République Algérienne Démocratique et Populaire
الجمهورية الجزائرية الديمقراطية الشعبية
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
وزارة التعليم العالي و البحث العلمي

Djilali Bounaama Khemis Miliana University
جامعة الجلالي بونعامه خميس مليانة

Faculty of Science and Technology Department of Mathematics and Computer Science

Final graduation Project

Thesis for obtaining a Master's degree in Computer Science

Option: Software Engineering and Distributed Systems

Analysis of NoC technologies for machine learning based on network-on-chip(M/L NoC)

Realized by:

Younes Messous
AbdLatif Hamadouche

In front of the jury members:

Mr. Belhacene Bahloul	President
Mr. Riad Meghatria	Examiner
Mr. Abdelhamid Hariche	Supervisor

Academic Year: 2022 / 2023

Abstract

Multicore integrated circuits such as Network on Chip or NoC are popular nowadays due of their high performance, they appeared to eliminate the increase in latency and power consumption thanks to the communication based on transmits digital packets instead of analog signals. The architecture of NoC is addressed as promising tool for the implementation of the machine learning algorithm. The routers in the NoCs will be trained by the M/L algorithm to transfer the packets from the source router to the destination routers. M/L algorithms based on supervised learning are very efficient in terms of speed and power consumption. The parameters are to be considered during the architecture design of M/L NoC like: features concerns, topologies and routing, QoS, protocols, fault tolerance, buffering concerns and real-time requirements. This work aims to identify significant parameters reliable for ML/NoC by using simulations on real case studies provided by the available NoC tools for the NoC researchers.Hence to discuss the results and extract a formula to identify some significant parameters reliable for ML/NoC.

Keywords : Network-on-Chip (NoC),Machine Learning (ML),Fault Detection,Routing Algorithms ,Simulations analysis,Evaluation Metrics, Model Training

Résumé

Les circuits intégrés multicœurs tels que le réseau sur puces ou NoC sont populaires de nos jours en raison de leurs hautes performances, ils sont apparus pour éliminer l'augmentation de la latence et de la consommation d'énergie grâce à la communication basée sur la transmission de paquets numériques au lieu de signaux analogiques.

L'architecture de NoC est considérée comme un outil prometteur pour l'implémentation de l'algorithme d'apprentissage automatique. Les routeurs des NoC seront formés par l'algorithme M/L pour transférer les paquets du routeur source vers les routeurs de destination. Ces algorithmes M/L basés sur l'apprentissage supervisé sont très efficaces en termes de vitesse et de consommation d'énergie.

Les paramètres à prendre en compte lors de la conception de l'architecture du M/L NoC sont variés : problèmes de fonctionnalités, topologies et routage, QoS, protocoles, tolérance aux pannes, problèmes de mise en mémoire tampon et exigences en temps réel.

Ce travail vise à identifier les paramètres significatifs fiables pour ML/NoC en utilisant des simulations sur des études de cas réels fournies par les outils NoC disponibles pour les chercheurs dans ce type de technologie. Par conséquent, discuter des résultats et extraire une formule pour identifier certains paramètres significatifs fiables pour ML/NoC.

Mots clés :Réseaux sur puces (RsP), Apprentissage Automatique(AA), Détection de pannes, Algorithmes de routage,Analyse des simulations, Mesures d'évaluation, modèles d'apprentissage.

ملخص

الدوائر المتكاملة متعددة النواة مثل الشبكة على الرقائق أوNoC شائعة في الوقت الحاضر نظرًا لأدائها العالي ، فقد ظهرت للقضاء على الزيادة في زمن الوصول واستهلاك الطاقة بفضل الاتصال القائم على الحزم الرقمية المرسلة بدلاً من الإشارات التناظرية.

تمت معالجة بنية NoC كأداة واعدة لتنفيذ خوارزميات التعلم الآلي M / L و سيتم تدريب الروترات المبنية على تكنولوجيا الـ NoCs بواسطة خوارزمية M / L لنقل الحزم من المصدر إلى الوجهة وتعد الخوارزميات القائمة على التعلم الألي فعالة للغاية من حيث السرعة واستهلاك الطاقة.

يجب أخذ مجموعة من المقومات في الاعتبار أثناء تصميم بنية M / L NoC و التي تعتبر جد متنوعة: السمات المحددة للتكنولوجيا ، والطبولوجيا وخوارزميات التوجيه ، وجودة الخدمة ، والبروتوكولات ، والتسامح مع الأخطاء ، ومخاوف التخزين المؤقت ، ومتطلبات الوقت الفعلي.

يهدف هذا العمل إلى تحديد المقومات المهمة التي يمكن الاعتماد عليها في ML / NoC باستخدام عمليات المحاكاة على دراسات الحالة الحقيقية التي توفرها أدوات محاكاة NoC المتاحة لباحثي هذه التكنولوجيا ومن ثم مناقشة النتائج واستخراج صيغة لتحديد بعض المقومات المهمة المعتمدة لـ ML / NoC.

الكلمات المفتاحية : Network-on-Chip (NoC) ، التعلم الآلي (ML) ، كشف الأعطال، خوارزميات التوجيه ، دراسة المحاكات، معايير التقييم ، نموذج التدريب.

Dedication

Our deep gratitude goes first to Allah Almighty for granting me the ability to complete this research successfully.

I would like to extend my heartfelt appreciation to my beloved mother, whose loyalty, generosity, and kindness have been a constant source of support and inspiration. Her love and guidance have illuminated my path and enriched my life in countless ways.

I am also immensely grateful to my dear father, whose unwavering dedication and sacrifices have shaped me into the person I am today. His wisdom and encouragement have been invaluable throughout this journey.

To my dear brother, Ayoub, I am profoundly thankful for your unwavering support and companionship. Your presence in my life has been a blessing beyond measure.

I would also like to express my gratitude to all my sisters, who have always been there for me with their love and encouragement.

Lastly, I would like to extend my appreciation to my cherished friends for their laughter, camaraderie, and the cherished memories we have shared. Your friendship has been a constant source of joy and support.

Younes

Dedication

Praise be to Allah who made it possible for me to complete my studies and complete this research. Praise be to Allah who made use of people who supported me and supported me.

I extend my sincere thanks to Mr. Hariche Abdelhamid for all the help he gave us, whether financial or moral support and also for his patience with us from the many questions and the many calls, especially at night, as he was staying up with us until after one in the morning, I thank him very much for helping us with every completion This search is successful.

I would like to express my sincere appreciation to my beloved mother, whose sincerity, generosity, and kindness have been a constant source of support and inspiration. Her love and guidance have enlightened my path and enriched my life in countless ways.

I am also extremely grateful to my dear father, whose unwavering dedication and sacrifice made me the person I am today. His wisdom and encouragement have been invaluable throughout this journey.

I would like to express my sincere appreciation to my wife, who supported me in everything, despite all the things that weighed her down from raising children (Abu Bakr and Ghaith) to household matters, but she did not complain, but rather assured me of completing my studies and even helped me in my research, thanks for everything

I would also like to express my gratitude to all my brothers, who have always been of help to me, especially my brother Abdul Karim, who carried all the burdens of the house on his shoulders and did not devote himself to studying.

I would also like to thank the teachers who contributed to my arrival at this stage, they are the ones who taught me everything I know and they

are the ones who guided me through them to get to where I am now, thank you

Finally, I would like to express my appreciation to my dear friends for their support, friendship, and fond memories we shared. Your friendship has been a constant source of joy and support.

AbdeLLatif

Acknowledgement

Our deep gratitude goes first to Allah, the Almighty for giving us strength and determination to fulfill this work.

We would like to take this opportunity to express our profound gratitude and deep regard to our supervisor, Dr.Harriche Abdelhmid whose constructive guidance and permanent motivation were the spring that fed our ambition to terminate this work.

Our sincere thanks go to the jury members who have honored us by accepting to examine this work. We are also grateful to all our teachers for their motivation and care during our master's studies and for believing in our capacities.

To all the people who have contributed from near or far, in a direct or indirect way to the development and the success of this work.

Table of content

List of Figures

List of Tables tableaux

Introduction

The rapid growth of complex embedded systems and the increasing demand for high-performance computing have led to the emergence of Network-on-Chip (NoC) architectures as a promising solution. NoCs provide efficient communication and data exchange among various components of these systems, such as processors, memory, and peripherals. However, the reliable and efficient operation of NoC systems faces significant challenges, primarily due to non-deterministic phenomena and the failure of routing algorithms.

One of the major issues in NoC systems is the occurrence of non-deterministic phenomena, which can lead to unexpected behavior and system failures. These phenomena include clock faults, communication errors, congestion, and other unpredictable events. Traditional deterministic approaches may not be effective in handling such dynamic and unpredictable situations.

Machine Learning (ML) has emerged as a powerful tool for addressing non-determinism and improving the performance and reliability of NoC systems. ML techniques enable NoCs to adapt, learn, and make intelligent decisions based on data inputs, thereby mitigating the impact of non-deterministic phenomena. By leveraging ML algorithms, NoCs can detect and prevent failures in routing algorithms, optimize system performance, and enhance fault tolerance mechanisms.

The focus of this master's thesis is to analyze NoC technologies for machine learning-based Network-on-Chip (M/L NoC) systems and investigate how ML can be implemented to detect and prevent failures in routing algorithms. The goal is to develop a comprehensive understanding of the challenges posed by non-determinism in NoC systems and explore the po-

tential of ML techniques in addressing these challenges. By analyzing NoC technologies and leveraging the capabilities of ML, this thesis aims to contribute to the development of robust and efficient M/L NoC systems. The findings and insights obtained from this research can provide valuable guidance for designing and implementing reliable and adaptive NoC architectures that can effectively handle non-deterministic phenomena and prevent failures in routing algorithms.

The manuscript is structured as follows: - Chapter 1 provides an overview of NoCs, including definitions, commonly used technologies, and approaches to ensure deadlock-free systems. This chapter lays the foundation for understanding the fundamental concepts of NoCs and their importance in modern computing systems. - Chapter 2 focuses on NoC Analysis Methods and NoC simulators, presenting a comparative study of existing techniques for analyzing and evaluating NoC architectures. This chapter explores various analysis methods. It also investigates different NoC simulators and their capabilities in simulating and studying the behavior of NoC systems. - Chapter 3 focuses on the implementation of ML techniques in NoC systems to detect and prevent failures in routing algorithms. It explores different ML algorithms, data processing techniques, and optimization strategies that can be applied to enhance the performance, adaptability, and fault tolerance of NoCs. - Chapter 4 presents the experimental results and findings obtained from implementing the ML/NoC model. It discusses the effectiveness of the proposed strategyies in detecting and preventing failures in routing algorithms, optimizing system performance, and improving the reliability of NoC systems. - Finally, the thesis concludes with a summary of the main contributions, a discussion of the obtained results, and outlines future perspectives for further research in this field.

Chapter 1

Dead-lock free NoC Architecture

Introduction

In modern computer systems, the increasing demand for high-performance and energy-efficient communication has led to the emergence of Network-on-Chip (NoC) architectures. NoCs provide scalable and efficient communication infrastructures by replacing traditional bus-based or point-to-point connections with a network of interconnected routers .

This chapter focuses on providing a comprehensive understanding of Network-on-Chip (NoC) architectures, including their definitions and benefits. By exploring the definitions and benefits of NoCs in greater detail, this chapter aims to shed light on the significance and advantages of adopting NoC architectures in high-performance and energy-efficient communication systems. By the next section we will start exploring the definitions and benefits of NoCs in greater detail.

1.1 NoC definitions

A Network-on-Chip (NoC) [34] is a communication architecture employed in computer systems to facilitate efficient and scalable communication among various components, including processors, memory units, and input/output devices. Unlike traditional approaches like shared buses or point-to-point connections, NoCs utilize a mesh or network of intercon-

nected routers to transmit data between these components. The routers collectively form the network infrastructure, communicating with each other by exchanging data packets through the network via a routing algorithm (Figure1.1).

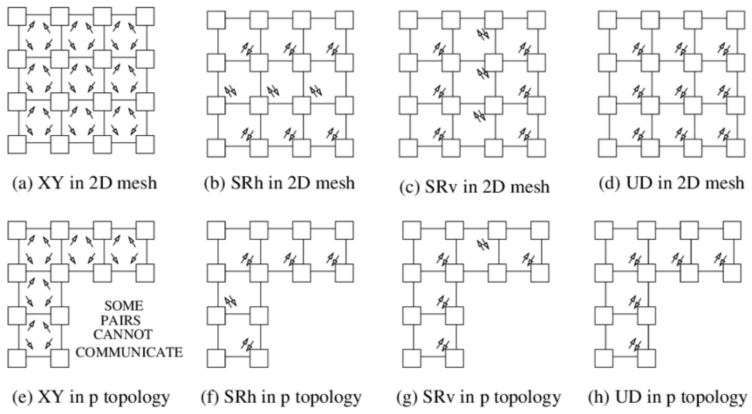


Figure 1.1: NoC routing algorithm.

In short, Network-on-Chip (NoC) is a communications architecture that uses interconnected routers between various components in computer systems, which makes NoCs suitable for a wide variety of applications and system designs. [34]

1.2 Advantages of Network on Chip (NoC)

NoCs offer numerous advantages and benefits that make them a valuable solution for addressing communication challenges in modern electronic systems. In this section, we will explore the advantages and benefits provided by Network-on-Chip (NoC) architectures.Here are some of the key advantages of NoC:

1.2.1 scalability

NoCs offer excellent scalability [1], allowing for the integration of a large number of components on a single chip. As the complexity of electronic systems continues to grow, NoCs provide a scalable solution by accommodating an increasing number of processing elements, memory units, and other functional units. This scalability [1] is essential for building com-

plex systems such as multi-core processors or highly integrated systems on chips (SoCs).

1.2.2 Performance

NoCs provide high-performance communication infrastructure for on-chip communication. By employing parallel communication paths and enabling concurrent data transfers, NoCs reduce communication latency and improve overall system throughput. The ability to transfer data efficiently and quickly between components enhances the overall performance of the system.

1.2.3 Flexibility

NoCs offer flexibility in terms of communication protocols and network topologies [19]. Different components can communicate using various protocols, and the network topology can be customized to meet the specific requirements of the system. This flexibility allows designers to optimize the communication infrastructure for different applications and adapt it to evolving needs.

1.2.4 Power efficiency

NoCs can be designed to be power-efficient, helping to minimize energy consumption in electronic systems. By employing efficient routing algorithms, power management techniques, and dynamic voltage and frequency scaling (DVFS), NoCs can reduce power consumption while maintaining high-performance levels. Power efficiency is crucial for battery-powered devices, mobile devices, and energy-constrained systems.

1.2.5 Fault tolerance and reliability

NoCs can incorporate fault tolerance mechanisms, ensuring reliable communication in the presence of failures or errors. Redundancy, error detection, and error correction techniques can be implemented at the network level to enhance system resilience. The fault tolerance capability of NoCs

contributes to the overall reliability of the system, particularly in safety-critical applications.

1.2.6 Design productivity

NoCs simplify the design process by providing a standardized communication infrastructure. With pre-defined interfaces and protocols, system designers can focus on the functional aspects of the components rather than dealing with the intricacies of point-to-point connections. NoCs reduce design complexity, enable better modularity, and facilitate the integration of intellectual property (IP) blocks, resulting in improved design productivity and faster time-to-market.

1.2.7 Traffic management and Quality of Service QoS

NoCs allow for efficient traffic management and Quality of Service QoS provisions. Different traffic types, such as real-time or latency-sensitive traffic, can be prioritized and allocated appropriate bandwidth to ensure their timely delivery. QoS mechanisms enable the efficient utilization of network resources and support the performance requirements of specific applications or tasks.

1.2.8 Integration with heterogeneous components

NoCs facilitate the integration of heterogeneous components, such as CPUs, GPUs, accelerators, and memory units, onto a single chip. This integration allows for the efficient utilization of specialized hardware for different tasks, optimizing performance, and energy efficiency. NoCs enable the seamless communication between these components, enhancing the overall system capabilities.

These advantages make NoCs a valuable solution for addressing the communication challenges in modern electronic systems. They provide scalability [1], high performance, flexibility, power efficiency, reliability, and design productivity, enabling the development of complex and efficient systems on chips. [5]

1.3 Characteristics of NoC

This kind of communication technology has several main properties that identify the complexity of such a system integrated on chips (or so called SoCs), here are some key characteristics :

1.3.1 Typologies

The topology of a Network on Chip (NoC) refers to the physical layout of the on-chip communication network connecting the processing elements, such as processors, memory, and input/output devices. The choice of topology is a critical design decision that affects the performance, power consumption, and scalability [1] of the NoC. There are several types of topologies [19] used in NoCs . [30]

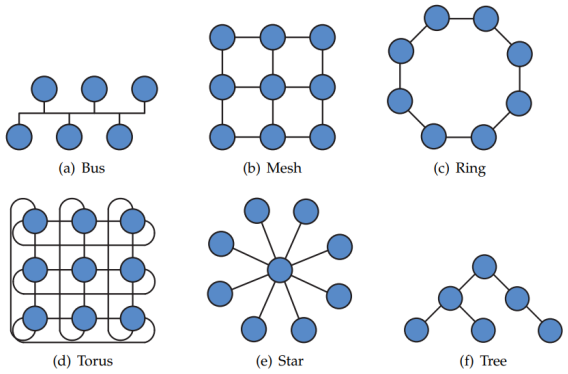


Figure 1.2: Six different common network topologies [16].

1.3.1.1 Bus

The topology shown in figure 1.2(a) represents a shared bus. It has low implementation cost, An efficient floorplan of a bus-based system is limited to a small number of components due to the long wires required in larger systems. The bisection width for N nodes is 1. Thus, it is independent of the number of nodes, limiting the scalability [1] of the bus.

1.3.1.2 Mesh

In the Mesh topology [16], processing cores and other components are arranged in a grid-like structure, with each component connected to its adjacent components. This creates a regular and homogeneous structure that is easy to implement and maintain. However, the Mesh topology suffers from high communication latency due to the long paths between distant components.

1.3.1.3 Butterfly

The Butterfly topology is a hierarchical structure that organizes the processing cores and other components in a tree-like fashion. Each level of the tree represents a different layer of hierarchy, with the top layer representing the root of the tree. The Butterfly topology is suitable for implementing large-scale systems with a high degree of fault tolerance, as it provides multiple redundant paths for data transmission.

1.3.1.4 Baseline

The Baseline topology is a simple and efficient structure that consists of a linear array of processing cores and other components. Each component is connected to its two neighbors, forming a one-dimensional structure. The Baseline topology is easy to implement and maintain, and it is suitable for applications that require low latency and high bandwidth.

1.3.1.5 Omega

The Omega topology is a mesh-based structure that uses multiple paths to reduce communication latency and increase bandwidth. The Omega topology organizes the processing cores and other components in a two-dimensional grid, with each component connected to its nearest neighbors in both horizontal and vertical directions. This creates multiple paths for data transmission, which reduces the communication latency and improves the system's overall performance. Overall, the choice of topology for a given NoC design depends on the specific requirements of the application,

such as latency, bandwidth, and power consumption. Each topology has its advantages and disadvantages, and designers must carefully evaluate them to select the most appropriate one for the application.

1.3.2 Buffering

Buffering refers to the temporary storage of data packets during transmission between the source and destination nodes within the network. The purpose of buffering is to handle variations in data traffic, accommodate different processing speeds of components, and prevent data loss or congestion .

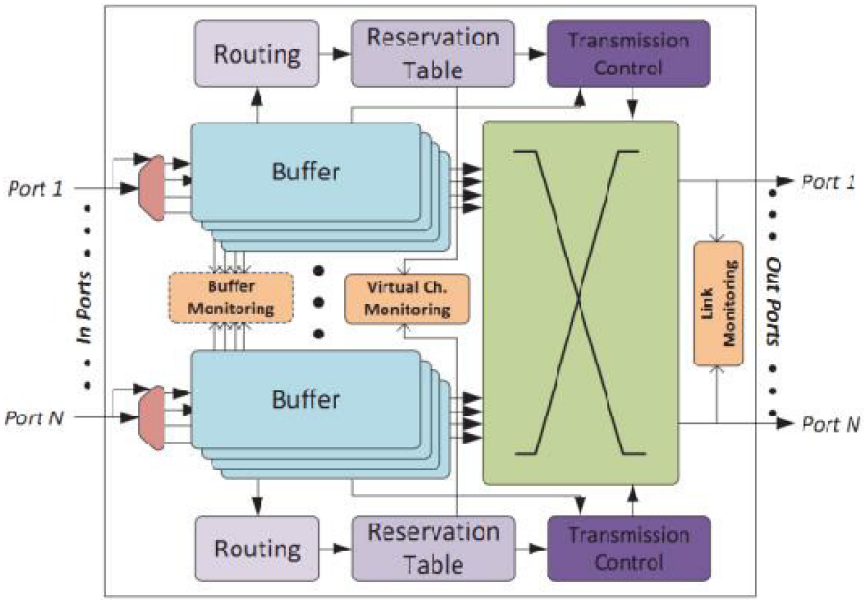


Figure 1.3: diagram of a packet switching router.

- **Buffer Types:**NoCs can employ different types of buffers (we can mention as example Figure 1.4) based on the specific requirements and design choices. Some common buffer types include input buffers, output buffers, and virtual channel buffers.

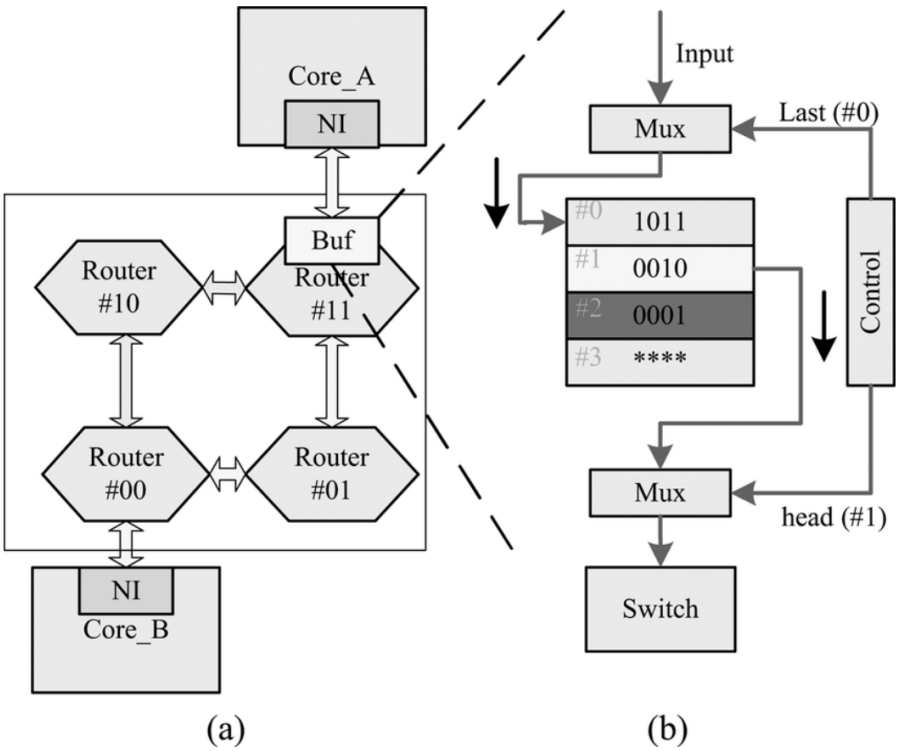


Figure 1.4: NoC architecture. (a) NoC. (b) Buffer.

- **Input Buffering:** Input buffers are located at the input ports of the routers and temporarily hold incoming data packets until they can be routed to the appropriate output ports.
- **Output Buffering:** Output buffers are situated at the output ports of the routers and store packets that are waiting to be transmitted to their destination nodes.
- **Virtual Channel Buffering:** Virtual channel buffering is a technique that uses separate buffers for different traffic classes or virtual channels within the NoC. It allows for better management of traffic with varying priorities, ensuring Quality of Service (QoS) requirements.
- **Buffer Size:** The size of the buffers in a NoC depends on factors such as the expected data traffic, the desired latency, and the overall system requirements. Larger buffer sizes can help accommodate bursts of traffic and reduce the likelihood of congestion or packet loss. However, larger buffers also require more area and power resources, so there is

a trade-off between buffer size and resource utilization.

- **Buffer Management:** The management of buffers in NoCs involves controlling the allocation and deallocation of buffer space as packets arrive and are transmitted. Various algorithms and policies can be employed to efficiently manage buffer space, such as First-In-First-Out (FIFO) scheduling, priority-based scheduling, or dynamic buffer allocation techniques based on traffic patterns.

1.3.3 Buffering Strategies

Different buffering strategies can be utilized in NoCs, depending on the specific system requirements and design goals. Some common buffering strategies include store-and-forward and wormhole routing.

- **Store-and-Forward:** In store-and-forward buffering, the entire packet is received and stored in the buffer before it is forwarded to the next hop. This strategy ensures complete packet integrity but introduces higher latency.
- **Wormhole Routing:** Wormhole routing breaks the packets into smaller flits (flow control digits) and transmits them in a pipelined manner through the network. This strategy allows for reduced latency but may require additional control mechanisms to handle flow control and ensure proper packet reassembly at the destination [39].

  Buffering is a critical component of NoC architectures as it helps manage data flow, handle congestion, and maintain reliable communication between nodes. Effective buffering strategies and management techniques play a significant role in optimizing performance, latency, and overall system efficiency in NoC-based systems.

1.3.4 NoC Routing algorithms

Routing algorithms [2] in a Network-on-Chip (NoC) system can be classified into two main categories: Deterministic Algorithms and Adaptive

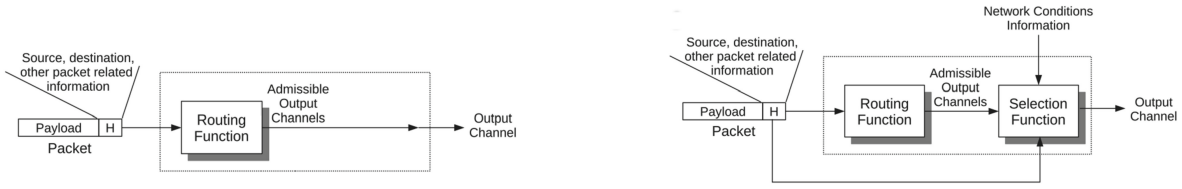
Algorithms. Each category offers different approaches for routing data within the NoC.

1.3.4.1 Deterministic algorithms

Deterministic algorithms in NoC use a predefined path for data routing, ensuring consistent and predictable communication. These algorithms do not require a selection block as they rely on a fixed path established beforehand. With deterministic algorithms, the routing function always returns the same fixed path, simplifying the routing process without the need for dynamic evaluations. This approach offers advantages in terms of simplicity, determinism, and lower hardware requirements and implementation complexity compared to adaptive algorithms [27]. However, deterministic algorithms have limitations in adapting to changing network conditions and dynamic traffic patterns. They cannot adjust the routing path based on factors like congestion or buffer occupancy. In scenarios where frequent network status changes or dynamic path selection is necessary, adaptive algorithms are more suitable. Adaptive algorithms offer the flexibility to select optimal paths based on real-time network conditions and can adapt to dynamic scenarios.(figure1.5a)

1.3.4.2 Adaptive Routing Algorithms

Adaptive algorithms in NoC dynamically adjust routing paths based on network conditions, optimizing data transfer. They offer flexibility, responding to changes and selecting efficient routes. A selection function evaluates factors like link utilization, congestion, choosing the best path. Although adaptive algorithms improve performance and reduce latency, they require extra memory and introduce processing overhead. The choice depends on system requirements: adaptive algorithms excel in dynamic scenarios, while deterministic algorithms suffice for simpler, predictable communication.(figure1.5b)



(a) The deterministic algorithm mechanism

(b) The mechanism of adaptive algorithms

Figure 1.5: The mechanism algorithms

1.3.5 On built Technologies of NoC

Each of the following technologies including ASIC (Application-Specific Integrated Circuit), FPGA [33] (Field-Programmable Gate Array), and VLSI (Very Large Scale Integration) serve a specific purpose and offer unique advantages in NoC design and implementation according to the performance/ flexibility rate.

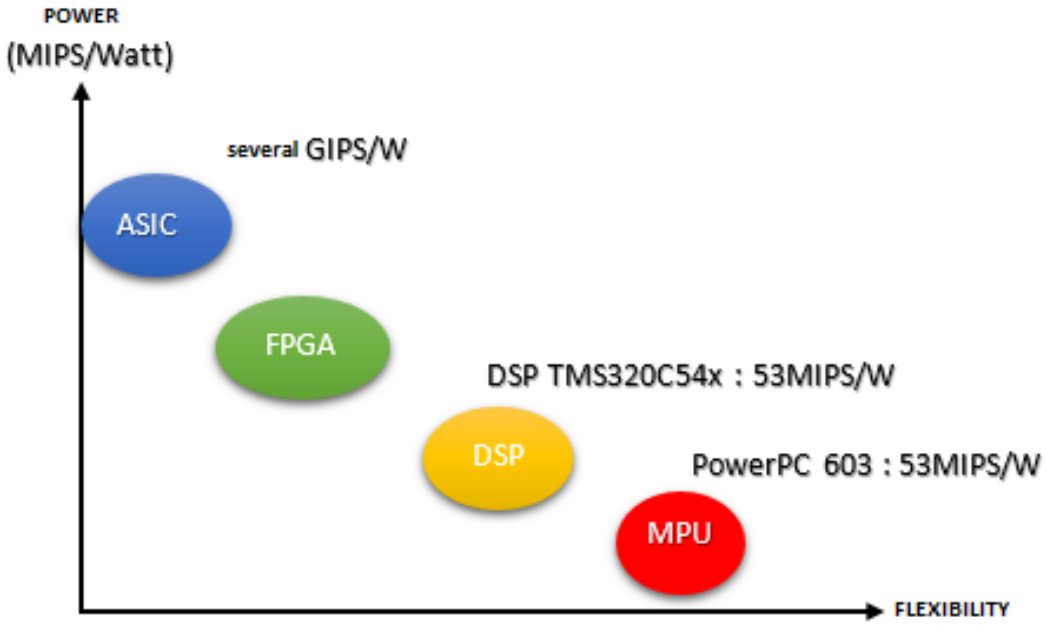


Figure 1.6: Performance / flexibility ratio for the main technologies.

1.3.5.1 ASIC (Application-Specific Integrated Circuit)

ASIC [14] refers to integrated circuits that are specifically designed for a particular application or task. These circuits are custom-built to perform a dedicated function, such as implementing the routing and switching functionalities of a NoC. ASICs offer high performance, low power consump-

tion, and optimized area utilization. They are typically designed using hardware description languages (HDLs) and undergo fabrication in semiconductor foundries. ASICs are widely used in commercial and industrial applications that require specialized and efficient hardware solutions.

1.3.5.2 FPGA (Field-Programmable Gate Array)

FPGAs [14] are integrated circuits that can be programmed and reprogrammed to implement various digital logic functions, including NoC components. Unlike ASICs [14], FPGAs offer flexibility and reconfigurability [8], allowing designers to modify the hardware design after fabrication. This flexibility makes FPGAs suitable for rapid prototyping, testing, and development of NoC architectures. Designers can use hardware description languages (HDLs) or high-level synthesis tools [3] to program FPGAs and implement the desired NoC functionality.

1.3.5.3 VLSI (Very Large Scale Integration)

VLSI [24] refers to the technology of integrating a large number of transistors and other electronic components into a single chip. VLSI technology enables the creation of complex and highly integrated circuits, including NoCs. It involves designing and fabricating integrated circuits with millions or even billions of transistors on a single chip. VLSI technology allows for compact and efficient NoC designs, with reduced power consumption and improved performance. It is widely used in the semiconductor industry for developing advanced electronic systems and integrated circuits.

These technologies, namely ASIC, FPGA, and VLSI, provide different options for implementing NoC architectures. Designers can choose the most suitable technology based on factors such as performance requirements, design flexibility, power consumption, and time-to-market considerations. By leveraging these technologies effectively, designers can create efficient and scalable NoC designs that meet the communication needs of modern computer systems.

1.4 On-Fault tolerant NoC

A ”deadlock-free” or ”On-fault tolerant NoC” Network-on-Chip (NoC) refers to a design or architecture of a communication network within a chip that guarantees the absence of deadlocks.

A deadlock is a situation in which two or more processes or entities are unable to proceed because each is waiting for the other to release a resource or take an action. In the context of NoCs, deadlocks can occur when multiple packets or messages are circulating in the network, and their paths intersect, leading to a situation where none of the packets can progress.

In a deadlock-free NoC, specific mechanisms and protocols are implemented to prevent the occurrence of deadlocks. These mechanisms may include:

- Routing algorithms

  The NoC employs routing algorithms that ensure packets are guided along conflict-free paths, avoiding situations where packets can get stuck in cycles or loops .

- Virtual Channels:

  Virtual Channels VC [16] provide independent lanes or buffers within the NoC for transmitting packets. Each VC has its own resources, and careful allocation and management of these resources help prevent deadlocks.

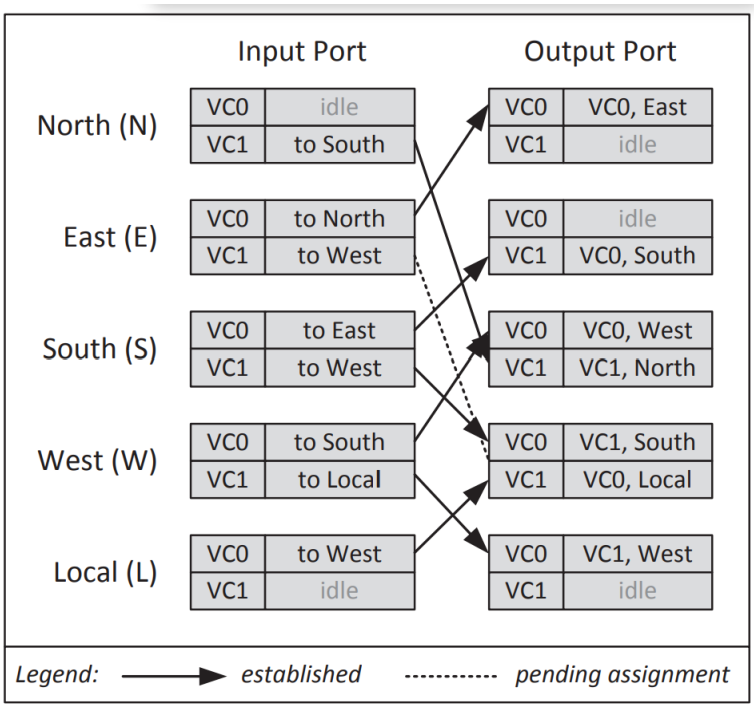


Figure 1.7: Virtual Channels [16].

- Flow control: Proper flow control mechanisms are employed to manage the movement of packets in the NoC. This can include techniques such as credit-based flow control or wormhole routing, which ensure that packets are only sent when there is sufficient capacity available in the network.

- Deadlock detection and recovery:

  In some cases, deadlock detection and recovery mechanisms may be implemented to identify and resolve deadlocks if they occur. This could involve aborting and re-routing packets or employing additional protocols to break the deadlock.

  By ensuring that these mechanisms are in place, a deadlock-free NoC minimizes the risk of communication bottlenecks and ensures the efficient and continuous flow of data within the network, leading to improved performance, reliability, and scalability [1] of the system. [17]

1.4.1 Failure source

In the context of Network-on-Chip (NoC) analysis, it is important to consider different types of failures that can occur in the system. Failures can be broadly categorized into two main types: hardware failures and software failures. Understanding these failures is crucial for ensuring the reliability and robustness of NoC architectures.

1.4.1.1 Hardware Failures

Hardware failures refer to failures or malfunctions that occur in the physical components of the NoC system. These failures can be caused by various factors, including manufacturing defects, aging effects, environmental conditions, voltage fluctuations, and physical damage. Hardware failures can manifest as component failures, interconnect failures, power supply failures, clock signal issues, and so on. When a hardware failure occurs in the NoC, it can lead to disruptions in communication, performance degradation, and even system crashes.

- Component Failure:

  This refers to the failure [19] of individual components in the NoC, such as routers, links, or buffers. It can occur due to manufacturing defects, wear and tear, or environmental factors. A component failure can lead to a loss of connectivity or degraded performance in the NoC.

- Interconnect Failure:

  Interconnect failures [19] involve issues with the connections between components in the NoC. It can be caused by wiring faults, signal integrity problems, or faulty routing paths. An interconnect failure can result in data corruption, packet loss, or increased latency.

- Power Supply Failure:

  Power supply failures [19] occur when there is a disruption in the delivery of power to the NoC components. It can be caused by voltage fluctuations, power surges, or inadequate power distribution. A power

supply failure can cause system crashes, loss of data, or incorrect operation of components.

- Clock Signal Issues:

  Clock signal failures involve problems with the synchronization of clocks in the NoC. It can be due to clock skew, jitter, or clock distribution problems. Clock signal issues can lead to timing violations, data synchronization errors, or improper coordination between components.

1.4.1.2 Software Failures

Software failures, on the other hand, pertain to failures or errors that occur in the software components of the NoC system. These failures can result from bugs, programming errors, design flaws, memory corruption, software conflicts, or external attacks. Software failures can cause communication errors, incorrect data processing, deadlock situations, timing violations, and other system-level issues. Detecting and addressing software failures is crucial to ensure proper functionality, performance, and security of the NoC.

- Bug or Programming Error:

  This type of failure occurs when there are coding mistakes or logical errors in the software running on the NoC. It can result in incorrect data processing, communication errors, or system instability. Bugs and programming errors are commonly addressed through debugging, testing, and code review processes.

- Memory Corruption:

  Memory corruption failures involve errors in memory access or management. It can lead to data corruption, program crashes, or security vulnerabilities. Memory corruption can occur due to buffer overflows, pointer errors, or uninitialized memory usage.

- Deadlock:

  A deadlock is a situation where two or more components in the NoC are waiting for each other to release resources, resulting in a system freeze. Deadlocks can occur due to improper resource allocation, synchronization issues, or incorrect locking mechanisms.

- Timing Violations:

  Timing violations refer to violations of timing constraints in the NoC. It can occur when tasks or processes do not complete within the specified time limits. Timing violations can lead to incorrect data transfers, missed deadlines, or performance degradation.

It is important to note that hardware and software failures are interconnected, as a hardware failure can trigger software errors and vice versa. Therefore, NoC analysis methods should consider both types of failures and their potential interactions. Various techniques, such as fault tolerance mechanisms, error detection and correction algorithms, redundancy schemes, and software testing approaches, can be employed to mitigate the impact of failures and enhance the reliability of NoC systems.

By studying and analyzing both hardware and software failures in NoC architectures, designers and researchers can gain insights into potential vulnerabilities, identify areas for improvement, and develop robust strategies to ensure the dependable operation of the system.

1.4.2 NoC-Failures examples

There are several types of faults that can occur in a Network-on-Chip (NoC) architecture [19]. Some of the most common faults include:

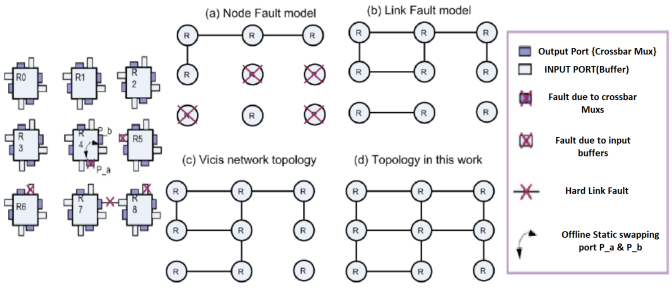


Figure 1.8: Fault models in NoC [19]

1.4.2.1 Link faults

These occur when there is a problem with the communication links between different components of the NoC, such as routers or processing elements. Link faults can result in data transmission errors or dropped packets.

1.4.2.2 Router faults

These occur when there is a problem with the routing functionality of the NoC, such as incorrect routing decisions or routing table errors. Router faults can result in data transmission errors or delays.

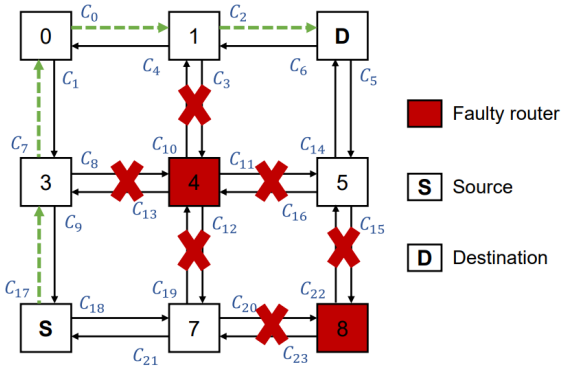


Figure 1.9: A mesh-based NoC of size 3x3 with two faulty routers [19]

1.4.2.3 Power supply faults

These occur when there is a problem with the power supply to the components of the NoC, such as voltage drops or power outages. Power supply faults can result in system crashes or data corruption.

1.4.2.4 Temperature faults

These occur when there is a problem with the temperature control of the NoC, such as overheating or thermal runaway. Temperature faults can result in system crashes or damage to the components of the NoC.

1.4.2.5 Clock faults

These occur when there is a problem with the clock signals used to synchronize the components of the NoC, such as clock skew or clock jitter. Clock faults can result in timing errors or synchronization issues.

To mitigate these faults, various fault-tolerant techniques can be employed, such as redundancy, error-correcting codes, and dynamic voltage and frequency scaling. Additionally, fault diagnosis and recovery mechanisms can be implemented to detect and isolate faults, and to restore the system to a stable state.

1.5 Network on Chips Examples

we explores several noteworthy examples of Network-on-Chip (NoC) implementations in diverse systems and applications:

1. ARM's AMBA Network Interconnect: ARM offers the AMBA Network Interconnect as part of its IP library. It comprises crossbars and bridges that can be flexibly assembled to create a hierarchical bus interconnect closely resembling an NoC. This design allows for greater adaptability in floor planning and enables the addition of buffering to prevent stalls effectively.

2. Tilera TILE-Gx processor: An impressive integration of 100 cores on a chip, interconnected by a 2D mesh network-based NoC. This architecture optimizes communication between cores for efficient parallel processing.

3. Arteris-based NoC: The TI OMAP platform uses an NoC-based interconnect from Arteris to establish seamless connections among its components. This approach enhances the platform's performance and scalability.

4. Research prototypes using NoC: Several innovative research prototypes, such as the TRIPS processor, Smart Memories, and Intel's Teraflops processor, incorporate NoCs to interconnect cores. This results in improved performance and communication efficiency.

5. GALS-based A NoC and multi-synchronous DSPIN NoC: Demonstrator chips featuring these NoC variants serve as system interconnects for the FAUST application, a sophisticated SoC platform with advanced telecommunication capabilities. The implemented quasi-mesh topology efficiently handles real-time communication demands.

6. BONE NoC group chips: The BONE NoC group has successfully fabricated various chips, including a NoC-based parallel processor for visual attention engines and an object recognition processor. Notably, they designed a memory-centric NoC for a homogeneous MPSoC using a hierarchical star topology, enabling dynamic task-to-processor mapping and improved performance over traditional 2D mesh-based CMPs.

In summary, NoCs have become an essential component in diverse applications, offering efficient and scalable interconnect solutions for multi-core processors and heterogeneous systems. These examples showcase the versatility and potential of NoCs in enhancing the performance of modern SoCs [11].

Conclusion

This chapter focused on the concept of deadlock-free Network-on-Chip (NoC) architectures and their significance in ensuring efficient and uninterrupted data flow. We discussed the benefits of implementing deadlock-free mechanisms in NoCs, including improved performance, reliability, and scalability. Additionally, we explored various aspects of NoCs such as scalability, fault tolerance, traffic management,integration capabilities and so many others. We also examined different NoC topologies, buffering techniques, routing algorithms, and the technologies used in NoC designs.

In the next chapter, we will delve into the analysis and evaluation of NoC technologies, exploring simulation and analysis techniques, performance analysis tools, and modeling approaches. These tools and methodologies will provide researchers and designers with valuable insights into the behavior and performance of NoC architectures, facilitating the development of efficient and optimized designs.

Chapter 2

Analysis of NoC-based Network Simulations

2.1 Introduction

The analysis of Network-on-Chip (NoC) [29] architectures have as a primary objective to gain insights into the performance, functionality, and temporal aspects of NoCs. By analyzing NoCs, researchers and designers can optimize their designs, improve efficiency, and meet specific requirements. This chapter focuses on various analysis methods and simulators used for NoC analysis.

This chapter is dedicated to analyzing NoC technologies that are specifically designed for machine learning-based applications in the context of deadlock-free operation. By examining these aspects, we can establish a strong foundation for understanding the unique requirements for this kind of NoC-based machine learning applications.

2.2 NoC Analysis Methods

NoC analysis involves evaluating different aspects of the architecture.Here are three key analysis methods commonly used:

2.2.1 Functional Analysis

This method focuses on the functional correctness of the NoC design. It involves verifying whether the NoC meets its intended purpose and ensures proper data transmission between various components. Functional analysis assesses the routing algorithms [2], flow control mechanisms, and error detection and correction techniques implemented in the NoC.

2.2.2 Non-functional Analysis

Non-functional analysis emphasizes performance-related aspects of the NoC. It includes evaluating metrics such as latency, throughput, and scalability [1]. Non-functional analy-

sis helps identify potential bottlenecks, optimize resource allocation, and enhance overall system performance.

2.2.3 Temporal Analysis

Temporal analysis deals with timing-related aspects of the NoC. It includes analyzing the timing constraints, synchronization mechanisms, and determining if the NoC meets the required timing specifications. Temporal analysis helps ensure that data transmission and synchronization occur within specified time bounds.

2.3 Simulators for NoC Analysis

Simulators for Network on Chip (NoC) are software tools [3] or platforms that are specifically designed to model and simulate the behavior and performance of NoC architectures. These simulators allow designers and researchers to evaluate the design choices, analyze the communication patterns, and assess the overall system performance of a NoC-based design before actual implementation.

2.3.1 Definition of simulators for NoCs

NoC simulators are software tools [3] that provide a virtual environment for modeling and simulating the communication infrastructure, traffic patterns, routing algorithms, and other key aspects of Network on Chip architectures. These simulators enable designers to experiment with different design parameters [38], evaluate the performance, and understand the behavior of NoC-based systems.

2.3.2 NoCs Simulators functionalities

Simulators play a crucial role in the analysis and evaluation of Network-on-Chip (NoC) architectures. They provide a virtual environment for studying the behavior and performance of NoC designs without the need for physical implementation. Simulators offer a range of functionalities that enable researchers, designers, and system architects to explore and assess different aspects of NoCs. In this subsection, we will discuss the typical functionalities offered by simulators for NoCs. These functionalities include network topology generation, traffic generation, routing algorithms, performance evaluation, power and energy analysis, fault injection and analysis, visualization and debugging, design space exploration, and integration with development tools [3]. By leveraging these functionalities, simulators serve as powerful tools for studying, optimizing, and validating NoC designs before actual implementation.

2.3.2.1 Modeling NoC Architecture

They provide the ability to model and configure the NoC architecture, including the number of nodes, their connectivity, routing algorithms, buffer sizes, and other architectural

parameters. This allows designers to create a virtual representation of the actual NoC design.

2.3.2.2 Traffic Generation

Simulators allow users to generate synthetic or real-world traffic patterns to simulate the data communication in the NoC. Different traffic patterns can be evaluated to analyze their impact on performance, latency, and throughput.

2.3.2.3 Performance Evaluation

Simulators measure and analyze various performance metrics, such as latency, throughput, packet loss, and network utilization. This helps designers assess the effectiveness of the NoC design and make informed decisions to optimize system performance.

2.3.2.4 Visualization and Analysis

Simulators often provide visualization capabilities to depict the NoC topology, traffic flows, and routing decisions. They may also offer analysis tools [3] to evaluate the performance results and identify potential bottlenecks or areas for improvement.

2.3.2.5 Experimentation and Optimization

Simulators allow designers to experiment with different design parameters, such as routing algorithms, buffer sizes, network topologies [19], and traffic patterns. By observing the simulation results, designers can optimize the NoC design to achieve desired performance goals.

Simulators for NoCs are valuable tools [3] in the design and evaluation process, enabling designers to gain insights into the behavior and performance characteristics of NoC architectures without the need for physical implementation. They aid in exploring design trade-offs, identifying potential issues, and optimizing the overall system design before committing to hardware implementation.

2.3.3 NoC Simulation Tools and Frameworks

Selecting the right NoC simulator presents a common challenge as available tools [3] often excel in certain aspects while lacking in others. NoC simulators can be categorized into two main groups. The first group consists of general network simulators suitable for NoC simulations, such as NS2, NS3, Omnet++, Wattch, Hotspot, Netsim, Gem5, Graphite, Hornet, Opnet, Fusionsim, Esece [28]-[32]. The second group comprises specific NoC simulators explicitly designed for NoC simulation, including BookSim, HNOCS, WormSim, Ocsim, Vnoc, Matrics, SICOSY, Tpzsimul, Garnet, SUNMAP, Ocintsim, Noxim, Nostrum, Nirgam, Occn, Nocsim, NoCTweak, Atlas, Gpnocsim, Xmulator, NONMAP, ReliableNoC, MapoNoC, Phoenixsim, Access Noxim.

This section aims to describe some essential simulators for NoC-based designs. Figure 4 illustrates the block diagram of a generic NoC simulator, along with its characteristic parameters. The NoC simulator may require the following input parameters:

2.3.3.1 Configuration options

These options define the type of application traffic to be simulated on the NoC tool. The traffic patterns can be either synthetic or derived from embedded application traces. Additionally, configuration options may include selecting a seed value for the simulation, specifying a log file for simulation outputs, setting a warm-up time for the network to stabilize, and choosing the simulation run time [12].

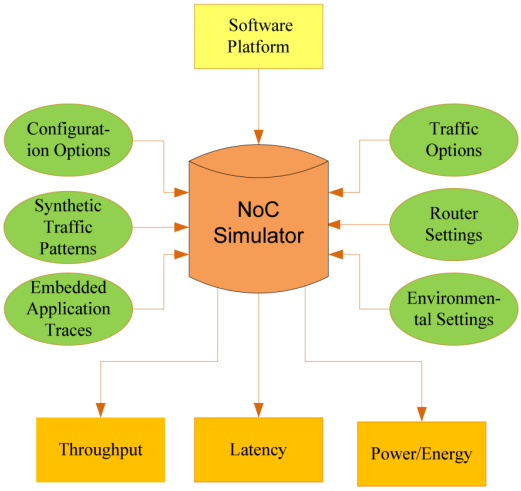


Figure 2.1: Generic NoC simulator. [23]

2.3.3.2 Synthetic options

These options determine the size and type of topology for the traffic, such as 2D mesh, as well as the type of synthetic traffic pattern, including random, transpose, bit-complement, bit-reverse, bit-shuffle, bit-rotate, and hotspot router selection. Hotspots refer to routers that receive data at a higher rate than they can handle, leading to reduced system performance and potential deadlocks. To address this, an intelligent routing algorithm can prevent hotspot formation.

2.3.3.3 Embedded application traces

This option involves using real application task graphs, such as VOPD, multimedia systems, multi-window displays, MPEG4 decoders, and E3S benchmarks, for simulation purposes.

2.3.3.4 Mapping option

The mapping option, which includes near-optimal mapping (NMAP), simulated annealing (SA), and branch and bound (BB), aims to achieve optimized latency, throughput, and

energy consumption.

2.3.3.5 Traffic options

Traffic options encompass the number of flits injected by each core per cycle (flit injection rate), the probability distribution of the period between two injected packets, packet length, and the selection of flits per packet.

2.3.3.6 Router settings

Router settings specify the type of router, such as wormhole router, virtual channel router, shared queues router, bufferless router, and circuit-switched router. It also defines pipeline type, the number of pipeline stages, and buffer depth.

2.3.3.7 Routing options

Routing options define the routing algorithm, such as XY dimension-ordered routing, west-first, north-last, and odd-even (OE) minimal adaptive routing. Additionally, it may include output port selection criteria, such as X dimension first, dimension nearest to the destination first, dimension farthest to the destination first, round-robin among output ports, output port with the highest credit first, switching arbitration policy, and inter-router link length.

2.3.3.8 Technology settings

This includes selecting the CMOS technology process (e.g., 90nm, 65nm, 45nm, 32nm, 22nm), clock frequency, and supply voltages.

2.3.3.9 Measurement options

Measurement options encompass output parameters such as throughput, power, latency, and energy consumption. These performance parameters provide insights into the behavior of the NoC multicore system before physical implementation.

2.3.4 List of various NoC simulation tools

A common challenge of selecting the right NoC simulator is that available tools [3] usually are strong in certain measurements and having deficits in others. NoC simulator can be divided into two broad categories:

(1) General network simulators that can be used for NoC simulations : NS2, NS3, Omnet++, Wattch, Hotspot, Netsim, Gem5, Graphite, Hornet, Opnet, Fusionsim, Esece).

(2) Specific NoC simulators, which are explicitly designed for NoC simulation : BookSim, HNOCS, WormSim, Ocsim, Vnoc, Matrics, SICOSY, Tpzsimul, Garnet, SUNMAP,

Ocintsim, Noxim, Nostrum, Nirgam, Occn, Nocsim, NoCTweak, Atlas, Gpnocsim, Xmulator, NONMAP, ReliableNoC, MapoNoC, Phoenixsim, Access Noxim and ORION) (Table2.1,2.2).

Table 2.1: High level NoC simulator characteristic. [37]

Simulator	Year	Team	Language	Topology	Heterogeneous Support	Availability	Output performance evaluation			
							Throu	latency	Power	Area
ATLAS [3]	2011	PUCRS,Brazil	Java	2D Mesh, Torus	-	+	+	+	-	-
GPNOCSIM [18]	2007	Bangladesh University	Java	2D Mesh Torus, Fat tree	-	+	+	+	-	-
DynMapNoC SIM [37]	2014	University of ORAN	Java	2D Mesh	+	+	+	+	-	-
Booksim [21]	2013	University of Stanford	C++	2D Mesh, Torus, Fat tree	-	+	+	+	-	-
DARSIM [26]	2009	MIT	C++	2D Mesh	-	-	+	+	-	-
GEM5 [28]	2011	Academic and industrial institutions, including AMD, ARM, HP, MIPS, Princeton, MIT, etc.	C++	2D Mesh, Torus, Fat tree	+	+	+	+	-	-
HNOCS [4]	2011	Technion – Israel Institute of Technology	OMNet ++	2D 3D Mesh	+	+	+	+	-	-
McSim-Cycle-accurate-Xbar [36]	2016	University of Montpellier []	C++	Mesh,Torus	-	+	+	+	+	-
McSim-Cycle-accurate-NoC [37]	2016	University of Montpellier	C++	2D Mesh	-	+	+	+	+	-

Table 2.2: Low level NoC simulator characteristic. [37]

Simulator	Year	Team	Language	Topology	Heterogeneous Support	Availability	Output performance evaluation			
							Throu	Latency	Power	Area
NC-G-SIM [40]	2013	Maharana Pratap University of Agriculture and Technology, India	SystemC	2D 3D Mesh, irregular NoC	-	-	+	+	+	-
Nirgam [9]	2007	University of Southampton	SystemC	2D 3D Mesh, Torus	-	+	+	+	+	-
Nostrum []	2012	Royal Institute of Technology of Stockholm	SystemC	2D Mesh, Torus	-	+	+	+	-	-
Noxim [31]	2010	University of Catania	SystemC	2D 3D Mesh	-	+	+	+	+	-
Orion 3.0 [23]	2013	Princeton University	C	All	-	+	-	-	+	+
SunFloor 3D [35]	2010	EPFL (Switzerland)	SystemC	2D 3D Mesh	-	-	+	+	+	-

2.3.5 Comparison of simulated NoCs

In order to conduct a comparative analysis of Network-on-Chip (NoC) tools [3], a standardized configuration setup is established. This setup allows for the evaluation of each simulator based on their performance metrics. Since the available simulators vary in their approach to selecting input/output parameters and employ different simulation models, it becomes challenging to thoroughly examine each simulator in a precise manner. Consequently, a common configuration setup is chosen to ensure that the comparison is carried out using the same input parameters. The analysis of the tools [3] utilizes the parameters presented in Table4.1 .

Table 2.3: Platform description. [23]

Platform	Description	Remarks
topology	2D mesh	most widely used
network size	4×4 (16 nodes)	selected for simulation
workload/benchmark	synthetic	most simulators support
fixed packet length	10 flits	nominal size
flit injection rate	0.1–0.7 flits/cycle/node	range 0–1
traffic type	uniform	predictable results
router type	wormhole	most common
routing algorithm	XY	commonly supported
buffer size	8 flits	selected for simulation
input voltage	1 V	selected for simulation
operating clock frequency	1000 MHz	selected for simulation
warm-up time	20,000 cycles	for accuracy

2.3.6 Synthetic benchmark

To facilitate the comparison of NoC simulators, a synthetic benchmark is chosen since certain simulators like Noxim and CinSim exclusively support synthetic traffic patterns. Additionally, the injected traffic is restricted to 0.7 flits per cycle per node to prevent congestion within the network. When analyzing latency, it is observed that Noxim and Nirgam exhibit similar latency trends with respect to the injected traffic (flit injection rate), as depicted in Figure .

At low traffic levels, NoCTweak and CinSim exhibit a non-linear response. However, when the flit injection rate surpasses 0.5 flits per cycle per node, their response becomes comparable to that of Noxim and Nirgam. This disparity in behavior might be attributed to the additional parameters required for analysis in NoCTweak and CinSim, which are limited in Noxim and Nirgam. Furthermore, the variations in results can be attributed to the absence of an exact and uniform embedded platform across these simulators. It should also be noted that these simulators have different design structures from one another.

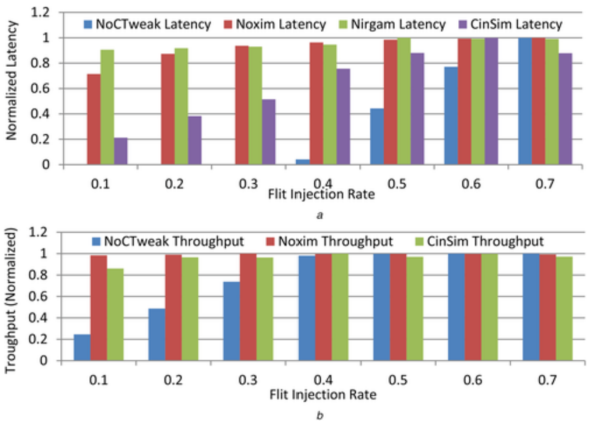


Figure 2.2: Performance parameters of NoC simulators. [23]

(a) Network latency of NoCTweak, Noxim, Nirgam and CINSim, (b) Network throughput of NoCTweak, Noxim and CinSim, (c) Average energy of NoCTwek and Noxim, (d) Network power of NoCTweak and Nirgam [23]

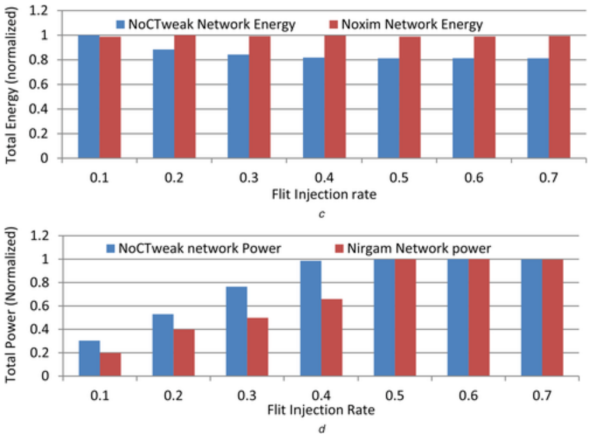


Figure 2.3: Performance parameters of NoC simulators. [23]

(a) Network latency of NoCTweak, Noxim, Nirgam and CINSim, (b) Network throughput of NoCTweak, Noxim and CinSim, (c) Average energy of NoCTwek and Noxim, (d) Network power of NoCTweak and Nirgam [23]

When examining throughput calculations (Figure 2.2b), it is observed that NoCTweak initially shows some variation in results. However, as the injection rate slightly increases, its response closely aligns with that of Noxim and CinSim. Likewise, in the analysis of network energy (Figure 2.2c), NoCTweak and Noxim exhibit minor differences in results when subjected to the same injected traffic. This distinction arises from the utilization of different energy models, with Noxim employing the ORION energy model [[22], while NoCTweak utilizes the CMOS standard cell library model. In terms of network power calculations (Figure 2.3d), once again, the results of NoCTweak and Nirgam are comparable, exhibiting a similar trend at high injection rates.

2.4 NoC-Based related works simulations

There are several simulations tool focusing or working on NoC analysis by the follow we try to give a small quick practical view of field application for this kind of simulation analysis :

2.4.1 omnet++

HaecComm is a simulator that was initially developed for the Diploma Thesis titled "Secure and Efficient Communication for Network-on-Chip under the Consideration of Multiple Paths". The research work was later condensed into a paper named "Lightweight Authenticated Encryption for Network-on-Chip Communications" [15], which was published at GLSVLSI 2019. The simulator is built upon the OMNeT++ framework and implemented in C++.

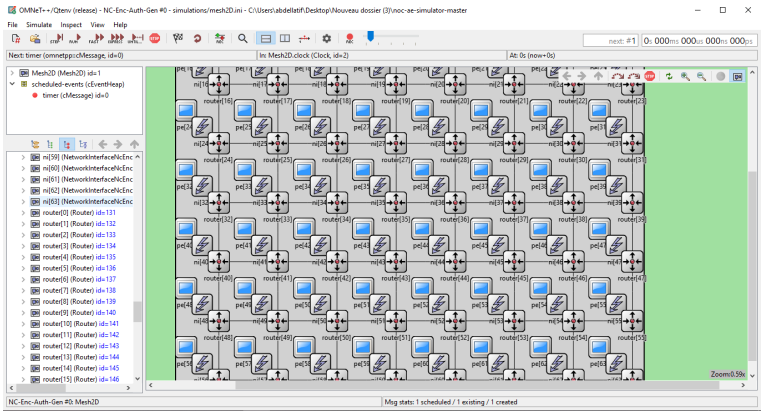


Figure 2.4: NoC simulation with omnet++

2.4.2 Atlas Simulator

Atlas is a popular and versatile simulator for Network-on-Chip (NoC) architectures. It supports various NoC topologies and routing algorithms, allowing researchers to explore different network configurations. With its traffic modeling capabilities, researchers can simulate realistic workloads and assess system behavior under different traffic scenarios. Atlas also provides performance analysis tools and visualization features, enabling the evaluation of metrics like latency, throughput, and packet delivery ratio. Overall, Atlas is a valuable tool for NoC simulation and analysis, offering flexibility and insights for researchers and designers.

2.5 Noxim simulator

Noxim [6] is a widely used functional simulator for NoC analysis. It is an open-source cycle-accurate simulator that provides a flexible [43] and customizable platform for study-

ing NoC architectures. Noxim offers various features and capabilities, including support for different NoC topologies [19], routing algorithms (adaptive and non-adaptive), traffic generation models, and performance metrics. It provides an easy-to-use interface, allowing researchers to configure NoC parameters and analyze simulation results efficiently.

The advantages of using Noxim include its simplicity, scalability [1], and versatility. It offers a lightweight simulation environment suitable for small-scale NoC designs, while also being capable of handling larger and more complex architectures. Noxim allows researchers to explore different design options, analyze performance under various traffic patterns, and evaluate the impact of routing algorithms. Additionally, being an open-source simulator, Noxim benefits from an active community that contributes to its development and provides support to users.

Researchers often choose Noxim for its balance between simplicity and functionality. It serves as a valuable tool for initial NoC analysis and prototyping, allowing researchers to gain insights into the behavior and performance of their designs before moving to more detailed and complex simulators.

2.5.1 Advantages of Noxim Simulator use

The selection of an appropriate simulator plays a crucial role in the analysis of Network-on-Chip (NoC) architectures. Noxim, an open-source functional simulator, offers several advantages that make it a valuable tool for NoC analysis. In this section, we will discuss why Noxim is a helpful choice for analyzing and evaluating NoC designs.

- **Flexibility and Customization:**

  Noxim provides a high degree of flexibility and customization options, allowing researchers and designers to tailor the simulation parameters according to their specific requirements. It supports various NoC topologies [19], routing algorithms, traffic patterns, and injection rates, enabling a wide range of experimentation and analysis. This flexibility makes Noxim suitable for exploring different design choices and optimizing NoC performance [10].

- **Open-Source Nature:**

  Noxim is an open-source simulator, which means its source code is freely available and can be modified and extended by the user community. This open nature promotes collaboration, encourages contributions, and allows researchers to customize and enhance the simulator based on their specific needs. Moreover, the availability of the source code ensures transparency and facilitates the understanding of the underlying simulation models and algorithms.

- **Ease of Use:**

  Noxim is designed to be user-friendly and easy to set up and configure. Its simple command-line interface and intuitive configuration files simplify the simulation setup process, making it accessible even to novice users. The simulator provides clear documentation and examples, aiding users in understanding its functionalities and getting started quickly with their NoC analysis.

- **scalability :**

  Noxim is designed to handle large-scale NoC simulations, making it suitable for analyzing complex architectures and large networks. It efficiently scales with the size of the NoC, allowing researchers to evaluate the performance and behavior of NoCs of varying scales. This scalability [1] is crucial when dealing with real-world applications and high-performance computing systems that require large and intricate NoC designs.

- **Active Community Support:**

  Noxim benefits from an active user community, including researchers, developers, and enthusiasts, who contribute to its development, provide support, and share their experiences. This community support ensures that users can find assistance, guidance, and solutions to their queries and challenges related to Noxim usage. The community also fosters the exchange of ideas, promotes collaborations, and facilitates the adoption of best practices in NoC analysis.

  Noxim offers flexibility, customization, ease of use, scalability [1], and community support, making it a valuable and helpful tool for NoC analysis. Its features and advantages make it suitable for both beginners and experienced researchers, allowing them to gain insights into NoC behavior, optimize designs, and make informed decisions for efficient NoC implementations.

Conclusion

This chapter provides a comprehensive analysis of Network-on-Chip (NoC) architectures, highlighting their importance in understanding performance, functionality, and temporal aspects. Through functional analysis, non-functional analysis, and temporal analysis, valuable insights are gained for optimizing NoC designs. Simulators play a crucial role in NoC analysis such as omnet++ ,ATLAS, and after evaluating several options, Noxim was chosen as the preferred simulator. This decision was made after comparing Noxim with other simulators such as Omnet++ and ATLAS. Noxim stands out for its flexibility, customization options, and support for various NoC topologies, routing algorithms, and

traffic generation models. Comparative studies aid in selecting the most suitable simulator for specific NoC analysis needs. The next chapter will delve into the integration of Machine Learning (ML) and NoC architectures, with a specific focus on ML/NoC routing to optimize designs and address complex challenges.

Chapter 3

Machine Learning of NoC

3.1 Introduction

In this chapter, we will delve into the intersection of machine learning (ML) and Network-on-Chip (NoC) and in a precise view the routing algorithms management. Our objective is to leverage ML techniques to optimize routing decisions within NoC architectures. We aim to develop algorithms or models that can autonomously select the best routing algorithm and determine the optimal number of buffers and flits based on the given data set. This section provides an overview of the topics covered in the subsequent sections.

3.2 Definition of NoC-based Routing machine learning

NoC-based Routing machine learning [20] is a specialized area within the field of Network-on-Chip (NoC) architecture that focuses on the application of machine learning (ML) algorithms and methodologies to optimize the selection of routing paths within a NoC. The primary goal of Machine learning of NoC-based Routing is to dynamically adapt the routing decisions based on the current network conditions and system requirements, aiming to improve overall efficiency, performance, and adaptability of NoC-based systems. its addresses the challenge of efficiently routing data packets within a complex network of interconnected routers in a NoC. Traditional routing algorithms in NoCs are often static and pre-defined, lacking the ability to adapt to changing traffic patterns or varying network conditions. ML/NoC routing seeks to overcome these limitations by leveraging ML techniques to make intelligent and adaptive routing decisions [7].

By utilizing ML algorithms and methodologies, its takes into consideration various factors that impact routing efficiency, such as network congestion, latency, power consumption, and system performance. ML models are trained on historical data or simulated scenarios to learn patterns, relationships, and trends in network behavior. This learned knowledge is then utilized to make real-time routing decisions that optimize the chosen metrics, such as minimizing latency or maximizing throughput.

Routing ML/NoC offers several benefits over traditional routing approaches. Firstly, it enables dynamic adaptation to changing network conditions, allowing for efficient utilization of network resources and improved system performance. Secondly, NoC-based Routing machine learning can handle complex and dynamic traffic patterns, optimizing routing decisions based on real-time data. This adaptability enhances the scalability [1] and flexibility of NoC-based systems, making them suitable for diverse applications and workloads.

Routing ML/NoC [20] involves the application of ML algorithms and methodologies to dynamically select efficient routing paths within a NoC. It aims to optimize routing decisions based on factors such as network congestion, latency, power consumption, and system performance. By leveraging ML, ML/NoC routing enhances the overall efficiency and adaptability of NoC-based systems, enabling them to handle diverse workloads and changing network conditions.

3.3 Steps of ML in NoC-based Embedded Systems

When incorporating Machine Learning (ML) into NoC-based embedded systems [13], the following steps can be followed:

3.3.1 Problem Definition

Clearly define the specific problem that ML will address within the NoC-based embedded system. This involves understanding the system requirements, identifying the challenges or bottlenecks, and determining the goals to be achieved through ML. For example, the problem could be improving traffic management to minimize congestion or optimizing power consumption for energy efficiency.

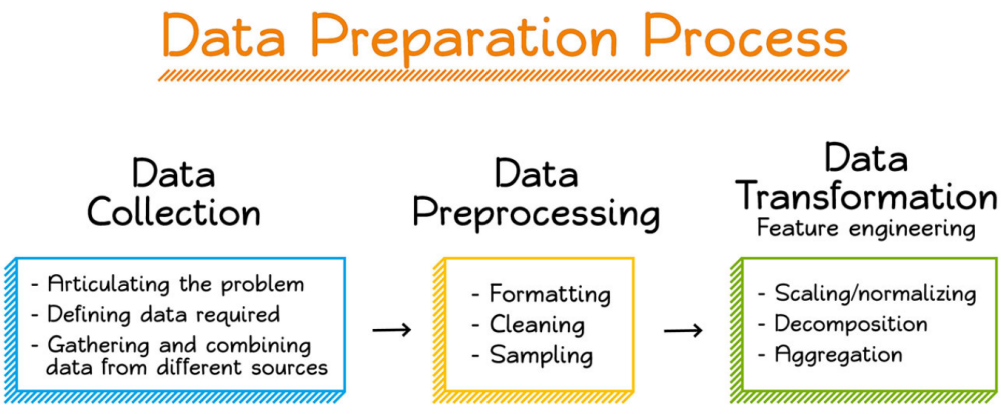


Figure 3.1: data preparation process.

3.3.2 Data Collection

Gather relevant data from the embedded system's NoC. This includes network traffic logs, performance metrics, power consumption data, and any other relevant information. The data should capture a diverse range of scenarios and conditions to ensure its representativeness and effectiveness for ML model training and evaluation (figure3.1).

To create a Machine Learning (ML) dataset for a NoC data collection, we'll follow several steps. Below is a general outline of the process:

- Define goals: We define the goals of the NoC simulation and create the dataset, including specific metrics, features, and labels required for the ML problem.
- Simulation tool selection: We select an appropriate NoC simulation tool based on the research requirements and required simulation capabilities.
- Design NoC topology: We define the NoC topology to be simulated, and specify the number of nodes, routers, links, and their interconnection scheme.
- Generate Traffic Patterns: We generate different traffic patterns that represent the workload that the NoC will handle, such as random, uniform, or application-specific traffic.
- Configure simulation parameters: We set up simulation parameters, including number of sessions, router configurations, traffic injection rate, routing algorithms, buffer sizes and other related settings.
- Run NoC Simulations: We perform NoC simulations for each traffic pattern and configuration using the simulation tool of choice.
- Simulation data collection: We record the performance metrics identified in step 1 while running the simulation. This data includes latency, throughput, packet loss, congestion, power consumption, etc.

3.3.3 Data Preprocessing

Process the collected data to ensure its quality and suitability for ML algorithms. This involves several tasks:

- Removing outliers: Identify and handle any data points that are significantly different from the majority of the data, as they can skew the analysis results.
- Handling missing data: Decide on an appropriate strategy for dealing with missing data, such as imputing missing values or removing the corresponding data points.
- Normalizing features: Normalize the data to bring all features to a similar scale. This helps prevent certain features from dominating the model training process.
- Handling categorical variables: Encode categorical variables into numerical values so that they can be effectively used by ML algorithms. This may involve techniques such as one-hot encoding, label encoding, or ordinal encoding.

Data preprocessing ensures that the data is clean, consistent, and suitable for ML model training (figure3.1).

3.3.4 Feature Selection/Extraction

Identify and select the most relevant features from the preprocessed data. This step aims to reduce the dimensionality of the data and focus on the key aspects that are most relevant to the problem at hand. Feature selection techniques include statistical analysis, domain knowledge, correlation analysis, or automated feature selection algorithms. Alternatively, feature extraction techniques like Principal Component Analysis (PCA) or t-SNE can be applied to derive a lower-dimensional representation of the data.

3.3.5 ML Model Selection

Choose the appropriate ML model(s) based on the defined problem and available data. Consider the characteristics of the problem, such as whether it is a classification, regression, clustering, or anomaly detection task. Additionally, take into account the size of the dataset, the complexity of the problem, and the available computational resources. Common ML algorithms include decision trees, random forests, support vector machines, neural networks, and k-means clustering, among others.

3.3.6 Training

Train the selected ML model(s) using the preprocessed data. Split the data into training and validation sets, typically using techniques like k-fold cross-validation or a train-test split. Feed the training data into the ML algorithm, and iteratively adjust the model's parameters to optimize its performance on the given task. This may involve techniques such as gradient descent or backpropagation for neural networks.

3.3.7 Evaluation

Assess the trained ML model's performance using appropriate evaluation metrics. The choice of metrics depends on the specific problem and ML technique used. For classification tasks, metrics like accuracy, precision, recall, and F1-score can be used. For regression tasks, metrics such as mean squared error (MSE) or mean absolute error (MAE) are common. Validate the model's effectiveness in addressing the defined problem within the NoC-based embedded system using the validation set or through additional testing [1].

3.3.8 Deployment and Integration

Once the ML model demonstrates satisfactory performance, integrate it into the NoC-based embedded system. This integration may involve implementing

the ML model in the hardware or software components of the system. Consider the compatibility and scalability of the ML model with the overall system architecture. This

step may require collaboration with system designers, hardware engineers, and software developers to ensure a seamless integration of the ML model into the existing infrastructure.

3.3.9 Monitoring and Fine-tuning

Continuously monitor the deployed ML model's performance in the embedded system. Collect additional data if necessary to further improve the model's accuracy and adaptability. Fine-tune the model periodically to account for changing conditions, evolving system requirements, or new data patterns. This step involves retraining the model with updated data and potentially adjusting the model's parameters to optimize its performance.

3.3.10 Data Visualization

Data visualization is an essential aspect of ML in NoC-based embedded systems. It involves representing the collected data, model outputs, and system performance metrics visually to gain insights and facilitate understanding. Various techniques can be employed, such as scatter plots, line charts, bar charts, histograms, heatmaps, box plots, and scatter matrices. Data visualization aids in identifying patterns, trends, correlations, and anomalies within the data, providing valuable information for decision-making and system optimization.

3.3.11 Model Interpretability and Explainability

In some cases, it is crucial to understand and interpret the ML model's internal workings and decisions. Model interpretability and explainability techniques help uncover the factors or features influencing the model's predictions. This can be achieved through methods like feature importance analysis, partial dependence plots, or model-agnostic approaches like LIME (Local Interpretable Model-Agnostic Explanations). Understanding the model's decision-making process enhances trust, enables debugging, and assists in identifying any biases or limitations.

By following these detailed steps, NoC-based embedded systems can effectively leverage ML techniques to address various challenges, optimize performance, and make data-driven decisions.

3.4 ML/NoC use in Embedded Systems

In this section, we delve into the application of machine learning (ML) techniques in embedded systems, with a specific emphasis on Network-on-Chip (NoC) architectures. We explore the intersection of ML and embedded systems, highlighting the benefits and challenges of integrating ML into NoC-based designs [20].

Embedded systems are specialized computing systems designed to perform specific functions within constrained environments. They are commonly found in a wide range of applications, including IoT devices, automotive systems, consumer electronics, and industrial automation. These systems often have limited computational resources, power constraints, and real-time requirements.

ML has emerged as a powerful tool for improving the capabilities of embedded systems. By leveraging ML algorithms, embedded systems can adapt, learn, and make intelligent decisions based on data inputs. ML can enhance various aspects of embedded systems, such as perception, control, optimization, and decision-making.

In the context of NoC architectures, ML plays a crucial role in optimizing and enhancing the communication capabilities of the system. NoCs provide the backbone for data exchange and communication between different components in embedded systems, including processors, memory, and peripherals. ML techniques can be applied to improve the routing decisions, congestion management, load balancing, and fault tolerance mechanisms within the NoC.

ML in embedded systems with an emphasis on NoC brings several advantages. It enables intelligent decision-making in real-time, allowing the system to adapt to changing conditions and optimize performance. ML can improve the overall system efficiency by dynamically adjusting the routing algorithms based on network traffic patterns and system requirements. Additionally, ML techniques can enhance fault detection and recovery mechanisms, improving the reliability and fault tolerance of the NoC.

However, integrating ML into embedded systems, especially within the resource-constrained environment of NoC architectures, poses challenges. ML algorithms can be computationally intensive and require significant memory resources, which may conflict with the limitations of embedded systems. Efficient implementation and optimization techniques are necessary to meet real-time requirements and minimize resource utilization.

Furthermore, ML algorithms require training data to learn patterns and make accurate predictions. Gathering and labeling representative data sets for training ML models in embedded systems can be a complex task. Additionally, ensuring the security and privacy of data in embedded ML systems is crucial, as sensitive information may be processed within the system.

the application of ML in embedded systems, with a focus on NoC architectures, opens up new possibilities for improving system performance, adaptability, and fault tolerance. By leveraging ML algorithms, embedded systems can make intelligent decisions and optimize the communication capabilities within the NoC. However, careful consideration must be given to resource constraints, real-time requirements, training data availability, and data security to effectively integrate ML into embedded systems.

3.5 ML-assisted Secured Solutions for NoC Architectures

Machine Learning (ML) can serve as a powerful tool in various aspects of Network-on-Chip (NoC) design and optimization. Here are a few examples:

3.5.1 Traffic Prediction and Adaptive Routing

ML can be employed to predict network traffic patterns within the NoC. By training ML models on historical traffic data , the system can learn to anticipate future traffic demands. These models can then be used to dynamically adjust the routing decisions in real-time based on predicted congestion areas. By avoiding congested paths and optimizing resource allocation, ML-based adaptive routing improves overall network performance and reduces latency.

In their study [25], Lin et al. proposed a novel deep reinforcement learning (DRL) framework for routerless networks-on-chip (NoC) and evaluated its performance using various metrics. By leveraging a deep neural network and parallel threads, the framework efficiently explored the extensive routerless NoC design space using a Monte Carlo search tree. The experimental results demonstrated significant improvements compared to conventional mesh-based approaches. Specifically, the (DRL) routerless design achieved a remarkable 3.25x increase in throughput, 1.6x reduction in packet latency, and 5x reduction in power consumption compared to traditional mesh architectures. Furthermore, when compared to state-of-the-art routerless NoC solutions, the DRL design exhibited a 1.47x increase in throughput, 1.18x reduction in packet latency, 1.14x reduction in average hop count, and 6.3% lower power consumption. These findings highlight the effectiveness of the deep reinforcement framework in optimizing routerless NoC designs, making it a promising approach for enhancing performance and energy efficiency in NoC systems (Lin et al., 2020).

3.5.2 Fault Detection and Recovery

ML algorithms can be applied to detect and predict faults or anomalies in the NoC. By analyzing real-time data from the network, ML models can learn normal behavior and identify deviations that indicate potential faults, such as link failures or excessive latency. This enables proactive fault detection and triggers appropriate recovery mechanisms, such as rerouting traffic or activating redundant paths, to enhance the network's resilience and minimize disruptions.

3.5.3 Power Optimization

ML techniques can be leveraged to optimize power consumption in NoC designs. By considering various parameters such as traffic patterns, application requirements, and network conditions, ML models can learn to dynamically adjust power levels or activate/deactivate components within the NoC. This adaptive power management approach helps conserve energy and extend the overall system's battery life, which is particularly beneficial for mobile or low-power embedded systems.

In their study [41], Wang et al. (2019) focused on addressing timing errors in Network-on-Chip (NoC) architectures. They highlighted that traditional reactive techniques for handling timing errors result in excessive power consumption and degraded performance.

To overcome these challenges and optimize energy efficiency and performance, the researchers proposed a proactive fault-tolerant mechanism utilizing reinforcement learning (RL).

The evaluation of the proposed mechanism demonstrated significant improvements. On average, the study reported a 55% reduction in end-to-end packet latency, a 64% improvement in energy efficiency, and a 48% decrease in retransmission caused by faults compared to reactive error correction techniques. These findings highlight the effectiveness of the proactive fault-tolerant mechanism using RL in optimizing energy efficiency, performance, and fault-tolerance in NoC architectures (Wang et al., 2019).

3.5.4 Quality of Service QoS Management

ML algorithms can assist in improving QoS in NoC designs by dynamically allocating network resources based on application requirements and priorities. ML models can learn to optimize resource allocation decisions, such as buffer sizes, routing policies, and arbitration mechanisms, to ensure that critical applications receive the necessary bandwidth and latency guarantees while efficiently handling non-critical traffic. This enhances the overall performance and user experience of the NoC-based system.

In their study [**?**], Khan et al. (2022) focused on predicting Quality of Service (QoS) violations in cloud computing environments. They emphasized the importance of reliable QoS and meeting Service Level Agreements (SLAs) for cloud service providers. Factors such as response time, accessibility, availability, and speed are critical for ensuring service reliability and meeting user expectations.

To address this challenge, the researchers explored the use of parallel mutant-Particle Swarm Optimization (PSO) for QoS violation detection and prediction. They compared the performance of Simple-PSO and Parallel Mutant-PSO techniques. The simulation results demonstrated that the proposed Parallel Mutant-PSO approach achieved a high accuracy rate of 94% in predicting cloud QoS violations. Additionally, it exhibited computational efficiency compared to conventional PSO methods.

3.5.5 Performance Modeling and Optimization

ML can be utilized to develop models that capture the complex relationships between various design parameters and performance metrics in NoC architectures. These models can then be used to optimize the design space, guiding decisions such as topology selection, routing algorithms, or buffer sizing to achieve desired performance objectives. ML-based performance modeling and optimization enable designers to explore a wider range of possibilities and make informed decisions for efficient NoC design.

In their study [32], Qian et al. (2013) proposed SVR-NoC, a learning-based support vector regression (SVR) model for evaluating the latency performance of Network-on-Chip (NoC) systems. Unlike traditional analytical models that rely on queuing theory, SVR-NoC utilizes machine learning techniques to analyze NoC latency based on training data.

This study introduces a novel approach to NoC latency analysis by leveraging machine learning techniques. SVR-NoC provides an efficient and accurate solution for evaluating the performance of NoC systems, contributing to the advancement of design and optimization in the field of Network-on-Chip.

These examples demonstrate how ML can be employed as a valuable tool in NoC-based systems, offering enhanced fault detection, power optimization, QoS management, and performance optimization. The application of ML techniques in NoC design empowers engineers to create more efficient and robust systems, leading to improved performance and reliability in various embedded applications.

3.6 ML for NoC Routing algorithms

we explore the application of machine learning (ML) techniques specifically in the domain of routing within Network-on-Chip (NoC) architectures. We discuss the design and implementation aspects of ML-based routing approaches and highlight the role of NoCs as a tool for ML experimentation and evaluation.

Routing is a critical component of NoC architectures as it determines how data packets are forwarded from source to destination within the network. Traditional routing algorithms, such as deterministic and adaptive routing, have been widely used in NoCs. However, ML-based routing approaches aim to leverage the power of ML algorithms to make routing decisions dynamically and adaptively based on various factors, including network conditions, traffic patterns, and performance requirements.

Designing ML-based routing approaches involves several key steps. Firstly, a suitable ML model or algorithm needs to be selected based on the specific routing objectives and system requirements. This can include supervised learning algorithms, reinforcement learning techniques, or even deep learning models, depending on the complexity of the routing problem and available data.

Once the ML model is chosen, training data needs to be collected and preprocessed. This data may include historical network traffic patterns, performance metrics, and other relevant features. The training data is used to train the ML model, enabling it to learn the underlying patterns and make accurate routing decisions.

Implementation of ML-based routing requires integrating the trained model into the NoC architecture. This can be achieved by utilizing the flexibility and programmability of NoCs, allowing the ML model to be deployed on the routers or intermediate nodes. The ML model can then dynamically adapt routing decisions based on real-time network conditions and optimize performance.

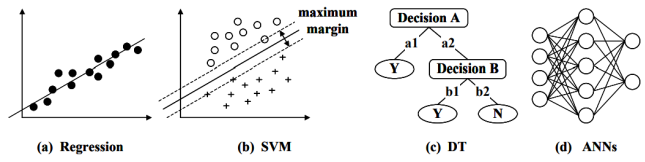
One significant advantage of using NoCs as a tool for ML in routing is the availability of rich simulation environments and evaluation frameworks. NoC simulators, such as Noxim, provide the capability to simulate and evaluate different ML-based routing approaches. These simulators offer insights into the performance, latency, throughput, and energy consumption of the ML-based routing algorithms, enabling researchers to fine-tune and optimize their designs.

By employing ML for routing in NoCs, several benefits can be achieved. ML-based routing approaches have the potential to adapt to dynamic network conditions, leading to improved network performance, reduced congestion, and optimized resource utilization. They can also enhance fault tolerance by dynamically rerouting packets in the presence of failures or congestion hotspots.

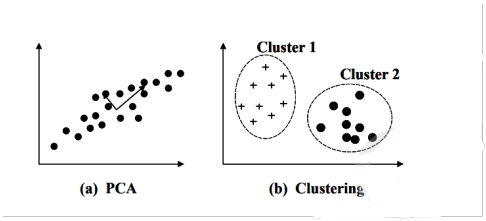
In conclusion, ML-based routing in NoC architectures offers a promising approach to address the challenges and limitations of traditional routing algorithms. By leveraging ML techniques, routing decisions can be made dynamically, adaptively, and intelligently based on network conditions and performance requirements. NoCs serve as valuable tools [3] [23] for designing, implementing, and evaluating ML-based routing approaches, providing simulation environments and performance evaluation metrics.

3.6.1 Models of Machine Learning in NoC

When it comes to machine learning (ML) models applied to Network-on-Chip (NoC) [42]. architectures, several approaches have been explored. Here are some models commonly used in ML with NoC:



(a) Examples of supervised learning include (a) Regression,(b) SVM,(c) DT, and (d) ANNs [42]



(b) Examples of unsupervised learning include (a) PCA and (b) Clustering [42]

Figure 3.2: Models of Machine Learning

Convolutional Neural Networks (CNNs

CNNs [42] are widely used in ML for NoC designs. They can be applied to tasks such as traffic prediction, congestion detection, or fault identification. CNNs excel at extracting spatial patterns from input data, making them suitable for analyzing NoC topologies [19], traffic patterns, or performance metrics.

Recurrent Neural Networks (RNNs)

RNNs [42] are effective for modeling temporal dependencies in NoC traffic. They are commonly used for tasks like traffic prediction, routing decision-

making, or workload balancing. RNNs can capture the sequential nature of traffic patterns and make predictions based on past and current states.

Long Short-Term Memory (LSTM)

LSTM [42] is a type of RNN that is particularly adept at handling long-term dependencies. NoC traffic patterns can exhibit long-term dependencies, and LSTM models can effectively capture and utilize such dependencies for tasks like traffic prediction, congestion control, or adaptive routing.

Reinforcement Learning (RL)

RL algorithms [42], such as Q-learning or Deep Q-Networks (DQNs) [42], have been applied to NoC routing and congestion control. RL agents can learn optimal routing policies through trial and error, maximizing long-term rewards based on feedback from the NoC environment. RL models can adapt to dynamic traffic conditions and optimize routing decisions accordingly.

Genetic Algorithms (GA)

GA is an evolutionary optimization technique that has been applied to NoC design and optimization. GA-based models can explore the design space, evolve NoC architectures, and optimize performance metrics such as latency, throughput, or power consumption. GA models use techniques like selection, crossover, and mutation to iteratively improve NoC designs.

Support Vector Machines (SVM)

SVMs [42] have been used in NoC research for tasks such as fault detection or classification. SVM models can learn decision boundaries to distinguish between different NoC states or conditions based on labeled training data. They can help identify anomalies or detect faults in the NoC system.

Ensemble Learning

Ensemble learning combines multiple ML models to make collective predictions or decisions. Ensembles can incorporate different ML algorithms or

variations of the same algorithm to enhance accuracy and robustness. Ensemble models can be used for tasks like traffic prediction, fault detection, or routing optimization in NoC systems.

These are some of the ML models commonly used in the context of NoC architectures. The choice of model depends on the specific task, available data, computational resources, and performance objectives. Researchers continue to explore and develop new ML models and techniques to address various challenges and optimize NoC designs and operations.

3.7 Evaluation a machine learning model

When evaluating a ML model, it is important to select appropriate evaluation metrics and techniques to assess its performance in making predictions. The choice of evaluation methods depends on the problem type and specific project goals. Here are some common approaches:

3.7.1 Evaluation and performance assessment

Evaluation and performance assessment are essential steps in machine learning to gauge the effectiveness of a model. Two common techniques for evaluation and performance assessment are train-test split and cross-validation.

Train-Test Split

Divide the dataset into a training set and a separate test set. Train the model on the training data and evaluate its performance on the test set. This helps assess the model's ability to generalize to unseen data.

Cross-Validation

Perform k-fold cross-validation, dividing the dataset into k subsets (folds). Train the model k times, each time using k-1 folds for training and one fold for validation. Average the performance across all folds to obtain a more reliable estimate of the model's performance.

3.7.2 Evaluation Metrics for Classification Problems

Evaluation metrics play a crucial role in assessing the performance and effectiveness of classification models. These metrics provide quantitative measures to evaluate the accuracy, reliability, and predictive power of a classifier.

Accuracy Measures the proportion of correctly classified instances.

Precision

Indicates the model's ability to correctly identify positive instances.

Recall

Measures the model's ability to find all positive instances.

F1 Score

The harmonic mean of precision and recall, providing a balanced measure.

Area Under the ROC Curve (AUC-ROC)

Evaluates the model's ability to discriminate between positive and negative instances.

3.7.3 Evaluation Metrics for Regression Problems

When evaluating regression problems, there are several commonly used evaluation metrics that provide insights into the performance and accuracy of regression models. Here are three widely used metrics:

Mean Absolute Error (MAE) MAE measures the average absolute difference between the predicted values and the true values. It provides a straightforward interpretation of the average magnitude of the errors without considering their direction. A lower MAE indicates better model performance, with a value of 0 indicating a perfect match between the predicted and true values.

Mean Squared Error (MSE) MSE calculates the average of the squared differences between the predicted values and the true values. It emphasizes larger errors due to the squared term, making it more sensitive to outliers. Like MAE, a lower MSE indicates better model performance, with a value of 0 representing a perfect match between the predicted and

true values.

Root Mean Squared Error (RMSE) RMSE is the square root of the MSE and provides a measure of the average magnitude of the errors in the same units as the target variable. It is often preferred over MSE as it provides a more interpretable and easily understandable metric. Like MAE and MSE, a lower RMSE indicates better model performance.

R-squared (R^2) R-squared measures the proportion of the variance in the dependent variable that can be explained by the independent variables in a regression model. It indicates the goodness of fit of the model. R-squared ranges from 0 to 1, with higher values indicating a better fit. However, it does not provide information about the accuracy or precision of the predictions and should be used in conjunction with other metrics.

These evaluation metrics help assess the performance and accuracy of regression models. Depending on the specific problem and requirements, different metrics may be more appropriate. It is recommended to consider multiple metrics to gain a comprehensive understanding of the model's performance.

3.7.4 Evaluation Metrics for Clustering Problems

there are several metrics that can be used to assess the quality and performance of clustering algorithms. Here are three commonly used evaluation metrics:

Silhouette Coefficient The Silhouette Coefficient quantifies how well each sample fits within its assigned cluster compared to other clusters. It considers the average distance between the sample and other samples in the same cluster (intra-cluster distance) as well as the average distance between the sample and samples in the nearest neighboring cluster (inter-cluster distance). The Silhouette Coefficient ranges from -1 to 1, with values closer to 1 indicating well-separated and distinct clusters.

Davies-Bouldin Index The Davies-Bouldin Index evaluates the similarity between clusters by assessing both the scatter within each cluster and the separation between different clusters. It takes into account the

average distance between each sample in a cluster and the centroid of that cluster, as well as the distances between cluster centroids. A lower Davies-Bouldin Index value indicates better clustering, with well-separated and distinct clusters.

Adjusted Rand Index (ARI) The Adjusted Rand Index adjusts the Rand Index for chance agreement, considering the expected agreement between clusters based on a random assignment of samples. It measures the similarity between the true class labels and the clustering results. The ARI ranges from -1 to 1, with values closer to 1 indicating better clustering performance.

These evaluation metrics provide insights into various aspects of clustering quality, such as cluster separation, compactness, and agreement with ground truth labels. It is crucial to select the most suitable metric based on the data characteristics and specific clustering objectives. Additionally, considering multiple metrics can offer a more comprehensive evaluation of clustering results.

3.7.5 Other Evaluation Techniques

In addition to the previously mentioned evaluation metrics for clustering problems, there are other techniques that can provide valuable insights into the performance and effectiveness of clustering algorithms. These techniques include: **Confusion Matrix** The confusion matrix is commonly used in supervised learning tasks, but it can also be adapted for clustering evaluation by comparing the cluster assignments with the ground truth labels. It provides a detailed breakdown of true positive, true negative, false positive, and false negative classifications, allowing for a more in-depth analysis of the clustering results.

Learning Curves Learning curves illustrate the performance of a clustering algorithm as the size of the dataset increases. By plotting the training and validation scores against the number of training instances, one can analyze the algorithm's ability to generalize and identify potential issues such as overfitting or underfitting.

Receiver Operating Characteristic (ROC) Curve The ROC curve is commonly used in binary classification tasks, but it can also be applied to clustering problems. It measures the performance of the clustering algorithm by plotting the true positive rate against the false positive rate at various threshold values. The area under the ROC curve (AUC-ROC) provides a measure of the algorithm's overall performance.

These additional evaluation techniques can complement the existing metrics and provide further insights into the performance and characteristics of clustering algorithms. It is important to choose the most appropriate techniques based on the specific requirements and objectives of the clustering problem at hand.

Conclusion

We emphasize the significance of integrating ML techniques into NoC routing to optimize routing decisions based on various parameters. By leveraging ML algorithms, we can enhance the performance and adaptability of NoC architectures. The chapter sets the stage for the subsequent chapter, which focuses on utilizing ML to achieve fault-free NoCs. The exploration of ML-based fault mitigation techniques in NoC architectures will further enhance the reliability and efficiency of these systems.

In the upcoming chapter, "Toward Fault-Free NoCs with ML Techniques," we will embark on a one way to enhance the fault tolerance of NoC architectures. We will explore the remarkable potential of ML techniques in detecting and mitigating faults within NoCs, ultimately striving for fault-free systems. By integrating ML algorithms into the routing and fault detection mechanisms of NoCs, we aim to significantly improve their reliability, efficiency, and overall performance. This chapter will delve into the methodologies and approaches employed to leverage ML techniques in achieving fault-free NoCs, shedding light on the transformative power of intelligent systems in the realm of network-on-chip design.

Chapter 4

Toward a faulty free NoCs with ML

4.1 Introduction

we delve into the application of machine learning (ML) techniques to enhance the fault tolerance of Network-on-Chip (NoC) architectures. Our goal is to develop an intelligent system that can autonomously detect and mitigate faults in NoCs, leading to more reliable and fault-free communication. By harnessing the power of ML algorithms, we aim to create a robust framework that can identify non-deterministic phenomena and proactively address them before they impact the performance and reliability of the NoC. We will explore various ML-based approaches, including anomaly detection, predictive maintenance, and fault diagnosis, to enable early fault detection and prevention. Furthermore, we will investigate how ML algorithms can be integrated into the NoC design process to improve its resilience against faults and ensure seamless communication between components. Through this work, we aim to pave the way towards the development of fault-free NoCs that can meet the stringent requirements of modern computing systems, enabling them to operate with utmost reliability and efficiency.

4.2 The process of ML/NoC

In this section, we will discuss the process of applying Machine Learning (ML) techniques in the context of Network-on-Chip (NoC) architectures. Throughout our research, we have explored various aspects of ML/NoC, and now we will summarize our process as follows:

4.2.1 Problem Identification and Formulation

The increasing popularity of Multicore Integrated Circuits, such as Network-on-Chips (NoCs), is driven by their high performance and ability to reduce latency and power consumption through digital packet-based communication. The architecture of NoCs holds promise for implementing machine learning algorithms, where routers can be trained using ML techniques to efficiently transfer packets between source and destination routers. However, to design an effective ML/NoC architecture, various parameters need to be considered, including feature concerns, topologies and routing, Quality of Service (QoS), protocols, fault tolerance, buffering concerns, and real-time requirements.

4.2.2 Analysis Methods

In the second part we analyzed the NoC structure and its requirements using the Noxim simulator. We have identified key parameters, performance measures and relevant factors that need to be taken into consideration during data collection.

4.2.3 Data Collection and Preparation

The next step is to collect the data needed to train and evaluate our ML model. This involves collecting data from NoC by running simulations multiple times with changing parameters. We then process the data with Parsing algorithm and then clean it up and convert it into a format suitable for ML parsing.

4.2.4 Feature Extraction and Selection

In this step, we extract relevant features from the pre-processed data, such as Buffer,Routing-Algorithm, Flit-Size, Power(W), Execution-Time(s) . These features represent the most informative NoC properties of the ML models.

4.2.5 Model Training and Validation

With our prepared data and selected features, we train our ML models. The training process involves feeding the data into the ML model, optimizing model parameters, and iteratively improving the model's performance. We then validate our trained models using separate validation datasets to ensure their generalization capability and robustness.

4.2.6 Model Evaluation and Performance Metrics

Once our models are trained and validated, we evaluate their performance by testing it by providing another example of a set of parameter values and seeing the result of the test if it is aware of falty or not, this evaluation provides us with insights about the effectiveness of the proposed model ML in knowing NoC faulty.

4.2.7 Results of Analysis

In this step we analyze and interpret the results obtained from our ML glsNoC process to gain a deeper understanding of its impact. This will involve examining the effects of ML techniques on NoC performance, fault tolerance, or other targeted areas. Through this analysis, I can identify strengths, weaknesses, and potential areas for improvement in future iterations.

4.2.8 Discussion and Conclusion

Upon completing our process, I will thoroughly analyze and interpret the results obtained from our ML/NoC endeavors to gain valuable insights

into its impact. The effects of various ML techniques on NoC performance, fault tolerance, and other targeted areas will be carefully examined. Throughout this analysis, I will diligently identify the strengths and weaknesses of the approach, enabling us to discern potential areas for future improvements.

By following this process, we can effectively leverage ML techniques to address various challenges and opportunities within NoC architectures. Our process provides a structured approach for problem-solving, data analysis, model development, and result interpretation in the ML/NoC domain.

4.3 On Fault tolerance NoC Analysis

we delve into the critical aspect of fault tolerance in Network-on-Chip (NoC) systems. Noxim, a widely used NoC simulator, lacks built-in support for fault tolerance. Recognizing the significance of fault tolerance in ensuring the reliability and performance of NoC architectures, we have undertaken the task of introducing a new strategy to simulate fault NoCs within the Noxim framework.

4.3.1 Fault Tolerance Analysis in Noxim

To Enabling Fault Tolerance Analysis in NoC in Noxim By following these steps and using Noxim as the simulator, we can perform simulations and analyze the performance of Fault Tolerance NoC configurations. Noxim provides a flexible and customizable platform for NoC research, allowing we to evaluate various performance metrics and draw insights from the simulation results.

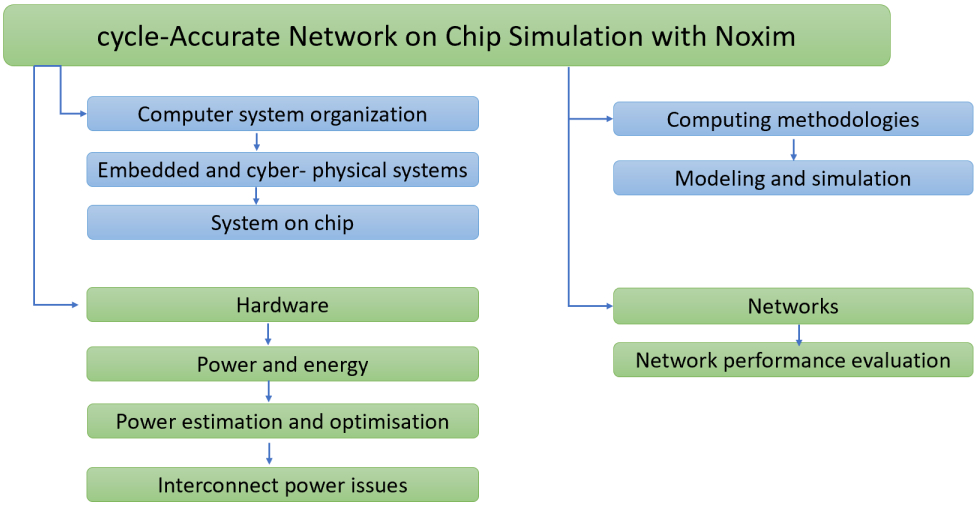


Figure 4.1: cycle-Accurate Network on Chip Simulation with Noxim.

To use Noxim as a tool to perform the simulations, we followed the steps outlined below to perform the analysis for the fault toleranceNoC case:

4.3.2 Architect simulator Noxim files

One of the key components that make up the Noxim simulator is its architecture, which consists of several important files and modules.

The main file of the Noxim simulator is ”main,” which serves as the entry point of the simulation. It orchestrates the overall simulation process and coordinates the execution of various components. Within the ”main.cpp” file, several other important files are included, such as ”node.cpp” and ”globalstatus.h”. These files contribute to the functionality and behavior of the simulator.

The ”main.h” file, which is part of the Noxim architecture, plays a crucial role in defining the simulator’s structure and organization. It acts as a central hub that connects and integrates different components of the simulator. Within ”main.h,” various header files are included, such as ”node.h”, ”globalstatus.h”, ”datatructure.h”, ”globalparameters.h”, and ”configurationmanager.h”. These header files define the data structures, global parameters, and configuration settings used throughout the simulation.(figure4.2)

To provide a clearer understanding of the Noxim architecture, a dia-

gram can be used to illustrate the relationships between these different files and modules. The diagram showcases the hierarchical structure of the simulator, with ”main.h” at the top, connecting to various header files and modules, which, in turn, interact with other components and functionalities:

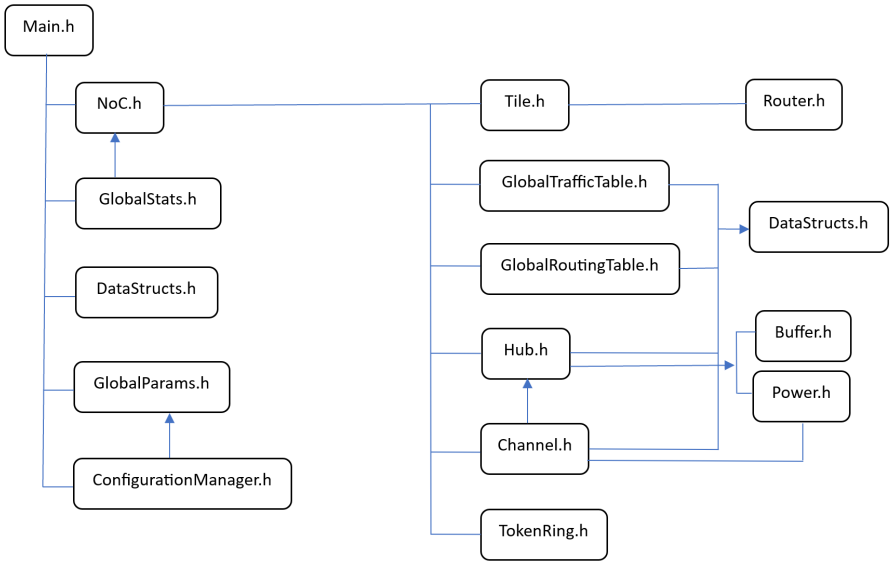


Figure 4.2: Hierarchy of the Noxim file.

4.4 Setting faulty NoC-based Nodes in Noxim

To enhance the fault tolerance capabilities of Noxim, we have developed a new strategies to address fault injection in the NoC.

4.4.1 Injection strategy

We begin by modifying the ”fault-injection.cpp” file, enabling us to inject faults into fault links, fault routers, and fault nodes within the NoC.(figure4.3)

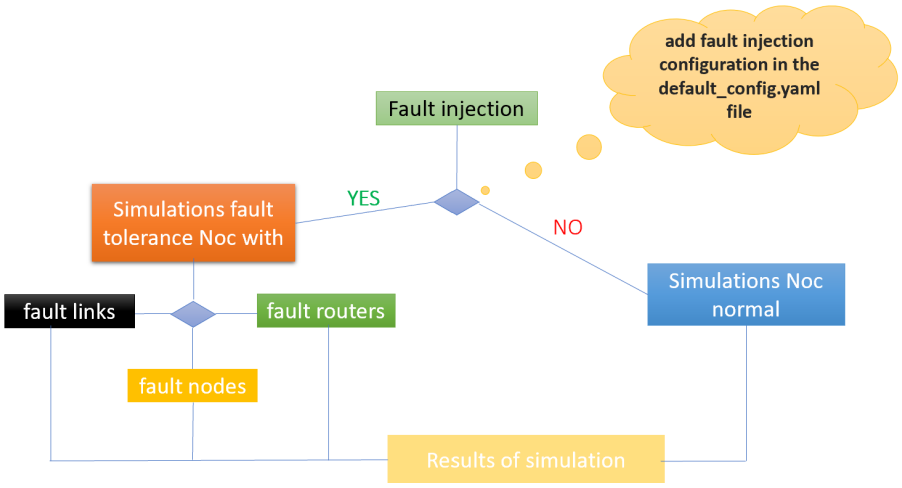


Figure 4.3: Hierarchy of add Fault Tolerance.

Incorporating this new strategy involves adding a ”fault-injection” function in the ”NoC.cpp” file. This function allows us to inject faults into specific components of the NoC, simulating various fault scenarios and evaluating the system's resilience.

Once we made the necessary changes, we remove the compiled object files, executables, and other build artifacts from the Noxim source code directory.we Run the make clean command to clean the Noxim build. After running ”make clean”, you will have a clean build environment and can proceed with building the Noxim simulator again using the ”make” command.

We took here examples for injection strategy as follwing :

Injected fault Node in router 2 with fault value 2

Injected fault Link in link 1 with fault value 5

Injected fault Router in router 2 with fault value 4

The result appears :

```
oem@abdellatif:~/Desktop/noxim-master/bin$ ./noxim -config ../config_examples/default_config.yaml
Injected NODE_FAILURE in router 2 with fault value 2
Injected LINK_FAILURE in link 1 with fault value 5
Injected ROUTER_FAILURE in router 2 with fault value 4
oem@abdellatif:~/Desktop/noxim-master/bin$
```

Figure 4.4: The result of the new strategy: Injections.

4.4.2 Saturation Strategy

In addition to the fault injection and fault recovery strategies, we have developed another approach to enhance fault tolerance in Noxim. This strategy focuses on addressing the issue of saturation in the NoC.

Saturation occurs when the network becomes overwhelmed with high traffic loads, leading to congestion and degradation in performance. To tackle this challenge, we introduce measures to mitigate saturation and maintain the efficiency and reliability of communication in the NoC.

One aspect of our saturation strategy is to dynamically adjust the injection rate of packets into the network. By monitoring the network conditions, such as buffer occupancy and traffic patterns, we can adaptively regulate the packet injection rate to prevent saturation. This ensures that the NoC operates within its capacity and avoids overwhelming any specific routers or links.

Furthermore, we implement intelligent routing algorithms that consider the current network load and congestion levels. These routing algorithms aim to distribute traffic evenly across the network and avoid congested paths. By dynamically selecting the optimal routing paths, we can alleviate congestion and prevent the saturation of specific areas in the NoC.

To validate the effectiveness of our saturation strategy, we conduct extensive simulations using different traffic patterns and varying levels of network load. By analyzing key performance metrics such as throughput, latency, and packet loss, we evaluate the ability of our strategy to mitigate saturation and maintain efficient communication.

In scenarios with high traffic loads (e.g., Pir 0.5), the network can experience a saturation level where the traffic overwhelms the available resources. Our saturation strategy ensures that even in such insanely saturated scenarios, the NoC can still maintain some level of functionality and meaningful communication.

By incorporating the saturation strategy into Noxim, we enhance the fault tolerance capabilities of the NoC. This strategy helps prevent network saturation, ensures efficient communication even under high traffic loads,

and contributes to the overall reliability and performance of the system.

In the following sections, we will delve deeper into the implementation details, evaluation methodology, and experimental results of this enhanced Noxim framework with fault tolerance capabilities.

4.4.3 Configure the simulation parameters

To perform simulations using the Noxim simulator for fault tolerance Network-on-Chip (NoC) cases, it is essential to properly configure the simulation parameters. These parameters define various aspects of the simulation, including the network size, topology, routing algorithm, traffic patterns, simulation duration, and other relevant settings. Noxim provides YAML configuration files that can be modified to specify these parameters.

The YAML configuration files serve as a convenient and flexible way to define the simulation parameters. By modifying these files,the simulation can be customized to our specific requirements and experimental settings. The following are some of the key parameters that can be configured:

1. Network Size: The size of the NoC grid can be specified by setting the number of rows and columns in the configuration file. This determines the total number of nodes in the network.

2. Topology: Noxim supports various topology options, such as mesh, torus, and ring. The desired topology can be selected by configuring the corresponding parameters in the configuration file.

3. Routing Algorithm: Different routing algorithms can be employed to control the packet forwarding in the NoC. The configuration file allows for selecting the desired routing algorithm from the available options.

4. Traffic Patterns: The traffic patterns determine the communication patterns among the nodes in the NoC. Noxim supports different traffic patterns, including random, transpose, bit-reversal, and many more. These patterns can be specified in the configuration file.

5. Simulation Duration: The duration of the simulation can be defined in terms of the number of cycles or the time units. This parameter controls the length of time for which the simulation will run.

6. Other Parameters: Additional parameters, such as injection rate, buffer size, flit size, and link latency, can be configured to match the desired experimental conditions and performance evaluation requirements.

By modifying the YAML configuration files provided in Noxim, We set up the simulation metrics for the 2D-NoC cases. This includes specifying network size, topology, routing algorithm, traffic patterns, simulation duration, and some other relevant parameters. . This flexibility allows for conducting experiments under various scenarios and conditions, enabling a comprehensive analysis of the NoC performance, fault tolerance, and other relevant metrics (Figure 4.5a 4.5b).

```yaml
# X and Y mesh sizes
mesh_dim_x: 16
mesh_dim_y: 16
buffer_depth: 4  # number of flits for each router buffer
flit_size: 32  # size of flits, in bits
r2h_link_length: 2.0   # lenght in mm of router to hub connection
# lenght in mm of router to router connection
r2r_link_length: 1.0
n_virtual_channels: 1
routing_algorithm: XY  # Routing algorithms
routing_table_filename: ""
# Routing specific parameters
dyad_threshold: 0.6  #   dyad_threshold: double
selection_strategy: RANDOM # Selection Strategies
# WIRELESS CONFIGURATION
Hubs:
    defaults:
    # channels from which Hub can receive/transmit
        rx_radio_channels: [0]
        tx_radio_channels: [0]
    # list of node tiles attached to the hub
        attached_nodes: []
    # size of buffers connecting the hub to tiles
        to_tile_buffer_size: 4
        from_tile_buffer_size: 4
    # size of antenna tx/rx
        rx_buffer_size: 64
        tx_buffer_size: 64
    0: attached_nodes: [17, 18, 33, 34]
    1: attached_nodes: [21, 22, 37, 38]
    2: attached_nodes: [25, 26, 41, 42]
    3: attached_nodes: [29, 30, 45, 46]
# Transmission channels configuration
RadioChannels:
    defaults:
        data_rate: 16  # flit transmission
        ber: [0, 0] # bit error rate (CURRENTLY UNSUPPORTED)
        mac_policy: [TOKEN_PACKET]  # mac policies

# SIMULATION PARAMETERS
clock_period_ps: 1000
reset_time: 1000  # duration of reset signal assertion, expressed in cycles
simulation_time: 10000   # overal simulation lenght, expressed in cycles
stats_warm_up_time: 1000   # collect stats after a given number of cycles
detailed: false # power breakdown, nodes communication details
# stop after a given amount of load has been processed
max_volume_to_be_drained: 0
show_buffer_stats: false
use_winoc: false   # enable wireless, when false, all wireless channel configuration
use_wirxsleep: false  # experimental power saving strategy
verbose_mode: VERBOSE_OFF # Verbosity level
trace_mode: false  # Trace
trace_filename: ""
min_packet_size: 12
max_packet_size: 12
packet_injection_rate: 0.01
probability_of_retransmission: 0.01
traffic_distribution: TRAFFIC_RANDOM  # Traffic distribution
traffic_table_filename: "t.txt"   # configuration file
```

(a) (b)

Figure 4.5: simulation parameters

4.4.4 Running Simulations

To run simulations (4.6) for 2D Network-on-Chip (NoC) cases using Noxim, we follow these steps:

- **Configure the simulation parameters:** we Modify the configuration file provided in Noxim to specify our desired simulation setup and parameters. This includes network size, topology, routing algorithm, traffic patterns, simulation duration, and other relevant settings.
- **Execute Noxim:**

  We run4.6 Noxim using the modified configuration file to initiate the simulations. This will start the simulation process and allow the NoC to operate according to the specified parameters.

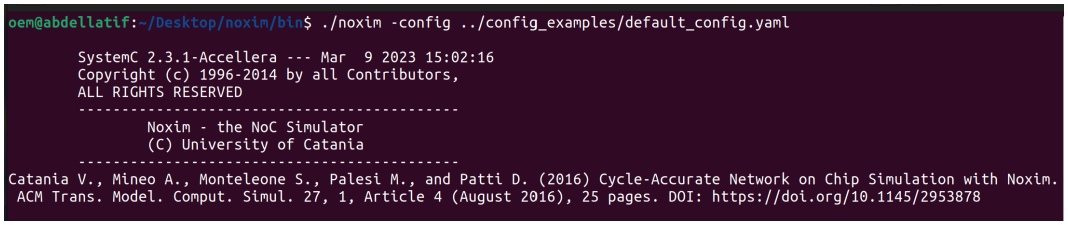


Figure 4.6: NoC run-simulation with noxim.

- **Data collection:**

  We allow the simulations to run for the required time to collect sufficient data for analysis. The duration will depend on the specific objectives of our study and the complexity of the simulated NoC.

By following these steps, we run simulations using Noxim and obtain data that can be further analyzed to gain insights into the performance and behavior of the simulated NoC. to adjust the simulation parameters and experiment with different configurations to explore various scenarios and evaluate the impact on the system.

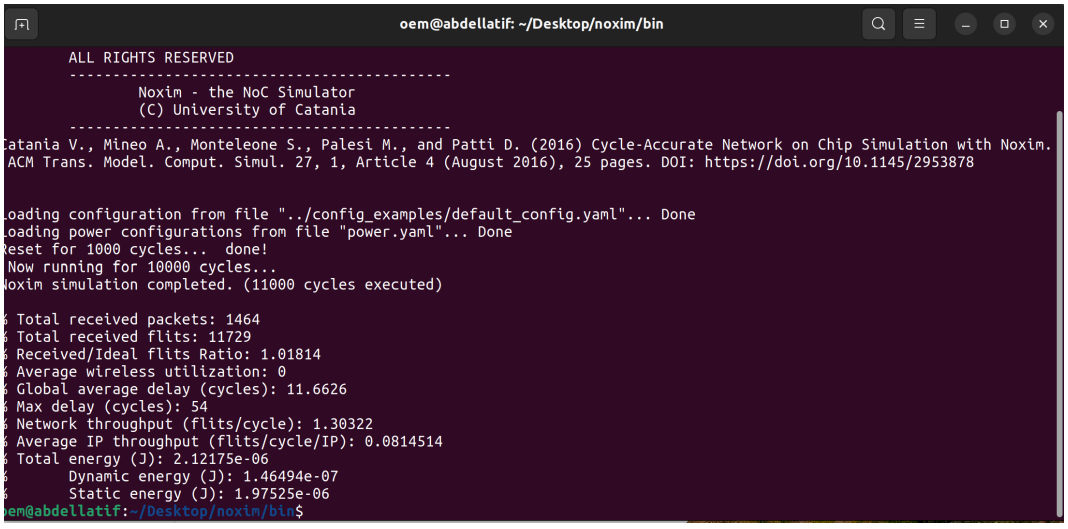


Figure 4.7: simulation NoC Results Format .

4.4.5 Parsing Strategy

The purpose of creating a parsing strategy for the simulation results stored in the ”my.log” file is to handle the large volume of data. Since the log file can contain more than 1,000,000 lines, it becomes challenging to extract specific information from it manually. Therefore, we utilize a parsing

approach using Python to efficiently retrieve the required data.

The parsing strategy enables us to extract trace information about all flits between the source and destination. By parsing the log file, we can obtain valuable details such as the number of nodes involved in the trace, the number of cycles taken, and the elapsed time.

Through this parsing process, we can analyze the behavior of the network-on-chip NoC system and gain insights into its performance. The extracted information allows us to evaluate various aspects of the NoC, such as the efficiency of routing algorithms, the impact of network congestion, and the overall reliability of the communication infrastructure.

4.4.6 Setting up Simulation Iteration

To comprehensively analyze the performance of the Network-on-Chip (NoC) system, we conducted simulations in Noxim by systematically changing one of the simulation parameters. This approach allowed us to explore all possible combinations of NoC simulation parameters, as presented in the table4.1.

Table 4.1: possibilities of Noxim simulation parameters NoC in this work.

Characteristic	Values	Number of possibilities
Topology	MESH, BUTTERFLY, BASE-LINE, OMEGA	4
Network Size	4x4 ...	-
Flit Size	16,32,64,128	4
Routing Algorithms	XY DELTA WEST-FIRST NORTH-LAST NEGATIVE-FIRST ODD-EVEN DYAD TABLE-BASED	8
Strategies	RANDOM BUFFER-LEVEL NOP	3
Traffic Distribution	TRAFFIC-RANDOM, TRAFFIC-TRANSPOSE1, TRAFFIC-TRANSPOSE2, TRAFFIC-HOTSPOT, TRAFFIC-TABLE-BASED, TRAFFIC-BIT-REVERSAL, TRAFFIC-SHUFFLE, TRAFFIC-BUTTERFLY	8
TOTAL	-	3072X(Number possibile of Network Size)

By systematically varying the simulation parameters, such as network size, topology, routing algorithm, traffic patterns, and other relevant factors, we were able to evaluate the impact of each parameter on the overall performance of the NoC. This iterative simulation process provided valuable insights into how different parameter settings affected key performance metrics, including throughput, latency, power consumption, and fault tolerance.

The systematic iteration of simulations in Noxim, covering all possibilities of NoC simulation parameters, allowed us to explore the full range of performance scenarios and identify the most effective parameter settings for achieving desired system performance and reliability.

4.5 Metric Detection

During our analysis of the NoC simulation using Noxim, we discovered the most critical metrics that significantly impact fault tolerance in the NoC system. Through repeated simulations and careful observation of the results, we identified the following metrics as highly important:

1. Throughput: This metric measures the rate at which data is successfully transmitted within the NoC. It provides valuable insights into the overall efficiency and capacity of the network.

2. Latency: Latency refers to the time it takes for a data packet to travel from the source node to the destination node. Lower latency is desirable as it ensures faster communication and reduces delays in data transmission.

3. Energy and Power Consumption: These metrics focus on the amount of energy consumed by the NoC system. By monitoring and optimizing energy consumption, we can enhance the energy efficiency and sustainability of the network.

4. Times: This metric measures the total time taken for the simulation to complete. It helps assess the overall performance and efficiency of the NoC system.

5. Number of Nodes in Tracer: The number of nodes in the tracer provides insights into the complexity and size of the network. This metric is

important for evaluating the scalability and capacity of the NoC architecture.

6. Number of Buffering: Buffering plays a crucial role in managing and regulating data flow within the NoC. By monitoring the number of bufferings, we can assess the efficiency of data handling and potential bottlenecks in the system.

7. Flit Size: Flit size refers to the size of data packets transmitted within the NoC. Optimizing the flit size can have a significant impact on the overall performance and throughput of the network.

8. Scalability: Scalability measures the ability of the NoC system to handle increasing demands and network size. Evaluating the scalability of the system helps identify potential limitations and areas for improvement.

By configuring Noxim to log and analyze these performance metrics during the simulation, we can gain valuable insights into the fault tolerance and overall performance of the NoC system. This knowledge allows us to make informed decisions and implement effective strategies for optimizing the system design and enhancing its resilience.

4.5.1 Simulations Algorithm:

To extract the information to be collected we use the parsing approach using Python, The analysis method we used to analyze and extract information from ”my-log.txt” files.

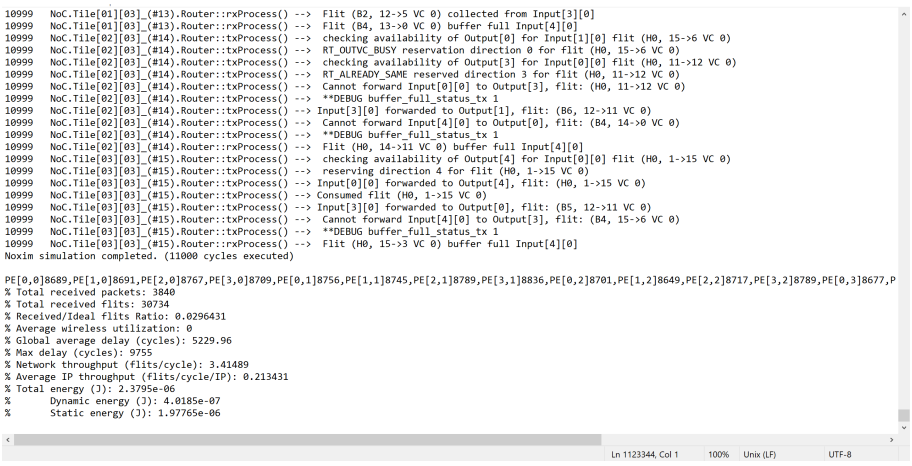


Figure 4.8: example of file ”my-log.txt”.

The diagrams in the next figures (4.9, 4.10) show how the parsing algorithm works. We divide it into two parts:

In this part, we search through all the lines to check if any of the following phrases (”rxProcess()–¿Flit (T7”, ”Consumed flit (T7 ”, ”RTOU-TVC-BUSY”, ”Cannot forward”, ”**DEBUG”, ”RT-ALREADY”) exist. If any of these phrases are found, the line is added to the output file.

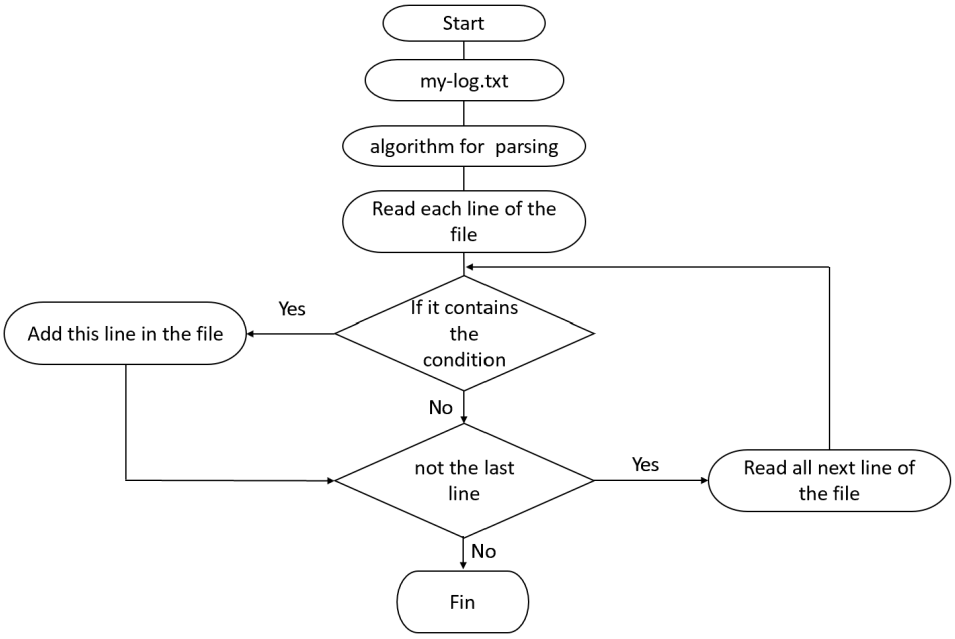


Figure 4.9: the first part of Algorithm parsing.

The second graph shows a part after get the result :

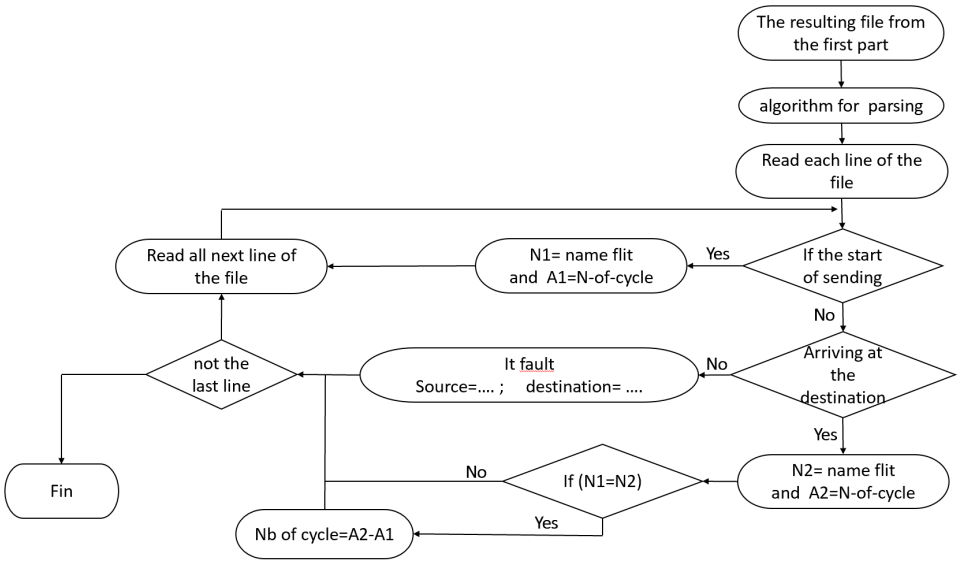


Figure 4.10: The second part of the parsing algorithm.

note : The second graph, the result that we were getting from the parsing algorithm contains a lot of errors because the simulations contain a lot of sending and receiving from the same sender and recipient and overlapping Due to time constraints and the parsing algorithm is not our goal in this research, we have done this part manually

4.5.2 Data collection and analysis

We collect Noxim-generated simulation results for each case. We extract significant performance metrics from output files or logs generated by Noxim during the simulations. We use several generators to extract the required data from the output files

We analyze the collected data using appropriate tools or scripts to compare the performance of 2D-NoC.

```python
def save_lines_with_text(file_path, text_list, output_file):
    with open('/content/my_log.txt', 'r') as file:
        lines = file.readlines()

    filtered_lines = [line.strip() for line in lines for text in text_list if text in line]

    with open('/content/output_file T7', 'a') as output:
        output.write('\n'.join(filtered_lines))
        output.write('\n')

# Usage example
file_path = 'path/to/your/file.txt'
output_file = 'path/to/your/output.txt'

text_to_search = ["rxProcess() -->  Flit (T7", "Consumed flit (T7"]
save_lines_with_text(file_path, text_to_search, output_file)
```

Figure 4.11: Generic parsing example of resulting NoC simulation with Noxim.

After this step we get to explore in the next section a set of resulting parameters .

4.5.3 NoC-based resulting dataset

The following parameters of NoC-based resulting dataset can be explained as follow:

1. Source: The source node ID from where the flit is originating.
2. Destination: The destination node ID to which the flit is being sent.

3. Nam-flit: The unique identifier for the flit.

4. Nb-of-cycle: The number of cycles taken by the flit for transmission.

5. Execution-Time(s): The execution time in seconds for the flit.

6. track node: The number of nodes involved in the trace.

7. Power(W): The power consumption in Watts for the flit transmission.

7. Routing-Algorithm: The routing algorithm used for determining the path of the flit.

8. Scalability: The scalability factor or parameter used in the simulation.

9. Flit-size: The size of the flit in bits.

10. Buffer: The size of the buffer for storing the flit at each node.

11. Fault: A binary value indicating if there is a fault or error in the transmission.

We have to mention some parameters that can not be traced correctly by Noxim such as time by flit and power by flit.Therefor we calculate the time for such flit and the power(W) consumed using the emulator's energy file by using two files : Power.yaml with Default-Config.yaml. It is as follows

By consequence the calculations can be illustrated like this followings:

- For the Time :

  we calculate the time for on flit Simulation Time (s) = Number of cycles * Clock period (s) clock-period-ps = 10,000,000 ps = 10 ms

- Example

  Simulation Time (s) = 4 cycles * 1 ns = 4 ns = 0.000004 milliseconds (ms) = 0.000000004 seconds (s)

- For the Power(W) :

  Power (W) = Dynamic Power (W) + Static Power (W)

  Dynamic Power (W) = Dynamic Energy (J) / Simulation Time (s)
  Static Power (W) = Static Energy (J) / Simulation Time (s)

Table 4.2: possibilities of parameters

Characteristic	Metrics	Number of possibilities
Routing-Algorithm	1, 2, 3, 4, 5, 6, 7	7
Scalability	4x4	1
Flit-Size	16,32,64,128	4
Buffer	4, 32	2
Nbr-cycle	-	-
track node	-	-
Execution-Time(s)	-	-
Power(W)	-	-
Fault	0, 1	2

The figure(4.12) represents the final result of our data set which analyse the state of NoC based node (faulty node :1, Safe node : 0).

	A	B	C	D	E	F	G	H	I	J	K	L
1	Source	Destination	Nam-flit	Nb-of-cycle	ecution-Tim	ode-de-la-t	Power(W)	iting-Algorit	Scalability	Flit-size	Buffer	Fault
2	11	15	0	4	4E-09	2	2.7808000E-11	1	16	32	4	0
3	1	12	0	9	9E-09	4	2.8155600E-10	1	16	32	32	0
4	6	5	0	77	2.2E-08	2	1.6823840E-09	1	16	32	4	1
5	3	4	5	105	2E-08	5	1.7380000E-09	1	16	32	32	1
6	6	1	0	6	6E-09	4	4.1328000E-10	1	16	128	4	0
7	9	4	6	123	1E-08	3	1.0590300E-08	1	16	128	4	1
8	11	9	1	6	1.09E-07	3	5.6309400E-09	1	16	128	32	0
9	14	0	7	173	6E-09	6	1.7874360E-08	1	16	128	32	1
10	13	3	7	12	1.91E-07	6	1.3408200E-19	2	16	32	4	0
11	14	5	7	110	6E-09	4	4.2120000E-21	2	16	32	4	1
12	4	6	7	6	3.75E-07	3	7.5337500E-19	2	16	32	32	0
13	7	4	7	117	1E-08	4	2.0090000E-20	2	16	32	32	1
14	5	8	7	6	8.73E-07	3	1.1183130E-18	2	16	128	4	0
15	4	6	7	6	1.19E-07	3	3.0797200E-19	2	16	128	32	0
16	12	0	7	8	8E-09	4	5.6160000E-21	3	16	32	4	0
17	5	8	7	6	2.03E-07	3	1.4250600E-19	3	16	32	32	0
18	7	14	7	8	3.32E-07	4	2.3306400E-19	3	16	128	4	0
19	12	3	0	14	1.4E-08	7	9.8280000E-21	3	16	128	32	0
20	6	14	7	8	8E-09	3	5.6160000E-21	4	16	32	4	0

Figure 4.12: part of the data set.

We will aim to perform a multiclass classification of error types in our NoC simulation, going beyond the 'faulty' and 'correct' binary outputs to classify different specific error types

4.6 NoC-based machine learning process

By leveraging machine learning algorithms, we can develop intelligent models that can effectively identify and classify faults within the NoC.

we will introduce a machine learning model using TensorFlow and Keras.

the model architecture is defined using the keras. Sequential class from the TensorFlow library. This class allows us to create a linear stack of layers for our neural network model.

The model consists of three dense layers, each created using the keras layers,Dense class, Dense layers are fully connected layers, where each neuron in a layer is connected to every neuron in the previous layer. Here's a breakdown of the layers:

The first dense layer:

Number of units: 64

Activation function: ReLU (Rectified Linear Unit)

Input shape: (11,) - Assumes there are 11 input features

The second dense layer:

Number of units: 64

Activation function: ReLU

The third dense layer:

Number of units: 1

Activation function: Sigmoid

Assumes binary classification, as the output is a single value between 0 and 1 representing the class probability The keras.Sequential class allows us to easily stack these layers one after another. This sequential structure ensures that the output of each layer serves as the input for the next layer, creating a flow of information through the network.

By compiling the model with the appropriate optimizer, loss function, and metrics, we can train the model using the provided training data (X-train and y-train). The training process aims to adjust the weights and biases of the model's neurons to minimize the defined loss function.

Once the model is trained, it can be used to make predictions on new data, as demonstrated in the code. The predicted class is determined by comparing the output value with a threshold of 0.5. If the predicted value is greater than or equal to 0.5, the class is considered "NOT OKEY"; otherwise, it is considered "OKEY".

In the case where the prediction is "NOT OKEY," the code further retrieves a separate dataset (other data) and preprocesses it in the same way as the original dataset. It then uses the Nearest Neighbors algorithm to find the k nearest lines in other6data that are similar to the new data.

The similar lines are then printed as proposals for optimizing network-on-chip performance.

Overall, this code demonstrates the construction, training, and utilization of a neural network model for classification tasks using the Keras API in TensorFlow.

4.6.1 NoC-based collected dataset

we provide details about the configuration data used for the ML-based routing in NoC architectures. The dataset includes various input parameters and corresponding output values.

The input parameters consist of the following features: Buffer size, Throughput, Latency, Power consumption, Flit size, Scalability,Number of cycle,Track node, Routing Algorithm, and Execution Time (in seconds). These parameters capture important characteristics and performance metrics of the NoC.

The output values include the Fault State and Rank. The Fault State that has two class indicates whether a fault or error occurred during routing, while the Rank represents the priority or preference of the selected routing option.

The dataset used for training and evaluation contains a total of 1265 rows. The data is divided into subsets for training and validation, with a specific ratio allocated for each. The training process involves multiple epochs, where the model is iteratively trained and adjusted to optimize performance.

The final test results show a Test Loss of 0.1905 and Test Accuracy of 0.9842. These metrics indicate the model's performance on unseen data, suggesting that it can effectively make routing decisions based on the provided inputs.

4.6.2 Model Evaluation and Performance Metrics

Once we have trained the ML-NoC model, it is crucial to evaluate its performance and assess its effectiveness in fault detection.

We evaluate our trained model on the testing dataset that we previously set aside. This dataset contains data that our model has not seen during training, allowing us to assess its generalization capability. By feeding the testing data into our trained model and comparing the predicted outputs with the ground truth labels, we can compute the selected performance metrics and analyze how well our model performs in fault detection.

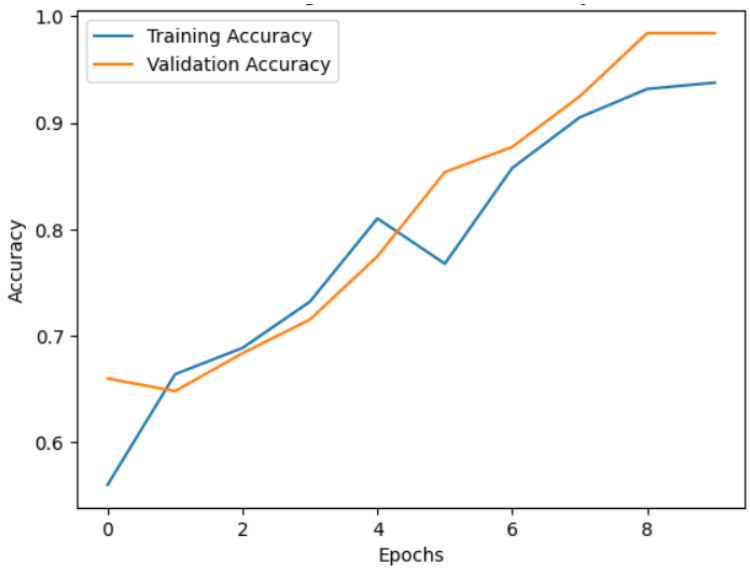


Figure 4.13: Training and validation Accuracy .

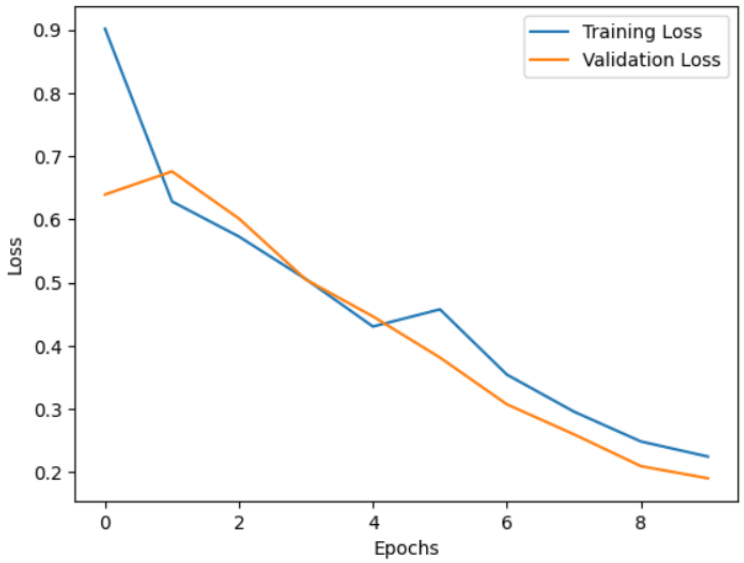


Figure 4.14: Training and validation Loss

The training accuracy curve and the training loss curve appear in the range of 2 to 4 periods, overshoot occurs, which means that the model

learns to fit the training data very closely.

4.7 Results and Discussions

In this section, we present the results and discuss the findings obtained from the evaluation of the ML-NoC model. The model was trained and tested using the provided code, and the following results were achieved:

Test loss: 0.2939 % Test accuracy: 0.9842 %

```
Test loss: 0.1905
Test accuracy: 0.9842
1/1 [==============================] - 0s 56ms/step
Prediction: Class NOT OKEY
```

Figure 4.15: Resutlt of ML-NoC training.

The obtained accuracy of 98.48 indicates that the ML-NoC model is highly effective in classifying samples correctly. This demonstrates the model's ability to accurately detect faults within the network-on-chip (NoC) system. Furthermore, based on the prediction result, if a sample is classified as belonging to the "Class NOT OKEY" category, we propose a solution that allows network-on-chips to operate optimally.

```
Proposal that allows network-on-chips to work at its best:
     Source  Destination  Nam-flit  Nb-of-cycle  Execution-Time(s)  \
217       3            9         0            9       9.000000e-09
218       3            9         1            9       9.000000e-09
219       3            9         2            9       9.000000e-09
220       3            9         3            9       9.000000e-09
221       3            9         4            9       9.000000e-09

     Nb-node-de-la-trace      Power(W)  Aouting-Algorithm  Scalability  \
217                    5  3.519450e-10                  1           16
218                    5  3.519450e-10                  1           16
219                    5  3.519450e-10                  1           16
220                    5  3.519450e-10                  1           16
221                    5  3.519450e-10                  1           16

     Flit-size  Buffer  Fault
217         32      32      0
218         32      32      0
219         32      32      0
220         32      32      0
221         32      32      0
```

Figure 4.16: first the train of data.

By extracting the destination value from the new-data and finding similar lines in another dataset, we can offer insights and recommendations for improving the network's performance. The proposed lines from the other dataset provide valuable information for further analysis and decision-

making.

Conclusion

we conducted a comparative study to evaluate the accuracy of our ML model in dealing with various NoC topologies. We explored the effectiveness of transfer learning and reformer algorithms in enhancing the performance and fault tolerance of NoC architectures. The results demonstrated the potential of these techniques in improving the accuracy and efficiency of our model. This chapter , the addition of a ML (Language Model) network on chip has proven to be a valuable enhancement in the field of network-on-chip (NoC) architectures

Conclusion and perspectives

In this master's thesis, we addressed the problem of identifying failures in Network-on-Chip (NoC) systems and proposed a machine learning model to detect and prevent such failures. The main objective was to develop a model that could accurately identify errors in the NoC and provide recommendations to optimize its performance.

Through our research, we successfully achieved a solution that can effectively identify errors in NoC systems and offer suggestions to ensure their proper functioning. By leveraging machine learning techniques, we were able to analyze the behavior and performance of NoCs and detect non-deterministic phenomena that can lead to system failures.

In Chapter 1, we focused on managing software failures in NoCs, exploring various techniques to mitigate the impact of failures and ensure deadlock-free operation. This chapter provided a comprehensive understanding of the challenges associated with software failures in NoC systems and proposed effective strategies to address them.

Chapter 2 presented a comparative study of NoC analysis methods and simulators. We examined different techniques for evaluating NoC architectures, such as performance evaluation, power consumption estimation, and communication modeling. This chapter provided valuable insights into the strengths and limitations of existing analysis methods and simulators, helping researchers and practitioners select appropriate approaches for assessing NoC systems.

Chapter 3 highlighted the limitations of traditional deterministic approaches in handling non-deterministic phenomena in NoC systems. We established the need for machine learning techniques to address these challenges effectively. By implementing machine learning algorithms, data

processing techniques, and optimization strategies, we demonstrated the potential of machine learning to detect and prevent failures in routing algorithms within NoCs.

Throughout our research, we obtained significant results that showcase the effectiveness of the proposed machine learning model. By accurately identifying errors in NoC systems and providing recommendations for optimal performance, our model can contribute to the development of robust and efficient NoC architectures.

Looking ahead, there are several promising research directions to explore. One such direction involves investigating other routing algorithms to further enhance the adaptability and efficiency of NoCs. Additionally, exploring alternative metrics, such as three-dimensional (3D) integration, can provide valuable insights into improving the scaleability and performance of NoC systems. Furthermore, addressing hardware failures in NoC systems is another area of interest for future research, as it complements the existing focus on software failures.

In conclusion, this master's thesis has shed light on the problem of failures in NoC systems and provided a machine learning-based solution for detecting and preventing such failures. By identifying errors and offering suggestions to optimize NoC performance, our research contributes to the advancement of reliable and high-performing NoC architectures. The findings and perspectives presented in this thesis can inspire further research and innovation in the field of Network-on-Chip technologies.

Perspective

Here is a concise and useful summary of the future research directions for enhancing NoCs:

Alternative routing algorithms: Investigate new routing algorithms to improve adaptability and efficiency, optimizing communication paths and minimizing delays.

Three-dimensional (3D) integration: Explore the benefits of integrating NoCs in a 3D space, enhancing scalability and performance by utilizing

additional dimensions for routing and resource allocation.

Address hardware failures: Focus on hardware failures in NoC systems by developing fault-tolerant mechanisms and error detection/correction techniques specific to NoC hardware components, enhancing reliability and robustness.

By pursuing these research directions, researchers can contribute to the improvement and advancement of NoC architectures, enabling more efficient and reliable communication in complex systems.

Bibliography

[1] Sergi Abadal, Mario Iannazzo, Mario Nemirovsky, Albert Cabellos-Aparicio, Heekwan Lee, and Eduard Alarcón. On the area and energy scalability of wireless network-on-chip: A model-based benchmarked design space exploration. *IEEE/ACM Transactions on Networking*, 23(5):1501–1513, 2014.

[2] Khurshid Ahmad and Muhammad Sethi. Review of network on chip routing algorithms. *EAI Endorsed Transactions on Context-aware Systems and Applications*, 7(22), 2020.

[3] K Balamurugan, S Umamaheswaran, Tadele Mamo, S Nagarajan, and Lakshmana Rao Namamula. Roadmap for machine learning based network-on-chip (m/l noc) technology and its analysis for researchers. *Journal of Physics Communications*, 6(2):022001, 2022.

[4] Yaniv Ben-Itzhak, Eitan Zahavi, Israel Cidon, and Avinoam Kolodny. Hnocs: modular open-source simulator for heterogeneous nocs. In *2012 international conference on embedded computer systems (SAMOS)*, pages 51–57. IEEE, 2012.

[5] Luca P Carloni, Partha Pande, and Yuan Xie. Networks-on-chip in emerging interconnect paradigms: Advantages and challenges. In *2009 3rd ACM/IEEE International Symposium on Networks-on-Chip*, pages 93–102. IEEE, 2009.

[6] Vincenzo Catania, Andrea Mineo, Salvatore Monteleone, Maurizio Palesi, and Davide Patti. Noxim: An open, extensible and cycle-accurate network on chip simulator. In *2015 IEEE 26th international*

conference on application-specific systems, architectures and processors (ASAP), pages 162–163. IEEE, 2015.

[7] En-Jui Chang, Hsien-Kai Hsin, Shu-Yen Lin, and An-Yeu Wu. Path-congestion-aware adaptive routing with a contention prediction scheme for network-on-chip systems. *IEEE Transactions on computer-aided design of Integrated circuits and systems*, 33(1):113–126, 2013.

[8] Subodha Charles and Prabhat Mishra. Reconfigurable network-on-chip security architecture. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 25(6):1–25, 2020.

[9] Naveen Choudhary, MS Gaur, and V Laxmi. Energy efficient network generation for application specific noc. *Glob J Comput Sci Technol*, 11(16):47–56, 2011.

[10] Sourav Das, Janardhan Rao Doppa, Dae Hyun Kim, Partha Pratim Pande, and Krishnendu Chakrabarty. Optimizing 3d noc design for energy efficiency: A machine learning approach. In *2015 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 705–712. IEEE, 2015.

[11] Giovanni De Micheli, Ciprian Seiculescu, Srinivasan Murali, Luca Benini, Federico Angiolini, and Antonio Pullini. Networks on chips: From research to products. In *Proceedings of the 47th Design Automation Conference*, pages 300–305, 2010.

[12] Anderson Roberto Pinheiro Domingues et al. Orca: A self-adaptive, multiprocessor system-on-chip platform. 2020.

[13] Wenkai Guan, Milad Ghorbani Moghaddam, and Cristinel Ababei. Quantifying the impact of uncertainty in embedded systems mapping for noc based architectures. *Microprocessors and Microsystems*, 80:103503, 2021.

[14] Abdelhamid HARICHE. *Approche à base d'opérateurs pour la validation formelle de systèmes micro-électroniques orientés NoC*. PhD thesis, Université Ibn Khaldoun-Tiaret-, 2019.

[15] Julian Harttung, Elke Franz, Sadia Moriam, and Paul Walther. Lightweight authenticated encryption for network-on-chip communications. In *Proceedings of the 2019 on Great Lakes Symposium on VLSI*, pages 33–38, 2019.

[16] Jan Heißwolf. A scalable and adaptive network on chip for many-core architectures. 2014.

[17] Rickard Holsmark, Maurizio Palesi, and Shashi Kumar. Deadlock free routing algorithms for irregular mesh topology noc systems with rectangular regions. *Journal of Systems Architecture*, 54(3-4):427–440, 2008.

[18] Hemayet Hossain, Mostak Ahmed, Abdullah Al-Nayeem, Tanzima Zerin Islam, and Md Mostofa Akbar. Gpnocsim-a general purpose simulator for network-on-chip. In *2007 International Conference on Information and Communication Technology*, pages 254–257. IEEE, 2007.

[19] Jie Hou. Performability analysis of networks-on-chips. 2021.

[20] Kwangok Jeong, Andrew B Kahng, Bill Lin, and Kambiz Samadi. Accurate machine-learning-based on-chip router modeling. *IEEE Embedded Systems Letters*, 2(3):62–66, 2010.

[21] Nan Jiang, George Michelogiannakis, Daniel Becker, Brian Towles, and William J Dally. Booksim 2.0 user's guide. *Standford University*, page q1, 2010.

[22] Andrew B Kahng, Bill Lin, and Siddhartha Nath. Orion3. 0: A comprehensive noc router estimation tool. *IEEE Embedded Systems Letters*, 7(2):41–45, 2015.

[23] Sarzamin Khan, Sheraz Anjum, Usman Ali Gulzari, and Frank Sill Torres. Comparative analysis of network-on-chip simulation tools. *IET Computers & Digital Techniques*, 12(1):30–38, 2018.

[24] Miyeon Lee, Inhwan Kim, Pureuna Shin, Hwayong Oh, and Keewon Joe. Prediction-reconstruction vlsi architecture with efficient pipelin-

ing for vvc decoder. In *2023 IEEE International Conference on Consumer Electronics (ICCE)*, pages 01–03. IEEE, 2023.

[25] Ting-Ru Lin, Drew Penney, Massoud Pedram, and Lizhong Chen. A deep reinforcement learning framework for architectural exploration: A routerless noc case study. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 99–110. IEEE, 2020.

[26] Mieszko Lis, Keun Sup Shim, Myong Hyon Cho, Pengju Ren, Omer Khan, and Srinivas Devadas. Darsim: a parallel cycle-level noc simulator. 2010.

[27] Feiyang Liu, Huaxi Gu, and Yintang Yang. Dtbr: A dynamic thermal-balance routing algorithm for network-on-chip. *Computers & Electrical Engineering*, 38(2):270–281, 2012.

[28] Jason Lowe-Power, Abdul Mutaal Ahmad, Ayaz Akram, Mohammad Alian, Rico Amslinger, Matteo Andreozzi, Adrià Armejach, Nils Asmussen, Brad Beckmann, Srikant Bharadwaj, et al. The gem5 simulator: Version 20.0+. *arXiv preprint arXiv:2007.03152*, 2020.

[29] Zhonghai Lu. *Design and analysis of on-chip communication for network-on-chip platforms*. PhD thesis, KTH, 2007.

[30] Sadia Moriam. On fault resilient network-on-chip for many core systems. 2018.

[31] Maurizio Palesi, Davide Patti, and Fabrizio Fazzino. Noxim, 2010.

[32] Zhiliang Qian, Da-Cheng Juan, Paul Bogdan, Chi-Ying Tsui, Diana Marculescu, and Radu Marculescu. Svr-noc: A performance analysis tool for network-on-chips using learning-based support vector regression model. In *2013 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 354–357. IEEE, 2013.

[33] Juan Ruiz-Rosero, Gustavo Ramirez-Gonzalez, and Rahul Khanna. Field programmable gate array applications—a scientometric review. *Computation*, 7(4):63, 2019.

[34] Pradip Kumar Sahu and Santanu Chattopadhyay. A survey on application mapping strategies for network-on-chip design. *Journal of systems architecture*, 59(1):60–76, 2013.

[35] Ciprian Seiculescu, Srinivasan Murali, Luca Benini, and Giovanni De Micheli. Sunfloor 3d: A tool for networks on chip topology synthesis for 3-d systems on chips. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 29(12):1987–2000, 2010.

[36] Manuel Selva, Charles Emmanuel Effiong, and Abdoulaye Gamatié. Mcsim-cycle-accurate-xbar: Manycore platform simulation tool for crossbar-based platform at a cycle-accurate level. 2016.

[37] Amit Kumar Singh. Survey of network-on-chip simulators.

[38] Anh T Tran and Bevan Baas. Noctweak: a highly parameterizable simulator for early exploration of performance and energy of networks on-chip. *VLSI Computation Lab, ECE Department, University of California, Davis, Tech. Rep. ECE-VCL-2012-2*, 2012.

[39] Mohammad Trik, Hoda Akhavan, Amir Massoud Bidgoli, Ali Mohammad Norouzzadeh Gil Molk, Hossein Vashani, and Saadat Pour Mozaffari. A new adaptive selection strategy for reducing latency in networks on chip. *Integration*, 89:9–24, 2023.

[40] Kartika Vyas, Naveen Choudhary, and Dharm Singh. Nc-g-sim: A parameterized generic simulator for 2d-mesh, 3d-mesh & irregular on-chip networks with table-based routing. *Global Journal of Computer Science and Technology*, 13(14), 2013.

[41] Ke Wang, Ahmed Louri, Avinash Karanth, and Razvan Bunescu. High-performance, energy-efficient, fault-tolerant network-on-chip design using reinforcement learning. In *2019 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, pages 1166–1171. IEEE, 2019.

[42] Xiaoyun Zhang, Dezun Dong, Cunlu Li, Shaocong Wang, and Liquan Xiao. A survey of machine learning for network-on-chips. *Available at SSRN 4440846*.

[43] Hao Zheng, Ke Wang, and Ahmed Louri. Adapt-noc: A flexible network-on-chip design for heterogeneous manycore architectures. In *2021 IEEE international symposium on high-performance computer architecture (HPCA)*, pages 723–735. IEEE, 2021.