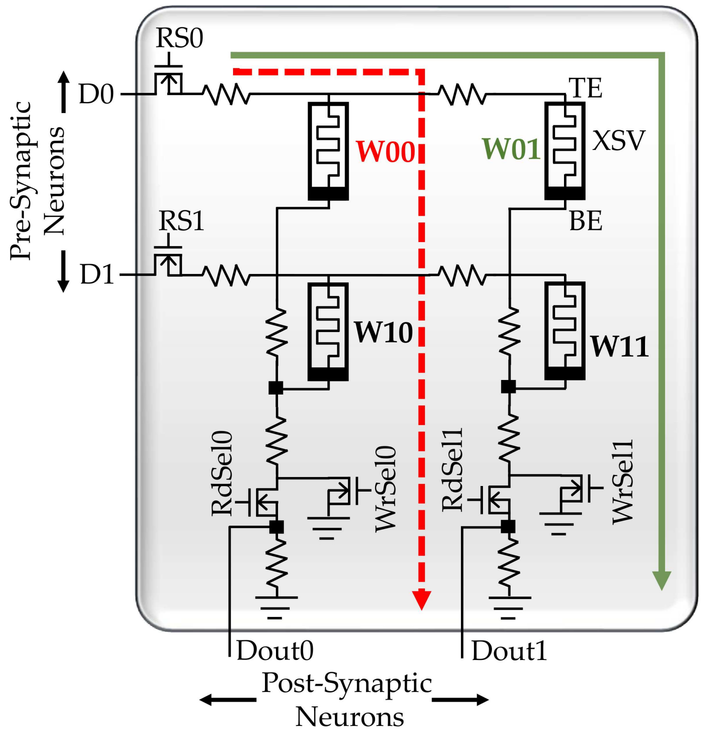


Fig. 3.6: Schematic of a 2 × 2 memristor crossbar array, to demonstrate the inherent properties that can be exploited by external attacks.

(SNN) architecture, based on IBM's truenorth [6], at three different levels of abstraction: *system, neurosynaptic core and memristor crossbar array.*

- At the system level (Figure 3.4(a)), the neuromorphic hardware is composed of many simple distributed processing units (neurosynaptic cores) that are sparsely connected and operate in parallel. Communication between the neurosynaptic cores (NS) is handled by a two-dimensional mesh network-on-chip interconnect fabric. The NS communicate using simple information packets which encompasses spikes (simplest possible temporal message) and the address to route the message over the network.

- NS (shown in Figure 3.4(b)) mimics the biological neuronal functionality and overcome the bottlenecks of a conventional von-neumann architecture. Each NS has its own local memory. Further, each NS includes a router, to route spike packets to other NS, and an axon scheduler to schedule spike events on selected axons based on incoming

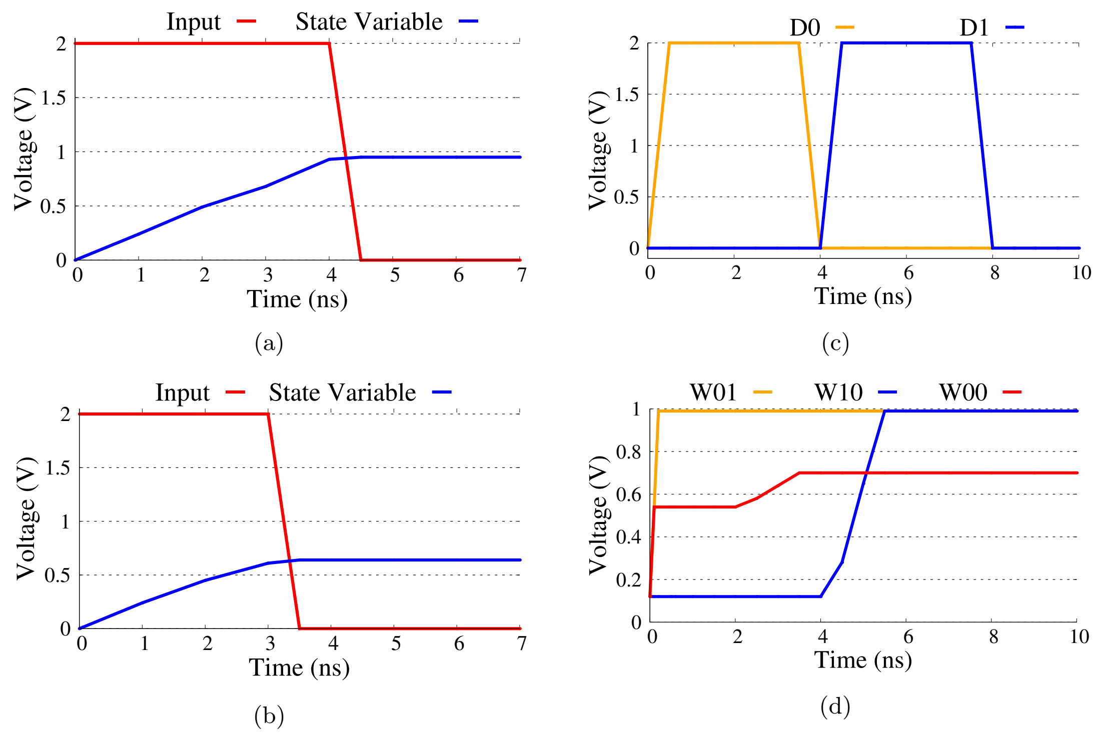


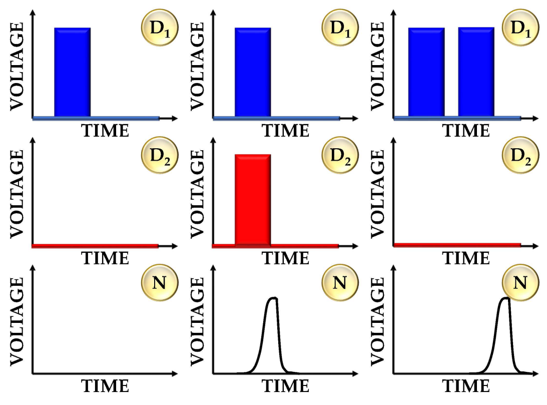
Fig. 3.7: Illustrates the inherent properties of a 2×2 memristor crossbar array, that can be exploited by external attacks. Figure 3.7(a) exhibits the non-volatile property of a memristor (state maintained even after voltage falls to zero). Figures 3.7(a) and 3.7(b), illustrate that the state value of the memristor explicitly depends on the flux injection (change in pulse width changes memristor state value). Finally, Figures 3.7(c) and 3.7(d) show the presence of parasitic leakage paths (red line in Figure 3.6) that has undesired effects on the memristor state ($W00$).

packets. From a functional standpoint, each NS has individually addressable axons, configurable synaptic crossbar and programmable neurons.

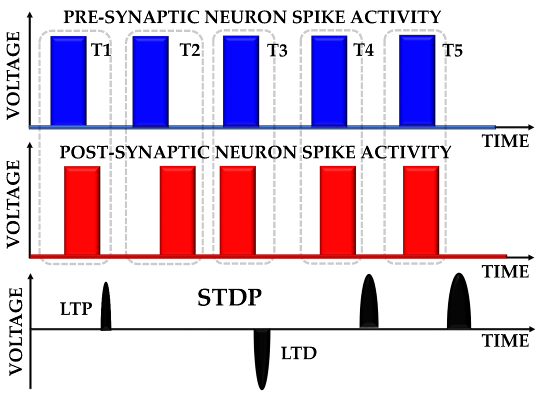
- The synapse unit (Figure 3.4(c)) consists of a $N \times N$ memristive crossbar array, along with other interface circuits. Each input neuron is connected to another output neuron with a two terminal memristor. The crossbar represents a recurrent network topology and can store N^2 synaptic weights among the N neurons. For our discussion, we use an integrate and fire model (discussed in Section 3.3.1) to capture the transient spiking of the biological neuron. Each neuron element tracks the membrane potential and has spike buffers to store input spikes arriving from other NS, as well as, the output spikes generated when the membrane potential reaches the threshold.

3.3.1 Operation

The dynamics of a neuron can be explained using a well known formal spiking model known as the leaky integrate and fire (LIF) [50–52]. The two basic principles of the LIF model are : *(a)* the membrane potential of the neuron evolves based on the prevalence of input spikes, and *(b)* neurons generate a spike event when the membrane potential breaches a set threshold. Figure 3.5 presents the equivalent circuit and the simulation of a single LIF neuron. When the integral of injected charge breaches the threshold, the neuron fires and the membrane potential is reset. When there is no charge injection in a time window, the membrane potential decreases (leaky phase). To model the biological neuron, a refractory period (delay) is considered during which the membrane potential does not increase, no matter how strong a stimulus is provided.



(a) Spatial and temporal summation


PRE-SYNAPTIC NEURON SPIKE ACTIVITY
T1
T2
T3
T4
T5
POST-SYNAPTIC NEURON SPIKE ACTIVITY
VOLTAGE
TIME
STDP
LTP
LTD

(b) STDP: synaptic weight update

Fig. 3.8: Properties of memristor based neuron design, and crossbar structure with sneak path mitigation mechanism. Figure 3.8(a) illustrates that the neuron can perform temporal and spatial summation of injected voltage. Figure 3.8(b) demonstrates the synaptic update stage following the STDP postulate.

Figure 3.4 illustrates a typical journey of a spike packet. The spike travels from axons (horizontal lines), through the active memristor based synapses in the crossbar to drive the inputs for the neurons (shown in Figure 3.4(c)). Axons are activated by spike events arriving from an external stimulus, or other post-synaptic neurons. When a neuron on an NS spikes, the router packetizes the spike event, destination address of the correlated axon,

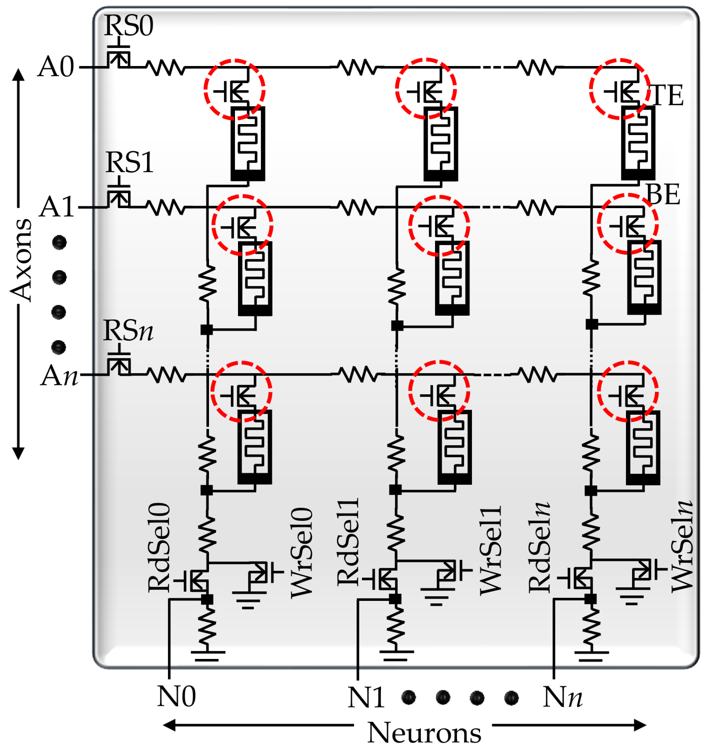


Fig. 3.9: Schematic of a 1 transistor - 1 memristor (1T-1M) based crossbar design to mitigate the sneak paths.

and the associated delay. The spike packet is injected into the interconnection fabric and directed towards the destination address (Figure 3.4(a)). Once the spike packet arrives at the destination NS, the router depacketizes and forwards the event to the axon scheduler, which then schedules it on the specific axon with the associated delay.

The temporal correlations between the spikes in pre-synaptic axon and post-synaptic neurons adjust the strength of the connections known as spike timing dependent plasticity (STDP). Consistent with Hebb's postulate for unsupervised learning, stronger synaptic connections increases the likelihood of pre-synaptic neuron induced firing of post-synaptic neuron [53].

3.3.2 Exploitable Properties

Inherent device/circuit properties can often expose vulnerabilities that can be exploited by external hacks during the system's operational lifetime. Knowledge of the implementation specifics is essential to root out vulnerabilities that may be exposed to the user environment. We present a few properties of neuromorphic design implementation using Figure 3.7 and Figure 3.8, that we use to frame the attacks.

- Figure 3.7(a) shows that the memristor maintains its state even when the supply voltage is removed (non-volatility), leaving it vulnerable to privacy attacks by probing.

- Figures 3.7(a) and 3.7(b), together demonstrate that the memristor state is a function of the flux injection (area under the voltage curve defined by pulse width and amplitude). Hence, altering the state by manipulating the voltage amplitude or pulse width, can harm system integrity.

- In Figure 3.6, when a voltage is applied across *D0* (Figure 3.7(c)), and *WrSel1* is enabled, the state of memristor *W01* changes (Figure 3.7(d)). However, we also observe an undesired state change in *W00*, due to the parasitic leakage paths (sneak paths). If left unmitigated, attacks can alter the synaptic weights stored as memristor state value.

- In SNN, the neurons are capable of performing spatial and temporal summation. In Figure 3.8(a), the stimulus from one dendrite (D_1) is insufficient for the neuron to fire. However, the aggregation of inputs from dendrites, D_1 and D_2, either spatially (i.e., stimulus from multiple dendrites) or temporally (i.e., stimulus accumulation in time window) can cause the neuron to fire.

- In SNN, the memristor synapse updates its state only when the pre-synaptic and post-synaptic neurons spike at close points in time. Figure 3.8(b) simulates the STDP process, and shows how voltages with different magnitude, polarity and pulse width are applied to the synapse. In window *T1, T4 and T5*, the pre-synaptic neuron fires before the post-synaptic neuron resulting in long term potentiation (LTP), whereas the opposite behavior in *T3*, results in long term depression (LTD). In window *T2*, the pre and post-synaptic neurons fire too far apart to update the memristor state.

3.4 Security Vulnerabilities

In this section, we present three attacks targeting crucial components of the neuromorphic design: *suppress neuron firing* (Section 3.4.1), *suppress spike communication* (Section 3.4.2), and *exploit crossbar sneak paths* (Section 3.4.3). For each attack, we present a brief evaluation on the impact of a hardware Trojan attack using a representative application.

3.4.1 Attack 1: Suppress Neuron Firing

In typical neuromorphic architectures, the neuron's overall firing rate is regulated within the operating boundaries, even with undesired changes such as temperature drifts, by a process called homeostasis. Homeostasis is implemented using automatic gain control (AGC) mechanisms to scale the synaptic weights and modulate the threshold voltage. Figure 3.10, illustrates the block diagram of an AGC loop based on the work presented in [54]. Additionally, the interface circuits allow individual control of the refractory period of each neuron. Programmable biases, and current sources can be used to reset the membrane potential for a fixed period, during which it will not be possible to stimulate the neuron.

We envisage a neuron suppression attack by manipulating the neuron threshold and the refractory period. A Trojan embedded in the gain control loop can maliciously scale the synaptic currents to produce output spikes at a firing rate proportional to the amplitude of the malicious control voltage. The Trojan in the feedback control affects the STDP resulting in compromised learning. Further, a minor modification is built into the neuron circuit to explicitly manipulate the refractory period and produce a wide range of spiking behavior.

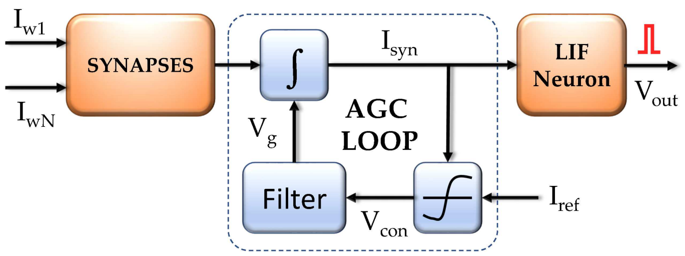


Fig. 3.10: Automatic gain control to maintain homeostasis.

3.4.2 Attack 2: Suppress Spike Communication

At the heart of neuromorphic architectures is a scalable network-on-chip (NoC) communication fabric that connects the highly parallel and distributed cores. NoCs designed for neuromorphic systems, present new challenges, as they are required to support direct axon addressing, time multiplexing, multicast routing, and must have very low latency. Previous works on NoC security in an untrustworthy design environment demonstrates the existence of security vulnerabilities [24,55]. The attack vector pursued by Rajesh et al. in [55] is very relevant here, and can prove to be more potent due to the added design complexity.

We adopt the attack vector from [55], where a rogue NoC has a malicious circuit embedded in it to inflict selective denial of service (DoS) attack, to a subset of the neuronal spike communication. The hardware Trojan will classify the neuron pairs, based on the most frequent firing rate during the presentation of a defined number of labeling samples. Once, a small set of important spike communication packets are identified, the Trojan will covertly deny/delay the delivery of packets affecting the STDP of the neurons.

3.4.3 Attack 3: Exploit Crossbar Sneak Paths

Sneak paths are associated with problems of increased power consumption, undesired state switching of neighboring memristors, and limiting the array size due to voltage drop away from the driver. In SNN, each of these effects can be exploited by external attacks. We envisage an attack exploiting the neighboring cell switching due to leakage current flowing through the sneak paths.

The attacker controls the input spike sequences by selecting appropriate input patterns (images), to cause undesired state changes in the crossbar array. The stream of input spikes establishes a current flow in the array and continually trains the network by updating the synapses on the path between the pre and post-synaptic neurons. However, the current leaks through alternate paths (sneak paths), and continually injects small pulses of charge. When sufficient charge is accumulated the sneak paths cause undesired neurons to fire, and hence alter the states of memristors in the vicinity of the actual path. In addition, the attacker introduces a systematic bias in the neuron synaptic mapping by causing read

Parameters	**Configuration**
Input Neurons	784
Output Neurons	$30 - 300$
Neuron Model	Linear-Integrate-Fire
Synapse Model	Simplified-STDP [57]
STDP window	$50ms$
Threshold voltage	$35mV$

Table 3.1: Baseline SNN configuration.

current variations. The undesired state changes, and the bias in the mapping will lessen the inference precision.

Contemporary works on memristor crossbar arrays, have proposed several sneak path mitigation techniques [56]. Figure 3.9 illustrates one such mechanism, where an additional gating transistor is added to each memristor to remove the alternate paths for current flow. Our SPICE simulations reveal that the mechanism is able to reduce charge through sneak paths nearly 20% of the original, but cannot fully eradicate it, due to transistor leakage. Hence, the crossbar is still vulnerable to the attack discussed above.

3.5 Evaluation Methodology

In this Section, discuss the tools, application and metrics used to evaluate our proposed attacks. Section 3.5.1 summarizes our simulation infrastructure. Section 3.5.2 introduces the benchmark application, and Section 3.5.3 elaborates on the metrics

3.5.1 Simulation Infrastructure

We employed a cross-layer simulation framework, described below, to investigate relevant memristor properties, and abstract the hardware trojan behavior in an SNN simulator.

- ***Device Layer:*** We use memristor SPICE model cards [58] to analyze the V-I characteristics and performed transient analysis, considering known device variations. We also investigated the mathematical models and abstractions to gain a comprehensive device physics characteristics and its associated properties.

- ***Circuit Layer:*** We use LTSpice simulator to implement memristor crossbar circuits of various sizes, and analyzed the sneak paths property. We carry forward these observations into the architecture layer by modeling their abstract behavior in the memristor-based spiking neural networks.

- ***Architecture Layer:*** We employ N2S3 simulator to model the proposed attack characteristics [59, 60] and evaluate the impact on image recognition application [61]. N2S3 supports various neuron models including leaky-integrate-fire and the Izhikevich models), as well as, synapse models such as Standard-STDP, Simplified-STDP. Querlioz et al. have proposed the methodology to employ the memristor device for storing the synapse weights for spiking neural networks [57].

3.5.2 Handwritten Image Recognition

We use *handwritten image recognition* application (MNIST dataset) to demonstrate the consequences of the proposed attacks. MNIST dataset consists of 28×28 pixel, 8-bit grayscale images of the handwritten digits (0 to 9). MNIST training dataset has $60,000$ handwritten images, whereas the testing set consists of $10,000$ images [60–62]. Table 3.1 demonstrate the configuration parameters used to develop a single layer neural network for the evaluation of our proposed attacks.

3.5.3 Evaluation Metrics

To fully comprehend the influence of the attacks in a neuromorphic system, we have identified two metrics: (i) *recognition rate*, and (ii) *malicious misprediction rate*, to evaluate the potency of our proposed attacks accurately. The recognition rate (R_r) and malicious misprediction rate (MM_r) can be defined as,

$$R_r = \frac{Tf_{cp}}{T_s}; \;\; MM_r = \frac{Ti_{mp}}{Tf_{cp}} + \frac{Ti_{mmp}}{T_s - Tf_{cp}}; \tag{3.1}$$

where T_s is the total samples used for evaluation, Tf_{cp} is the samples correctly predicted without the influence of attack (Trojan-free), Ti_{mp} is falsely mispredicted samples due to

Output Neurons	No Attacks	Attack 1	Attack 2	Attack 3
30	74.02	50.22	39.92	35.03
50	78.13	54.17	42.30	40.68
100	83.79	58.04	43.30	48.01
300	87.39	59.88	46.37	55.02

Table 3.2: Recognition Rate (R_r) of baseline system and our proposed attacks.

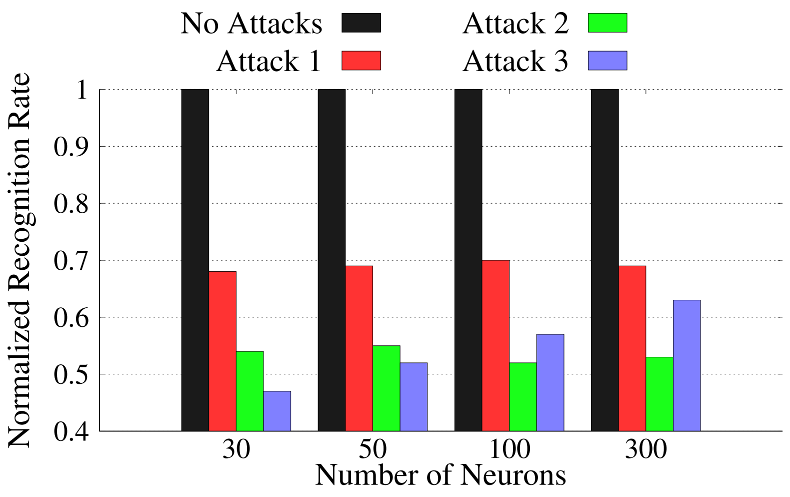


Fig. 3.11: Normalized image recognition of the three attacks (higher is better accuracy): **suppress neuron firing** (red bar), **suppress spike communication** (green bar) and **exploit crossbar sneak paths** (blue bar). Under attack, the recognition rate drops significantly.

attack (Trojan-injected) and Ti_{mmp} is falsely predicted samples due to attack.

3.6 Experimental Results

In this section, we evaluate the potency of our proposed attacks, *Suppress Neuron Firing Attack* (in Section 3.6.1), *Suppress Spike Communication Attack* (in Section 3.6.2) and *Exploit Crossbar Sneak Paths Attack* (in Section 3.6.3), using metrics presented in Section 3.5.3. To obtain a comprehensive understanding of attack potency, we evaluate the impact of attacks on various sizes of neural network output layer (30, 50, 100 and 300 output neurons). Table 3.2 showcases the baseline recognition rate under the ”No Attacks” column, and reveals that as the network size increases, the recognition rate also improves. Figure 3.12(a) shows the heatmap of synaptic weights after learning the MNIST digits.

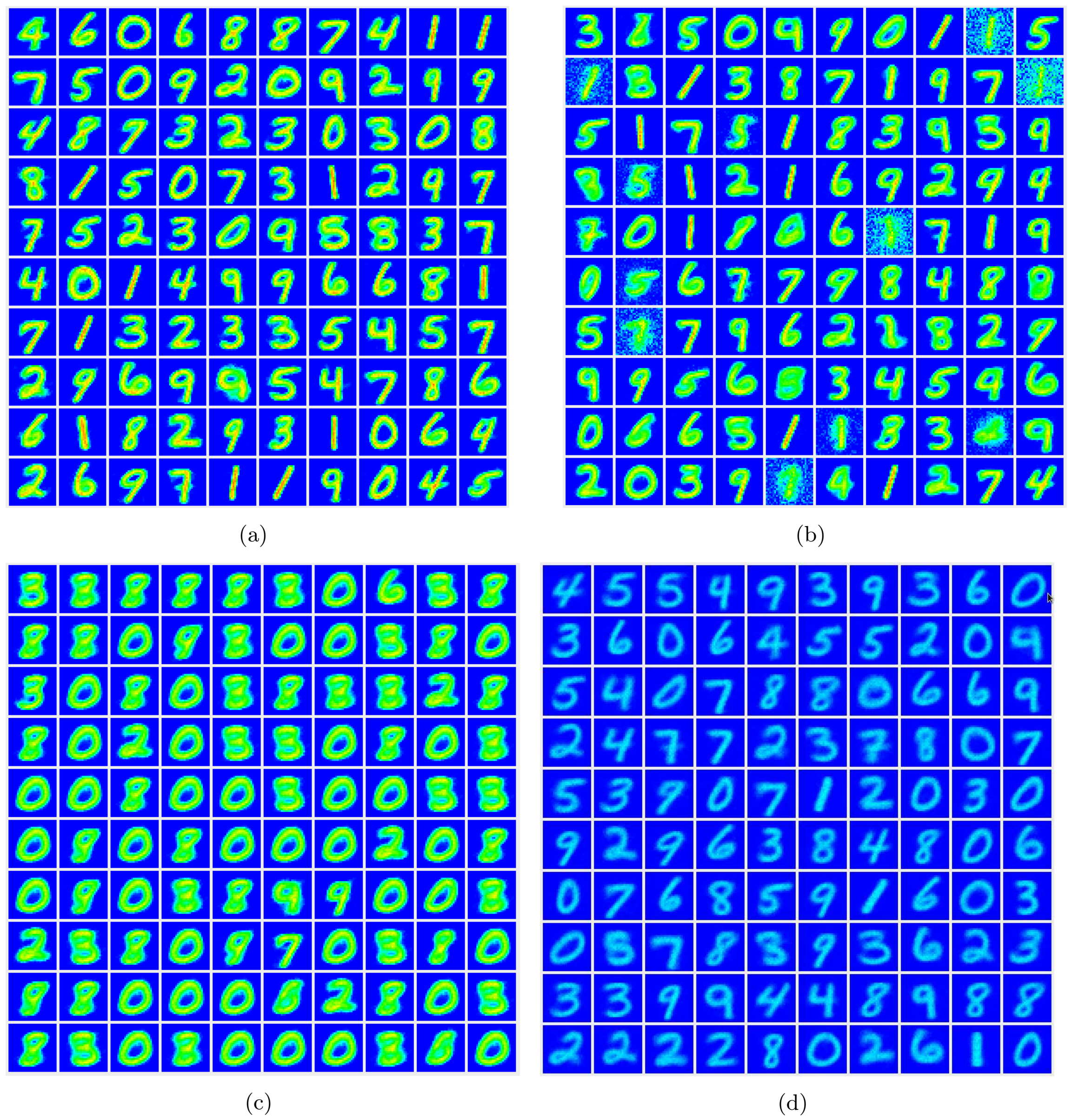


Fig. 3.12: Potency of the three proposed attacks. Figure 3.12(a) shows the heat map of 100 output neurons after ideal Trojan free network training. Figure 3.12(b) illustrates the heat map of 100 output neurons under the **suppress neuron attack**. Figure 3.12(c) illustrates the heat map of 100 output neurons after network training under **suppress spike communication attack**. Figure 3.12(d) illustrates the heat map of 100 output neurons after network training under **exploit crossbar sneak paths attack**.

3.6.1 Attack 1: Suppress Neuron Firing

Column *Attack 1* in Table 3.2 reveals the impact on recognition rate due to suppress neuron firing attack. On an average, the recognition rate of the system under attack is

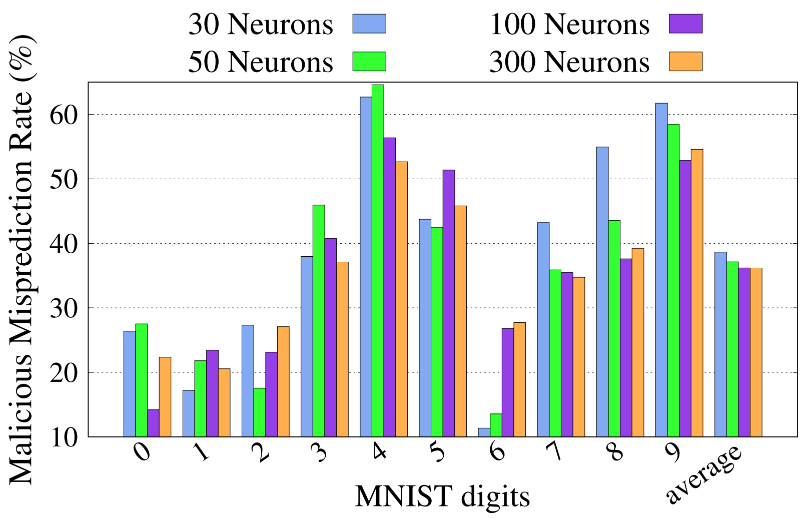


Fig. 3.13: Attack induced misprediction rate signifies the increase in mispredictions caused specifically due to a Trojan attack on the SNN. Figure 3.13 shows misprediction rate caused distinctively due to the **suppress neuron firing** attack.

at 55.58%. Figure 3.11 (red bar) demonstrates the normalized drop in recognition rate of the MNIST dataset. On an average, the recognition rate suffers by 30% due to the attack, across the different sizes of output layers. Figure 3.13 projects a more analytic view of the attack potency, as it charts the attack induced *malicious misprediction rate* for the different MNIST digits. By suppressing a subset of the neurons from firing, we observe that the degree of attack varies significantly across the digits. SNN mapping, in conjunction with the neurons suppressed during the attack, determine the impact on the malicious misprediction rate for different handwritten digits. We notice that recognition of digits 4, 5, 8 and 9 are severely affected in our experiments, as compared to other digits.

Figure 3.12(b) shows the heat map of synaptic weights after training for an neural network with 100 output neurons. The Trojan suppresses certain neurons from firing correctly, thereby affecting the training of synaptic weights. From Figure 3.12(b), we can observe that there is uncertainty in recognizing handwritten digits 4, 5 and 9.

3.6.2 Attack 2: Suppress Spike Communication

Column *Attack 2* in Table 3.2 demonstrates the hit on recognition rate due to a Trojan

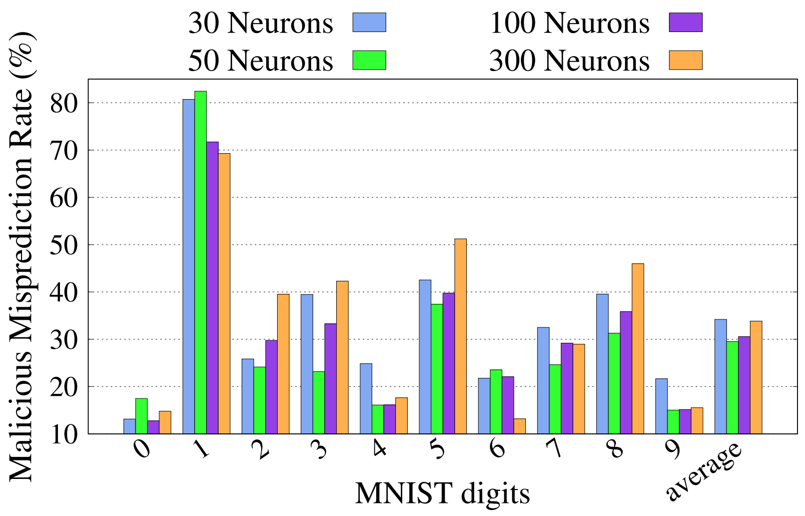


Fig. 3.14: Attack induced misprediction rate signifies the increase in mispredictions caused specifically due to a Trojan attack on the SNN. Figure 3.14 illustrates the misprediction rate induced individually by the **suppress spike communication** attack.

in the NoC router suppressing specific inter-neuron communication. The recognition rate of the SNN is degraded to 42.97% due to the attack. Figure 3.11 (green bar) demonstrates the normalized degradation in the recognition of MNIST handwritten digits. We observe, that on an average, the recognition rate degrades by 48%.

Figure 3.14 shows a detailed impact of the attack, by charting the attack induced *misprediction rate* for different digits of the MNIST dataset. We observe that, in our simulations, the suppression of spike communication has had a smaller impact on digits 0 and 6, when compared to other digits. Figure 3.12(c) shows the heat map of synaptic weights after network training for an SNN with 100 output neurons. Comparing the figure with the Trojan free version (Figure 3.12(a)), we observe that several digits are not trained and do not appear in the heat map. Figure 3.12(c) represents an extreme case of DoS attack, and different flavors of DoS can be realized.

3.6.3 Attack 3: Exploit Crossbar Sneak Paths

Column *Attack 3* in Table 3.2 shows the recognition rate due to the attack exploiting the crossbar sneak paths. Since the attack affects the synaptic weights stored in the memristor

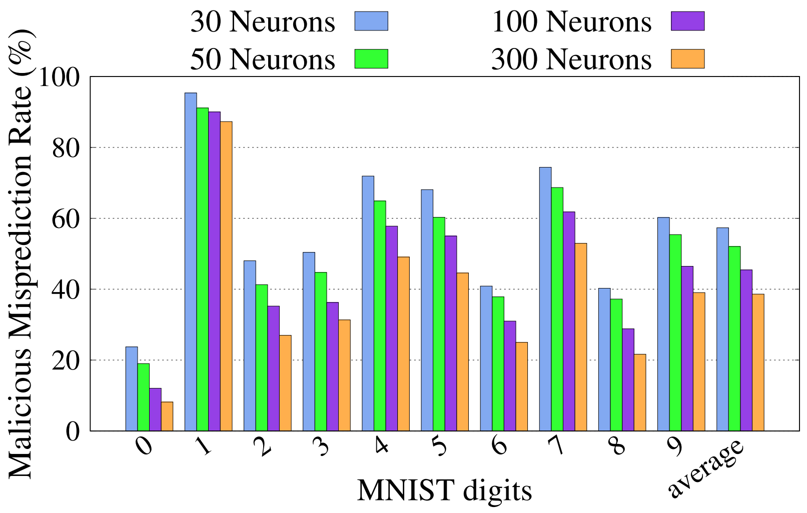


Fig. 3.15: Attack induced misprediction rate signifies the increase in mispredictions caused specifically due to a Trojan attack on the SNN. Figure 3.15 reveals the misprediction rate inflicted by **exploiting crossbar sneak paths**.

crossbar, exploiting this property leads to significant reduction in recognition rate. On an average, the recognition rate of SNN is 44.68%. As the size of the neural network increases, the impact of cross bar sneakpaths become more localized, thereby reducing the impact of the attack. Figure 3.11 (blue bar) illustrates the normalized recognition rate degradation due the attack exploiting the crossbars sneak paths.

Figure 3.15 shows the in-depth analysis of the attack, by charting out the attack induced *malicious misprediction rate* for different MNIST digits. Figure 3.12(c) shows the heat map of synaptic weights after network training for an SNN with 100 output neurons. Comparing with the Trojan free (Figure 3.12(a)), we notice that strength of the synaptic weights across the output neurons are weak and hence, MNIST digits are improperly trained due to crossbar sneak paths attack.

CHAPTER 4
REVIVING TRUST IN 3PIP BY DETECTING TASP TROJANS

4.1 Background and Contributions of This Work

The growing popularity of Third Party Intellectual Property (3PIP) components in Multi-Processor System-on-Chips (MPSoCs) is posing significant challenges for secure hardware design [7]. Malicious functionality in a 3PIP, or hardware trojans, can be used for causing a range of harms at the system/application level covering the three central pillars of security assurance: confidentiality, integrity, and availability. For example, the confidentiality of a MPSoC can be compromised if a hardware trojan gives the attackers back door access to sensitive information [63]. Likewise, other harmful effects of trojans include functionality change, performance degradation, and denial of service [64].

In this work, we explore *3PIP architectural state preserving trojans (TASP)*. These are trojans embedded in a 3PIP core that preserve the architected state of the processor similar to a trojan free core. Figure 4.1 illustrates the concept of a TASP relative to typical trojans. While most commonly studied trojans corrupt portions of the externally visible architected states, including the registers and the memory, a TASP does not. For example, a TASP might flip the prediction bit of branch prediction, or increase access latency in the cache. Such actions do not corrupt the architected state of the processor. Consequently, a carefully designed TASP may evade detection, while continuously degrading the performance delivered to an end-user.

A TASP brings unique challenges in detection and mitigation, despite its apparent similarity with performance degrading trojans described in the trojan taxonomy [64]. Two of the key challenges include: a) limited gate level visibility inside a 3PIP core, and b) false positives generated by noise in the system. First, most 3PIP cores are delivered as "black boxes" to protect the intellectual property of the third party [26]. Consequently,

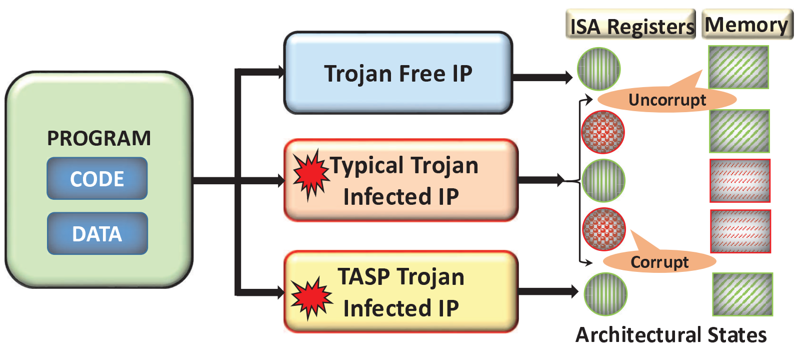


Fig. 4.1: Unique challenges of a TASP trojan.

malicious functionality embedded in these components can easily bypass many existing techniques for hardware trojan detection that rely on gate-level inspection [25]. Second, false positives can be flagged due to performance fluctuation from process variation effects and non-deterministic architectural events in a healthy core.

To detect such a potent threat in modern MPSoCs, we present three detection techniques covering a spectrum of TASP attacks. These comprise: (1) an analytical model based method to detect always-on TASPs; (2) a runtime technique to detect intermittent TASPs using another core; and (3) a runtime method to detect intermittent TASPs without using other cores.

We make following contributions in this work.

- We explore a potent threat model as 3PIPs grow in prominence in modern MPSoCs. We evaluate the performance impact of TASPs on a modern processor (Section 4.2).
- We propose a *Specification-centric Performance-aware Analytical Representation (SPAR)* that uses an analytical model to detect always-on TASPs (Section 4.3.2).
- We propose two runtime techniques: *Covert Performance Monitoring System (CPMS)* and *Employing Side-channel analysis for TASP Detection (ESTD)* to detect intermittent TASPs during the lifetime of the chip (Sections 4.3.3 and 4.3.4).
- We demonstrate the efficacy of *SPAR*, *CPMS*, and *ESTD* (Section 4.5).

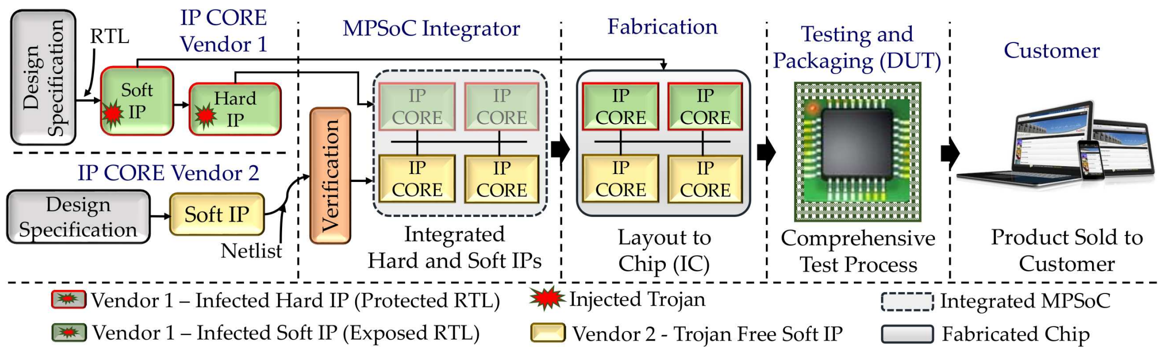


Fig. 4.2: TASP threat model in the context of the existing supply chain of a 3PIP block.

4.2 TASP: A Conceptual Overview

In this section, we first present a broad overview of a TASP, its relevance (Section 4.2.1) and its major classifications (Section 4.2.2). We then discuss the design of the TASP and its resulting damaging impact in Sections 4.2.3 and 4.2.4, respectively. Finally, we evaluate its design footprint in Section 4.2.5.

4.2.1 TASP Relevance in IC Supply Chain

Figure 4.2 illustrates the TASP threat model in the context of a typical supply chain for 3PIP integration in a MPSoC. To foster a compelling competitive edge, MPSoC integrators rely heavily on 3PIP vendors for common MPSoC blocks such as processing elements. 3PIP vendors may deliver these blocks as either technology independent soft IP (exposed RTL) or technology dependent hard IP (protected RTL). The hard IP—a preferred mode for 3PIP vendors—offers only a functional signature (input, output, and functional description) to the MPSoC integrator. The 3PIP vendors then provide the fully exposed RTL directly to the fabrication house, where the entire design can be seamlessly integrated and fabricated. This widely popular model has serious implications towards circuit level vulnerability. Without a golden model and gate level visibility, a TASP can easily bypass existing post-silicon verification methods implemented by the MPSoC integrator. A vendor or a malicious design engineer in the 3PIP design house can use this potent model to harm a MPSoC integrator, or other clients such as government agencies [65].

4.2.2 Types of TASP Trojans

TASPs can be classified into two major categories based on their activity period: always-on and intermittent.

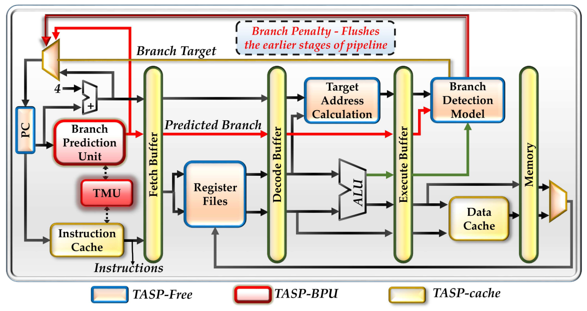
Always-on: An always-on TASP is active throughout the lifetime of the chip. The malicious designer in the 3PIP design house activates the TASP before the 3PIP core is delivered to MPSoC integrator. Moreover, these types of trojans are difficult to detect in a MPSoC system due to the absence of the golden model.

Intermittent: These TASPs are active sporadically, and thus interleave between inactive and active phases. Their active phase may be triggered through a variety of conditions, such as certain input patterns or external conditions (e.g., temperature). The period of active and inactive phases are assumed to be random as it is depends on discretion of that attacker. For the purpose of this work, we focus on the impact of intermittent TASPs when active and explore techniques to detect them irrespective of the trigger mechanisms.

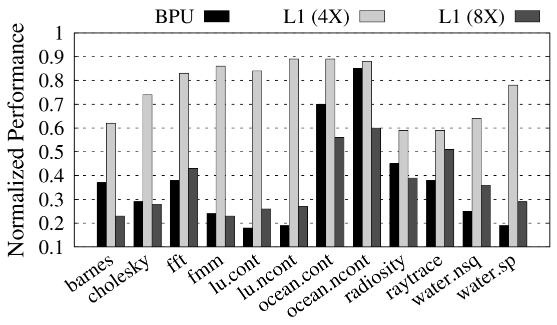
An intermittent TASP can have several major operational phases: (i) *Dormant*; (ii) *Activation*; (iii) *Operational*. In the dormant phase, the trojan is simply awaiting a specific trigger mechanism to become active [7]. Delaying this activation essentially allows the TASP to evade post-silicon detection. After activation, a TASP becomes operational, where it performs its intended function by altering internal data and control signals in the 3PIP core.

4.2.3 Design of a TASP

Figure 4.3(a) illustrates the design and operation of *TASP-BPU* and *TASP-cache*. *TASP Monitoring Unit (TMU)* monitors the instruction sequence being fetched from instruction cache. The TASP in the BPU, once activated, flips the branch prediction outcomes. As a result, incorrect instructions are fetched from I-cache and go through the various stages of pipeline for execution. At the end of execute stage, the branch detection model, shown in Figure 4.3(a), identifies the incorrect branch prediction and therefore, the instructions present in the fetch, decode and execute pipe stages are flushed. Then, the correct instructions is fetched from the I-cache and propagated through the pipeline. Similarly,



(a) Design of TASP.



(b) Performance Impact due to TASP.

Fig. 4.3: (a) TASP-BPU and TASP-cache Design and Operation. (b) Performance degradation from: (i) BPU TASP, (ii) L1 TASP-4X latency increase, (iii) LI TASP-8X latency increase.

TASP in L1 I-cache and L1 D-cache (shown in Figure 4.3(a)), once activated, increases the L1 cache access time. When the memory instructions, such as load and store, are executed, *TASP-cache* inserts additional latency when the L1 cache is accessed. As a consequence, the memory access time of those instructions are significantly affected. Both *TASP-BPU* and *TASP-cache* affects the performance of the 3PIP core without altering its architectural states such as register files and memory.

4.2.4 Performance Impact of a TASP

We outline our methodology and results to demonstrate the performance impact of a TASP.

Evaluation Methodology: Using the Sniper simulator [66], we inject TASPs in the Branch Prediction Unit (BPU) and the L1 cache of a modeled single core Intel i7-4650U Processor [67]. The TASP in the BPU flips the branch prediction bit at every prediction. Similarly, the TASP in the L1 cache increases the data access latency. We use the Splash2 benchmark suite [68] in our simulations.

Results: Figure 4.3(b) shows the performance from three different TASPs, normalized to the performance of a TASP free core. We find that the TASPs in the BPU, and the L1 cache with 4X and 8X slower access latencies, degrade the performance of the MPSoC system by 62%, 24%, and 64%, respectively. For the *lu.cont* and *lu.ncont* benchmarks, the

Metric	Baseline	TASP	Overhead
Area (μm^2)	30635.92	31757.21	3.66%
Power (mw)	382.50	383.75	0.32%

Table 4.1: Design footprint of the TASP infected core.

BPU TASP reduces the performance by 82% and 81%, while the 8X LI cache TASP reduces the performance by 74% and 73%, respectively.

4.2.5 Design Footprint

To evaluate the area and power overhead of adding a TASP in a 3PIP core, we modify the RTL core generated by Fabscalar [69]. We add the logic for the BPU and L1 TASPs. We synthesize the TASP injected core with the TSMC 45nm library using the Synopsys Design Compiler. Table 4.1 shows that the TASP has low area and power overheads making it difficult to detect.

4.2.6 Significance of TASPs in 3PIP cores

Our results demonstrate a compelling need for techniques to detect TASPs in 3PIP cores, presented in the next section.

4.3 TASP Detection Techniques

In this section, we describe the three TASP detection techniques. We first outline the key design challenges in detecting a TASP (Section 4.3.1). We next present our TASP detection techniques: (a) *SPAR* - that detects always-on TASPs using an analytical method (Section 4.3.2); (b) *CPMS* - that detects an intermittent TASP using another comparative core (Section 4.3.3); and (c) *ESTD* - that detects intermittent TASPs without any other cores. Finally, we evaluate the performance overhead of CPMS and ESTD in Section 4.3.5.

4.3.1 Design Challenges

Limited RTL Information: The RTL of a 3PIP core is protected, and unavailable to the system integrator. The system integrator can only use the IP specifications or monitor the activity at the external interface of the core to detect a TASP.

Lack of a Golden Model: Since the 3PIP core is obtained from an external design house, there is no ideal model to compare its behavior with.

Managing Overheads: A practical detection scheme must limit the performance, power, and energy overheads.

4.3.2 SPAR

In this section, we present an overview of SPAR—an analytical technique to detect always-on TASPs, outline SPAR models for TASP free and TASP infected cores, and validate SPAR.

Overview

Many 3PIP providers offer detailed processor specifications for their IP [70, 71]. Based on these specifications, we develop an analytical model to characterize the performance of a trojan-free incarnation of the target processing core. Subsequently, we augment this model to analytically model different TASPs, so that we can estimate their expected performance characterizations. Using this technique, we are therefore able to estimate the performance of 3PIP cores *without a golden model*. Comparing observed performance with our models presents a way to detect always-on TASPs.

CPI for TASP Free Cores

The key insight of our proposed technique is to recognize that the processor performance is heavily dependent on predictive structures like the BPU and caches [72]. Thus, we can estimate the *Cycles per Instruction (CPI)*—a standard performance metric—by discounting the impact of realistic predictors from the estimate using oracle prediction. The expected performance (CPI_{exp}) of a 3PIP core is:

$$CPI_{exp} = CPI_{oracle} + CPI_{cache} + CPI_{bpu} \tag{4.1}$$

where CPI_{oracle} is the perfect performance of the 3PIP core. CPI_{cache} and CPI_{bpu} are the performance penalties due to the cache and the BPU, respectively.

We estimate the CPI_{exp} using the architectural parameters from the IP specification and application parameters generated by the compiler [73].

$$CPI_{exp} = \frac{1}{N}\{\frac{N}{D} + \frac{D-1}{2D}(c_m + br_{mp}) + ld_p + st_p + br_p\} \tag{4.2}$$

where D is the processor dispatch width and N is the total instruction count. c_m and br_{mp} are the cache misses and branch mispredictions, respectively, used to model the effect of inherent dispatch inefficiency caused by miss events [73]. ld_p, st_p, and br_p are the time penalties due to loads, stores, and branch mispredictions, respectively. The parameters for both miss events and penalties are generated by the compiler using dynamic and static prediction [74].

CPI for TASP Infected Cores

The TASP activity in the BPU affects the branch misprediction events in a processor. This BPU TASP misprediction (br_{mpTASP}) can be modeled using:

$$br_{mpTASP} = br_{mp} + n_{TASP} \tag{4.3}$$

where n_{TASP} is the additional mispredictions induced by the TASP. Based on the type of TASP, we can adjust n_{TASP} to reflect 100% mispredictions or other high misprediction rates. CPI_{TASP} for the infected BPU can be calculated by substituting br_{mpTASP} in place of br_{mp} in Equation 4.2.

The TASP activity in the L1 cache can be modeled using:

$$ld_{TASP} = ld_p + TASP_{lat}; \;\; st_{TASP} = st_p + TASP_{lat} \tag{4.4}$$

where $TASP_{lat}$ is the additional latency induced by the TASP for both a cache hit and a cache miss. ld_{TASP} and st_{TASP} are the load and store penalties of the infected core

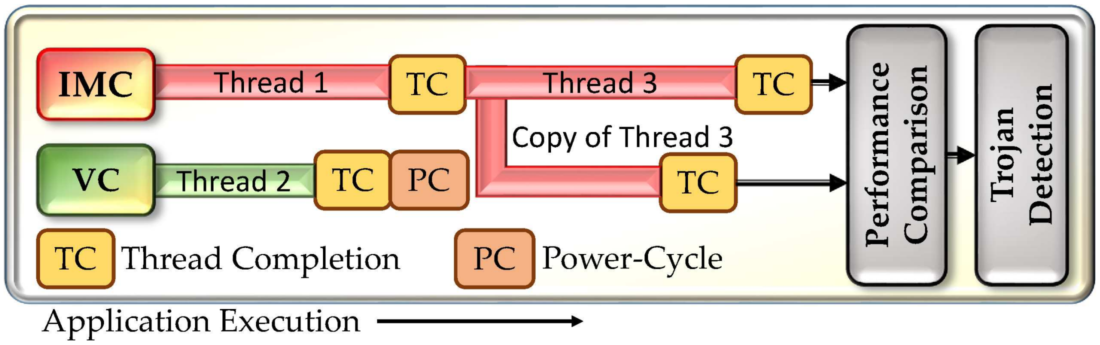


Fig. 4.4: Detection phase of the CPMS.

respectively. We substitute ld_{TASP} and st_{TASP} for ld_p and st_p in Equation 4.2 to estimate the CPI_{TASP} for an infected L1 cache.

Validation: For a specific application, we extract the application parameters from the compiler and the architectural parameters from the IP specification to calculate the *CPI* using Equation 4.2. Next, we model the 3PIP core in Sniper [66] using the architectural parameters provided in the IP specification. We then simulate the same application on the modeled core and determine the *CPI*. We validate SPAR by comparing the calculated *CPI* with the simulated *CPI*.

4.3.3 CPMS

To detect intermittent TASPs, CPMS periodically repurposes one of the on-chip cores to monitor another on-chip core and compares their performance on the same workload. The active and inactive phases of the TASP is varied at runtime depending upon the attacker discretion. However, during the overlap period between the active TASP and detection period, CPMS is able to identify the active TASPs. We next outline the CPMS comparative core choices, its operational phases, and its TASP detection threshold.

Comparative Core Choices: In the first step of CPMS, we identify a core, referred as *Intensive Monitoring Core (IMC)*. To detect a TASP in the IMC, we choose another on-chip core, termed as the *Validation Core* (VC), which is functionally and micro-architecturally identical to the IMC. Consequently, when both cores are free of TASPs, we expect comparable performance in them. In practice, either/both cores may have an active TASP.

Operational Phases:

- **Inactive Phase (Phase 1):** CPMS is inactive, and all cores operate normally to maximize performance.

- **Detection Phase (Phase 2):** Figure 4.4 shows the operational flow of CPMS. First, CPMS identifies the VC with low utilization and reschedule the scheduled tasks to another core to assure that no new tasks are scheduled on the VC. After the task completion, once idle, it is power-gated to erase its internal state. Next, when a new thread is scheduled on the IMC, the operating system also schedules a copy of the thread on the VC. Both cores then execute identical workloads in separate memory spaces to avoid data races. To detect the presence of a TASP, we compare the performance difference from these cores using an estimated threshold.

Action Post Detection: If a TASP is detected, the IMC is power-gated and the TASP is flagged to the operating system. Subsequently, the validation core continues to execute the program initially assigned to the IMC for continuous program execution. If no TASP activity is identified, the VC stops mirroring the IMC and returns to Phase 1.

TASP Detection Threshold: One of the key design challenges in CPMS is identifying a practical threshold for detecting anomalies between a VC and an IMC under attack. Having a lower threshold can increase false positives as noises like voltage, frequency fluctuation, or manufacturing variation can affect a core's performance. On the other hand, having a higher threshold can increase false negatives by occasionally discounting an active TASP as a noise.

$$\chi^2 = \sum_{k=1}^{n} \frac{(O_k - E_k)^2}{E_k} \tag{4.5}$$

We use a Chi-Squared (χ^2) distribution across different frequencies to estimate the goodness of fit between the VC and the IMC. Equation 4.5 estimates the χ^2 value, where O_k indicates the observed value (performance of IMC) and E_k indicates the expected value (performance of VC) for n degrees of freedom [75]. The n degrees of freedom are the different threads which are executed on the VC and the IMC. Using the χ^2 value, we determine

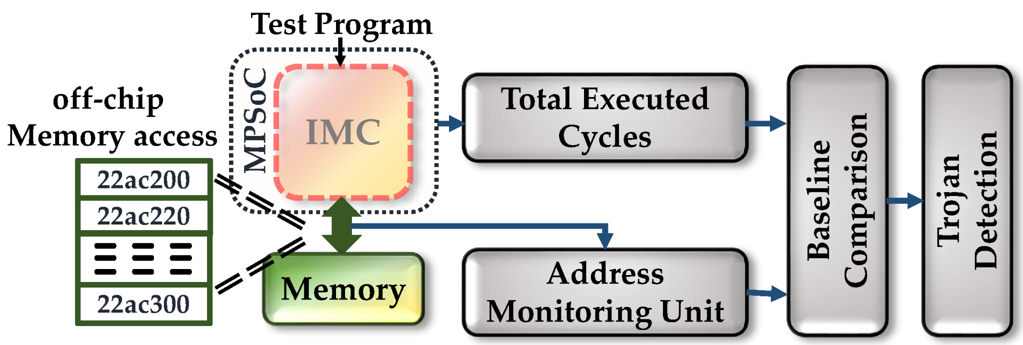


Fig. 4.5: Test program execution phase of the ESTD.

the probability density function (PDF). If the PDF between the VC and the IMC is lower than a certain value, it indicates a significant deviation in their performance with a high confidence, suggesting potential TASP activity [75].

If both VC and IMC contain an active TASP, CPMS cannot identify the TASP activity. However, this limitation is overcome by employing ESTD which has ability to identify the TASP activity in the absence of a VC.

4.3.4 ESTD

The ESTD technique employs a novel side-channel approach to detect intermittent TASPs in the absence of a VC. ESTD overcomes the limited RTL visibility of a 3PIP core by relying on the memory access patterns of the IMC. We next outline the baseline estimation, and detail the operational phases of ESTD.

Baseline Estimation: ESTD runs a test program on a 3PIP core. This test program is specifically designed to have very predictable memory access patterns in uncompromised cores. Algorithm 1 presents an example of such a test program. This test program includes: (1) a predictable branching pattern; (2) every predictable branch accessing a location in the cache; and (3) every mispredicted branch resulting in a cache miss and subsequent memory access. The memory access patterns of this program will be substantially different in the presence or absence of an active *TASP-BPU*. Likewise, the number of cycles of this program will widely vary in the presence and absence of an active L1 TASP.

Algorithm 1 ESTD test program flow

```
procedure PredictableBranch(array $x$, bool Prediction)
    //ideally use non-linear memory structure (e.g. c++ list)
    type localVar1
    for $i = 1 \rightarrow k$ do
        if Prediction then
            //access same cache location
            localVar1 = 10
        else
            // access different cache block each iteration
            localVar1 = $x[i * CacheBlockSize]$
        end if
    end for
    return localVariable
end procedure
```

After SPAR has ruled out always-on TASPs, the ESTD test program is executed to capture the total executed cycles along with the memory access patterns as a baseline reference. The limitation of the ESTD is that *TASP-cache* must be active for certain period during the TASP detection phase.

Operational Phases:

- **Inactive Phase (Phase 1):** ESTD is inactive, and all cores operate normally to maximize performance. An IMC is also chosen in a cyclic fashion to ensure all system cores are tested.

- **Test Program Execution (Phase 2):** In phase 2 (Figure 4.5), computation on the chosen IMC is suspended or transferred to other cores in the system. The operating system then executes the test program on the IMC, and compares the total executed cycles and memory access patterns with the baseline. A significant anomaly flags active TASPs to the operating system and the system returns to phase 1.

4.3.5 Performance Overhead Estimation

The performance overhead of CPMS and ESTD depends on the length of the detection phase (L_D), and the interval between the detection phases (T_D). For both CPMS and

ESTD, T_D and L_D represent the amount of time the system spends in phase 1 and phase 2, respectively. The performance of the system (pf_{sys}) is calculated using following method,

$$pf_{sys} = \frac{T_D * pf_{p1} + L_D * pf_{p2}}{T_D + L_D}; \;\; pf_{od} = \frac{pf_{p1} - pf_{sys}}{pf_{p1}} \tag{4.6}$$

where pf_{p1} and pf_{p2} are the system performances in phase 1 and phase 2, and pf_{od} is the performance overhead.

4.4 Evaluation Methodology

In this section, we outline our architectural (Section 4.4.1), and our area/power estimation methodology (Section 4.4.2).

4.4.1 Architecture Configuration

We model an Intel i7-4650U Processor [67] using the Sniper Multi-Core simulator [66]. These cores are configured based on the parameters listed in Table 4.2. To simulate the effect of the TASP, we modify the behavior of the BPU and the L1 cache. We execute 12 benchmarks from the Splash2 Benchmark Suite [68] in *detailed mode*. Splash2 benchmark suite contains a set of diversified workloads that depicts the characterization of real world applications used in-the-field. Therefore, the results obtained from Splash2 simulations will not significantly vary for actual applications used-in-the-field. We simulate these benchmarks for 6 different frequency ranges specified for Intel i7-4650U (3.3GHz - 1.7GHz) [67].

Parameters	**Processor Configuration**
Processor	Intel i7-4650U
Cores	1 - 2
CPU Type	Out-of-Order Engine (OoO)
Frequencies	(1.7, 2.1, 2.4, 2.7, 3.0, 3.3) GHz
L1 I-Cache	32KB/4-way
L1 D-Cache	32KB/4-way
L2 Cache	512KB/8-way
L3 Cache	4MB/16-way
Memory	1GB

Table 4.2: Configuration parameters used to model a 3PIP core.

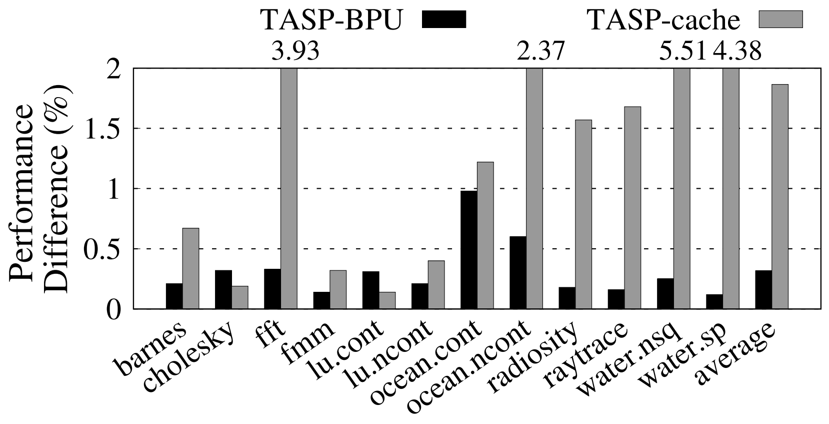


Fig. 4.6: SPAR Validation: Performance difference between SPAR and 3PIP core simulation for BPU and cache TASPs.

4.4.2 Estimating Area and Power Overhead

We design the additional hardware required for CPMS in Verilog HDL and synthesize using Design Compiler with the 45nm technology.

4.5 Experimental Results

In this section, we present the results of SPAR (Section 4.5.1), CPMS (Section 4.5.2), and ESTD (Section 4.5.3). We discuss the impact of intermittent TASP activity periods (Section 4.5.4) and then we present the performance, area, power, and energy overheads of CPMS and ESTD (Section 4.5.5 and 4.5.6).

4.5.1 SPAR

Figure 4.6 illustrates the differences in performance obtained from SPAR and 3PIP core simulation for Splash2 benchmarks. In Figure 4.6, *TASP-BPU* and *TASP-cache* indicate the performance difference between the SPAR technique and the 3PIP core simulation with the infected BPU and cache, respectively. The average performance difference for the TASP-BPU and the TASP-cache are 0.32% and 1.89%, respectively. The higher performance difference in TASP-cache is from the architectural simulator artifact, as Sniper itself shows 6% error in simulating an idealized cache [66]. Overall, the low performance difference shows that SPAR can effectively detect TASP induced performance degradation.

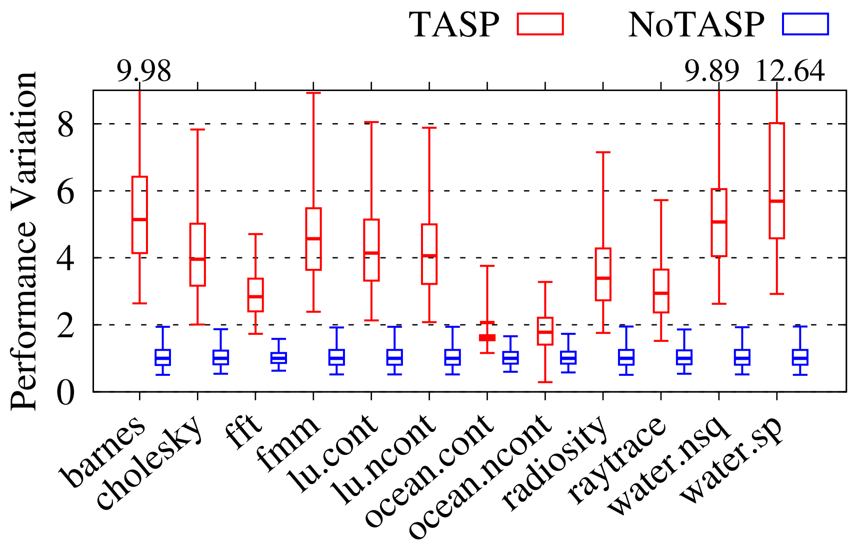


Fig. 4.7: Compares performance variation between TASP and NoTASP models. TASP indicates the variation between VC and infected IMC (TASP-BPU and TASP-cache), and NoTASP indicates the variation between VC and non-infected IMC. Data is shown as a box graph, indicating min, quartiles, and max.

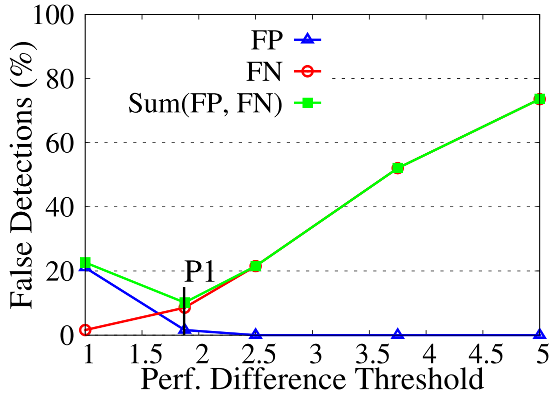
4.5.2 CPMS

Figure 4.7 compares the performance variation between TASP and NoTASP models across different p-states to illustrate the effect of internal noise. In the absence of a TASP, there is a small performance variation between the VC and the IMC caused by internal noise. However, in the presence of a TASP, the performance variation between the VC and the IMC is increased significantly by the additional branch mispredictions and longer cache access latencies.

Figure 4.8(a) illustrates the variation between the performance of the VC and the infected IMC using χ^2 distribution (Equation 4.5). A PDF of less than 0.05 indicates that there is a significant variation between the performance of the IMC and the VC. Figure 4.8(b) shows the false positives and false negatives of CPMS for different detection thresholds. In Figure 4.8(b), point P1 shows that the optimum detection threshold to identify the TASP is 1.85 with FP and FN of 2% and 9%, respectively. We therefore select the CPMS detection threshold to be 1.85. The *ocean* benchmark accounts for 6.94% of the total false negative detections. A large number of synchronization points in *ocean* obfuscate

Frequency	χ^2	PDF
3.3GHz	34.32	0.0006
3.0GHz	31.22	0.0018
2.7GHz	28.67	0.0044
2.4GHz	25.02	0.0147
2.1GHz	22.73	0.002
1.7GHz	20.83	0.05

(a)



(b)

Fig. 4.8: (a) Chi-Squared distribution for the performance of the VC and the infected IMC (TASP-BPU and TASP-cache). (b) Analysis of CPMS false positives and false negatives.

the impact of TASP, and hence degrade the effectiveness of CPMS.

4.5.3 ESTD

To ensure a good tradeoff between overhead and detection accuracy, we need to carefully select the length of the detection phase. Table 4.2(a) shows the excerpt of memory patterns accessed by the test program when executed on non-infected and infected IMCs. When both TASP-BPU and TASP-cache are inactive, the first load instruction of the test program accesses the data from the memory and the subsequent load instructions access the data from the cache. When TASP-BPU is active, it mispredicts most of the branches and results in accessing the data for all load instructions from the memory. When TASP-cache is active,

(a)

	IMC T-free	IMC T-BPU
1st ld	1f15010	22ac060
2nd ld	—	22ac080
3rd ld	—	22ac0a0
4th ld	—	22ac0c0

(b)

Loops	DRAM Access	Cycles
1K	6%	94%
10K	61%	121%
100K	616%	347%
1000K	22, 231%	1257%

Table 4.3: (a) Memory trace excerpt for the test program executed on a TASP-free (T-Free) and TASP-BPU (T-BPU) IMC. (b) Percent increase in DRAM accesses for the TASP-BPU IMC and percent increase in total executed cycles for the TASP-cache IMC after executing the test program.

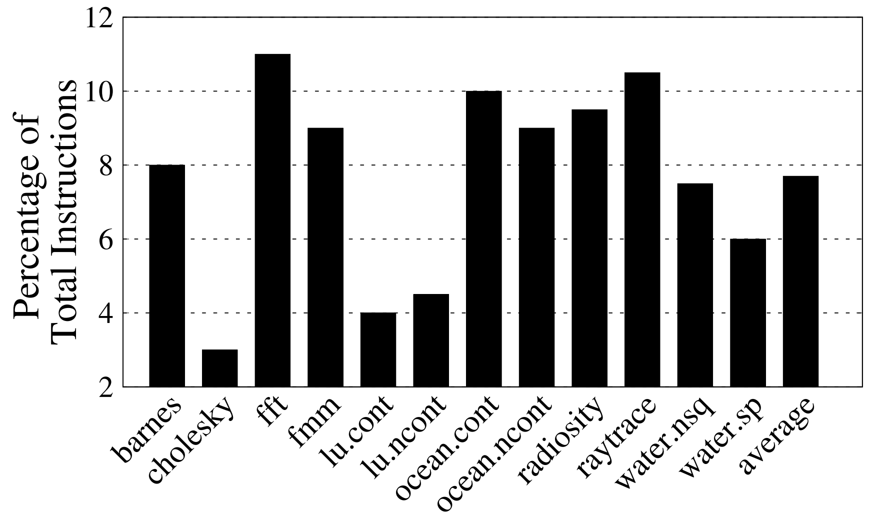
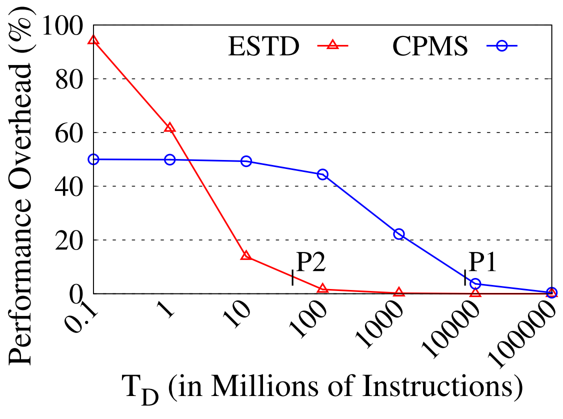


Fig. 4.9: Impact of TASP Activity Periods on CPMS.

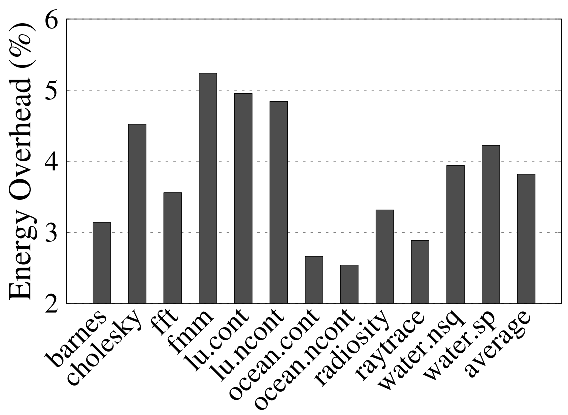
it increases the cache access latency. Thus, the TASP-BPU increases the DRAM memory access count, while the TASP-cache increases the total executed cycles for the infected IMC. Table 4.2(b) reports the percentage increase in DRAM memory accesses and percentage increase in total executed cycles for the infected IMC using different loop iterations. With 1000 and $10K$ loop iterations, this test program will not provide a significant increase in the DRAM access count for an infected IMC and therefore result in false negatives. We therefore select the length of the detection phase (L_D) to be 100K loop iterations.

4.5.4 Impact of TASP Activity Periods

CPMS and ESTD are independent of TASP activity patterns. Figure 4.9 demonstrates the minimum number of instructions (in %) the TASP should be active during the detection phase of CPMS to successfully identify the active TASPs. The pattern of TASP activity period is varied at random. If the TASP (*TASP-BPU* and *TASP-cache*) remains dormant during the detection phase, CPMS will not identify the TASP. In order to CPMS identify the active TASPs, the TASP must be active for certain number of instructions during the detection phase. For Splash2 benchmarks, on an average TASP must be active for 8% of the total instructions, with *fft* benchmark incurring the highest overlap of 11%.



(a)



(b)

Fig. 4.10: (a) Performance overhead of CPMS and ESTD based on the number of instructions between detection events (T_D). (b) Energy overhead for ESTD executing a test program.

For ESTD, *TASP-BPU* can be identified even if there is a small variation in memory access patterns. However, the variation in memory access pattern of TASP activity affects the detection of *TASP-cache*. ESTD requires *TASP-cache* to be active for at least 10% of the total instructions executed by the test program with 100K loop iterations to successfully flag the active TASPs. With 100K loop iterations, the increase in total cycles executed by test program is significantly higher (122%), compared to the baseline.

4.5.5 Performance Overhead

Figure 4.10(a) shows the performance overheads of CPMS and ESTD using Equation 4.6 for different values of T_D. CPMS and ESTD require a T_D of 7.2 billion instructions (P1) and 30.4 million instructions (P2) to achieve a 5% overhead.

4.5.6 Area, Power, and Energy Overhead

We synthesized a RTL model of CPMS using Design Compiler. We observe marginal overheads in area (2.31%) and in power (0.61%) for the 45*nm* technology. Since a test program is executed on the IMC for ESTD, it consumes additional energy. Figure 4.10(b) shows the energy overhead of ESTD, where T_D is 30.4 million instructions. To determine the energy overhead, we use Sniper to generate the energy consumption for Splash2 benchmarks and the test program. The energy consumption for the test program is constant. Therefore, the overhead depends on the relative energy of the test program compared to real world applications.

CHAPTER 5
ENERGY-EFFICIENT NOC ARCHITECTURES FOR MANYCORE NTC SYSTEM

5.1 Background and Contributions of This Work

Modern many-core chip design is plagued by barriers of prohibitive energy constraints and restrictive power budgets, while it is still expected to cater to the demands of diversified applications. *Near threshold computing* (NTC) comes as a saving grace to the energy-efficient computing paradigm by aggressively operating all computing platforms with a supply voltage close to the transistor threshold voltage. However, the tremendous increase in energy efficiency comes at the cost of a steep performance loss and performance variability (due to process variation) [10]. Further, traditional many-core architectures designed to perform at nominal voltages yield sub-optimal performance at NTC. While a majority of existing literature focuses on optimizing the computing cores, research on the on-chip communication's impact at NTC has taken a back seat. In this context, we meticulously evaluate the application level and hardware performance characteristics of many-core NTC systems to specifically isolate the impact of the on-chip communication fabric—network-on-chip (NoC).

NTC circuits typically employ more devices to exploit application parallelism and compensate for the performance loss of a single device [10]. A direct consequence of this approach is the increased communication demand on the NoC owing to simultaneous interaction of many cores. This heightened communication demand, along with the following three prominent factors, delivers a severe blow to the on-chip communication latency and performance. First, we see an increase in the *inter-core packet hop distance* by virtue of an increase in the computational core count. Second, the supply voltage scaling to near threshold results in a *massive reduction of the NoC operational frequency.* Finally, the unavoidable effects of *process variation* (PV) presents a tremendous challenge in NTC systems. In this work, we demonstrate that the *traditional on-chip communication fabric creates a severe performance bottleneck in NTC systems.* In addition, we seek a solution to regain the lost performance without compromising on the energy efficiency of the system.

Contemporary research on NoC topology and architectures such as clustered NoC [40], hierarchical NoC [41] and tile based NoC [36,42], have aimed to reduce the inter-core packet hop distance. While these works are an important step forward, they do not adequately address the challenges of reduced operational frequency and PV induced performance variation posed by the NTC regime. Hence, to improve the NoC performance without compromising on the energy efficiency, we propose *BoostNoC*— a power efficient, multi-layered NoC architecture that efficiently caters to the demands of many-core NTC systems. *BoostNoC* is made up of two architecturally homogeneous layers contrasted in their design characteristics. While one layer is optimized for power, the other is optimized to boost the NoC performance under high communication loads. *To the best of our knowledge, this is the first work to exploit the unique opportunity presented by the variation in communication load across epochs to efficiently boost the NoC performance in NTC regime.*

We make the following key contributtions in this work:

- We analyze the factors affecting performance in many core NTC systems and isolate the impact of the on-chip communication (Section 5.2).
- We explore the detailed design of a power efficient multi-layerd NoC architecture called *BoostNoC* to improve the performance under high communication load in many core NTC systems (Section 5.3).
- We examine the router occupation over the duration and study the traffic characteristics of applications using our *BoostNoC* architecture and determine that the use of *drowsy routers* can further improve the energy efficiency (Section 5.4.3).
- We propose *PG BoostNoC*— a design augmentation to power gate the unused routers in the *BoPeL*, to improve the peak power consumption, and energy efficiency of the *BoostNoC* architecture (Section 5.4.2).

- Using a rigorous cross-layered circuit-architectural analysis (Section 5.5), we evaluate the performance, energy efficiency and peak power improvement of *BoostNoC* architecture and its two design augmentations, considering the in-die process variations prevelant to near threshold systems. Our analysis reveals that *BoostNoC* and its variants delivers up to 1.4× higher performance per watt compared to a conventional NoC operating at NTC and nearly 3× that of a NoC operating at super threshold computing (STC) (Section 5.6).

- We present the difference between the two *BoostNoC* design augmentations—*PG BoostNoC* and *Drowsy BoostNoC*, and correlate the obtained data with application characteristics to make recommendations based on application characteristics (Section 5.6.6).

5.2 Motivation

In this section, we quantitatively assess the performance bottlenecks in a NTC many-core system. The performance of a many-core NTC has two major contributing factors: application level and hardware performance characteristics. To understand application level characteristics, we study the performance scalability of various applications under an *idealized hardware*[1] in Section 5.2.1. To carefully understand the impact from hardware performance characteristics, we decouple two of its major components: off-chip memory latency and on-chip interconnect latency. Our *rigorous experimental data clearly demonstrates that on-chip interconnect latencies are the most dominant performance bottlenecks in a NTC many-core system.*

5.2.1 Application Performance Characteristics

Application speedups from parallel execution are bound by the prevailing fraction of serial code and do not improve linearly with an increase in the computational core count.

[1] Ideal hardware signifies a system with no hardware performance penalties. The associated hardware penalties related branching, memory access, on-chip communication, among others, are all considered to be 1 cycle.

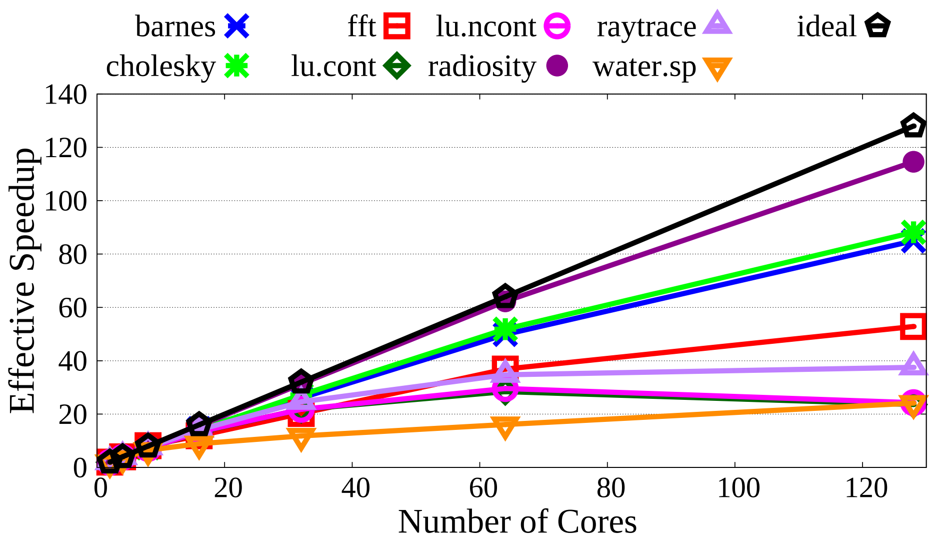


Fig. 5.1: Limitation due to application characteristics.

Since the fraction of serial code varies across applications, it is critical to understand this application level bottleneck when we comparatively analyze STC and NTC systems.

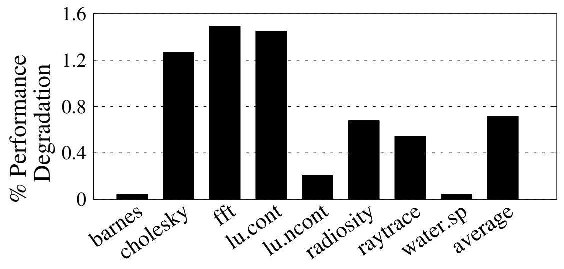
Figure 5.1 shows the effective application speedups obtained when a representative set of parallel workloads (SPLASH2 benchmarks) are executed on *ideal hardware* by scaling the processor count from 1 to 128 cores. The evaluation methodology used for this analysis is presented in detail in Section 5.5. We observe that only a couple of applications in this diverse set of benchmarks, can effectively scale beyond 60 cores. Benchmarks like *radiosity*, *cholesky* and *barnes*, have nearly ideal speedup indicating very little overheads due to the serial portions of the code. Other applications like *water.sp* and *raytrace* have large portions of serial code. Deploying these applications in NTC systems with hundreds of cores will result in decidedly sub-optimal performance.

5.2.2 Hardware Performance Characteristics

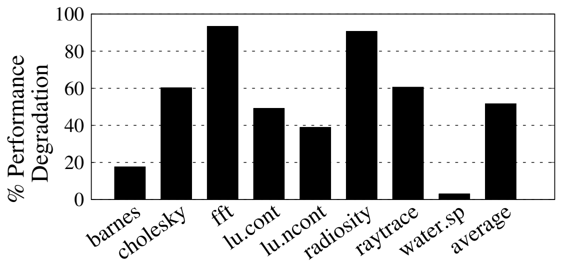
To quantify the impact of notable hardware characteristics such as memory access latency and inter-core communication on system performance, we consider a popular tile based 128-core architecture as our baseline NTC system [36,76]. The 128 cores are organized as 32 tiles (8×4) interconnected by a mesh network, with each tile consisting of 4 cores. The

Parameters	**Ideal System**	**Interconnect Bottleneck**	**Memory Bottleneck**
Architecture	Intel Xeon Processor E5 Series		
Cores	Tile-Based 128 Cores		
Voltage	0.35V		
Frequency	200MHz		
Technology	22nm		
Memory Latency	1 Cycle	1 Cycle	10 Cycles
NoC Latency	1 Cycle	2D Mesh NoC (1 Cycle/hop)	1 Cycle

Table 5.1: System configurations used to quantitatively analyze the performance bottleneck in many-core NTC systems.

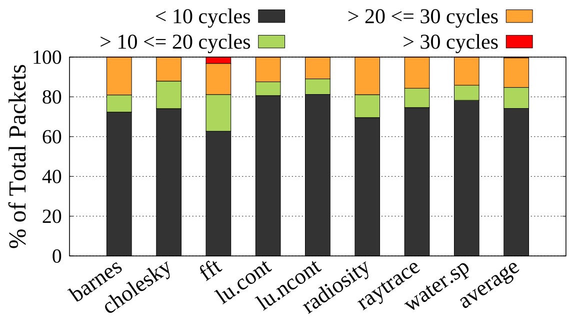


(a) Performance degradation due to off-chip memory access.

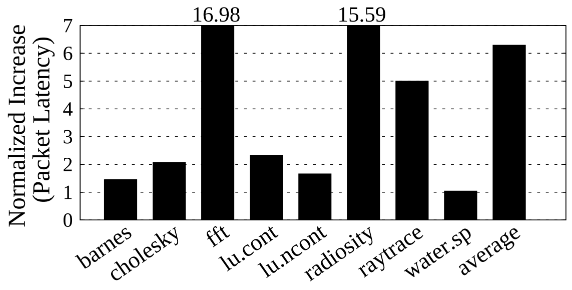


(b) Performance degradation due to a NoC.

Fig. 5.2: Quantitative analysis of hardware characteristics to identify the cause of sub-optimal system level performance in many-core NTC systems.



(a) Distribution of packet latency.



(b) NTC packet latency normalized to its STC counterpart.

Fig. 5.3: Characterizing the loss in NoC performance in NTC. Figure 5.3(a) presents the distribution of packet latency (in cycles) and Figure 5.3(b) shows the degradation in packet latency in NTC systems.

test configuration parameters are shown in the Table 5.1 (Section 5.5 presents a detailed discussion of the methodology).

Memory Access: Figure 5.2(a) illustrates the performance degradation due to off-chip memory access latency in a 128-core NTC system. Our analysis proves that *memory access is not a prominent cause for performance bottleneck in NTC systems.* We observe that the average performance degradation due to memory access latency is a mere 0.7% and the highest degradation suffered is 1.5% for the *fft* application. The baseline is considered to be an 128-core NTC system with ideal memory access latency as shown in Table 5.1.

On-Chip Communication: Figure 5.2(b) shows that the system performance degrades significantly due to the on-chip communication (network-on-chip) latency in a 128-core NTC system. Compared to an ideal system, the average performance degradation is a significant 50%, while *radiosity* and *fft* suffer from nearly 90% degradation in performance.

Our evaluations reveal that the following three factors play a decisive role in the degradation in NoC performance.

- *Increase in communication demand*: When comparing the volume of packets injected in a 128-core NTC system to an isopower 16-core STC system [2], we found that the volume of injected packets increased by more than 3× in the NTC system. The rise in core count results in the increase of both inter-core, as well as, cores-memory communication.

- *Diverse latency distribution in NTC*: Figure 5.3(a) illustrates the distribution of communication latency in a 128-core NTC system. We observe that, on an average, more than 30% of the packets have a latency greater than 10 cycles. A similar analysis in the STC system showed that a mere 5% of the packets have a latency greater than 10. This diversity in latency distribution is the resultant of increased inter-core packet hop distance owing to a rise in the core count.

- *Reduced NoC operational frequency*: Figure 5.3(b) shows that the packet latency degrades by more than 6×, on average, in a tile-based 128 core NTC system. Applications such as *fft* and *radiosity*, suffer a latency degradation of nearly 16×. The

[2]While scaling the 16-core STC system to a 128-core NTC system we ensure that both systems have a constant power budget.

increase in inter-core packet hop distance, along with the added detriment of reduced operating frequency, considerably increase the average packet latency.

5.2.3 Significance

The degradation in performance due to application (Section 5.2.1) and hardware characteristics (Section 5.2.2) help us characterize the demand in NTC systems. Our findings clearly demonstrate that ***the on-chip communication is a severe bottleneck in many-core NTC systems***. Hence, we propose **BoostNoC**, a novel power-efficient NoC architecture for NTC systems to efficiently reclaim the lost performance.

5.3 BoostNoC Architecture

In this section, we provide a detailed description of the *BoostNoC* architecture. Section 5.3.1 presents the design overview and we establish the insight behind our approach in Section 5.3.2. In Section 5.3.3, we detail our two layers, and analyze the intricacies of switching between the layers in Section 5.3.4. Section 5.3.5 reveals the required hardware control mechanism.

5.3.1 Design Overview

We envisage a multi-layered NoC architecture, where the layers are architecturally homogeneous but optimized to contrasting design considerations. Our work demonstrates

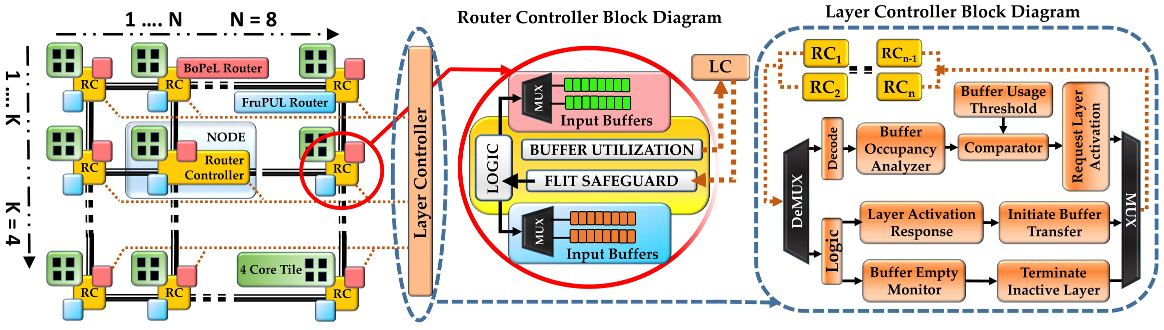
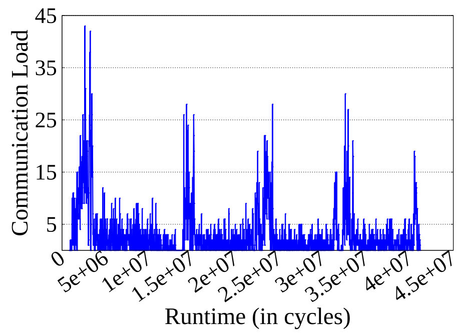
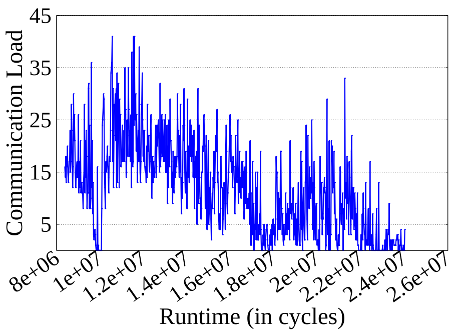


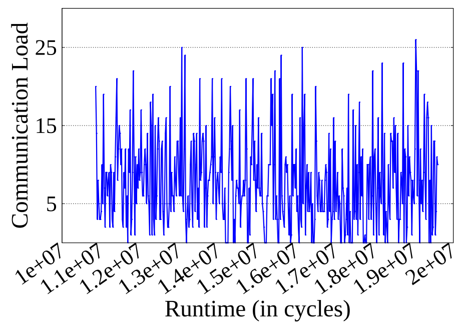
Fig. 5.4: BoostNoC Architecture. The figure also shows the functional diagrams of the router and layer controllers.



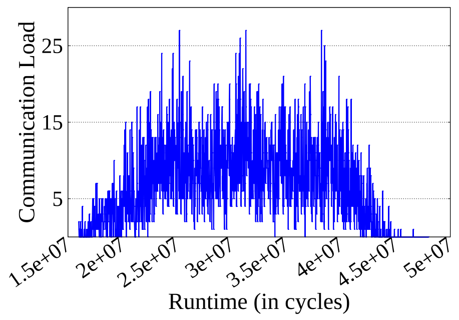
(a) barnes



(b) cholesky



(c) radiosity



(d) raytrace

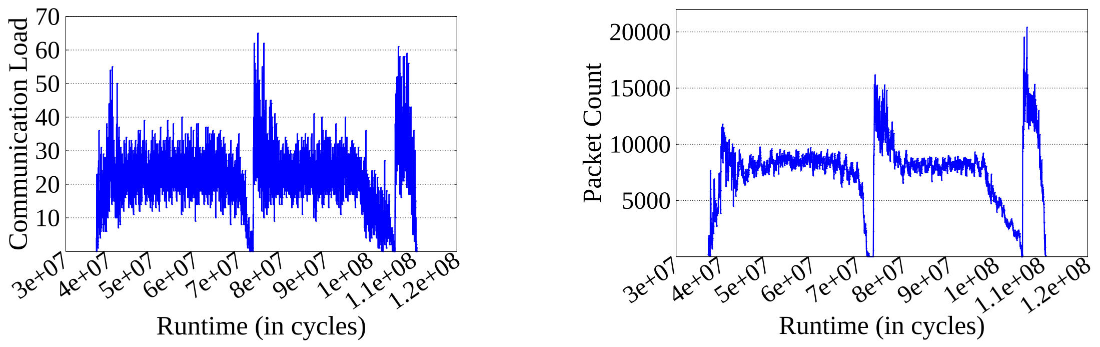
Fig. 5.5: Temporal variation of communication load for 4 different benchmarks. The plots illustrate network communication load (in %) during consecutive intervals of 20000 cycles for the whole application runtime. We see discernible patterns in all applications.

a novel incarnation of this concept—*BoostNoC*— that exploits the temporal nature of communication demand in NTC systems. The temporal nature refers to the variation of communication load across different epochs due to the inherent application characteristics.

Figure 5.4 illustrates the framework of our novel *BoostNoC* architecture. *BoostNoC* combines two architecturally homogeneous layers that are optimized to contrasting design parameters. Based on the communication load, *BoostNoC* dynamically switches between the layers. While one layer is optimized for power efficient data transmission, the other layer is used to bolster the NoC performance. We detail the technicalities of *BoostNoC* in the following sections.

5.3.2 Temporal Communication Demand

Figure 5.5 shows the on-chip communication network utilization trend of 4 representative applications, running on a 128-core NTC system. The *x-axis* represents consecutive



(a) Communication load - **fft**

(b) Volume of long distance communication.

Fig. 5.6: Correlation between communication load and volume of long distance communication for the **fft** application. Figure 5.6(b) shows the volume of long distance packets in consecutive epochs of 20000 cycles. The x-axis represents the application runtime in cycles.

intervals during the application runtime. In most benchmarks, we see discernible patterns in the communication demand that fluctuates between epochs. In few epochs the cores are highly voluble[3] creating a high load on the communication fabric, while in other epochs most cores are quiet (low communication demand). This temporal variation of network utilization can be correlated to the volume of injected packets experiencing long inter-core packet hop distance[4]. Figure 5.6 illustrates this correlation for the *fft* benchmark. We see a sharp rise in network utilization in epochs with a high volume of long distance packets.

Our novel *BoostNoC* architecture aims to exploit this temporal variation in communication demand by trading off chip area to bolster the NoC performance and energy efficiency.

5.3.3 BoostNoC Layers

Two architecturally homologous layers of NoC routers are interconnected in a mesh topology to frame the *BoostNoC* architecture. The two layers share the links between the routers as shown Figure 5.4. The two layers are:

- ***Frugal power usage layer (FruPUL)***: The routers in this layer are optimized to operate in the near threshold voltage regime to provide power-efficient operation at a low communication load.

[3]high volume of inter-core communication

[4]Considering the tiled architecture, we define packets needing more than 3 hops to reach their destination as long distance communication.

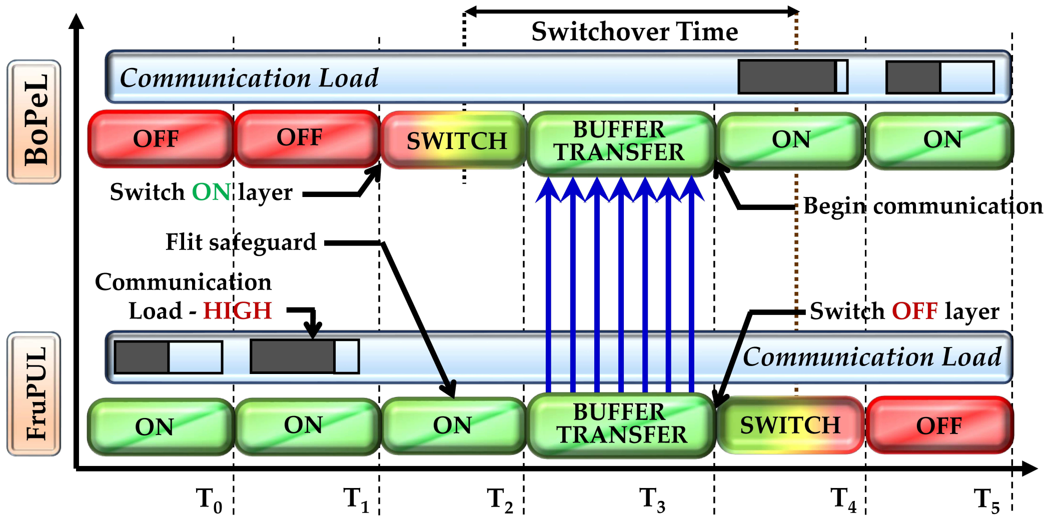


Fig. 5.7: Operational phases of the switchover mechanism.

- ***Boost performance layer (BoPeL)***: The routers in this layer are optimized to operate at the nominal voltage to bolster the NoC performance under a high communication load. The objective of *BoPeL* is to drain the in-flight packets at a quicker rate and offset the latency degradation caused by voluminous long distance communication.

At any given time, only one layer plays an active role in the communication fabric and the other layer is turned off. *FruPUL* is the default active layer as the cores are considered to be operating in the NTC regime. During epochs with high communication loads, *BoPeL* is activated (and *FruPUL* deactivated) to meet the demand and boost the NoC's performance. The layer switchover mechanism and the cost associated with it are discussed in Section 5.3.4.

5.3.4 Switchover Mechanism

The switchover between the layers is the crux of the *BoostNoC* architecture. The primary constraint while switching between the two layers is to maintain lossless communication of packets while incurring minimal switching overheads. Figure 5.7 illustrates the process of switching between layers. Keeping the defined constraints in mind, we envisage four operational phases of the switchover mechanism explained below in conjunction with Figure 5.7.

- *Pre-initiate*: During normal NoC operation, one of the layers is active and the other is powered off. In this interval, the aggregate buffer occupancy of the routers in the active layer is carefully monitored. The buffer occupancy information serves as an indicator of the communication load on the network. It is the cardinal parameter behind the decision making process involved in switching between the layers. In Figure 5.7, we observe that *FruPUL* is active and the communication load is being monitored. When the load increases, the decision to switch to *BoPeL* is made.
- *Initiate*: Based on the decision, *BoPeL* is signaled to switch on. During the same time, all the routers in *FruPUL* are instructed to process the in-flight flits in each router and forward them to the input buffers of their respective downstream routers. The flits already present in each router's input buffers maintain status quo. We call this process *flit safeguarding*. The process of *flit safeguarding* is allowed to continue and complete until *BoPeL* (the other layer) is switched on and ready to handle traffic.
- *Transfer*: Once *BoPeL* signals ready, the packets in the input buffers of routers in *FruPUL* (one layer) are transferred to the corresponding routers in *BoPeL* (the other layer). The novel buffer content transfer mechanism overcomes the need to drain packets from the network.
- *Terminate*: On receiving a signal from *FruPUL* that the buffer content transfer is successful and that all its buffers are empty, the layer is signaled to be powered off. Simultaneously, *BoPeL* is waved to begin normal operation.

5.3.5 Hardware Control Mechanism

BoostNoC architecture requires specific hardware enhancements to carry out its functions in an orderly fashion. We adopt two hardware control mechanisms known as *Layer Controller* and *Router Controller* to efficiently resolve and regulate the layer operations in

Algorithm 2 Layer Controller Operation

```
1: Initialize: $Routers = N$;                                   ▷ Number of routers
2: Acknowledge: $ack\_LX$
3: WaitForAcclimatizationPd();
4: for $k = 1 \rightarrow Routers$ do
5:     Evaluate BufferOcupancy(k);
6: end for
7: for $k = 1 \rightarrow Routers$ do
8:     Evaluate RouterLocation(k);
9:     if $(BufferUsage > Usage_{threshold})$ then
10:        RouterRequested++;
11:    end if
12: end for
13: if $(RouterRequested > Router_{significant})$ then
14:     Enable $reqactiv\_LX$;
15: end if
16: for $k = 1 \rightarrow Routers$ do
17:     Enable $init\_fs(k)$;
18: end for
19: WaitFor $respactiv\_LX$;
20: if $(respactiv\_LX)$ then
21:     Enable $init\_transbuf()$;
22: end if
23: WaitFor $resp\_bufempty$;
24: if $(resp\_bufempty)$ then
25:     Enable $term\_LX(OLD)$;
26:     Enable $begin\_comm$;
27: end if
```

the NoC. Each controller plays a definitive role to efficiently boost the NoC performance in NTC systems.

Layer Controller (LC): The role of the layer controller is to monitor the network communication load by aggregating the information sent from individual router controllers. It functions like the brain of *BoostNoC*, and plays a central role in the decision process to switch between layers. Algorithm 2 shows the basic operation of the *LC*. As an initial setup, *LC* acknowledges the active layer ($ack\_\boldsymbol{LX}$) and records the buffer occupancy of the routers in that layer (lines 1-6). It then continually monitors the information sent by individual router controllers during each epoch and based on the rules set in lines $7 - 15$, it decides if a switchover in layer will yield a better outcome. Once the decision is made to switch between layers, *LC* signals to turn on the alternate layer ($reqactiv\_\boldsymbol{LX}$) and instructs the individual router controllers (RC) to trigger *flit safeguarding* (*init_fs*). On receiving a response from the newly activated layer (*respactiv_L**X***), it instructs all *RCs* to begin inter-layer buffer content transfer (*init_transbuf*) and waits for all *RCs* to signal for transfer completion (*resp_bufempty*). At this point, the *LC* terminates the old layer (*term_L**X***), activates the new layer (*begin_comm*) and goes back to monitoring the communication load.

Router Controller (RC): *RCs* are distributed agents with a three-fold functionality: (a) to sense local changes in the network, (b) to report gathered information to the *LC* and (c) to actuate responses when directed by the *LC*. Each individual *RC* reports its buffer occupancy to the *LC* at regular intervals (*report_bufoc*) and waits for a decision. On receiving the *init_fs* signal, the *RC* performs buffer content transfer, reports successful transfer back to the *LC* and waits for *begin_comm* to restart communication in the active layer.

Figure 5.8 illustrates the sequence of handshake signals between *LC* and *RC*, highlighting the operation of *BoostNoC*. *LC*, additionally ensures that once a layer is activated, it stays active for a set minimum period known as *acclimatization period*. The acclimatization

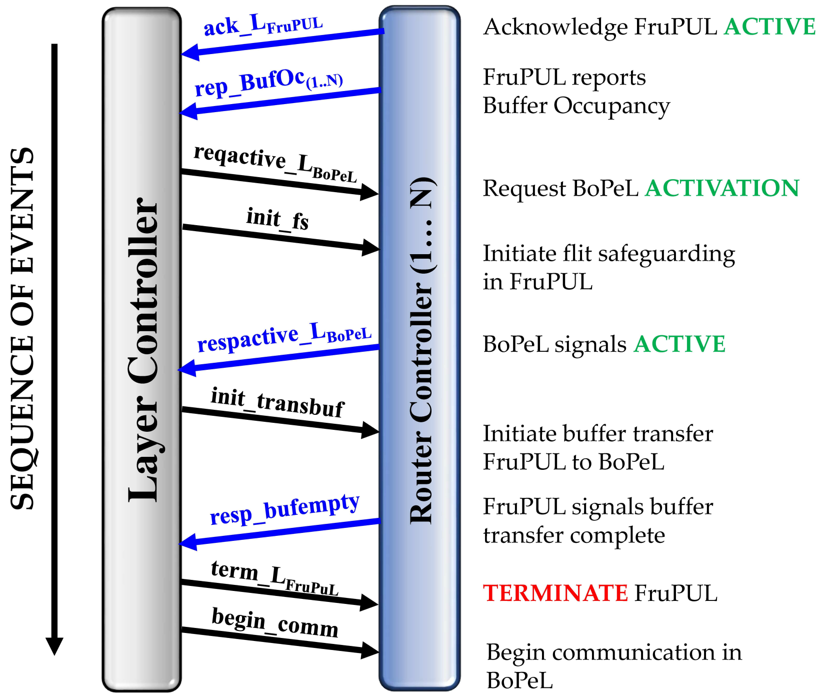


Fig. 5.8: Handshake communication between Layer and Router controller.

period is added to amortize the cost associated with the layer switchover and to avoid the effect of thrashing between layers.

Inter-Layer Buffer Content Transfer: The router controller (shown in Figure 5.4) plays a critical role in the inter-layer transfer of packets. The router in *FruPUL* is connected to its counterpart in the *BoPeL* using a bi-directional physical link controlled by the *RC*. The router in each layer consists of n buffers. Once the process of *flit safeguarding* is complete, *RC* evaluates the buffer occupancy of the active layer. The buffer contents of the active layer are serially copied to the buffers of the router in the alternate layer by selecting the appropriate MUX and DeMUX signals. A counter keeps track of all transactions between the two layers and once the value matches the buffer occupancy estimated before the process, the *RC* signals the successful completion of buffer transfer. This process happens simultaneously in the entire network. The serial transfer and transaction tracking between the two layers ensure a lossless transition between the two layers during a switchover. The cost associated with the switchover directly correlates to the buffer occupancy at the start of the process and the worst case switchover overhead depends on the buffer size of the routers.

5.4 Energy Efficient BoostNoC Architectures

In this section, we examine the traffic characteristics and network utilization of the *BoPeL* layer for various applications, to further optimize the *BoostNoC* architecture. Based on the key observations (Section 5.4.1), we propose two design derivatives of *BoostNoC*—*PG BoostNoC* (Section 5.4.2) and *Drowsy BoostNoC* (Section 5.4.3)—to further improve the energy-efficiency.

5.4.1 Key Observation

Breaking down the *BoostNoC* layer switching rule in Algorithm 2, we can state that *BoPeL* activation predominantly requires a high communication load on the network. Figure 5.9, presents an interesting observation of the *BoPeL* operation. In the majority of applications, a large percentage of the NoC routers remain idle intermittently, indicating that the

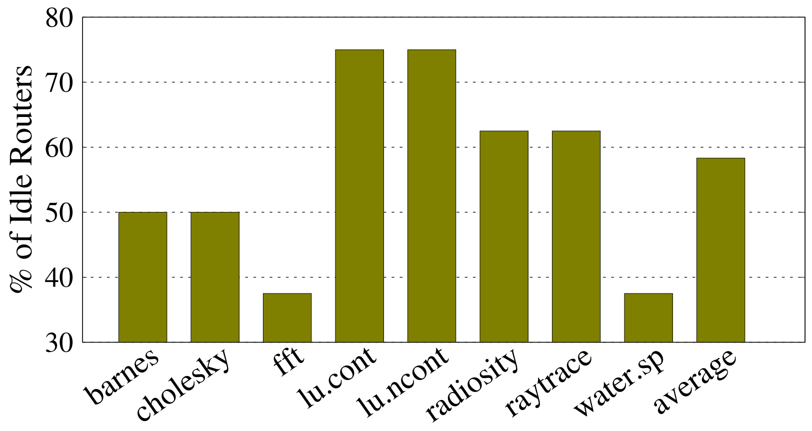


Fig. 5.9: Percentage of idle routers in *BoPeL*.

communication load at any given time is spatially concentrated among a few routers. On an average, nearly 60% of the routers remain idle intermittently in each epoch of *BoPeL* operation, giving us the impetus to further optimize the *BoostNoC* energy efficiency.

While the spatial concentration of communication load in *BoPeL*, provides the necessary information for *BoostNoC* optimization, it is not sufficient. On that account, we also examine the temporal distribution of communication in *BoPeL* to classify the *BoPeL* traffic trends into two categories, and propose *BoostNoC* derivative designs for optimization.

- **Sparse communication**: The network traffic in *BoPeL* is temporally scattered, with only few routers sporadically exercised. For example, the *BoPeL* traffic characteristics of *lu.cont* in one epoch (Figure 5.10(a)), reveal that the packets are quickly drained from the network, and for the rest of the epoch, the communication is sparse. During this period, the idle routers can be power gated with minimal effect on the NoC performance, giving rise to our first *BoostNoC* augmentation—*PG BoostNoC* (Section 5.4.2).

- **Frequent communication**: The network traffic in *BoPeL* gradually reduces, with a subset of the routers frequently being exercised. For example, *fft* endures a continued load, with frequent communication spikes as seen in Figure 5.10(b). Power gating routers is a sub-optimal design choice under these circumstances, due to the wait

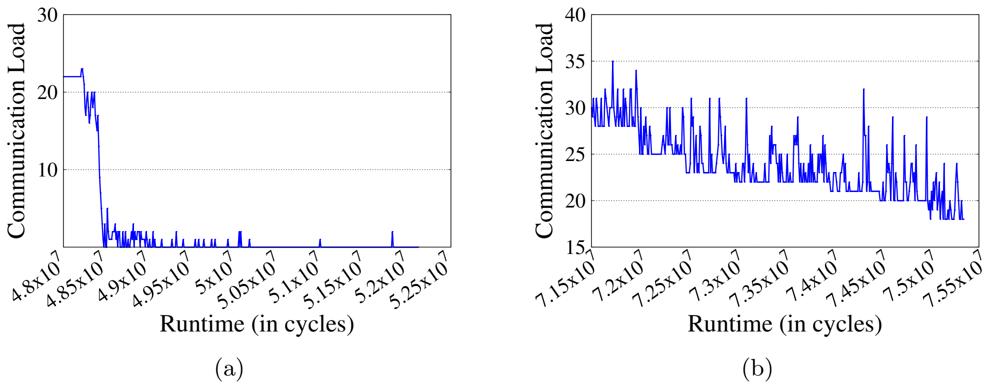


Fig. 5.10: Classification of network traffic characteristics for an epoch in BoPeL. (a) lu.cont benchmark. (b) fft benchmark.

time associated with bringing the routers online. However, to extract optimal energy efficiency from the intermittent idle routers, we propose the use of drowsy SRAM as the input buffers—*Drowsy BoostNoC* (Section 5.4.3).

5.4.2 PG BoostNoC

The routers in the *BoPeL* operate at the nominal voltage, and hence have a significantly high power consumption. Idle routers in the *BoPeL* present an excellent opportunity to further improve the energy efficiency of *BoostNoC*. We employ power-gating of individual *BoPeL* routers on observing a drop in the communication load by the layer controller within the layer transition time to *FruPuL*. Router controllers that sense the local changes, decouples the idle routers from the network, signaling the surrounding routers about the change in the state. The state transition between OFF-ON/ON-OFF consume extra cycles and incur performance and energy penalties. We evaluate the performance and energy efficiency of power gating individual routers in *BoPeL* in Section 5.6.

5.4.3 Drowsy BoostNoC

The routers in the *BoPeL* operate at the nominal voltage and hence have a significantly high power consumption. By introducing *drowsy SRAMs* as buffers in the router, we add an additional low power operation mode to improve the energy efficiency. In this mode, a

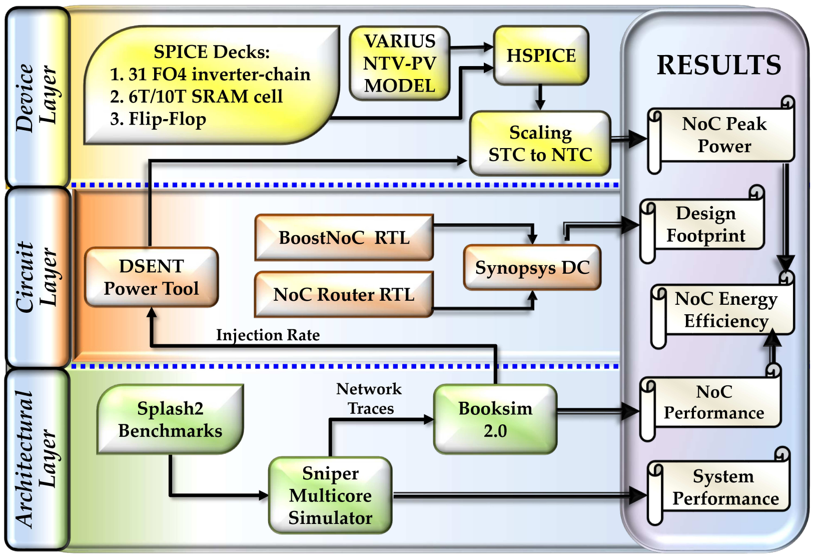


Fig. 5.11: BoostNoC cross-layer methodology.

Parameters	**STC Configuration**	**NTC Configuration**
Architecture	Intel Xeon Processor E5 Series	
Cores	16	128
Voltage	1.0V	0.35V
Frequency	2.3GHz	200MHz
Technology	22nm	22nm

Table 5.2: STC and NTC system configuration.

low voltage is supplied to the inactive routers, thereby reducing the leakage current. The idle routers are periodically put into *drowsy mode* and are woken up when the upstream router requests credit information. A single cycle cost is added to wake up a router in the *drowsy* state [77]. The decision to put the idle routers into the low power mode can be made by the router controller based on buffer utilization changes. We evaluate the improvement in energy efficiency due to *drowsy routers* in *BoPeL* in Section 5.6.

5.5 Evaluation Methodology

Figure 5.11 presents the comprehensive cross-layer methodology we use to evaluate the efficacy of *BoostNoC* architectures using three metrics: *peak power*, *performance* and *energy efficiency*. Architectural simulations are performed to assess the performance (Section

5.5.1), while the circuit layer analysis contributes valuable information regarding the design footprint and power characteristics (Section 5.5.2). Section 5.5.3 presents the procedure for device level analysis to obtain process variation parameters and STC to NTC scaling data.

5.5.1 Architectural Layer

Multi-core Simulation: We model an Intel Xeon E5 series processor on Sniper multi-core simulator [66] with the configuration shown in Table 5.2. The STC system models 16 cores interconnected using a NoC (4×4 2D mesh topology). The NTC system models 128 cores in a tile based architecture interconnected using a 8×4 2D mesh NoC, with each tile housing 4 cores [76]. We use highly parallel large-set workloads from the Splash2 benchmark suite to assess the performance of these systems and collect traces of the communication. We use booksim 2.0 [78] to simulate and evaluate the NoC behavior. Splash2 benchmark suite consists of parallel and well-diversified applications that can scale to 128 cores [79].

Chen et al. showed that the maximum delay deviation due to within-die PV is a colossal 200% for the NTC regime at 22nm and thus cannot be discounted [80]. We therefore used this delay variation to model PV-affected NTC core.

NoC Simulation: We model a 8x4 2D mesh NoC mimicking a 32 tile-based NTC system on the Booksim Simulator [78]. The router has a 4-stage pipeline of route computation, virtual channel allocation, switch allocation and switch traversal. We simulate the traces collected from Splash2 benchmarks and observe the NoC behavior and study various traffic characteristics. We implement the *BoostNoC* architecture with functionality detailed in Section 5.3 and evaluate the performance of the NoC. Our evaluation carefully considers the impact of PV on the NoC performance.

5.5.2 Circuit Layer

To estimate the design footprint and hardware overheads of our architecture, we augment the open source NoC router RTL [81] with the hardware control mechanisms discussed in Section 5.3.5. We synthesize the NoC router RTL using the $32nm$ standard cell library using Synopsys Design Compiler. We use the DSENT power modeling tool [82] to determine the NoC leakage and dynamic power estimates considering the PV parameters evaluated in the device layer. The network and router configuration are identical in Sniper, Booksim, as well as, DSENT to maintain uniformity.

5.5.3 Device Layer

We obtain the $22nm$ PTM model for HSPICE simulations and customize it in order to generate leakage and dynamic power behavior at STC and NTC regimes [83]. NTC circuits are highly susceptible to process variation. Our HSPICE evaluations model the effect of PV based on VARIUS-NTV [84] and we use these results while scaling from STC to NTC. The details of our scaling methodology follows.

Power Scaling from STC to NTC

Scaling the entire power from the STC to the NTC region presents a methodological challenge. HSPICE simulation of an entire NoC architecture is computationally intense. To manage the complexity, we scale the STC power to NTC using the following three categories [85].

- *Combinational logic*: This is scaled using the STC/NTC characteristics of the canonical 31 fanout-of-4 inverter-chain as the representing circuit [11].
- *Storage elements*: We scale the on-chip SRAM power by investigating the power scaling trend from the STC 6T SRAM cell to the NTC-friendly 10T SRAM cell [86].
- *Interconnect*: We estimate the interconnect power to be 50% of the dynamic power based on previous work [85]. Since scaling the supply voltage equally affects both interconnect power and dynamic power, we assume that their relative weight remains unchanged for STC and NTC.

5.6 Experimental Results

In this section, we discuss the experimental results obtained from our simulation of the *BoostNoC* architecture and its energy-efficient variants considering the with-in die PV.

Section 5.6.1 summarizes the different schemes that we use in our simulations for baseline comparison. We evaluate the effectiveness of our proposed architectures using three metrics *performance* (Section 5.6.2 and Section 5.6.3), *peak power* (Section 5.6.4) and *energy efficiency* (Section 5.6.5). In Section 5.6.6, we present a detailed trade-off analysis between the two variants of *BoostNoC*, based on the application characteristics. We conclude our results section by presenting the design footprint in terms of area overhead in Section 5.6.7.

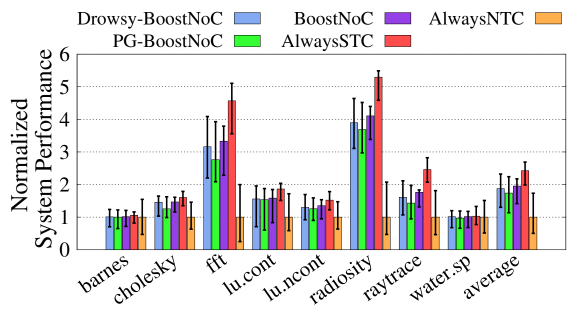
5.6.1 Evaluation Schemes

We use five schemes to evaluate our proposed architectures. They are:

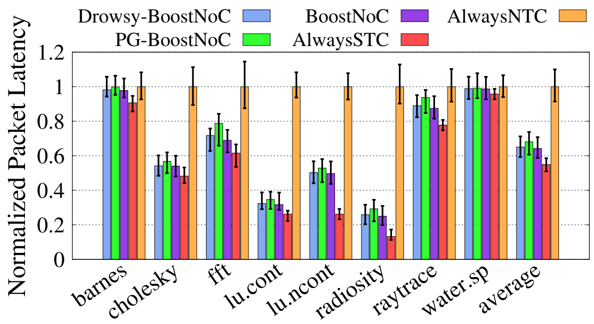
- ***Always NTC***: The NoC and the cores are both operated in the NTC regime throughout the application runtime. In theory, this scheme is extremely energy efficient however at the cost of a substantial degradation in performance. Moreover, the with-in die process variation significantly affects the performance/power characteristics of both the cores, as well as, the NoC in this scheme.

- ***Always STC***: In this scheme, the cores are operating in the NTC regime, while the NoC is operating at nominal voltage (STC). This configuration offers the best performance while taking a significant hit in peak power and energy efficiency. The cores substantially suffer from the effect of process variation. However, the NoC exhibits lower variation in performance/power characteristics as it operates at the STC regime.

- ***BoostNoC***: Our proposed *BoostNoC* architecture, discussed in Section 5.3, uses two layers (*FruPUL* and *BoPeL*) to provide the best of both worlds. The architecture sacrifices chip area to deliver better performance and energy efficiency. The process variation affects both cores and NoC significantly. Since NoC operates in *FruPUL* layer during most of the application runtime, the effect of process variation is high compared to an *always STC* scheme.



(a) Normalized system level performance.



(b) Normalized packet latency.

Fig. 5.12: (a) System level performance improvement of our proposed BoostNoC architectures normalized to AlwaysNTC scheme (Higher is better. (b) Normalized reduction in packet latency of BoostNoC compared to AlwaysNTC scheme (Lower is better).

- ***PG BoostNoC***: In this scheme, the unused routers in the *BoPeL* are power gated to reduce the static power consumption. As discussed in Section 5.4.2, the objective of this technique is to improve the NoC's peak power and energy efficiency, while taking a small hit on performance.

- ***Drowsy BoostNoC***: In this scheme, we employ drowsy SRAMs as the NoC router buffers (Section 5.4.3) in the *BoPeL*. The unused routers are able to improve NoC energy efficiency by transitioning into a drowsy state, with a minuscule wake up delay.

5.6.2 System Level Performance Analysis

Figure 5.12(a) shows the normalized system level performance of the *BoostNoC* architecture and its two variants, considering within-die process variation. The performance is normalized to the baseline PV-free *always NTC* scheme. Our results demonstrate that on an average, the *BoostNoC* improves the system performance by nearly 2×. Benchmarks with a high communication demand such as *fft* and *radiosity* show even higher performance improvement (nearly 4×). However, applications with low communication demand (*barnes* and *water.sp*) are less sensitive to the boost in operating frequency and hence deliver a small improvement in the system level performance.

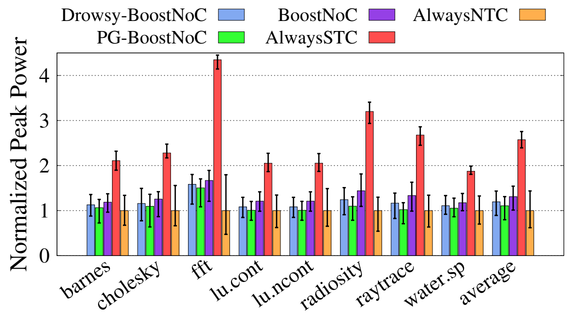
Our evaluations show that the two variants of *BoostNoC*—*PG BoostNoC* (1.74×) and *Drowsy BoostNoC* (1.8×), slightly compromise on the system performance. This variation in performance is due to the additional time required for the routers to transition from sleep/drowsy state to the operational state. As expected, Figure 5.12(a) demonstrates that, *Drowsy BoostNoC* offers better performance than *PG BoostNoC*, due to the faster state transition times. We observe that the degree of variation in performance between the two variants are application dependent and is discussed in detail in Section 5.6.6.

5.6.3 NoC Performance Analysis

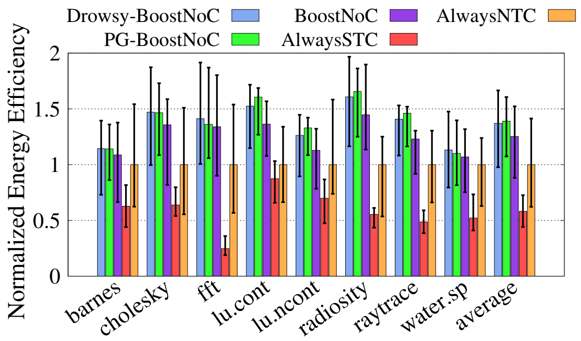
Figure 5.12(b) illustrates the packet latency reduction due to *BoostNoC* and its variants. This reduction in packet latency directly translates into a system level performance improvement. Figure 5.12(b) demonstrates the on-chip communication performance as compared to the baseline PV-free *always NTC* NoC. On an average, *BoostNoC* improves the packet latency by nearly 40% compared to a conventional *always NTC* scheme. The performance of *Drowsy BoostNoC* is fairly identical to the *BoostNoC* architecture due to negligible overhead in the transition from *drowsy mode* to ON state. On the other hand, *PG BoostNoC*, suffers from larger state transition time (power gated OFF to ON), and hence achieves a slightly higher packet latency compared to *BoostNoC*. *Always STC* performs better than *BoostNoC* as the NoC operates at a higher frequency throughout the application runtime. Our results demonstrate that applications with high communication loads significantly benefit from the *BoostNoC* architecture, and, the application characteristics determine the performance variation between the two *BoostNoC* variants (Section 5.6.6).

5.6.4 NoC Peak Power Analysis

Figure 5.13(a) compares the peak power dissipated among the different simulation schemes. The values obtained are normalized to baseline PV-free *always NTC* peak power which is expected dissipate the least power. *BoostNoC* suffers from a nearly 30% rise in the peak power on an average, due to the switchover to *BoPeL* which operates at the nominal voltage. However, this is noticeably better than the *always STC* scheme, which incurs more



(a) Normalized peak power.



(b) Normalized energy efficiency.

Fig. 5.13: (a) Normalized peak power of BoostNoC architectures compared to PV-free AlwaysNTC (Lower is better). (b) Energy efficiency of BoostNoC Architectures normalized to PV-free AlwaysNTC (Higher is better).

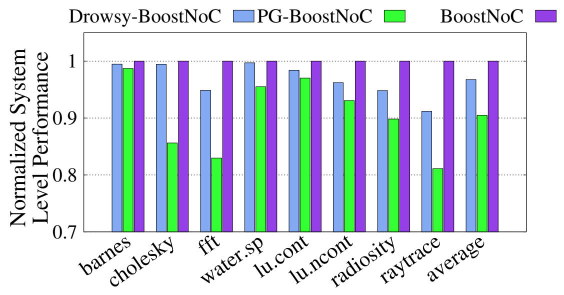
than 2× increase in peak power.

Drowsy BoostNoC and *PG BoostNoC* experience peak power dissipation values of 20% and 10% over *always NTC*, respectively, which is fairly better than the peak power seen for *BoostNoC*. The improvement in peak power seen in *PG BoostNoC*, over *BoostNoC* is due to the reduction in static power by power-gating the idle routers in the *BoPeL*.

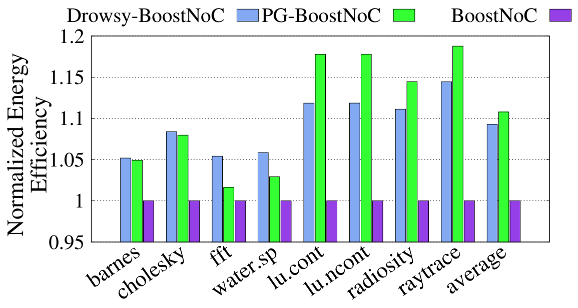
5.6.5 NoC Energy Efficiency Analysis

Figure 5.13(b) compares the normalized energy efficiency of the schemes. In a sense, *performance delivered per watt* is an accurate measure for comparison of the schemes as it accounts for both performance, as well as, power. Our analysis shows that, though the performance of the *always STC* scheme is significantly higher than other schemes, it is highly energy inefficient. The proposed *BoostNoC* provides a favorable trade-off between power and performance, and hence surpasses conventional NTC architectures by 25%. Both *Drowsy BoostNoC* and *PG BoostNoC* further improves the energy efficiency over *always NTC* by nearly 40%.

Water.sp has a low runtime and a low communication demand limiting the duration of operation in *BoPeL*. These characteristics of *water.sp* prohibits *BoostNoC* from improving its energy efficiency. Similarly, the meager improvement in *barnes* is due to its high compute and low communication attributes. Applications such as *Radiosity*, and *lu.cont* experience



(a)



(b)

Fig. 5.14: Analysis between Drowsy-BoostNoC, PG-BoostNoC and BoostNoC. (a) Normalized system level performance (Higher is better). (b) Normalized energy efficiency (Higher is better).

larger improvements in energy efficiency, leveraging the *BoostNoC* architecture well.

Figure 5.13(b) reveals an interesting consequence of the variations in application characteristics on the two design derivatives of BoostNoC. Section 5.6.6 presents a detailed analysis on why *PG BoostNoC* performs better for some applications, while *Drowsy BoostNoC* is favorable for others.

5.6.6 Analysis : *PG BoostNoC* vs *Drowsy BoostNoC*

While Figure 5.12 and Figure 5.13 demonstrate the relative impact of *BoostNoC* compared to *always NTC* and *always STC*, Figure 5.14 primarily analyzes the variation in performance and energy efficiency between the design derivatives of *BoostNoC*. *PG BoostNoC* and *Drowsy BoostNoC* differ primarily in the operation in BoPeL. We observe that application characteristics play a key role in deciding an optimal choice of *BoostNoC* architecture. We correlate the results in Figure 5.14 to the following three key application characteristics:

- ***Communication load*** : The duration *BoostNoC* operation in BoPeL is a direct resultant of the high communication load in applications. Applications such as, *fft*, *radiosity* and *raytrace* process a large percentage of the total packets in BoPeL, thereby seeing a larger impact due to *BoostNoC*. Figure 5.12(a) illustrates the relative impact of *BoostNoC*, and clearly reveals that *BoostNoC* seizes the opportunity to reclaim the lost performance of *always STC*. Applications such as *barnes* and *water.sp* have relatively low communication load, echoing the data that for higher communication load, the *BoostNoC* architecture is more effective.

- ***Communication Frequency*** : The temporal distribution of packets impact the transition between *on* and *drowsy/off* states, thereby affecting both energy and performance. Frequent arrival of packets favor the use of drowsy SRAMs, over power gating, due to the wake up delay overheads associated with power gating the routers. Communication patterns of *cholesky*, when compared with *radiosity* (Figure 5.5) serve as a good example to analyze the the two *BoostNoC* designs. The spike in network activity in *radiosity* is even, and temporally well distributed, whereas in *cholesky*, we see varying network activity with frequent network spikes. In Figure 5.14(a), the larger difference in performance between *PG BoostNoC* and *Drowsy BoostNoC*, in *cholesky* as compared to *radiosity* emphasizes the impact of communication frequency.

- ***Idle Routers*** : The spatial distribution of packets in the BoPeL also play a significant role in the NoC efficiency, as it determines the number of routers that can be put to drowsy/off state. Correlating Figure 5.9, with Figure 5.14(b), we observe that for applications that have a higher percentage of idle routers in the BoPeL, *PG BoostNoC* has an edge over *drowsy BoostNoC* in efficiency. For *fft* and *water.sp*, which have the lowest percentage of idle routers in BoPeL among our tested applications, *Drowsy BoostNoC* is clearly the better choice between the two *BoostNoC* derivative designs.

5.6.7 Design Overheads

The overheads due to the cost associated with switching between layers is accounted for in our performance evaluations. *BoostNoC sacrifices chip area to deliver better performance and energy efficiency.* The relative chip area increases by 1.16× as *BoostNoC* consists of two architecturally homogeneous layers. However, the design overhead of the *LC* and the *RC* logic is a mere 1.77% of the single layered NoC.

CHAPTER 6
CONCLUSION

This dissertation investigates the critical security vulnerabilities stemming from emerging neuromorphic computing architectures and 3PIP cores. Neuromorphic computing marks the beginning of a new era in computing system design. However, the security implications of the same is still at its nascent stage. This dissertation uncovers security vulnerabilities in the emerging neuromorphic computing paradigm. This work lays a strong foundation to understanding the broad security implications of emerging nano-ionic device based neuromorphic hardware. This work analyzed the potential cause for security vulnerabilities using two lateral threat models. Further, this work also investigated three specific attacks, exploiting the untrustworthy design environment, and the security loopholes arising from the implementation specific properties. The dissection of specific attacks, will help systematically develop collective security primitives and design-for-trust solutions to minimize the attack surface and protect against security vulnerabilities.

Trustworthiness is an emerging concern in the development of MPSoCs, which often include 3PIPs. This dissertation uncovers an imminent threat model that originates from pervasive use of 3PIP cores. This work demonstrates that a malicious TASP can affect the performance of a core without altering its architected state. Hence, disrupting the availability of on-chip resources in a MPSoC system. Further, this work determines the feasibility of TASP design by evaluating its design footprint. This work proposes three detection mechanisms to identify always-on and intermittent TASPs in a 3PIP core. These proposed mechanisms can effectively detect TASPs with marginal overheads.

Finally, this dissertation quantitatively demonstrates the severe performance bottleneck created by a conventional NoC architecture in many-core NTC system. This work demonstrates the key factors that affect the NoC performance at NTC and propose *Boost-NoC*—a novel power-efficient, multi-layered homogeneous NoC architecture that exploits the temporal variation in on-chip communication demand. *BoostNoC* efficiently switches between the two layers, *FruPUL* (optimized for power consumption), and *BoPeL* (optimized for performance), to extract the performance benefits of nominal voltage operation, while largely sustaining the energy benefits of NTC. This work meticulously analyzes the spatial and temporal communication patterns in *BoostNoC*, to further optimize by employing two energy-efficient design primitives: *PG BoostNoC* and *Drowsy BoostNoC*. This work demonstrated that *BoostNoC* and its derivatives improve the energy-efficiency of a many-core NTC system by more than 35%.

REFERENCES

[1] S. Li, K. Lim, P. Faraboschi, J. Chang, P. Ranganathan, and N. P. Jouppi, "System-level integrated server architectures for scale-out datacenters," in *IEEE/ACM International Symposium on Microarchitecture (MICRO).* ACM, 2011, pp. 260–271.

[2] S. Bhunia, M. S. Hsiao, M. Banga, and S. Narasimhan, "Hardware trojan attacks: threat analysis and countermeasures," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1229–1247, 2014.

[3] C. Liu, J. Rajendran, C. Yang, and R. Karri, "Shielding heterogeneous mpsocs from untrustworthy 3pips through security-driven task scheduling," in *IEEE Symposium on Defect and Fault-Tolerance in VLSI Systems (DFT)*, 2013, pp. 101–106.

[4] H. Li, Q. Liu, J. Zhang, and Y. Lyu, "A survey of hardware trojan detection, diagnosis and prevention," in *Computer-Aided Design and Computer Graphics (CAD/Graphics), 2015 14th International Conference on.* IEEE, 2015, pp. 173–180.

[5] R. Lee, S. Sethumadhavan, and G. E. Suh, "Hardware enhanced security," in *ACM Conference on Computer and Communications Security (CCS)*, 2012, p. 1052.

[6] F. Akopyan, J. Sawada, A. Cassidy, R. Alvarez-Icaza, J. Arthur, P. Merolla, N. Imam, Y. Nakamura, P. Datta, G. J. Nam, B. Taba, M. Beakes, B. Brezzo, J. B. Kuang, R. Manohar, W. P. Risk, B. Jackson, and D. S. Modha, "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct 2015.

[7] M. Tehranipoor and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design and Test for Computers*, vol. 27, no. 1, pp. 10–25, 2010.

[8] X. Zhang and M. Tehranipoor, "Case study: Detecting hardware trojans in third-party digital ip cores," in *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2011, pp. 67–70.

[9] C. Rajamanikkam, K. Brenning, A. Deakin, S. Roy, and K. Chakraborty, "Taspdetect: Reviving trust in 3pip by detecting tasp trojans," *Microprocessors and Microsystems*, vol. 56, pp. 76 – 83, 2018.

[10] R. G. Dreslinski, M. Wieckowski, D. Blaauw, D. Sylvester, and T. N. Mudge, "Near-threshold computing: Reclaiming moore's law through energy efficient integrated circuits," *Proc. of the IEEE*, vol. 98, no. 2, pp. 253–266, 2010.

[11] N. R. Pinckney, K. Sewell, R. G. Dreslinski, D. Fick, T. N. Mudge, D. Sylvester, and D. Blaauw, "Assessing the performance limits of parallelized near-threshold computing," in *DAC*, 2012, pp. 1147–1152.

[12] D. Markovic, C. C. Wang, L. P. Alarcon, T.-T. Liu, and J. M. Rabaey, "Ultralow-power design in near-threshold region," *Proceedings of the IEEE*, vol. 98, no. 2, pp. 237–252, 2010.

[13] C. Rajamanikkam, R. J. Shridevi, S. Roy, and K. Chakraborty, "Boostnoc: Power efficient network-on-chip architecture for near threshold computing," in *IEEE International Conference on Computer-Aided Design (ICCAD)*, 2016.

[14] B. V. Benjamin, P. Gao, E. McQuinn, S. Choudhary, A. R. Chandrasekaran, J. M. Bussat, R. Alvarez-Icaza, J. V. Arthur, P. A. Merolla, and K. Boahen, "Neurogrid: A mixed-analog-digital multichip system for large-scale neural simulations," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 699–716, May 2014.

[15] E. Painkras, L. A. Plana, J. Garside, S. Temple, F. Galluppi, C. Patterson, D. R. Lester, A. D. Brown, and S. B. Furber, "Spinnaker: A 1-w 18-core system-on-chip for massively-parallel neural network simulation," *IEEE Journal of Solid-State Circuits*, vol. 48, no. 8, pp. 1943–1953, 2013.

[16] A. Sengupta and K. Roy, "Spin-transfer torque magnetic neuron for low power neuromorphic computing," in *Neural Networks (IJCNN), 2015 International Joint Conference on*. IEEE, 2015, pp. 1–7.

[17] S. Park, A. Sheri, J. Kim, J. Noh, J. Jang, M. Jeon, B. Lee, B. Lee, B. Lee, and H. Hwang, "Neuromorphic speech systems using advanced reram-based synapse," in *Electron Devices Meeting (IEDM), 2013 IEEE International*. IEEE, 2013, pp. 25–6.

[18] L. Huang, A. D. Joseph, B. Nelson, B. I. Rubinstein, and J. D. Tygar, "Adversarial machine learning," in *Proceedings of the 4th ACM Workshop on Security and Artificial Intelligence*. ACM, 2011, pp. 43–58.

[19] N. Papernot, P. D. McDaniel, and I. J. Goodfellow, "Transferability in machine learning: from phenomena to black-box attacks using adversarial samples," *CoRR*, vol. abs/1605.07277, 2016. [Online]. Available: http://arxiv.org/abs/1605.07277

[20] A. Bagheri, O. Simeone, and B. Rajendran, "Adversarial training for probabilistic spiking neural networks," in *19th IEEE International Workshop on Signal Processing Advances in Wireless Communications, SPAWC 2018, Kalamata, Greece, June 25-28, 2018*, 2018, pp. 1–5.

[21] B. Liu, C. Yang, H. Li, Y. Chen, Q. Wu, and M. Barnell, "Security of neuromorphic systems: Challenges and solutions," in *Circuits and Systems (ISCAS), 2016 IEEE International Symposium on*. IEEE, 2016, pp. 1326–1329.

[22] C. Yang, B. Liu, H. Li, Y. Chen, W. Wen, M. Barnell, Q. Wu, and J. Rajendran, "Security of neuromorphic computing: thwarting learning attacks using memristor's obsolescence effect," in *Proceedings of the 35th International Conference on Computer-Aided Design*. ACM, 2016, p. 97.

[23] M. Farag and M. Ewais, "Smart employment of circuit redundancy to effectively counter trojans (secret) in third-party ip cores," in *International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, Dec 2014, pp. 1–6.

[24] D. M. Ancajas, K. Chakraborty, and S. Roy, "Fort-nocs: Mitigating the threat of a compromised noc," in *IEEE/ACM Design Automation Conference (DAC)*, 2014, pp. 1–6.

[25] A. Waksman, M. Suozzo, and S. Sethumadhavan, "Fanci: identification of stealthy malicious logic using boolean functional analysis," in *ACM Conference on Computer and Communications Security (CCS)*, 2013, pp. 697–708.

[26] A. Al-Anwar, Y. Alkabani, M. W. El-Kharashi, and H. Bedour, "Hardware trojan protection for third party ips," in *Euromicro Conference on Digital System Design (DSD)*, 2013, pp. 662–665.

[27] G. Bloom, B. Narahari, R. Simha, A. Namazi, and R. Levy, "FPGA soc architecture and runtime to prevent hardware trojans from leaking secrets," in *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST).* IEEE, 2015, pp. 48–51.

[28] M. Hicks, M. Finnicum, S. T. King, M. M. K. Martin, and J. M. Smith, "Overcoming an untrusted computing base: Detecting and removing malicious hardware automatically," in *IEEE/ACM International Symposium on Security and Privacy*, 2010, pp. 159–172.

[29] M. Rathmair, F. Schupfer, and C. Krieg, "Applied formal methods for hardware trojan detection," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, June 2014, pp. 169–172.

[30] J. Rajendran, V. Vedula, and R. Karri, "Detecting malicious modifications of data in third-party intellectual property cores," in *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, June 2015, pp. 1–6.

[31] M. Banga and M. S. Hsiao, "Trusted rtl: Trojan detection methodology in pre-silicon designs," in *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2010, pp. 56–59.

[32] K. Hu, A. N. Nowroz, S. Reda, and F. Koushanfar, "High-sensitivity hardware trojan detection using multimodal characterization," in *IEEE/ACM Design Automation & Test in Europe (DATE)*, 2013, pp. 1271–1276.

[33] K. Xiao and M. Tehranipoor, "Bisa: Built-in self-authentication for preventing hardware trojan insertion," in *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2013, pp. 45–50.

[34] Y. Jin and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *Proceedings of IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, 2008, pp. 51–57.

[35] J. Zhang, F. Yuan, L. Wei, Y. Liu, and Q. Xu, "Veritrust: Verification for hardware trust," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 7, pp. 1148–1161, 2015.

[36] C. Silvano, G. Palermo, S. Xydis, and I. S. Stamelakos, "Voltage island management in near threshold manycore architectures to mitigate dark silicon," in *DATE, Dresden, Germany, March 24-28, 2014*, 2014, pp. 1–6.

[37] U. R. Karpuzcu, K. B. Kolluru, N. S. Kim, and J. Torrellas, "Varius-ntv: A microarchitectural model to capture the increased sensitivity of manycores to process variations at near-threshold voltages," in *IEEE Dependable Systems and Networks (DSN)*, 2012, pp. 1–11.

[38] S. Mukhopadhyay, S. Ghosh, K. Kim, and K. Roy, "Low-power and process variation tolerant memories in sub-90nm technologies," in *Proceedings of the 1st ACM Symposium on Cloud Computing (SoCC)*, Sept 2006, pp. 155–159.

[39] E. Krimer, R. Pawlowski, M. Erez, and P. Chiang, "Synctium: a near-threshold stream processor for energy-constrained parallel applications," *IEEE Computer Architecture Letters*, vol. 9, no. 1, pp. 21–24, Jan 2010.

[40] M. R. Seifi and M. Eshghi, "Clustered noc, a suitable design for group communications in network on chip," *Journal on Computer and Electrical Engineering*, vol. 38, no. 1, pp. 82 – 95, 2012.

[41] A. Lankes, T. Wild, and A. Herkersdorf, "Hierarchical nocs for optimized access to shared memory and io resources," in *Euromicro Conference on Digital System Design (DSD)*, 2009, pp. 255–262.

[42] M. Janidarmian, A. Khademzadeh, and M. Tavanpour, "Onyx: A new heuristic bandwidth-constrained mapping of cores onto tile-based network on chip," *IEICE Electronics Express*, vol. 6, no. 1, pp. 1–7, 2009.

[43] G. V. Research. (2017, July) Artificial intelligence market analysis by solution (hardware, software, services), by technology (deep learning, machine learning, natural language processing, machine vision), by end-use, by region, and segment forecasts, 2014 - 2025. [Online]. Available: http://www.grandviewresearch.com/industry-analysis/artificial-intelligence-ai-market

[44] P. R. Cohen and E. A. Feigenbaum, *The handbook of artificial intelligence*. Butterworth-Heinemann, 2014, vol. 3.

[45] W. Haensch, "Scaling is overwhat now?" in *Device Research Conference (DRC), 2017 75th Annual*. IEEE, 2017, pp. 1–2.

[46] F. Rahman, B. Shakya, X. Xu, D. Forte, and M. Tehranipoor, "Security beyond cmos: Fundamentals, applications, and roadmap," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2017.

[47] M. Tehranipoor, H. Salmani, and X. Zhang, "Hardware trojan detection: Untrusted third-party ip cores," in *Integrated Circuit Authentication*. Springer, 2014, pp. 19–30.

[48] N. Bostrom, "Strategic implications of openness in ai development," *Global Policy*, vol. 8, no. 2, pp. 135–148, 2017.

[49] M. Hengstler, E. Enkel, and S. Duelli, "Applied artificial intelligence and trustthe case of autonomous vehicles and medical assistance devices," *Technological Forecasting and Social Change*, vol. 105, pp. 105–120, 2016.

[50] A. N. Burkitt, "A review of the integrate-and-fire neuron model: I. homogeneous synaptic input," *Biological cybernetics*, vol. 95, no. 1, pp. 1–19, 2006.

[51] E. Orhan, "The leaky integrate-and-fire neuron model," *no*, vol. 3, pp. 1–6, 2012.

[52] Ş. Mihalaş and E. Niebur, "A generalized linear integrate-and-fire neural model produces diverse spiking behaviors," *Neural computation*, vol. 21, no. 3, pp. 704–718, 2009.

[53] M. C. Van Rossum, G. Q. Bi, and G. G. Turrigiano, "Stable hebbian learning from spike timing-dependent plasticity," *Journal of neuroscience*, vol. 20, no. 23, pp. 8812–8821, 2000.

[54] G. Rovere, Q. Ning, C. Bartolozzi, and G. Indiveri, "Ultra low leakage synaptic scaling circuits for implementing homeostatic plasticity in neuromorphic architectures," in *IEEE International Symposium on Circuits and Systems (ISCAS)*, 2014, pp. 2073–2076.

[55] R. J. Shridevi, D. M. Ancajas, K. Chakraborty, and S. Roy, "Runtime detection of a bandwidth denial attack from a rogue network-on-chip," in *ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, 2015, pp. 8:1–8:8.

[56] M. A. Zidan, H. A. H. Fahmy, M. M. Hussain, and K. N. Salama, "Memristor-based memory: The sneak paths problem and solutions," *Microelectronics Journal*, vol. 44, no. 2, pp. 176–183, 2013.

[57] D. Querlioz, O. Bichler, and C. Gamrat, "Simulation of a memristor-based spiking neural network immune to device variations," in *Neural Networks (IJCNN), The 2011 International Joint Conference on.* IEEE, 2011, pp. 1775–1781.

[58] C. Yakopcic, T. M. Taha, G. Subramanyam, and R. E. Pino, "Memristor spice model and crossbar simulation based on devices with nanosecond switching time," in *Neural Networks (IJCNN), The 2013 International Joint Conference on.* IEEE, 2013, pp. 1–7.

[59] P. Boulet, P. Devienne, P. Falez, G. Polito, M. Shahsavari, and P. Tirilly, "N2s3, an open-source scalable spiking neuromorphic hardware simulator," Ph.D. dissertation, Université de Lille 1, Sciences et Technologies; CRIStAL UMR 9189, 2017.

[60] M. Shahsavari and P. Boulet, "Parameter exploration to improve performance of memristor-based neuromorphic architectures," *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

[61] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[62] L. Deng, "The mnist database of handwritten digit images for machine learning research [best of the web]," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[63] A. Waksman and S. Sethumadhavan, "Silencing hardware backdoors," in *IEEE/ACM International Symposium on Security and Privacy*, 2011, pp. 49–63.

[64] R. Karri, J. Rajendran, K. Rosenfeld, and M. Tehranipoor, "Trustworthy hardware: Identifying and classifying hardware trojans," *Proceedings of IEEE*, vol. 43, no. 10, pp. 39–46, 2010.

[65] J. Brill, "The intersection of consumer protection and competition in the new world of privacy," *Competition Pol'y Int'l*, vol. 7, pp. 7–313, 2011.

[66] T. E. Carlson, W. Heirman, and L. Eeckhout, "Sniper: exploring the level of abstraction for scalable and accurate parallel multi-core simulation," in *International Conference on Supercomputing*, 2011.

[67] Intel, "Intel core i7-4650u processor," http://ark.intel.com/products/75114/Intel-Core-i7-4650U-Proce

[68] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, "The splash-2 programs: Characterization and methodological considerations," in *International Symposium on Computer Architecture (ISCA)*, 1995, pp. 24–36.

[69] N. K. Choudhary, S. V. Wadhavkar, T. A. Shah, H. Mayukh, J. Gandhi, B. H. Dwiel, S. Navada, H. H. Najaf-abadi, and E. Rotenberg, "FabScalar: composing synthesizable rtl designs of arbitrary cores within a canonical superscalar template," in *International Symposium on Computer Architecture (ISCA)*, 2011, pp. 11–22.

[70] *Intel 64 and IA-32 Architectures Optimization Reference Manual*, Intel, September 2014, http://www.intel.com/content/www/us/en/processors/core/core-i7-processor.html.

[71] ARM, *ARM Cortex-A72 MPCore Processor Technical Reference Manual*, 2015, http://infocenter.arm.com/help/index.jsp?topic=/com.arm.doc.100095_0001_02_en/index.html.

[72] T. S. Karkhanis and J. E. Smith, "A first-order superscalar processor model," in *International Symposium on Computer Architecture (ISCA)*. IEEE, 2004, pp. 338–. [Online]. Available: http://dl.acm.org/citation.cfm?id=998680.1006729

[73] E. Stijn, L. Eeckhout, T. Karkhanis, and J. E. Smith, "A mechanistic performance model for superscalar out-of-order processors," *ACM Transactions on Computer Systems*, vol. 27, no. 2, pp. 3:1–3:37, May 2009. [Online]. Available: http://doi.acm.org/10.1145/1534909.1534910

[74] H. Wang, Y. Guo, I. Koren, and C. Krishna, "Compiler-based adaptive fetch throttling for energy-efficiency," in *IEEE International Symposium on Performance Analysis of Systems and Software*, March 2006, pp. 112–119.

[75] E. B. Wilson and M. M. Hilferty, "The distribution of chi-square," *Proc. Nat. Acad. Sci. USA*, vol. 17, no. 12, pp. 684–688, December 1931.

[76] I. Labs. (2010, 24 May) The scc platform overview. [Online]. Available: http://http://www.intel.com/content/dam/www/public/us/en/documents/technology-briefs/intel-la

[77] K. Flautner, N. S. Kim, S. Martin, D. Blaauw, and T. Mudge, “Drowsy caches: Simple techniques for reducing leakage power,” in *Proc. of 29th ISCA*, 2002, pp. 148–157.

[78] N. Jiang, D. Becker, G. Michelogiannakis, J. Balfour, B. Towles, D. Shaw, J. Kim, and W. Dally, “A detailed and flexible cycle-accurate network-on-chip simulator,” in *IEEE International Symposium on Performance Analysis of Systems and Software*, 2013, pp. 86–96.

[79] S. C. Woo, M. Ohara, E. Torrie, J. P. Singh, and A. Gupta, “The splash-2 programs: Characterization and methodological considerations,” in *International Symposium on Computer Architecture (ISCA)*. ACM, 1995, pp. 24–36.

[80] L. Chang, D. J. Frank, R. K. Montoye, S. J. Koester, B. L. Ji, P. W. Coteus, R. H. Dennard, and W. Haensch, “Practical strategies for power-efficient computing technologies,” *Proceedings of IEEE*, vol. 98, no. 2, pp. 215–236, 2010.

[81] D. Becker. (2012, August) Open source noc router rtl. [Online]. Available: https://nocs.stanford.edu/cgi-bin/trac.cgi/wiki/Resources/Router

[82] C. Sun, C.-H. Chen, G. Kurian, L. Wei, J. Miller, A. Agarwal, L.-S. Peh, and V. Stojanovic, “Dsent - a tool connecting emerging photonics with electronics for opto-electronic networks-on-chip modeling,” in *ACM/IEEE International Symposium on Networks-on-Chip (NOCS)*, 2012, pp. 201–210.

[83] W. Zhao and Y. Cao, “Predictive technology model,” June 2012. [Online]. Available: http://ptm.asu.edu/

[84] S. 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 Transactions on Semiconductor Manufacturing*, vol. 21, pp. 3 –13, 2008.

[85] H. Chen, D. Manzi, S. Roy, and K. Chakraborty, “Opportunistic turbo execution in NTC: exploiting the paradigm shift in performance bottlenecks,” in *IEEE/ACM Design Automation Conference (DAC)*, 2015, pp. 63:1–63:6.

[86] N. Weste and D. Harris, *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th ed. USA: Addison-Wesley Publishing Company, 2010.

CURRICULUM VITAE

Chidhambaranathan Rajamanikkam

Published Journal Articles

- **Chidhambaranathan Rajamanikkam**, Kurt Brenning, Andrew Deakin, Sanghamitra Roy and Koushik Chakraborty, *"TASPDetect: Reviving Trust in 3PIP By Detecting TASP Trojans"*, Proceedings of the 2018 Elsevier Science Publishers Journal in Microprocessors and Microsystems (MICPRO), vol. 56, pp. 76-83.
- **Chidhambaranathan Rajamanikkam**, Rajesh JS, Koushik Chakraborty and Sanghamitra Roy, *"Energy Efficient Network-on-Chip Architectures for Many-core Near-Threshold Computing System"*, Proceeding of the Journal of Low Power Electronics (JOLPE) (Under Review).
- **Chidhambaranathan Rajamanikkam**, Rajesh JS, Sanghamitra Roy and Koushik Chakraborty, *"Understanding Security Threats in Emerging Neuromorphic Computing Architecture"*, Proceedings of the Journal of Hardware and Systems Security (HaSS) (Under Review).
- Shamik Saha, Prabal Basu, **Chidhambaranathan Rajamanikkam**, Aatreyi Bal, Koushik Chakraborty and Sanghamitra Roy, *"SSAGA: SMs Synthesized for Asymmetric GPGPU Applications"*, Proceedings of the 2017 ACM Transactions on Design Automation of Electronic Systems (TODAES), vol. 49, pp. 1-20.
- Prabal Basu, **Chidhambaranathan Rajamanikkam**, Aatreyi Bal, Pramesh Pandey, Trevor Carter, Koushik Chakraborty and Sanghamitra Roy, *"FIFA: Exploring a Focally Induced Fault Attack Strategy in Near-Threshold Computing"*, Proceedings of 2018 IEEE Embedded Letters, pp 76-83.

- Prabal Basu, Pramesh Pandey, Aatreyi Bal, **Chidhambaranathan Rajamanikkam**, Koushik Chakraborty and Sanghamitra Roy, *"TITAN: Uncovering the Paradigm Shift in Security Vulnerability at Near-Threshold Computing"*, Proceedings of the 2018 IEEE Transactions on Emerging Topics in Computing (TETC).

- Rajesh JS, **Chidhambaranathan Rajamanikkam**, Koushik Chakraborty and Sanghamitra Roy, *"Securing Data-Center against Power Attacks"*, Proceedings of the 2019 Journal in Hardware and Systems Security (HASS) (Accepted for Publication).

- Milan Patnaik, **Chidhambaranathan Rajamanikkam**, Chirag Garg, Arnab Roy, V. Devanathan, Shankar Balachandran, and Kamakoti Veezhinathan, *"ProWATCh: A Proactive Cross-Layer Workload-Aware Temperature Management Framework for Low-Power Chip Multi-Processors"*, Proceedings of the 2015 ACM Journal of Emerging Technologies in Computing Systems (JETC), vol. 12, pp. 22:1–22:25.

Published Conference Papers

- **Chidhambaranathan Rajamanikkam**, Rajesh JS, Koushik Chakraborty and Sanghamitra Roy, *"BoostNoC: Power efficient network-on-chip architecture for near threshold computing"*, Proceedings of the 2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD), Austin, TX, 2016, pp. 1-8.

- Rajesh Jayashankara Shridevi, **Chidhambaranathan Rajamanikkam**, Koushik Chakraborty and Sanghamitra Roy, *"Catching the flu: emerging threats from a third party power management unit."*, Proceedings of the 2016 IEEE/ACM Design Automation Conference (DAC), 86:1-86:6

- Seetal Potluri, Satya Trinadh Adireddy, **Chidhambaranathan Rajamanikkam** and Shankar Balachandran, *"LPScan: An algorithm for supply scaling and switching activity minimization during test"*, Proceedings of the 2013 IEEE International Conference on Computer Design (ICCD), Asheville, NC, 2013, pp. 463-466.