# Resource Optimisation of Networks On Chip for High Performance System On Chip Applications

A Thesis submitted for the Degree of

Doctor of Philosophy

By

Ahmed Al-Alousi


Department Electrical and Electronic Engineering

College of Engineering, Design and Physical Sciences

Brunel

University London

April 2024

# Author's Declaration

The work described in this thesis has not been previously submitted for a degree in this or any other university and unless otherwise referenced it is the author's own work.

# Abstract

This thesis presents a novel methodology for resource optimisation of 3D Network-on-Chip (NoC) architectures in High-Performance System-on-Chip (SoC) applications.

The proposed approach combines hypergraph-based modelling and genetic algorithm optimisation to efficiently explore the design space and identify optimal combinations of topology and routing algorithms tailored to specific application requirements. Two compute-intensive, yet different use cases were chosen to validate the approach; namely, the double SHA256 attack as applied to Bitcoin mining, and real-time facial recognition.

In addition to hypergraph modelling and GA optimisation, a unique aspect of this work is the development of the Performance-to-Cost-Ratio function concept, as an effective, yet simple method to steer the fitness evaluation during the optimisation phase.

In contrast to existing research that focuses on narrow perspectives of the problem, this work offers a more holistic approach, by considering the interplay of routing strategies, buffer and message sizing, bandwidth, and latency factors, together with resource utilisation.

The results and insights obtained contribute to the overall understanding of the design and optimisation of 3D NoC architectures and their impact on system performance and resource utilisation.

Three specific research questions are addressed. First, the effective utilisation of hypergraphs as a design and implementation space exploration modelling tool for 3D NoCs; Second, is the justification and use of genetic algorithms as an optimisation technique, once a suitable topology is identified; and finally, the combination of the ideas of hypergraph modelling and GA optimisation as framework in high-performance SoC designs employing 3D NoCs.

Extensive simulations and comparative analyses carried out showed significant performance improvements in latency, throughput, bandwidth and resource utilisation, versus the chosen 3D Mesh baseline architecture.

The optimised architectures also lead to an observable energy efficiency characteristic, when tested on an actual FPGA implementation.

The proposed methodology provides a systematic and automated design space exploration and optimisation approach in the domain, eliminating the need for manual design space

exploration, this enabling architects and designers to make informed decisions based on the specific requirements of the target application, from the onset.

The insights and techniques presented in this thesis have far reaching implications for developing efficient and scalable NoC solutions in the era of kilo- and mega-core SoC applications.

# Dedication

For ***The Glorious Iraq***, the cradle of civilisation; you are but the distance of heartbeat away.

For my late parents ***Abdulsalam and Fetooh***; you are both everything I have become.

For my children: ***Adil, Katrin, Nadine and Reema***; who never stopped believing in their dad, came what may.

# Acknowledgments

I would like to express my deepest gratitude and thanks to my supervisor, Professor Maozhen Li, for his unwavering support, calm and unassuming demeanour, compassion, guidance, and mentorship throughout this journey. His constant encouragement was priceless in both the darkest and the brightest times. I am truly fortunate to have worked under his supervision, and I am forever indebted to him for his invaluable contributions to my academic and personal growth.

I wish to also extend my sincere appreciation to Vicky Maladeni of the Postgraduate Support Office for her compassion, understanding, and exceptional assistance. Vicky's tireless efforts speaks volumes about the person that she is, and the organisation's success in providing a supportive and inclusive environment, which has been crucial to my well-being and success as a researcher. I am deeply grateful to Vicky for her prompt responses, helpful advice, and genuine concern for my welfare throughout my studies.

I wish to thank members of my family, and specifically acknowledge two without whome I would not be writing this today: my auntie Nesrat and my eldest sister Thanaa. Words are simply not enough.

Last, and by no means least, I wish to particularly acknowledge my daughter Nadine, my guardian angel: your love, support, enthusiasm and natural engineering brilliance made all of this work worthwhile. I look forward to being at your graduation ceremony, my precious.

# Contents

# List of Figures

# List of Tables

# List of Code Listings

# List of Abbreviations

| Term | Meaning |
|---|---|
| 2D | Two Dimensional |
| 3D | Three Dimensional |
| ACO | Ant Colony Optimisation |
| ASIC | Application-Specific Integrated Circuit |
| BFT | Butterfly Fat Tree |
| BT | Binary Tree |
| CFET | Complementary Field Effect Transistor |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| Digraph | Directed Graph |
| NSGA | Digital Signal Processor |
| DTA | Dynamic Timing Analysis |
| FinFET | Fin Field Effect Transistor |
| FPGA | Field Programmable Gate Array |
| GA | Genetic Algorithm |
| GPU | Graphics Processing Unit |
| HDL | Hardware Description Language |
| Hypergraph | Multi-dimensional graph |
| IC | Integrated Circuit |
| IoT | Internet of Things |
| IP | Intellectual Property |
| LP | Linear Programming |
| ML | Machine Learning |
| MOSFET | Metal Oxide Semiconductor Field Effect Transistor |
| MPI | Message Passing Interface |

| Term | Meaning |
|---|---|
| NI | Network Interface |
| NoC, NOC | Network On Chip |
| NS | NanoSheet |
| NSGA-II | Non-dominant Sorting Genetic Algorithm |
| PE | Processing Element |
| PoW | Proof of Work |
| QoS | Quality of Service |
| ROSS | Rensselaer's Optimistic Simulation System |
| SA | Simulated Annealing |
| SoC, SOC | System On Chip |
| STI | Shallow Trench Isolation |
| TSV | Through Silicon Via |
| VC | Virtual Channel |
| VLSI | Very Large Scale Integration |

# Chapter 1

# Introduction

## 1.1 Background and Motivation

The rapid advancements in semiconductor technology and the increasing demand for high-performance computing have driven the development of complex System-on-Chip (SoC) architectures. These SoCs integrate many processing elements, memory modul es, and specialised accelerators on a single chip, enabling the execution of computationally intensive applications[1]. However, as the number of on-chip components grows, the communication infrastructure becomes a critical bottleneck, limiting the overall system performance and scalability[2].

Network-on-Chip (NoC) architectures have emerged as a promising solution to address the communication challenges in modern SoCs as far back as demonstrated in [3]. NoCs provide a structured and scalable communication fabric that enables efficient data transfer among the on-chip components [4]. By abstracting the communication infrastructure from the processing elements, NoCs offer several advantages over traditional bus-based architectures, including higher bandwidth, lower latency, and improved scalability [5].

The design of efficient NoC architectures is a complex task involving various design choices, such as topology selection, routing algorithm design, and resource allocation. These design choices significantly impact the performance, power consumption, and area overhead of the NoC and, ultimately, the overall system performance [6]. Therefore, effective design space exploration and optimisation techniques are crucial for identifying the most suitable NoC configurations for specific applications or system requirements.

Three-dimensional (3D) NoC architectures have gained significant attention in recent years due to their potential to enhance the performance and scalability of SoCs. By leveraging 3D integration technologies, such as through-silicon vias (TSVs) and monolithic 3D integration, 3D NoCs enable stacking multiple layers of processing elements and interconnects [7]. This

vertical integration offers several benefits, including reduced wire lengths, higher bandwidth, and increased package density [8].

However, the design and optimisation of 3D NoCs pose additional challenges compared to their 2D counterparts, such as the increased complexity of the topology, the impact of inter-layer communication, and the need for efficient vertical link placement [9].

To tackle these and other challenges in effectively explore the design space of 3D NoCs, more holistic modelling and optimisation techniques are required. In that sense, Hypergraph theory has emerged as a robust mathematical framework for representing and analysing complex systems, including NoC architectures.

Hypergraphs extend the concept of traditional graphs by allowing edges to connect any number of vertices, enabling the capture of higher-order interactions and dependencies. By modelling 3D NoCs as hypergraphs, designers can accurately represent the complex interconnections among the processing elements and comprehensively analyse of the network properties and performance characteristics, in contrast with the more traditional approach of using vendor-specific tools which, while effective, will often come at the expense of portability and flexibility, as well as cost.

Taking this further and superimposing it on time-to-market, skilled design resource costs and efficiency drives, it soon becomes apparent that vendor commitment, particularly at the early conceptual stages, will have a drastic impact on the end product characteristics [10].

In view of the above, the need for using more vendor-agnostic tools, while retaining accuracy of the design space characteristics modelling, lends even more weight to the use of hypergraphs as an accurate, relatively simple, well founded and well understood approach to the question of modelling 3D NoCs; after all, they are inherently 3D architectures [11].

Another aspect of the problem is the sheer size of the design space 3D NoCs tend to span, which is complicated even further where conflicting constraints, such as high performance and low power consumptions need to be satisfied [12].

In fact, the vastness of the design space 3D NoCs on its own, necessitates efficient modelling and optimisation techniques, to aid identify the most suitable configurations for a given application [13]. Genetic algorithms (GAs) have been widely used for solving complex optimisation problems in various domains, including NoC design [14]. The principles of natural evolution inspire GAs and employ mechanisms such as selection, crossover, and mutation to explore the search space and evolve optimal solutions [15]. By encoding the NoC design parameters as chromosomes and defining appropriate fitness functions, GAs can effectively navigate the design space and identify NoC configurations that optimise performance metrics such as throughput, latency, and power consumption [16].

Given the above, it is then apparent that applying hypergraph-based modelling and GA-based

optimisation techniques to 3D NoCs clearly represent new opportunities for resource optimisation in high-performance SoC designs, as well as answering the fundamental requirements of a vendor-independent, yet faithful and accurate tools for design space exploration and optimisation.

By considering the specific requirements and constraints of the target application, this approach can guide the selection of optimal NoC topologies, routing algorithms, and resource allocation strategies.

Once modelled, a 3D NoC can then be optimised using a genetic algorithms process, so as to to maximise the utilisation of the available hardware resources, while meeting the performance and power budget constraints [17]. The resulting optimised 3D NoC architectures can significantly enhance the system performance, reduce power consumption, and improve the overall efficiency of the SoC.

## 1.2 Research Questions

With Section 1.1 setting the stage for motivation, we set out to address three key research questions related to designing and optimising 3D NoC architectures for high-performance SoC designs. These are:

### 1.2.1 RQ1: How can hypergraphs be effectively used as a design and implementation space exploration tool for 3D Networks on Chip?

The first research question focuses on applying hypergraph theory to arrive at a consistent approach, appropriate for modelling and analyse 3D NoC architectures.

As stated earlier, hypergraphs provide a robust mathematical framework for representing complex systems with higher-order interactions and dependencies.

Given that 3D NoCs are by definition 3D structures, hypergraphs can effectively and accurately capture the intricate interconnections among the processing elements, memory modules, and other on-chip components, as well as the characteristics of said interconnects. By modelling 3D NoCs as hypergraphs, designers can gain valuable insights into the network's structural properties and performance characteristics.

Indeed, hypergraph-based analysis techniques, such as partitioning, clustering, and connectivity analysis, can be applied to identify optimal NoC topologies, and to evaluate the impact of different design choices and explore the trade-offs between performance, power, and area.

Hypergraph models can also facilitate the study of various network properties, such as bandwidth, latency, and congestion, enabling a comprehensive understanding of the NoC behaviour.

As a design and implementation space exploration tool for 3D NoCs, Hypergraphs capture complex relationships and dependencies among the network elements. By representing the NoC as a hypergraph, designers can perform detailed analysis and optimisation at a higher level of abstraction without the need for extensive low-level simulations. This allows for rapid exploration of different design alternatives and identifying promising NoC configurations.

Moreover, hypergraph-based techniques can be integrated with existing NoC design flows and tools, providing a seamless and efficient way to explore the design space. The insights gained from hypergraph analysis can guide the selection of optimal NoC architectures, enabling designers to make informed decisions and trade-offs based on the specific requirements of the target application.

### 1.2.2 RQ2: How can genetic algorithms be justified and used for optimising the identified topologies in 3D Networks on Chip?

The second research question addresses the use of genetic algorithms (GAs) to optimise the identified topologies in 3D NoCs.

GAs are a class of evolutionary algorithms successfully applied to various optimisation problems, including NoC design. They are particularly well-suited for exploring large and complex search spaces, where an exhaustive evaluation of all possible solutions is impractical.

In the context of 3D NoC optimisation, GAs can be used to search for the best combination of design parameters, such as topology, routing algorithm, buffer sizes, and link bandwidth, that maximise the desired performance metrics. The justification for using GAs lies in their ability to efficiently navigate the vast design space and find near-optimal solutions in a reasonable amount of time [18],[19].

The optimisation process using GAs typically involves the following steps:

- Encoding: The NoC design parameters are encoded as chromosomes, where each gene represents a specific parameter value. The encoding scheme should capture all the relevant design choices and their respective ranges.
- Fitness evaluation: A fitness function is defined to evaluate the quality of each chromosome (NoC configuration) based on the desired performance metrics, such as throughput, latency, and power consumption. The fitness function assigns a score to each chromosome, indicating its suitability for the target application.
- Selection: The selection operator chooses the fittest chromosomes from the current population to serve as parents for the next generation. Various selection methods, such as tournament or roulette wheel selection, can balance exploration and exploitation [20].
- Crossover: The crossover operator combines the genetic information of the selected

chromosomes to create their offspring. This operation promotes the exchange of good genetic material and explores new regions of the search space.

- Mutation: The mutation operator introduces random changes to the genes of the offspring chromosomes, helping to maintain diversity in the population and prevent premature convergence to suboptimal solutions.
- Termination: The GA iteratively applies the selection, crossover, and mutation operators to evolve the population over multiple generations until a termination criterion is met, such as reaching a maximum number of generations or achieving a satisfactory fitness level.

By iteratively evolving a population of NoC configurations and selecting the fittest individuals, GAs can effectively explore the design space and identify optimised topologies that meet the desired performance objectives. Using GAs in 3D NoC optimisation is further justified by their capability to handle complex design constraints, such as power, budget, and area limitations, by incorporating them into the fitness function or through specialised operators.

GAs offer flexible optimisation objectives and easily adapt to different application requirements. By adjusting the fitness function and the genetic operators, designers can prioritise specific performance metrics or find trade-offs between conflicting objectives.

Combined with development and use of the Performance-Cost-Ratio functions concepts detailed in Section 3.7.1 as a set of simple, yet very effective means of gauging fitness, the use GAs is then made to be very specific it its fitness evaluation for a given optimisation constraint or constraints.

### 1.2.3 RQ3: How can hypergraph-based modelling and GA-based optimisation techniques be effectively used for resource optimisation in high-performance SoC designs employing 3D NoCs?

The third research question explores the effective use of hypergraph-based modelling and GA-based optimisation techniques for resource optimisation in high-performance SoC designs employing 3D NoCs . Resource optimisation is critical to SoC design, as it directly impacts the system performance, power consumption, and cost.

In the context of 3D NoCs, resource optimisation involves efficiently allocating and utilising hardware resources, such as processing elements, memory modules, communication links, and buffers, to meet the performance and power constraints of the target application. The goal is to find the optimal balance between resource provisioning and system performance while minimising power consumption and area overhead.

Hypergraph-based modelling and GA-based optimisation techniques can be effectively employed for resource optimisation in 3D NoC-based SoCs, offering a comprehensive and automated approach to design space exploration and optimisation. The integration of these techniques

enables designers to systematically analyse the impact of different resource allocation strategies and identify the most suitable NoC configurations for the given application requirements .

The effectiveness of using hypergraph-based modelling and GA-based optimisation for resource optimisation in high-performance SoC designs can be attributed to several factors, most prominent of which are:

- Comprehensive design space exploration: Hypergraph-based modelling allows for a detailed representation of the NoC architecture, capturing the complex interactions and dependencies among the resources. By analysing the hypergraph model, designers can explore various resource allocation strategies and evaluate their impact on system performance and power consumption.
- Automated optimisation process: GA-based optimisation techniques automate the search for optimal resource allocation configurations [21]. GAs can efficiently explore the vast design space and identify the most promising solutions by encoding the resource allocation parameters as chromosomes and defining appropriate fitness functions [22], [23]. This automation reduces the manual effort required for design space exploration and enables the discovery of optimised resource allocation strategies that human designers might overlook [24].
- Multi-objective optimisation: Resource optimisation in SoC designs often involves multiple conflicting objectives, such as maximising performance while minimising power consumption and area. Hypergraph-based modelling and GA-based optimisation techniques can effectively handle multi-objective optimisation problems by incorporating multiple fitness functions or using specialised multi-objective optimisation algorithms, such as NSGA-II [25], [26]. This allows designers to find Pareto-optimal solutions representing the best trade-offs among the different objectives.
- Application-specific optimisation: Hypergraph-based modelling and GA-based optimisation techniques can be tailored to target the application's specific requirements and constraints. By incorporating application-specific knowledge into the modelling and optimisation process, designers can focus on the most relevant performance metrics and resource constraints. This application-specific optimisation leads to more efficient resource utilisation and better system performance than generic optimisation approaches.
- Scalability and adaptability: Hypergraph-based modelling and GA-based optimisation techniques are scalable and can handle the increasing complexity of modern SoC designs. The number and size of NoC components are growing. These techniques can effectively explore the design space and find optimised resource allocation solutions. Moreover, these techniques are adaptable to different NoC architectures, routing algorithms, and application domains, making them suitable for a wide range of high-performance SoC designs.

Using hypergraph-based modelling and GA-based optimisation techniques for resource optimisation in high-performance SoC designs employing 3D NoCs can significantly improve system

performance, power efficiency, and overall resource utilisation as shown by the analysis and use cases in Chapter 3, Chapter 4 and Chapter 5, respectively. By automating the design space exploration and optimisation process, these techniques enable designers to make informed decisions and find the best trade-offs between performance, power, and area, ultimately resulting in more efficient and cost-effective SoC designs [27], [28].

## 1.3 Thesis Contributions

This thesis presents several novel contributions to the field of Networks-on-Chip optimisation for high-performance many-core System-on-Chip architectures. As processor designs advance towards kilo-core scales and beyond, the significance of efficient on-chip communication becomes paramount. The primary contributions are as follows:

1. Development of a novel vendor-independent theoretical framework combining hypergraphs and genetic algorithms. The hypergraph component provides a mathematically elegant and precise modelling approach, enabling straightforward representation of complex multi-point communications and superior expressiveness compared to conventional graph-based models. This is enhanced by an adapted version of the Non-dominated Sorting Genetic Algorithm II (NSGA-II), featuring NoC-specific mutations and fitness functions incorporating bandwidth, latency, throughput and routing algorithm as objectives.
2. Debugging and enhancement of the ROSS cycle-accurate simulator to support next-generation architectures through comprehensive models for three-dimensional Network-on-Chip topologies, at scale. This includes scalable three-dimensional mesh, torus, and hypercube topology models supporting thousands of cores, along with a flexible traffic pattern generator suited to modern high-performance 3D NoC applications.
3. Empirical validation through two compute-intensive use cases demonstrating significant performance improvements: For real-time facial recognition, achieving over 29% reduction in average packet latency and 40% improvement in network throughput; and for double SHA-256 cryptographic operations, achieving over 33% reduction in latency and 40% increase in throughput. These improvements were further validated through hardware implementation using SystemC on a Xilinx Zynq AP SoC, demonstrating over 30% reduction in power consumption and 20% improvement in resource utilisation.

These contributions advance the state-of-the-art in Network-on-Chip design and optimisation, particularly addressing the challenges of scaling to kilo-core architectures. The mathematical robustness of the hypergraph modelling approach, combined with its inherent simplicity and precision, provides a powerful yet accessible framework for describing and optimising complex Network-on-Chip architectures.

## 1.4 Thesis Structure

Aside from the Literature Review (*Chapter 2*), this thesis is organised into three main research chapters, and one conclusion chapter (*Chapter 6*), which address the above research questions and contribute to advancing 3D NoC design and optimisation methodologies.

*Chapter 3* focuses on developing and applying hypergraph-based modelling techniques for 3D NoC architectures. It presents a comprehensive framework for representing 3D NoCs as hypergraphs. It also demonstrates the effectiveness of hypergraph-based analysis in exploring the design space and evaluating the performance characteristics of different NoC topologies. The chapter also introduces novel hypergraph-based metrics and algorithms for assessing the structural properties and communication patterns of 3D NoCs. The insights gained from this chapter lay the foundation for the subsequent optimisation techniques discussed in the following chapters.

*Chapter 4* investigates the use of genetic algorithms for optimising 3D NoC architectures. It presents a GA-based optimisation framework that automates the search for optimal NoC configurations based on the specified performance objectives and constraints. The chapter details the encoding scheme, fitness function design, and genetic operators used in optimisation. It also demonstrates the effectiveness of the proposed GA-based approach in finding optimised NoC topologies and routing algorithms for different application scenarios. The results presented in this chapter highlight the potential of GA-based optimisation in enhancing the performance and efficiency of 3D NoCs.

*Chapter 5* explores hypergraph-based modelling and GA-based optimisation techniques for resource optimisation in high-performance SoC designs employing 3D NoCs. It presents a holistic approach that integrates these techniques into the SoC design flow, enabling designers to make informed decisions regarding resource allocation and optimisation. The chapter demonstrates the effectiveness of the proposed approach in finding optimal resource configurations that maximise performance while minimising power consumption and area overhead. It also presents case studies and experimental results showcasing the benefits of using hypergraph-based modelling and GA-based optimisation for resource optimisation in real-world SoC designs.

*Chapter 5* concludes the thesis, discusses findings and future directions.

This thesis focusses on providing a comprehensive and in-depth analysis of the proposed methodologies, their theoretical foundations, and their practical applications. The chapters are structured to progressively build upon each other, starting from the fundamental concepts of hypergraph-based modelling and GA-based optimisation and culminating in their integration for resource optimisation in high-performance SoC designs.

The research questions posed in this thesis are addressed through theoretical analysis, algorith-

mic development, and experimental validation. The results and insights obtained from each chapter contribute to the overall understanding of the design and optimisation of 3D NoC architectures and their impact on system performance and resource utilisation.

By the end, readers will understand the effectiveness and practicality of using hypergraph-based modelling and GA-based optimisation techniques for 3D NoC design and optimisation. The contributions made in this work have the potential to significantly advance state-of-the-art NoC design methodologies and pave the way for more efficient and high-performance SoC architectures.

## 1.5 Chapter Summary

This chapter has introduced the background and motivation for the 3D NoC design and optimisation research using hypergraph-based modelling and GA-based optimisation techniques. It has highlighted the importance of effective design space exploration and optimisation in high-performance SoC designs employing 3D NoCs.

Three key research questions, that drive the research efforts of this thesis are presented. The first research question focuses on the effective use of hypergraphs as a design and implementation space exploration tool for 3D NoCs. The second research question addresses the justification and use of genetic algorithms for optimising the identified topologies in 3D NoCs. The third research question explores the effective use of hypergraph-based modelling and GA-based optimisation techniques for resource optimisation in high-performance SoC designs.

This chapter also overviews the three main research chapters that form this thesis's core. Each research chapter addresses one or more research questions and contributes to advancing 3D NoC design and optimisation methodologies.

Chapter 2 presented a literature review of existing works in the field, as well as a review of some current application-specific use cases.

Chapter 3 focused on developing and applying hypergraph-based modelling techniques for 3D NoC architectures, laying the foundation for subsequent optimisation techniques.

The research presented in this thesis aims to provide a comprehensive and in-depth analysis of the proposed methodologies, their theoretical foundations, and their practical applications. The insights and contributions made in this work have the potential to significantly advance the state-of-the-art NoC design methodologies and enable the development of more efficient and high-performance SoC architectures. We have shown the effectiveness of using hypergraph modelling Genetic Algorithms optimisations in the proposed framework to identify optimal topologies and configurations for two compute-intensive, yet slightly different use cases - the

Double SHA256 in Bitcoin mining presented in Chapter 4 and Real-time facial recognition in both still and video streams as presentd in Chapter 5

This chapter now sets the stage for the research on 3D NoC design and optimisation using hypergraph-based modelling and GA-based optimisation techniques. It has highlighted the importance and relevance of the research questions addressed in this thesis and provided an overview of the research chapters that contribute to advancing the field.

# Chapter 2

# Literature Review

Expanding on the earlier introduction in Chapter 1, this literature review will address these scientific enquiries as follows:

- Section 2.1 and subsections give an overview of performance concerns in multicore SoCs
- Section 2.2 and subsections explore the current state of research in various areas surrounding NoC architectures, topologies and associated performance and FT characteristic of each topology
- Section 2.3 and subsections will explore the current state of research in load balancing, resource distribution and multithreading concerns as applied to H-P SoCs and the NoC paradigm

## 2.1 Performance Aspects in Systems on Chip (SoC)

Many works have recently emerged addressing design challenges of fast, scalable solutions in multicore systems. These focused on improving execution times in the machine learning domain in particular, with the goal of improving element capabilities by algorithmic, as well as hardware means [29], [30].

The above not withstanding, only a few works focused on the interconnection subsystem as a potential source of performance improvement, with examples including [31], [32], especially given that wrapping many cores together offers excellent parallelism, adequate interconnections, energy efficiency, thermal management, and silicon real estate efficiencies, particularly in area-constrained applications [33], [34].

Research on scalable, power-aware interconnects is helping to manage challenges posed by the increasing number of cores and demanding applications.

In the future, SoCs will have 1024 or more cores. This will substantially increase the number of streams supported by a single device in the next two to four years [7]. It is imperative to consider

a comprehensive range of factors, including individual element capabilities, interconnects, energy efficiency, and load distribution to optimise for high-performance SoC applications featuring 1024 or more cores, with the NoC paradigm emerging as a viable solution for addressing multifaceted challenges [35].

With that in mind, and while the research interest in this domain is active, it tended to focus on narrow perspectives. Furthermore, the impact of silicon node processes on the construction of HP many-core SoCs, in both the physical and logical synthesis senses, while the subject of active research, has not yet been fully exploited [36].

## 2.2 Network on Chip (NoC)

Advances in VLSI technology, in tandem with silicon fabrication processes, have enabled the packing of hundreds, even thousands, of pre-designed IPs in one SoC, fuelled by the billion-transistor drive of this millennium. This was recognised early on by such works as [37], and more recently by [38] and [39].

As the industry increasingly shifted towards higher integration, the push for multi-core and many-core paradigm architectures has similarly intensified, resulting in the integration of hundreds, even thousands of cores on a single die.

This shift created the need for a high-performance, flexible, scalable and designer-friendly interconnection architecture. Traditional external bus interconnects became inadequate for managing the extensive on-chip integration. The solution emerged in a new type of on-chip interconnect, inspired by well-established and understood practices of packet-switched communication networks, leading to the *Network on Chip*, or *NoC* paradigm.

NoCs have many advantages over direct wiring and buses. These advantages include high bandwidth, low latency, low power consumption, scalability, and opportunities to integrate security and cryptography in the fabric of IP and communication.

In a NoC, messages are shuttled through interconnect networks between IP cores and other elements. Consequently, the architecture of on-chip interconnects significantly impacts communication across the chip, which affects the overall chip's performance. Key factors such as end-to-end delay, throughput, and packet loss are critical considerations, on a par with packet switched networks.

These constraints also characterise the different topologies that abound in any packet-switched architectural proposition, which is the concern of subsequent sections of this work. Each topology has its relative merits and limitations [40].

## 2.3 NOC Topologies and Components: an overview

Before delving into topologies and characteristics thereof, it is worth a prelude to the subject with a general view of what makes and constitutes a NoC and its associated components.

NoCs are used to build parallel and multi-core processing platforms, usually on a single chip. It is an architecture for interconnecting multiple cores on a chip, consisting of a set of Routers (R), Links (L), Intellectual Property (IP) cores, and Network Interfaces (NI), also called a Network Adapter (NA).

Figure 2.1 gives an overview of this paradigm.

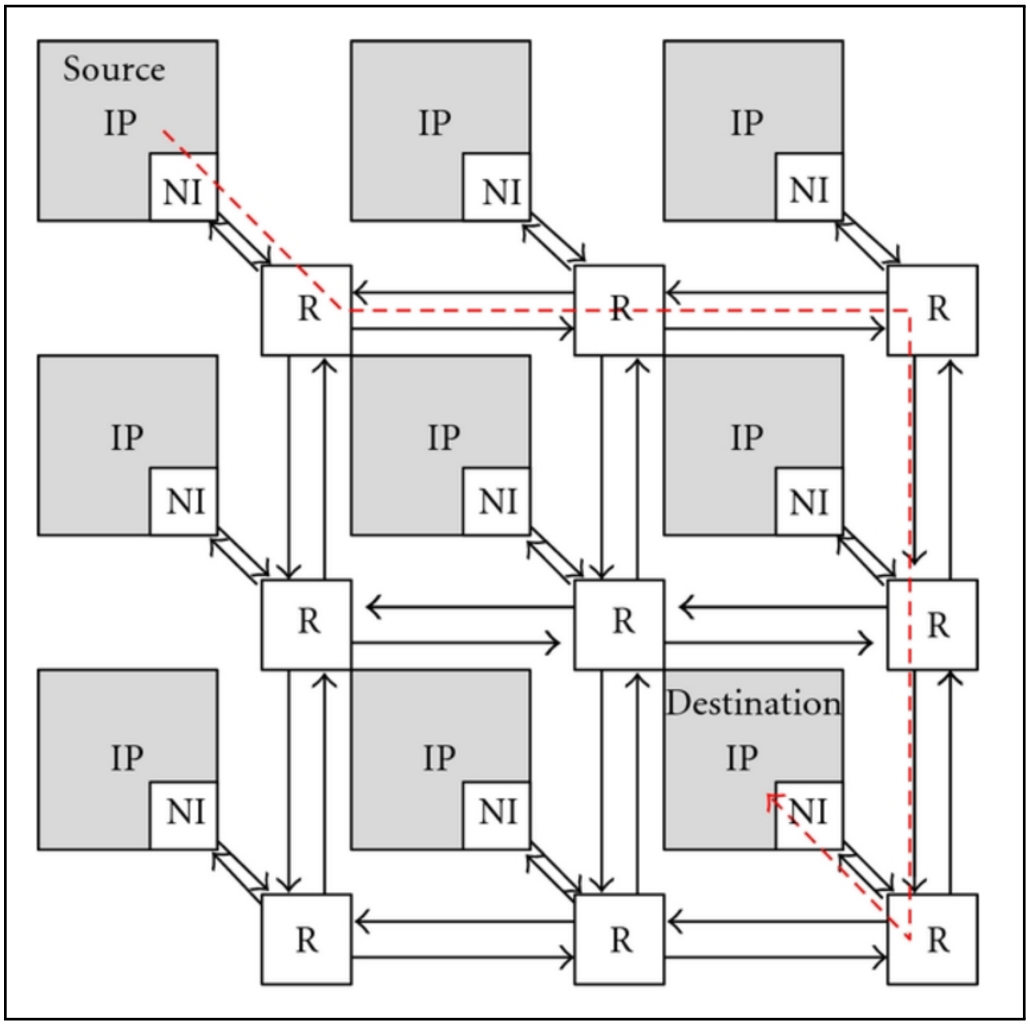



Figure 2.1: NoC Paradigm

Next, each component listed in Figure 2.1 is described.

### 2.3.1 Links

In a NoC context, a link is the physical interconnect between two routers. It comprises one or more logical or physical channels, each comprising a set of wires. Messages passing through the NoC are broken down into fixed-length packets. These packets are then composed as datagrams called *flits* (*flow control digits or units*). Packets are then individually transferred flit-by-flit style. Most of the time, a flit will match a *phit* (*physical transfer unit*) in which a packet is divided and transmitted through the network, or the minimum amount of data that can be transmitted in one link transaction. Synchronisation is achieved with a synchronisation protocol implemented through dedicated wires or mixed-time FIFO or globally asynchronous locally synchronous (GALS) [41], assuming local handshake protocols.

### 2.3.2 Network Interfaces

In a NoC context, a Network Interface (NI) is the interface between a router and the local IP. It transposes the IP's communication perspective to the router's as shown in Figure 2.2 below:

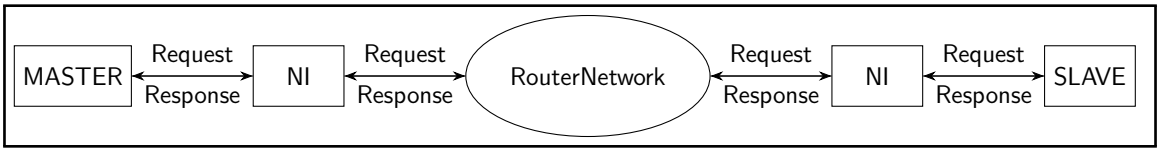



Figure 2.2: Network Interface

This transposition can be viewed as a type of communication service which packages the raw data into the packets at the point of origin for transmission onto the network and unpacks them at the destination [42]. Having interfaces at the edge of routers allows for implementing IP-specific protocols without affecting the router's core functionality. Thus, by implementing NIs as services, cores can be seamlessly integrated within the NoC platform and paradigm. Finally, implementing NI functionality as a service effectively decouples interdependence between IPs and pages to reuse many reusable IPs when implementing SoCs.

### 2.3.3 Routers

Arguably, the router is the most important component in a NoC architecture (Figure 2.3).

In a NoC context, the router switches the appropriate messages arriving at its input ports to its output port(s) at the right time, following a predetermined path or route. A router also supports switching, virtual channels and flow control, some of which will be reviewed next.

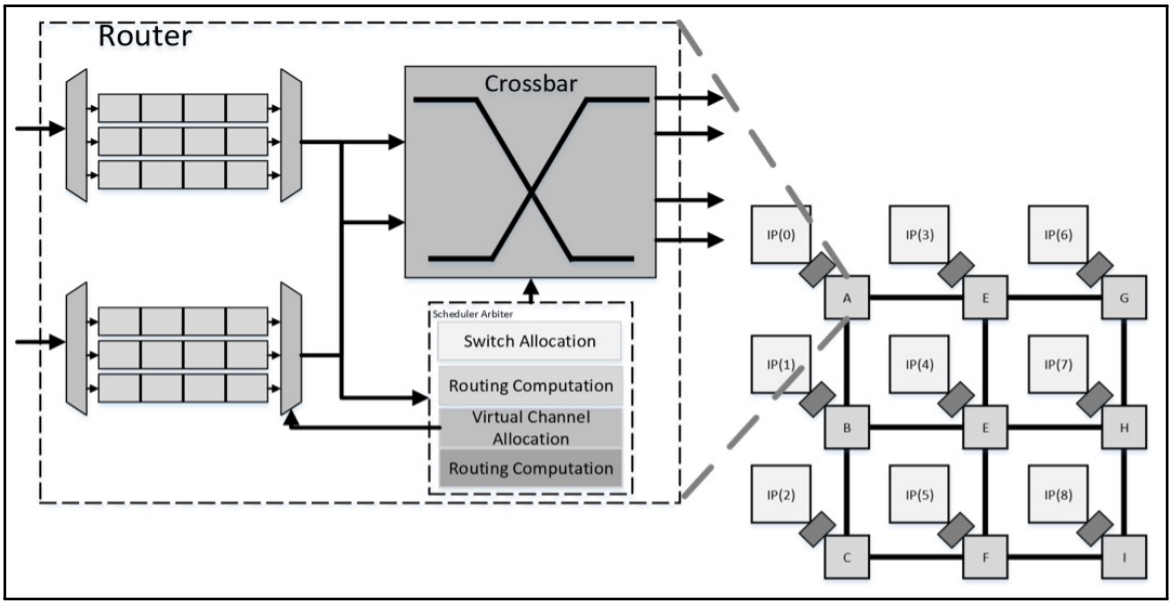



Figure 2.3: Internals of a Router

### 2.3.4 Routing and Timing

Routing determines the path a packet takes from the point of origin to the intended target. The routing algorithm decides which output port and channel(s) to traverse and which next neighbour to forward.

The key aspect of a routing algorithm is timing. It refers to where or when a routing decision is made. This concept is commonly used to classify routing algorithms into centralised, source, and distributed [43].

The router only knows its neighbourhood as packets travel across the network. A header containing only the destination address is used to select output channel/s. Source routing algorithms will predetermine complete routing paths as a header on source nodes before injecting packets into the network. The router switches along a path will be configured accordingly by the header.

### 2.3.5 Switching

Switching is the process of constructing a path for packet propagation. One way to break down the techniques of a switch is shown in Figure 2.4, and discussed next:

**Circuit Switching** constructs an end-to-end path reserved on each intermediate router before the data transition through a routing probe. This reservation is released by the final destination or upon completion of the transmission.

Arbitrary messages can be propagated and transported to the destination without interruption upon the sender receiving an acknowledgement flit from the receiver. A circuit switching

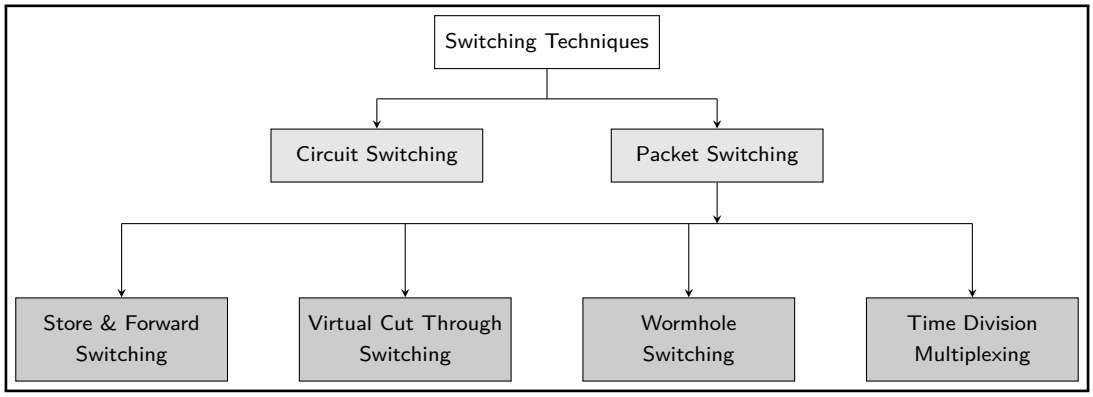



Figure 2.4: Switching Techniques

mechanism with separated data and control plains is developed to reduce the overall latency of the circuit establishment[44].

**Packet switching** on the other hand, does not reserve the entire channel. It is classified into *Store & Forward (SF)*, *Virtual Cut Through (VCT)* and *Wormhole Switching (WH)*. SF is a technique appropriate for the embryonic transfer of short and frequent packets requiring input and output buffering. In other words, routing decisions are made by each intermediate router as long as the entire packet has been buffered. Header flits can be forwarded to the next hop if the routing decision has been made and the available buffer space in the next hop is sufficient.

**VCT packet switching** allows the header flit to cut through and move to the next hop as soon as a routing decision is made, letting the remaining flits follow the same output channel as their predecessors. In VCT, it is not possible to interleave or multiplex packets and packet streams onto the same physical channel. The packet must be stored along a path on the intermediate router(s). They will all behave the same as SF if the next hop is occupied or out of buffer space, which contrasts with WH, which will be discussed next.

In **WH packet switching**, a header is used to build a path for subsequent flits that belong to the same packet. Packets will then be transmitted in a pipeline-type arrangement, whereby the path may span several routers. If the header cannot be processed for some reason, then the wormhole chain will stall, thus occupying space in the flit buffers of each router along the path(s) constructed thus far.

This will then possibly lead to blocking other communications. In extreme conditions, this could lead to chain blocking, resulting in packet(s) being subjected to multi-blocking. This will give rise to problematic timing analysis. WH offers low network latency and buffer costs. However, its congestion level is high and deadlock-prone without special measures such as Virtual Channel.

Last but not least, there is **Time Division Multiplexing (TDM)**, which might be viewed as a switching technique, particularly in settings with high(er) resource utilisation.

In a TDM setting, resources are allocated through a timetable of a fixed number of time slots at each router node. Each slot is reserved for one connection. Tables on all routers are kept in sync with a global TDM schedule. This allows virtual channel (discussed in the next section) reservations to be guaranteed, independently of connections. This does not require arbitration and flow control as a prerequisite to the switching operation , discussed earlier in Section 2.3.5.

A summary comparison of packet switching techniques discussed in this section is given in Table 2.1 below:

| **Switching** | **Communi-cation Entity** | **Path Reserva-tion** | **Buffer Size** | **Resource Utilisation** | **Comments** |
|---|---|---|---|---|---|
| Circuit Switching | Flit | Yes | Small | Low | Requires setup acknowledgement and path tear down phases |
| Store & Forward | Packet | No | Large | High | Header must wait for entire packet before routing |
| Virtual Cut Through | Packet | No | Large | High | Header can be forwarded before tail arrives |
| Wormhole | Flit | Yes | Small | Moderate | Header blocking reduces efficiency |

Table 2.1: Comparison between different packet switching techniques

### 2.3.6 Virtual Channels

A *Virtual Channel (VC) (Figure 2.5*, is a technique used to enhance network performance through the use of shallow buffers instead of a single deep buffer at the input/output ports of routers (discussed in Section 2.3.5). Typical performance gain is in the range of 20% - 50%.

Without VCs, packets are stalled and stored in the local buffer(s) if the destination router's buffer is full or locked. It becomes evident under heavy load, particularly in WH-based switching as discussed earlier. VCs vastly reduce this problem since other unblocked packets in the VC can almost bypass blocked packet(s) towards the next hop router. This technique is a control mechanism to expedite data packets in more generalised transmission protocols such as TCP.

The decision as to which packet(s) get access to physical channels is down to prioritisation following the arbitration policy. Higher-priority packets take precedence over lower-priority ones, which is evident in NoCs applying priority pre-emptive arbitration[45].

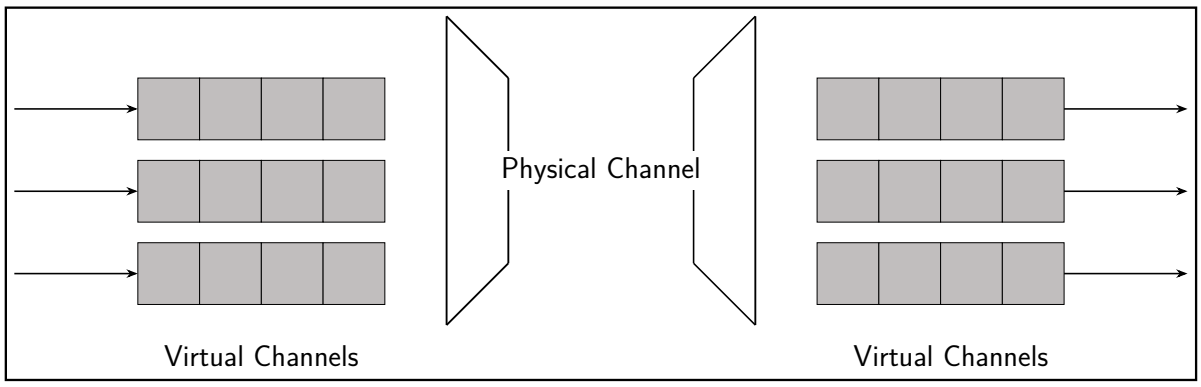



Figure 2.5: Example of a Virtual Channel (VC)

To conclude, higher throughput and switching at the rate of physical channel bandwidth are the main advantages of using VCs[46]. Using VCs in conjunction with wormhole switching also realises several other advantages, e.g. deadlock-free switching, more efficient use of network channels and the ability to implement service levels by using a class of service, while still keeping switching complexity moderate. Indeed, this combination has become increasingly prevalent in NoC architectures.

We will now discuss flow control as the final critical component of a router's functionality.

### 2.3.7 Flow Control

*Flow Control (FC)* allocates network resources to packets traversing the NoC. Examples of such resources are buffer space, control state and channel bandwidth[47]. Flow Control can be decomposed as in Figure 2.6

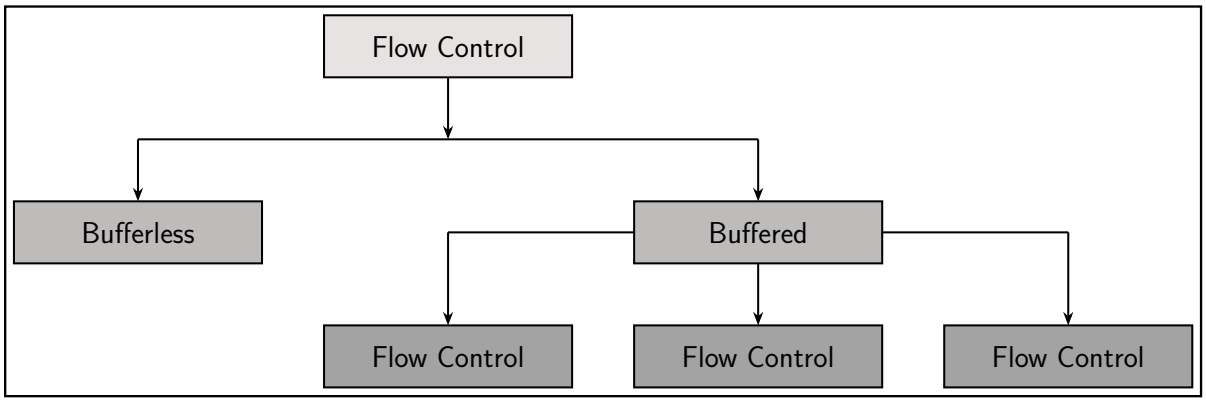



Figure 2.6: Flow Control Classifications [48]

FC can be broadly classified as buffered or bufferless. Bufferless flow control is mainly used in circuit-switched network applications. It provides a mechanism for dedicated end-to-end transmission path construction. In contrast, buffered flow control focuses on packet-switched networks.

In this context, basic handshaking involves using a validation signal to establish communication. These are sent during flits (ARQ) transmission and returned as an acknowledgement upon successful receipt and validation of the same (ACK). This mechanism's main attraction is the low-cost implementation. This comes at the expense of low link utilisation, leading to inefficiencies. [49] gives good examples of using handshaking as a method of flow control in the context of NOCs. There are also good examples of using a handshake as a means of flow control, generally in [50] and [51].

When implementing ACK/NACK, flits are copied from the source router's buffer and sent to the next hop router until an ACK signal is received back from the destination router. If instead a NACK is received, the flit is retransmitted.

Credit-based flow control involves the destination router keeping a counter of the available buffer space in the source router – the credit counter. This counter is decremented as flits are transmitted by the sender and incremented if the receiver accepts the flit. This guarantees packet integrity. Examples of NoCs which implement ACK/NACK as a flow control include SPIN and QNoC.

### 2.3.8 IP cores

We have so far discussed the layer-1 and layer-2 aspects of NoCs, and in doing so, have eluded that the NoC exists to connect the higher-level functions of other components tasked with performing the objective of the SoC. These components are IP cores, and mapping them onto the NoC itself achieves the all-important design step of integrating the IP core functionality and the NoC [52].

In theory and practice, IP cores can be any synthesisable entity, be they CPU cores, DSPS, memory blocks, video processors, or very complex subsystems. A single is often portioned into functional blocks or tiles interconnected by a dedicated or shared NoC. This paradigm has existed since the early propositions of NoCs, as noted in [53]. It is also prolific and prevalent in today's SoC applications, specifically in commodity graphics and GPU applications [54]. Figure 2.7 shows an NVIDIA Tegra K1 SoC, partitioned in this fashion:

## 2.4 Topology Considerations in NOC Design

As a design principle, a NoC-based system typically starts with a study of the design space. The design's principle objective is to arrive at an optimal topology. Parameters such as network topology, routing, and switching strategies influencing NoC's performance are also studied.

Figure 2.8 below gives an overview of some popular topologies, which we will consider in turn:

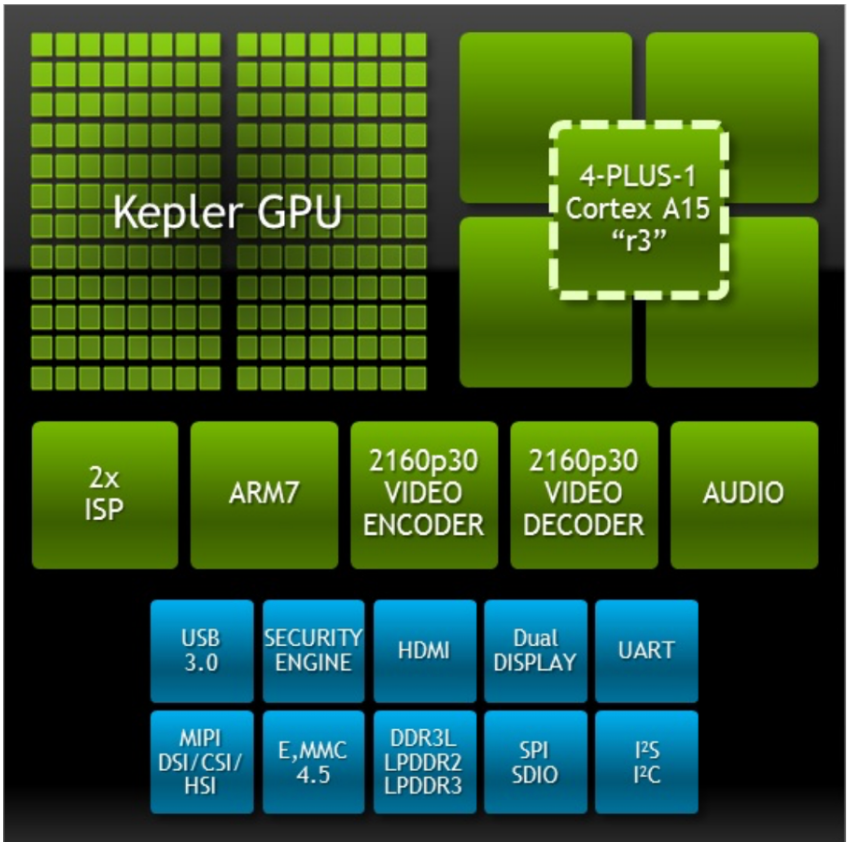



Figure 2.7: NVIDIA Tegra K1 SoC with an Interconnecting NoC

It is also worth mentioning that any discussion on NoC topologies will invariably fall into two broad categories: flat and hierarchical or 2D and 3D.

In this literature review, we have elected to tackle the subject of the evolution of hierarchical or 3D topologies rather than considering the two in isolation. By definition, these topologies are an evolution of 2D or flat architectures.

In this spirit, we shall examine standard NoC topologies, from the classic to the more recent and emerging topologies in NoC design. We will then summarise each topology's characteristics and provide an overview of its relative merits.

We will consider 11 topologies, depicted in Figure 2.8. Each topology is considered in its specific section. We will then discuss the evolution of 3D topologies by looking at some common examples.

### 2.4.1 Mesh Topology

In a mesh topology (Figure 2.8$a$), nodes are connected to form a grid. One of the advantages of this topology is easy expandability: this is a simple matter of adding more nodes to the existing architecture. Another advantage is the existence of multiple paths amongst any pair of nodes, making it tolerant to link failure to an extent.

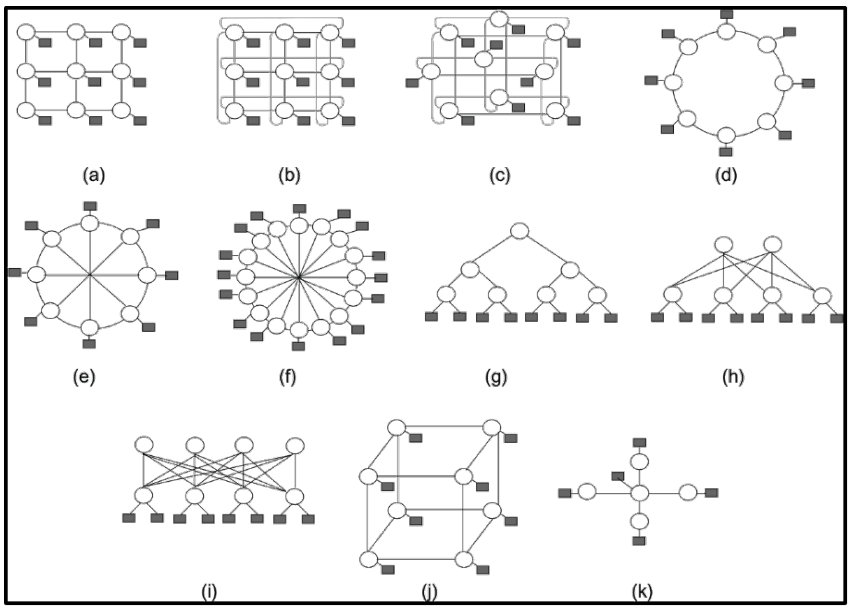



Figure 2.8: Common NOC Topologies

Its disadvantages include irregularity (corner nodes have a degree of 2, edge nodes a degree of 3 and inner nodes a degree of 4, respectively). This leads to non-uniform fault-tolerant patterns and non-uniform bandwidth characteristics: corner nodes have the least degrees (interconnects) and hence the least bandwidth, whereas inner nodes have the most.

Another disadvantage of the mesh topology is that it can grow quite large in diameter, leading to less efficient silicon area utilisation.

### 2.4.2 (Folding) Torus Topology

Torus (Figure 2.8$b$) is a mesh topology derivative, obtained by augmenting the mesh direct connections for each pair of end nodes, in each row and column, respectively. A 9-node torus is shown in Figure 8b. Compared to a mesh layout, this topology exhibits a reduced diameter. Amongst its disadvantages is the length of paths. This can be overcome with a Folding Torus arrangement, at the cost of doubling the wire length.

### 2.4.3 Ring Topology

In a ring architecture, each node is interconnected in a ring fashion (Figure 2.8$d$). Every node has two neighbours, irrespective of the diameter of the ring. Ring architecture exhibits a desirable small degree, but its diameter is linearly proportional to the number of interconnected nodes.

One advantage of the ring architecture is the ease with which link faults can be located, which simplifies troubleshooting. Compared to other architectures, expanding a ring with additional

nodes is relatively easy. Ring topology limitations include the disruption necessary to insert or delete nodes in the ring , and even a single fault in the ring can disrupt the entire network.

### 2.4.4 Octagon Topology

In an Octagon architecture (Figure 2.8$e$), each node requires two hops to form a communication path with any other. At its simplest, the Octagon model is made up of eight nodes with twelve bi-directional links. We refer to its simplest form as this topology can infinitely expand with more nodes and links while the basic architecture remains the same.

It's trivial to notice that this topology expands the ring architecture discussed in 2.4.3 above. We observed that nodes are arranged in a ring, with the addition of a third link to the central connection point, in addition to left and right neighbours.

This topology has some distinct advantages. For example, there is only a maximum of two hops between any two nodes, creating shorter, faster pathways between nodes, compared to the ring architecture discussed in 2.4.3 above. Another advantage is high throughput due to the added number of non-overlapping communication pathways created. Finally, this type of architecture naturally lends itself to shortest-path and shortest-path-first routing when combined with interconnect scheduling design techniques [55].

### 2.4.5 Spidergon (Spider) Topology

The Spidergon architecture in Figure 2.8$f$ was proposed by ST Microelectronics as a topology for Systems on Chip.

It comprises an even number of nodes $N$ and is similar to a Ring, augmented by cross-links between opposing nodes. It can be that the Spider is an extension of the Octagon topology discussed above.

This architecture exhibits unique topological characteristics. Firstly, the network is regular, vertex symmetrical (identical topology at each node), and edge transitivity (any two edges in the network are automorphic).

This topology is also interesting because the routing algorithms like Across First (aFirst), Across Last (aLast) and Across Equalised (aEqualised) have evolved out of this proposal.

### 2.4.6 Binary Tree (BT) Topology

In a Binary Tree topology(Figure 2.8$g$), nodes are deployed as an inverted tree, originating at a root node. Each node has a set of coordinates consisting of its level and position. The level denotes the vertical level in the tree, and the position is the actual horizontal placement,

customarily ordered from left to right. Each non-leaf node is linked to a pair of nodes in the level directly below. Leaf nodes, also known as the resource nodes, are placed at the bottom of the tree.

Some advantages of this topology include high performance in terms of latency and throughput, and the traversal of BTs is very well understood, with abundant methodologies for doing so, leading to robust switching and routing. Its drawbacks include relative complexity of the structure, maintaining tree balance, and a linear increase in traversal complexity relative to size $O(n)$, where n is the number of nodes in the tree.

### 2.4.7 Butterfly Fat Tree (BFT) Topology

Butterfly Fat Tree or BFT(Figure 2.8$h$) is derivative of the BT topology discussed in Section 2.4.6.

In this topology, the network is modelled as a tree, with butterfly-like interconnections between levels. Nodes have the same co-ordinates of reference as in a BT. Leaf (or resource) nodes are placed at the bottom of the tree, such that every two pairs of resource nodes (4 resource nodes) are linked to a non-leaf node, and each non-leaf node is either linked to four non-leaf nodes at the level directly above it, or four leaf nodes as explained earlier.

Added advantages of this layout include increased bandwidth and reduced latency due to additional links and pathways. The opportunities for localised switching at neighbouring nodes eliminate the need for global synchronisation. Amongst its disadvantages is the increase in silicon area required for interconnects and additional pathways needed for these interconnects. This only amounts to an insignificant increase versus the overall complexity of SoCs. Although on the older side, [56] also gives good foundational work and proposal for such a switch-based architecture. [57] and [58] also provide an excellent discussion of BFTs and other topologies.

### 2.4.8 SPIN Topology

SPIN (Figure 2.8$i$) stands for Scalable, Programmable, Integrated Network. It is an implementation of a BFT design discussed in Section 2.4.7. The illustration Figure 2.8 shows a 4-ary SPIN. We can observe that there are as many non-leaf (router or switch) nodes as there are leaf.

This topology's advantages include forming a non-blocking packet switching paradigm, in which performance scales well while keeping size reasonable. For a tree of $n$ leaves, there is a total $\frac{3n}{4}$ of switches; the number of required parent ports is the same as the number of child ports for every switch node in the tree, and every level of switch has the same number of switches. Growth of the network is of the order of $\frac{nlogn}{8}$.

Disadvantages include higher on-chip space consumption and power efficiency trade-offs for higher throughput compared to BFT and BT architectures.

### 2.4.9 Hypercube (Hyper) Topology

Of all topologies discussed so far, the Hypercube (Figure 2.8$j$), or simply Hyper, is a 3D or hierarchical NOC implementation, which is a prelude to discussing the evolution of the hierarchical NOC from flat 2D implementations to 3D implantation. This has been and continues to be, a drive instigated by the demand for faster processors and more cores over the same or smaller area ratios.

An n-dimensional Hypercube, $Q(n)$, is a 3D graph structure obtained using interconnecting 2n nodes with n2(n-1) edges. One way of addressing nodes in a Hyper is to address every node with an n-bit binary number b, such that any two given nodes, u and v, are joined if and only if u and v differ by exactly one bit; thus, $Qn$ is a regular graph since each node (vertex) has the same number of neighbours. Following this, it is easily observed that hyperarchitectures form simple recursive structures and are therefore highly scalable. [59] provides an excellent discussion of a high-performance routing algorithm proposal for Hyper, targeted at DSP and Image Processing applications.

One major limitation of a Hypertopology is the network size restriction, due to the degree limitation, with some proposed solutions around derivations of folded hypercubes, crossed cubes and hierarchical cubes, amongst others.

### 2.4.10 Star Topology

The Star topology (Figure 2.8$j$) is the last of the selected 11 architectures for this review.

In this architecture, $n-1$ nodes in a network of $n\ nodes$ interconnect to a central node. The centre node has a degree of $n-1$, whilst every other node has a degree of $1$.

Some of the $Star$ interesting features, and perhaps strengths, is that the diameter of a star is a constant $2$, regardless of size, and therefore a small average pathway distance is exhibited. Inherent redundancy is also exhibited in that any single node failure does not impact the rest of the network, with the important exception being the centre node.

Limitations of this topology include the single point of failure due to complete dependency on the central node for interconnect. The bandwidth and latency limitation comes from a central interconnection point being a bottleneck.

## 2.5 Topology Evolution: 2D to 3D

As mentioned earlier, demand for faster processors and multiple cores over the same or smaller area ratios quickly led to the boundaries and limitations of 2D NoC topologies being reached, so it was natural that researchers turned their attention to 3D topologies.

Very active research and development further enabled these efforts' derive in the 3D IC domain, primarily driven by Through Silicon Via (TSV) advancements [60].

Indeed, the drive for both energy efficiency, coupled with the need for higher integration, was very much challenged by the need to develop new MOSFET transistors, as planar structures became a blocker at around the 28nm node process [61], as will be discussed shortly. Thus, the move from planar MOSFET designs to 3D structures ensued, is summarised in Figure 2.9:

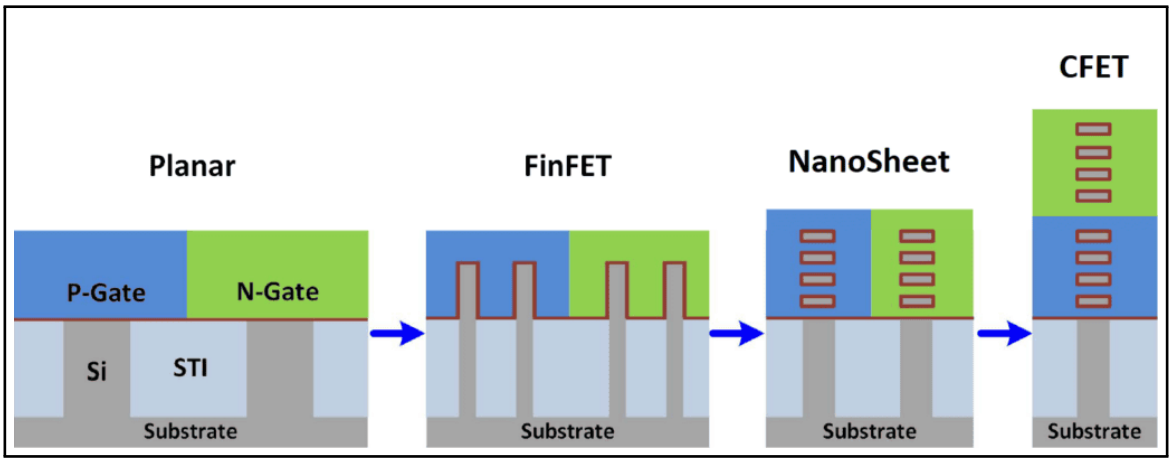



Figure 2.9: Evolution from Planer (2D) MOSFET to 3D CFET

The characteristics of each iteration of MOSFET architecture gives good insight as to the evolutionary pathways that 3D NoCs and SoCs have taken. It represents a remarkable journey in semiconductor technology.

As mentioned above, planar MOSFETs, characterised by their flat, two-dimensional structure with the gate positioned atop a single conducting channel, and Shallow Trench Isolation (STI) used to isolate adjacent transistors, dominated semiconductor manufacturing until the early 2000s [62]. However, as device scaling approached sub-28nm nodes and beyond, planar designs encountered severe short-channel effects and increasing power leakage [63].

This led to the development of FinFET technology, where the conducting channel is raised into a three-dimensional 'fin' structure, with the gate wrapping around three sides, significantly improving electrostatic control [64].

As manufacturing processes advanced towards 7nm nodes, NanoSheet (NS) transistors emerged, featuring multiple stacked silicon channels completely surrounded by the gate material, offering superior electrical characteristics and better scaling potential [65].

The latest advancement in this progression is the Complementary FET (CFET), which vertically stacks n-type and p-type transistors, enabling unprecedented device density and performance while maintaining compatibility with existing manufacturing processes [66].

This architectural evolution from planar to CFET represents a fundamental shift from two-dimensional to three-dimensional design paradigms, significantly improving power efficiency and computing density in modern semiconductor devices [67].

All of this resulted in an equally vibrant research effort in newly developed 3D NoC topologies, which the NoC, both as a standalone structures as an end, as well as core components of the SoCs equally remarkably evolved to utilise this advancement in both design and engineering processes (Figure 2.10)

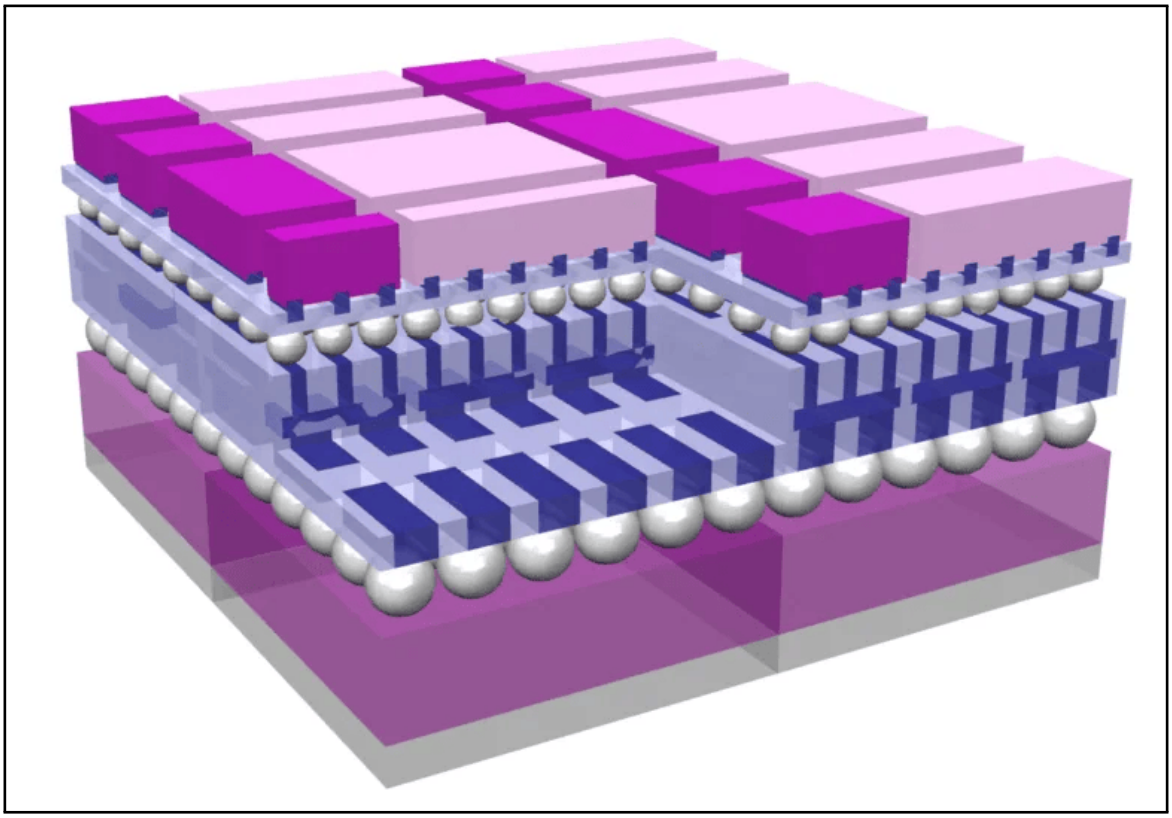


Figure 2.10: Representation of a 3D NoC in Silicon

It is the subject of our discussion in this section and subsections, examining the popular Hypercube and Hyper-Mesh topologies.

### 2.5.1 Mesh and Cube in 3D: 3D Hypercube and 3D Hyper Mesh

We introduced the Hypercube architecture in Section 2.4.9 earlier and touched upon some of its features and characteristics. In keeping with the conventions set out thus far, we will also model this topology using graph notation.

With reference to Figure 2.8, the following propositions are made, guided by Graph Theory, to arrive at the Hypercube representation in Figure Figure 2.11:

1. Let $G(V, E)$ be an undirected graph, with V and E representing the vertex and edge sets of G, respectively.
2. Let hypercube $Qn$ be an undirected graph, where $n$ represents the dimension of the hypercube and a set of vertices labelled $0$ to $2n\ \ 1$ and a set of $n2n-1$ edges, such that there is an edge between any two vertices if and only if their labels differ by exactly one positional bit.

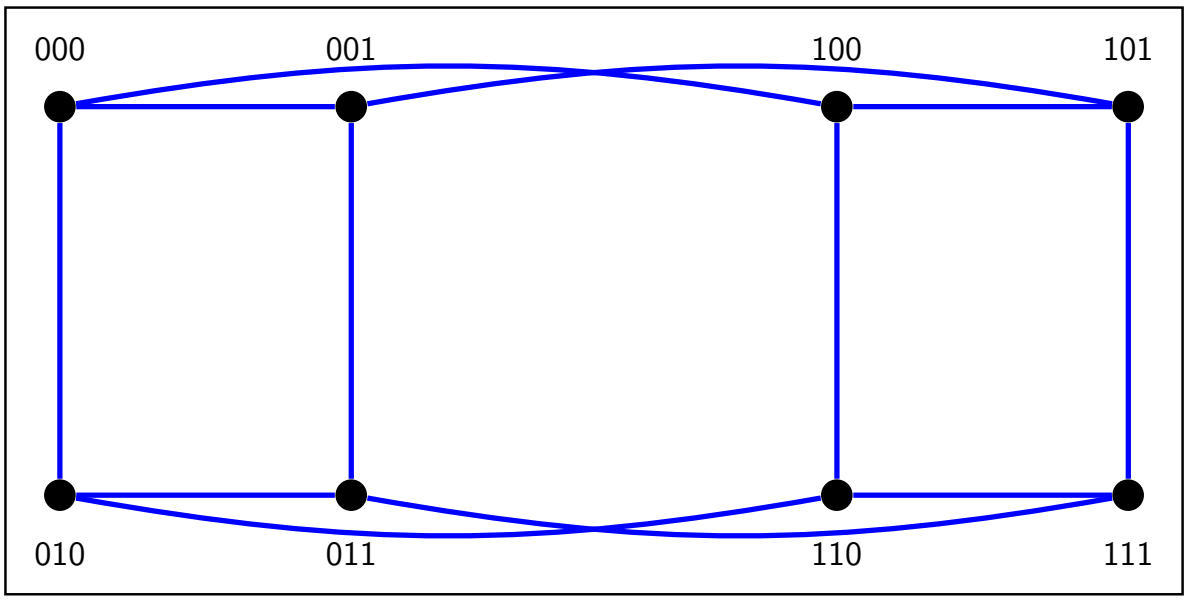



Figure 2.11: Hypercube (n=3)

1. Let $M(r, c)$ be a 2D mesh topology, consisting of $r$ rows and $c$ columns, with a total of $r * c$ vertices.
2. Let vertices be labelled $(x, y)$, where $1 \leq x \leq r$ and $1 \leq y \leq c$.
3. Each interior vertex has exactly 4 neighbours. The degree of $M(r, c)$ is $4$, diameter is $r + c - 2$.

Similarly, we define a mesh with the following characteristics to arrive at the mesh in 2.12:

1. Let $M(r, c)$ be a 2D mesh topology, consisting of $r$ rows and $c$ columns, with a total of $r * c$ vertices.
2. Let vertices be labelled $(x, y)$, where $1 \leq x \leq r$ and $1 \leq y \leq c$.
3. Each interior vertex has exactly 4 neighbours. The degree of $M(r, c)$ is $4$, diameter is $r + c - 2$.
4. There exist at least 2 node-disjoint paths between any two nodes.
5. WLOG assume $r = c = k\ where\ k \in \mathbb{N}\ and\ k\ is\ even.$

Containing the graph theory theme, the cross product of two graphs, $\otimes$, is a handy and

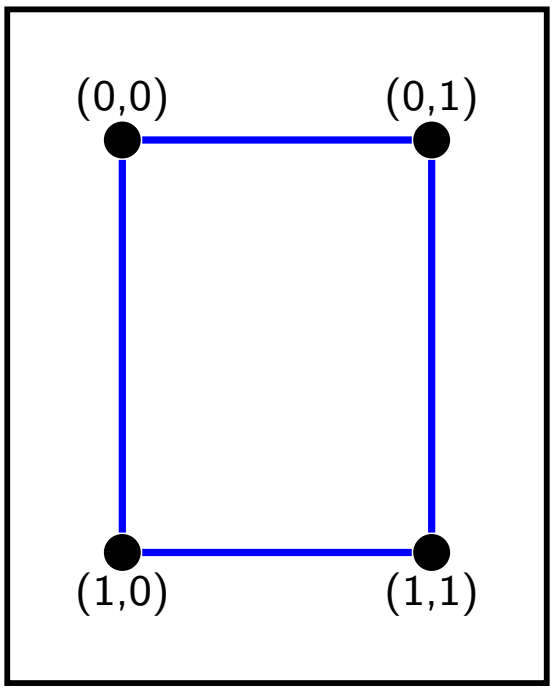



Figure 2.12: $M(k = 2)$

powerful tool for deriving new variations of common networks. A full discussion of graph theory is beyond the scope of this work.

WLOG, the cross product of the Mesh and Hypercube structures introduced thus far will yield a Hyper Mesh architecture, depicted in Figure 2.13 below, such that $G(V, E) \otimes M(k) = QM(n, k)$, will satisfy the following conditions:

1. Size: $QM(n, k)$ has $2^n k^2$ nodes.
2. Degree: $QM(n, k)$ is of degree $n + 4$
3. Diameter: let n be the dimension of a hypercube $QM(n, k)$ of size $k^2$, then the diameter is $n + k - 2$.
4. Bisection width: let n be the dimension of a hypercube $QM(n, k)$ of size $k^2$, then the bisection width is $2^n - 1k^2$.
5. Connectivity: let u and v be two nodes in $QM(n, r, c)$, then the number of node-disjoint paths is the minimum of (neighbours of node $u$, neighbours of node $v$). It is at most n+2k-2 in length.
6. Cost: let n be the dimension of a hypercube $QM(n, k)$ of size $k^2$, then the cost (number of links) of the structure is $2^n + 1(k^2 - k) + nk^2 2^{n-1}$.

Having completed our survey of NoC topologies, we now focus on load balancing, resource distribution multithreading and optimisation techniques. After summarising what we have discussed, this is the subject of Section 2.3 and subsections thereof.

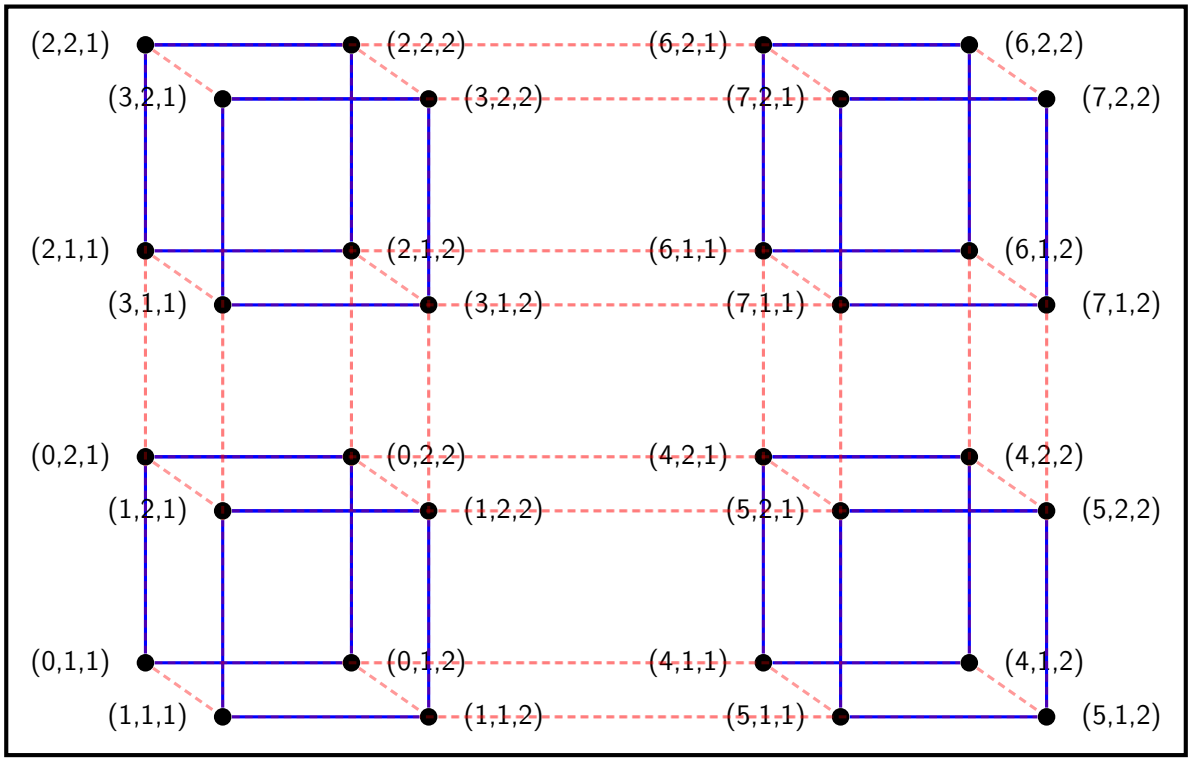



Figure 2.13: Hyppermesh $QM(3,2)$

### 2.5.2 NOC Architecture Summary

Having discussed topologies and components in the previous sections, we shall now look at a comparison between the characteristics of the various architectures considered thus far, utilising undirected graphs as a model, with the following observations:

- $n$: Number of vertices
- $cost$: Maximum number of edges for a given topology, of size $n$ number of vertices
- $k$: for a mesh $M$ of $(r, c)$ rows and columns, we assume that $r = c = k\ w.l.o.g$
- degree $d$: the number of links for a given vertex
- network diameter $(ND)$: maximum shortest path length between any pair of nodes in a topology
- performance: this is a general performance indicator taking into account throughput and average network latency

| Topology | Max Degree | Diameter | Cost | Performance |
|---|---|---|---|---|
| Mesh | 4 | $2k-2$ | $2n(n-1)$ | Average |
| Folding Torus | 4 | $(n-2)\lfloor k/2 \rfloor + \max(2n-4,\ \lfloor k/2 \rfloor)$ $(n-2)\lfloor k/2 \rfloor + \max(2n-4,\ \lfloor k/2 \rfloor)$ | Variable (out of scope) | Good |

Continued on next page

**Table 2.2 – continued from previous page**

| Topology | Max Degree | Diameter | Cost | Performance |
|---|---|---|---|---|
| Ring | 2 | $n/2$ | $2n$ | Good |
| Octagon | 3 | 2 | 12 | Very Good |
| Spidergon | $n/2$ | $2n+4$ | $3n/2$ | Excellent |
| BT | $2^L$ | $\mathcal{O}(\log n)$ | TBD | Very Good |
| BFT | $2^L$ | $\forall L = \log_4 n, ND = L$ | TBD | Very Good |
| SPIN | $n$ | Irregular $(N \log N)/8$ | Irregular | Very good |
| Heube | $n$ | $n$ | $n2^{n-1}$ | Excellent |
| Star | $n$ | 1 | $n$ | Poor |

Table 2.2: Comparison of Network-on-Chip Topologies

Thus far, we have addressed the different architectures, components and relative merits of various NoC topologies with Graph Theory tools. Any discussion around NoC performance, particularly in multi-core SoC settings, inevitably includes load balancing, resource distribution, routing and multi-threading. This will be our focus in Section 2.6 and further sections of this literature review.

## 2.6 On Load balancing, Resource Allocation and Multi-threading in NOC Designs

Whilst a dissection of operating systems theory and practices is not in the scope of this research, it is trivially known that allocating a large number of objects in multi-threaded settings on multicore problems will inevitably lead to a degradation in performance, contrary to the expected twice-fold increase. A basic understanding of resource management and operating systems tells us that the reason behind this is embedded deeply in software-based resource allocation mechanisms, such as the well-known glibc allocator in *NIX environments [68]

To this extent, the problem of resource allocation in a NoC context is compounded and unique. Addressing resource allocation in multi-core applications is complex; scaling this to meet 1K+ cores is even more complex. If one is mindful that a typical OS, operating on commercially available server-type hardware, is at most dealing with a few tens of cores, given the state of commercially available CPU technology.

### 2.6.1 Load Balancing and Routing

Literature on load balancing in NoC designs usually tackles the matter as a routing problem. Works on the subject include [69]. Their novel routing policy design is ACO-based and involves regional routing design inspired by entomology research, specifically observing live ants. Others, such as [70], considered global network load to devise a centralised, adaptive routing paradigm. A type of hybrid approach was presented in [71], with their proposed route aggregation using a super-router node, whereby nodes with dense traffic and long link distances are grouped together as one routing node, with a routing protocol for the super-router to improve overall performance. It is worth noting that although a good body deals with routing and load balancing, approaches focus on load distribution as a routing question rather than an overall resource allocation problem. Moreover, not much work seems to have gone into load-balanced routing in an FPGA context; one such literature is [72], which, although superficial in its approach, still manages to cover the subject of routing and load balancing in an FPGA context.

### 2.6.2 Resource Allocation

Resource allocation plays a crucial role in optimising the performance, power consumption, and cost of Networks-on-Chip (NoCs).

Resource allocation, in a NoC context, refers to the process of assigning and managing on-chip resources, such as processing elements, memory modules, and communication channels, to different applications or tasks running on the NoC.

Effective resource allocation strategies ensure that on-chip resources, such as processing elements, memory modules, and communication channels, are efficiently utilised to meet the demands of applications while adhering to constraints.

In other words, the goal of resource allocation is to optimise the utilisation of available resources and ensure efficient execution of applications while meeting performance requirements and constraints.

The problem is more pronounced and indeed critical in general purpose NoC. It was recognised in works as early as [73]).

Another good good example is [74]. Another more thorough example is [75], although no attempt is made to address resource allocation in an FPGA context.

Another interesting treatment can be found in [76], whereby the efficiency problem is considered a fragmentation question. A possible answer is proposed in their DeFrag algorithm.

Further treatment of resource allocation as a communication problem is found in this interesting and rather novel approach of bringing cycle-accurate SystemC modelling as an instrument of

choice [77]; a set of load balancing, broadcast and scatter primitives are proposed and put to good use.

We will now survey the different strategies used in resource allocation as applied to NoC.

#### 2.6.2.1 Application Mapping

As stated earlier, one of the core questions at the heart of NoC applications in SoC contexts is the allocation of processing and memory resources.

This invariably leads to the need to examine application behaviour and traffic generation patterns, and this is where application mapping becomes relevant.

Application mapping involves assigning application tasks to specific processing elements in the NoC. The goal is to minimise communication overhead and maximise resource utilisation.

Various mapping techniques exist, each with its own advantages and disadvantages, the most common of which are discussed below, with their relative merits and shortcomings:

- **Clustering-based mapping:** as a strategy centring around the communication behaviour, applications with a high level of communication demands are grouped and mapped to nearby processing elements, to reduce communication distances and improve performance [34, p. 102685]. This approach is particularly effective for applications with localised communication patterns. However, it can be challenging to determine the optimal clustering strategy, and poor clustering can lead to increased congestion and contention
- **Priority-based mapping:** Tasks are assigned priorities based on their criticality or performance requirements, and higher-priority tasks are allocated to processing elements with better resources or connectivity [74, p. 4]. This strategy ensures that critical tasks receive adequate resources, but it may lead to unfairness and starvation for lower-priority tasks.
- **Topology-aware mapping:** Mapping algorithms consider the NoC topology and its characteristics to make informed allocation decisions. For instance, tasks with high bandwidth requirements may be placed closer to the network's centre to minimise contention [10, p. 109]. This approach can improve overall system performance, but it requires a detailed understanding of the NoC topology and its impact on application performance.

#### 2.6.2.2 Communication Scheduling

This aspect of resource mapping is more focussed on and concerned with the timing and scheduling of communication resources. It determines the order and timing of data transfers between processing elements.

Efficient scheduling algorithms minimise contention, prevent deadlocks, and ensure timely delivery of data. Common scheduling techniques include:

- **Time-division multiple access (TDMA):** time is divided into slots, and each communication channel is assigned specific slots for data transmission. This approach ensures fairness and avoids collisions but may lead to under utilisation of resources if traffic patterns are not uniform.
- **Round-robin scheduling:** Channels are given access to the network in a cyclic manner, ensuring fairness and simplicity. However, it may not be optimal for applications with varying communication demands.
- **Priority-based scheduling:** Communication requests are prioritised based on their urgency or criticality. This approach ensures that high-priority data is transmitted promptly, but it may lead to starvation for lower-priority requests.

#### 2.6.2.3 Quality-of-Service (QoS) Management

QoS management aims to guarantee specific performance levels for different applications or traffic flows in the NoC.

This involves allocating resources and prioritising communication based on QoS requirements. Techniques for QoS management include:

- **Traffic shaping:** a technique for regulating the rate of data transmission, to prevent congestion and ensure fairness among different traffic flows. Can improve overall network performance but may require complex traffic monitoring and control mechanisms.
- **Virtual channels:** involves the creation of multiple virtual channels within a physical channel, to isolate and prioritise different traffic flows, with varying QoS requirements [78]. Virtual channels can improve performance and isolation, but may increase resource overhead and complexity.
- **Adaptive routing:** involves dynamic adjustment of routing paths, based on network conditions and QoS requirements, to minimise latency and avoid congestion [79]. Adaptive routing can improve performance and fault tolerance, but may require sophisticated routing algorithms and real-time monitoring of network conditions.

Table 2.3 below summarises aspects discussed above:

| Strategy | Description | Advantages | Disadvantages |
|---|---|---|---|
| Application Mapping | Assigning tasks to processing elements | Minimises communication overhead<br>Maximises resource utilisation | Can be challenging to determine optimal mapping<br>May lead to uneven resource utilisation |

Continued on next page

**Table 2.3 – continued from previous page**

| Strategy | Description | Advantages | Disadvantages |
|---|---|---|---|
| Communication Scheduling | Determining order and timing of data transfers | Minimises contention<br>Prevents deadlocks<br>Ensures timely data delivery | May lead to underutilisation or starvation<br>Can be complex to implement and manage |
| QoS Management | Guaranteeing performance levels for different traffic flows | Improves overall network performance<br>Ensures fairness and isolation<br>Provides fault tolerance | May require complex mechanisms<br>Can increase resource overhead |

Table 2.3: Summary of Resource Allocation Strategies in NoCs

#### 2.6.2.4 Emerging Trends in Resource Allocation

As stated earlier in Section 2.6, and subsequent sections, resource allocation plays a critical role in the design and application of NoC. This is even more pronounced and critical in the many-core state of today's high-performance SoC applications and VLSI ASICs, where kilo-core deployments are increasingly common place [80], and mega-core is only a matter of time [81].

In that sense, the need for ever more creative techniques of resource management and allocation is driving the current trend in future development. Noteworthy directions are:

- **Machine Learning:** ML techniques are being used to predict traffic patterns and dynamically adjust resource allocation to optimise NoC performance. This approach can adapt to changing workloads and improve resource utilisation. The major drawback in using ML is the need for extensive training data, and complex models.
- **Application-specific resource allocation:** Recognising that different applications have unique communication patterns and performance requirements, there is a growing trend towards tailoring resource allocation strategies to specific application domains [12]. This approach can optimise performance for specific applications but may require specialized knowledge and design effort.
- **3D NoC resource allocation:** As stated earlier, the natural move to and evolution of the 3D pradigm in NoC applications offered both new opportunities , and posed new challenges for resource allocation. This has lead to more and more high performance SoC and ASIC

design application of the 3D NoC model to address questions of density, performance and energy efficiency, in dense node processing at 5nm and below [82]. Efficiently managing vertical links and balancing resource utilisation across multiple layers are crucial for optimising 3D NoC performance.

## 2.7 Optimisation Techniques and Trends for NoC

The design and optimisation of Networks-on-Chip (NoCs) is a multifaceted challenge with significant implications for system performance, power consumption, and cost. As the number of cores on a chip increases and the communication demands of applications become more intricate, efficient NoC design becomes paramount. Various optimisation techniques have been proposed to address these challenges, each with its own strengths and weaknesses. This section surveys some prominent NoC optimisation techniques, including machine learning, genetic algorithms, linear programming, simulated annealing, and game theory, and discusses current trends in the field. It concludes with a discussion of why genetic algorithms are particularly well-suited for optimising NoCs modelled as hypergraphs.

### 2.7.1 Machine Learning for NoC Optimisation

ML techniques have emerged as a promising avenue for NoC optimisation, in very much the same way they are being applied to address resource allocation questions. This is due to their ability to discern complex patterns and relationships in data. These techniques can be employed to predict NoC performance metrics such as latency and throughput, based on architectural parameters and application characteristics. This predictive capability enables rapid design space exploration and optimisation without necessitating time-consuming simulations.

One example of ML application in NoC optimisation, is the use of supervised learning techniques to predict NoC performance and power consumption. This approach involves training a model on a dataset of NoC configurations, and their corresponding performance metrics. The trained model can then be used to predict the performance attributes of interest for a new NoC design, enabling efficient design space exploration [83].

Another trend is the use of reinforcement learning techniques. This involves training an agent to interact with an environment, and then learn optimal policies through trial and error. In the context of NoC optimisation, the agent can be trained to select optimal configurations based on feedback from the NoC environment. This approach has the potential to adapt to dynamic workload conditions and optimise NoC performance in real-time.

Furthermore, ML can be used to optimise specific aspects of a given NoC design, such as

routing algorithms and buffer allocation. The models can be trained to predict traffic patterns and dynamically adjust routing decisions to minimise congestion and improve latency.

Despite its potential, ML applications in NoC optimisation settings also have drawbacks, some of which are:

- **Data dependency:** ML models require large and representative datasets for training, to be useful. Acquiring such datasets can be challenging and time-consuming, especially for complex NoC designs.
- **Black box nature:** ML models can be difficult to interpret and understand, making it challenging to rationalise and cite the reasoning behind their decisions. This lack of transparency represents a not so insignificant concern for NoC designers, who need to understand the trade-offs and implications of specific design choices.
- **Generalisation limitations:** ML models may not generalise well to unseen NoC configurations and/or application workloads. This limitation very often restricts their applicability to specific design scenarios, and require retraining for new applications and/or architectures.

### 2.7.2 Genetic Algorithms for NoC Optimisation

Genetic algorithms (GAs) are a class of evolutionary algorithms inspired by natural selection and genetics. They have been widely used for NoC optimisation due to their ability to efficiently explore large and complex design spaces. GAs operate on a population of candidate solutions (NoC configurations) and iteratively evolve them through genetic operations such as selection, crossover, and mutation. The fitness of each candidate solution is evaluated based on predefined objective functions, which consider the desired performance metrics and constraints.

One of the key advantages of GAs for NoC optimisation is their ability to handle multi-objective optimisation problems. NoC design often involves balancing conflicting objectives, such as maximising performance while minimising power consumption and area overhead. GAs can effectively explore the tradeoffs between these objectives and identify Pareto-optimal solutions, which represent the best possible compromises.

Another benefit of GAs is their adaptability to different NoC architectures and application domains. By adjusting the fitness function and genetic operators, GAs can be tailored to optimise NoCs for specific application requirements and constraints. This adaptability makes GAs a versatile tool for NoC optimisation across various domains.

### 2.7.3 Linear Programming for NoC Optimisation

Linear programming (LP) is a mathematical optimisation technique used to achieve the best outcome (such as maximum profit or lowest cost) in a mathematical model whose requirements are represented by linear relationships. In the context of NoC design, LP can be used to formulate and solve optimisation problems related to resource allocation, routing, and scheduling. By defining objective functions and constraints as linear equations or inequalities, LP solvers can efficiently find optimal solutions that satisfy the given requirements. For instance, LP can be used to optimise buffer allocation in NoCs, where the objective is to minimise buffer sizes while ensuring deadlock-free routing and meeting performance constraints.

The above discussion not withstanding, LP as applied to the NoC optimisation question, has some distinct disadvantages in two key areas: linearity assumption and scalability. In the first aspect, LP requires the objective function and constraints to be linear. This assumption may not hold for all but the most trivial NoC optimisation problems, especially those involving complex non-linear relationships between design parameters and performance metrics. In the second regard, LP can become computationally expensive for large scale NoC designs, with many variables and constraints. This limitation can hinder its applicability in even moderately complex optimisation problems.

### 2.7.4 Simulated Annealing for NoC Optimisation

Simulated annealing (SA) is a probabilistic technique for approximating the global optimum of a given function. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For problems where finding an approximate global optimum is more important than finding a precise local optimum in a fixed amount of time, simulated annealing may be preferable to alternatives such as gradient descent. In NoC optimisation, SA can be used to explore the design space and find optimal or near-optimal solutions for various design parameters, such as topology, routing algorithms, and buffer sizes [79, p. 185]. SA starts with an initial solution and iteratively explores neighbouring solutions by randomly modifying the current solution. The acceptance of a new solution is based on a probability function that depends on the difference in quality between the current and new solutions and a temperature parameter that gradually decreases over time.

That being said, SA has its own disadvantages, when applied to NoC optimisation, including the requirement for careful tuning of parameters, such as the initial temperature and cooling schedule, to achieve good performance. This tuning process can be time consuming, and often requiring expert knowledge. Moreover, SA can be slow to converge to the optimal solution, particularly where complex NoC designs, with large search spaces are concerned.

### 2.7.5 Game Theory for NoC Optimisation

Game theory is a mathematical framework that studies strategic interactions between rational decision-makers. In the context of NoC design, game theory can be used to model and analyse the interactions between different NoC components, such as routers and processing elements. By considering the individual objectives and strategies of these components, game theory can help to design NoCs that achieve optimal overall performance and resource utilisation [84]. For example, game theory can be used to design routing algorithms that balance the competing interests of different traffic flows in the NoC, ensuring fair and efficient resource allocation.

The above not withstanding, game theory has its own limitations in this domain, most noteworthy of which are complexity and reliance on assumptions. In the first regard, applying game theory to NoC optimisation can be complex, requiring careful modelling of the interactions between components, and the definition of appropriate payoff functions. In the second regard, game theory relies on certain assumptions of rationality and behaviour of NoC components. These assumptions may not always hold true in practice, particularly where complex NoC designs with dynamic workloads are concerned.

### 2.7.6 Current Trends

The field of NoC optimisation is constantly evolving, with new techniques and approaches being proposed to address the growing complexity and challenges of NoC design. Some of the current trends in NoC optimisation include:

- **Hybrid optimisation algorithms:** Combining different optimisation techniques, such as GAs and machine learning, can leverage their respective strengths and improve the overall optimisation process. For example, GAs can be used for global search and exploration, while machine learning techniques can be used for local refinement and prediction.
- **Application-specific NoC design:** Recognising that different applications have unique communication patterns and performance requirements, there is a growing trend towards designing NoCs tailored to specific application domains. This approach involves incorporating application-specific knowledge into the optimisation process to achieve optimal performance and resource utilisation.
- **3D NoC optimisation:** 3D NoCs, which stack multiple layers of NoC architectures, offer significant potential for performance and scalability improvements. However, they also introduce new challenges for optimisation, such as thermal management and vertical link placement. Current research is exploring new techniques for optimising 3D NoCs, considering these unique challenges.
- **Design space exploration and automation tools:** To assist NoC designers in navigating the vast and complex design space, there is a growing need for design space exploration

and automation tools. These tools can leverage various optimisation techniques, such as GAs and machine learning, to automate the exploration process and identify optimal NoC configurations.

### 2.7.7 Suitability of the GA-Hypergraph Combination

Genetic algorithms offer a compelling approach to NoC optimisation, particularly when NoCs are modelled as hypergraphs. This suitability stems from the key factors:

1. **Handling complexity:** Hypergraph models of NoCs can be quite complex, capturing intricate interconnections and dependencies among processing elements, memory modules, and other on-chip components. GAs excel at efficiently exploring this complex design space and identifying optimal configurations. Unlike linear programming, which struggles with non-linear relationships often present in NoC designs, GAs can navigate these complexities with ease.
2. **Multi-objectivity:** NoC design often involves balancing multiple conflicting objectives, such as performance, power consumption, and area overhead. GAs can effectively handle multi-objective optimisation problems and identify Pareto-optimal solutions that represent the best possible trade-offs. This capability sets GAs apart from techniques like simulated annealing, which may struggle to balance multiple objectives simultaneously.
3. **Adaptability:** GAs can be easily adapted to different NoC architectures and application domains by adjusting the fitness function and genetic operators. This adaptability makes GAs a versatile tool for optimising NoCs modelled as hypergraphs, as the fitness function can be tailored to consider the specific characteristics and constraints of the hypergraph model. This contrasts with machine learning, where models often require retraining for new applications or architectures.
4. **Scalability:** GAs can handle the increasing complexity of NoC designs, including the growing number of cores and the intricate interconnections in 3D NoCs. As NoC designs become more complex, GAs can effectively scale to explore the larger design space and identify optimal solutions. This scalability is crucial in the era of kilo- and mega-core systems, where other techniques might falter.
5. **Explainability:** Unlike machine learning models, which often operate as "black boxes," GAs offer a more transparent optimisation process. The genetic operations of selection, crossover, and mutation are readily interpretable, allowing designers to understand how the algorithm arrives at its solutions. This transparency is valuable for NoC design, where understanding the trade-offs and implications of different design choices is crucial.

In conclusion, GAs really do present several features that make them a powerful and versatile tool for optimising NoCs, when modelled as hypergraphs. Their ability to handle complexity, multi-objective optimisation, adaptability, scalability, and explainability makes them a

compelling choice for addressing the challenges of NoC design and optimisation in the contemporary landscape of high-performance computing, and the exponential growth in many-core applications.

Table 2.4: Comparison of NoC Design Space Optimisation Techniques

| **Technique** | **Features** | **Advantages** | **Disadvantages** |
|---|---|---|---|
| Machine Learning | • Predicts NoC performance metrics<br>• Enables rapid design space exploration<br>• Adapts to dynamic workloads<br>• Optimises routing and buffer allocation | • Fast exploration of design space<br>• Can handle complex relationships<br>• Potential for real-time adaptation | • Requires large, representative datasets<br>• Black box nature<br>• Limited generalisation ability |
| Genetic Algorithms | • Evolutionary approach<br>• Operates on solution populations<br>• Uses genetic operators<br>• Evaluates fitness objectives | • Handles multi-objective optimisation<br>• Adaptable to various architectures<br>• Efficient space exploration<br>• Scalable to large designs<br>• Explainable process | • Computationally expensive<br>• Requires careful parameter tuning |
| Linear Programming | • Mathematical optimisation<br>• Linear relationships<br>• Uses equation solvers | • Efficient for linear problems<br>• Guaranteed optimal solutions | • Limited to linear relationships<br>• Expensive for large problems<br>• May miss NoC complexities |

Continued on next page

**Table 2.4 – continued from previous page**

| Technique | Features | Advantages | Disadvantages |
|---|---|---|---|
| Simulated Annealing | • Probabilistic technique<br>• Explores neighbouring solutions<br>• Temperature-based acceptance | • Can escape local optima<br>• Simple implementation | • Needs parameter tuning<br>• Slow convergence<br>• Limited for complex NoCs |
| Game Theory | • Models strategic interactions<br>• Considers component objectives<br>• Optimises resource usage | • Insights into interactions<br>• Fair resource allocation | • Complex to model<br>• Relies on assumptions<br>• Not suitable for all cases |

## 2.8 Chapter Summary

There is arguably a good body of existing work and knowledge around some of the issues relating to the performance of NoCs in multi-core applications.

Research in NoC optimisation techniques has evolved from basic genetic algorithms to more sophisticated approaches including machine learning and hybrid methods, as highlighted in Section 2.7. This not withstanding, current state-of-the-art research tends towards a narrow, disjointed approach that fails to fully address the challenges of next-generation architectures requiring 1K+ cores and beyond.

That said, current state-of-the-art research would suggest a narrow, disjointed approach to resource allocation questions in NoCs for high performance multi-core applications. The world continues to move rapidly towards 1M+ cores being the norm and massively spiky neural network implementations, particularly in 5G application, AI, and ML. We can no longer treat the question of resource allocation and NoC optimisation as a disjointed set of smaller problems.

Further, research in resource allocation and optimisation in FPGAs is scarce. This is especially true of partially re-configurable FPGAs, for which the demand has never been greater due to factors such as time to market, cost, performance, and the ever-increasing pressure on silicon real estate.

As highlighted in the sections earlier, these are some of the motivations for undertaking this research, and it is for these reasons that this research is both viable and relevant. This is summarised in Table 2.5 below:

Table 2.5: 2D NoC vs. 3D NoC: A Comparison [85]

| Feature | 2D NoC | 3D NoC | Justification |
|---|---|---|---|
| Topology | Mesh/Ring | Mesh/Torus | - |
| Diameter [86] | $O(\sqrt{N})$ / $O(N)$ | $O(\sqrt[3]{N})$ | 3D reduces diameter, enabling shorter communication paths between nodes. |
| Scalability | Limited | Improved | Lower diameter allows for better scalability with increasing core counts. |
| Bandwidth | Limited | Increased | More connections and vertical links in 3D increase the available bandwidth. |
| Wire length | Longer | Shorter | Vertical stacking reduces wire length, leading to lower latency and power consumption. |
| Density | Lower | Higher | 3D integration enables higher core density and closer proximity of processing elements. |
| Resource Allocation | Less Efficient | More Efficient | Shorter communication distances and increased connectivity allow for more balanced resource utilisation. |
| Latency | Higher | Lower | Shorter paths and higher bandwidth contribute to reduced latency. |
| Energy Efficiency | Lower | Higher | Shorter wires and reduced communication reduce energy consumption. |

*Continued on next page*

Table 2.5 – continued from previous page

| Feature | 2D NoC | 3D NoC | Justification |
|---|---|---|---|
| Performance | Lower | Higher | Lower latency, higher bandwidth, and efficient resource allocation improve performance. |

# Chapter 3

# Modelling 3D NOC as Hypergraphs

First introduced by Leonhard Euler to solve the 7 bridges of Konigsberg[87], *Graph Theory* is useful in many disciplines for modelling real-life problems involving topology, traversal and optimisation.

It gained popularity in VLSI design, with applications as a tool for modelling the Dynamic Timing Analysis (DTA) of VLSI components [88], [89].

This chapter will use higher-order graph theory to investigate the modelling and analysis of 3D Network-on-chip (3D NOC) architectures. We will then go on to develop a uniform method An *Undirected Graph* or simply *Graph* $G = (V, E)$ consists of finite nonempty sets of vertices $V$ and edges $E$[90, p. 148-149]. If each edge is a set $\{v, w\}$ of vertices, the graph is undirected. Whereas if each edge is an ordered pair $[v, w]$ of vertices, the graph is a *Undirected Graph* or simply *Digraph*. A graph can be represented using an adjacency matrix $A = |V| \times |V|$, where an entry $A[i, j] = 1$ will exist if and only if there is an edge from vertex $i$ to vertex $j$, and $A[i, j] = 0$ if there is no edge. This is a special case of a distance matrix $D = |V| \times |V|$ with the same unit distance for all edges [91].

## 3.1 Hypergraphs: Characteristics

An *Hypergraph* $H$ is defined as a generalisation of the idea of a graph, in which an edge, called an *hyperedge*, can link any number of vertices. More formally, an Hypergraph $H$ is a pair $H = (V, E)$, where $V$ is a set of vertices and $E$ is a set of non-empty subsets of $V$[92]. Each subset $e \in E$ is a *hyperedge*, and each hyperedge will connect at least one pair of $V(i, j)$.

In an hypergraph, an *edge* (often called a *hyperedge*) can connect more than two vertices. In this situation, a 3-dimensional representation can be considered; more formally, for a set of vertices $V$, A *hyperedge* in this context could be a set of three vertices $\{v_1, v_2, v_3\}$.

A *3-dimensional adjacency matrix*(also referred to as an *adjacency tensor*[93], or simply Tensor) extends the concept to be useful in representing higher-dimensional data structures or networks.

An hypergraph typically denotes a graph visualised or embedded in the n-dimensional space. In the context of this work on 3D NoC, we are modelling structures where n=3; hence, we will use Hypergraph (also, *hypergraph*) to refer to hypergraphs where n=3, without explicit specification. Henceforth, we will use *n* to refer to the number of nodes in the network, the plane, or in general, without further specifying that n=3.

From fundamental Graph Theory, an *adjacency tensor* $T$ for a hypergraph of shape $n \times n \times n$, where $n$ is defined as the number of nodes in the graph. An element $T_{ijk}$ in this tensor will thus have a value of 1 if and only if there is a relationship (i.e. an edge) between node $i$, node $j$, and node $k$, respectively, else it will have a value of 0. This adjacency tensor provides a convenient representation for hypergraph relationships among the nodes in the network.

In other words, a 3-dimensional structure is required to represent a hypergraph. The tensor idea described earlier serves the purpose well; taking $H$ with dimensions $|V| \times |V| \times |V|$ as an example, then an entry $H[i, j, k]$ in that structure would be set to 1 if there exists an *hyperedge* connecting vertices $i, j$, and $k$. Otherwise, it would be set to 0, as described above. More succinctly:

1. If $H[i, j, k] = 1$ and all indices are distinct, it indicates a hyperedge connecting vertices $i, j$, and $k$.
2. If two indices are the same, for instance, $H[i, j, j]$, it might represent a traditional edge in the graph connecting $i$ and $j$.
3. If all three indices are the same, for example, $H[i, i, i]$, it might denote a self-loop on vertex $i$.

To take the ideas developed above further, in the context of hypergraphs and 3D Topologies, and without loss of generality, a Distance Tensor is three-dimensional array (tensor) $\mathcal{D}$ of size $n \times n \times n$, where each element $\mathcal{D}_{ijk}$ represents the shortest distance (or weight) from node $i$ to node $j$ in the $k^{th}$ layer of the 3D structure. Given a 3D topology, the Distance Tensor would capture the distances between every pair of nodes for each topology layer. This structure allows one to store and analyse multi-layer or multi-relational data in networks.

Just like the adjacency matrix can be seen as a special case of a distance matrix, the 3D adjacency matrix can be seen as a special case of a 3D distance tensor. Therefore, an element $D[i, j, k] = 1$ in this tensor represents the distance between a hyperedge connecting vertices $i, j$, and $k$, respectively. If there is no such hyperedge, this could be represented as infinity (or some very large number).

Next, we will now focus our attention on 4 primary topologies of interest: 3D Mesh, 3D Torus, 3D Folding Torus, and 3D Hypercube. Henceforth, topologies are assumed to be 3D structures, unless stated otherwise.

For each topology of interest, we will proceed to derive the following characteristics:

1. **Adjacency Matrix**: An adjacency matrix is a square matrix representing a finite graph. The matrix elements indicate whether pairs of vertices are adjacent or not in the graph. For a graph $G$ with $n$ vertices, its adjacency matrix $A$ is an $n \times n$ matrix where $A[i, j]$ is 1 if there is an edge between vertices $i$ and $j$; otherwise, it's 0.
2. **Distance Matrix**: The distance matrix is a square matrix containing the shortest distance between every pair of nodes in the graph. For a graph $G$ with $n$ vertices, its distance matrix $D$ is an $n \times n$ matrix where $D[i, j]$ is the shortest path length from vertex $i$ to vertex $j$.
3. **Weight Matrix**: The weight matrix is similar to the adjacency matrix, but instead of binary values indicating the presence or absence of edges, it contains the weights of the edges. The corresponding entry is usually set to infinity (or a large value) if there is no edge between a pair of vertices.

We will now proceed to these features in the context of our topologies of interest, namely *Mesh, Torus, Folding Torus and Hypercube*, as follows:

**3D Mesh:** A 3D mesh consists of vertices arranged in a regular grid. Let us assume the grid dimensions are $n \times m \times p$. Each node in the 3D mesh has a coordinate $(x, y, z)$ where $0 \leq x < n$, $0 \leq y < m$, and $0 \leq z < p$.

**Adjacency Matrix:**

An adjacency matrix $A$ for this structure would be of size $(n \times m \times p) \times (n \times m \times p)$. For two nodes $i$ and $j$:

$$A[i, j] = \begin{cases} 1 & \text{if } iandj \text{ are adjacent in the 3D mesh} \\ 0 & otherwise \end{cases} \tag{3.1}$$

For nodes to be adjacent, their coordinates $(x_i, y_i, z_i)$ and $(x_j, y_j, z_j)$ should differ by 1 in one dimension and be the same in the other two dimensions[94].

**Distance Weight Matrix:** This is similar to the adjacency matrix but with distances. If nodes are adjacent, the distance is 1; otherwise, it is infinity (or a large number).

**3D Cube:**

A 3D cube is a specific case of a 3D mesh where $n = m = p = 2$.

**3D Torus:**

A 3D torus connects the edges of a 3D mesh, making the topology wrap around. This means nodes on the boundary are adjacent to nodes on the opposite boundary.

**Adjacency Matrix:** This matrix is similar to the 3D mesh but with added connections for wrapping. For example, nodes with $x = 0$ are adjacent to nodes with $x = n - 1$ (and similar for $y$ and $z$). [95]

**Distance Weight Matrix:** As with the 3D mesh, adjacent nodes have a distance of 1 but with added wrap-around connections.

**3D Folding Torus:**

A 3D Folding Torus topology is complex. The boundaries wrap around (akin with 3D Torus), with the addition of folding connections, hence the name.

**Adjacency Matrix:** Besides the regular torus connections, nodes are also connected across a folding plane. Depending on the specifics of the folding torus, connections will vary[96].

**Distance Weight Matrix:** This will depend on the specific fold connections and the regular torus connections.

## 3.2 Topology descriptions as graphs

This section and the following paragraphs will consider each topology's vertex and edge characteristics. The end goal will culminate towards modelling each topology's characteristics as a set of Performance Cost Ratio functions (PCR), introduced in Section 3.7.1

**3D Mesh:**

**Vertices:** Every point in the 3D mesh is a vertex. If the mesh has dimensions $n \times m \times p$, there are $n \times m \times p$ vertices. Each vertex can be identified by its 3D coordinates: $(x, y, z)$ where $0 \leq x < n$, $0 \leq y < m$, and $0 \leq z < p$[97, p. 109].

**Edges:** There is an edge between any two adjacent vertices in the mesh. Specifically:

- If two vertices share the same $y$ and $z$ coordinates but have $x$ coordinates that differ by 1, they're connected.
- If two vertices share the same $x$ and $z$ coordinates but have $y$ coordinates that differ by 1, they're connected.
- If two vertices share the same $x$ and $y$ coordinates but have $z$ coordinates that differ by 1, they're connected.

**Hyper Cube:**

**Vertices:** The 3D cube is a special case of the 3D mesh with 8 vertices, represented as the corners of a cube.

**Edges:** Each vertex is connected to 3 other vertices - those it shares a face with. There are 12 edges in total.

**3D Torus:**

**Vertices:** Same as the 3D mesh.

**Edges:** The 3D torus has all the edges of the 3D mesh plus additional edges that "wrap around". Specifically:

- Vertices on the left boundary (where $x = 0$) are connected to vertices on the right boundary (where $x = n - 1$), and vice versa if they share the same $y$ and $z$ [98].
- Similarly, vertices on the top boundary (where $y = 0$) are connected to vertices on the bottom boundary (where $y = m - 1$), and vice versa if they share the same $x$ and $z$.
- Vertices on the front boundary (where $z = 0$) are connected to vertices on the back boundary (where $z = p - 1$), and vice versa if they share the same $x$ and $y$.

**3D Folding Torus:**

The 3D folding torus is more complex and can vary based on the exact fold type. Here's a basic interpretation:

**Vertices:** Same as the 3D mesh.

**Edges:** The 3D folding torus has all the edges of the 3D torus plus additional edges for the fold. The exact nature of these edges depends on the fold's specifics. For example, there could be Edges connecting vertices on the "top" of the torus to vertices on the "bottom", creating a fold that's akin to a Möbius strip, but in three dimensions.

The exact specification of the "folding" in the 3D folding torus will dictate additional edges. The interpretation above provides a high-level overview, but the specific topology's exact details must be clarified to give an accurate graph representation.

**Orthogonal & Diagonal Edges**

In the context of the 3D topologies mentioned, "orthogonal" and "diagonal" edges pertain to how nodes are connected in the structure.

**Orthogonal Edges:**

These edges connect two nodes that share two of the three coordinates and differ by exactly one unit in the third coordinate. In our 3D topologies:

1. 3D Mesh & 3D Cube: For any given vertex with coordinates $(x, y, z)$, it has orthogonal edges to:
   - $(x \pm 1, y, z)$ (if within bounds)
   - $(x, y \pm 1, z)$ (if within bounds)
   - $(x, y, z \pm 1)$ (if within bounds)
2. 3D Torus:
   - Same as the 3D Mesh, but with wrapping. So, a node at one boundary has an orthogonal connection to a node at the opposite boundary in the torus.
3. 3D Folding Torus
   - Same as the 3D Torus, but with additional orthogonal connections depending on the specifics of the folding.

**Diagonal Edges:**

Diagonal edges connect two nodes that differ in more than one coordinate.

**3D Mesh and 3D Cube**

- A vertex $(x, y, z)$ could have diagonal edges to:
  - $(x \pm 1, y \pm 1, z)$
  - $(x \pm 1, y, z \pm 1)$
  - $(x, y \pm 1, z \pm 1)$
- And the fully diagonal edges would be to:
  - $(x \pm 1, y \pm 1, z \pm 1)$

However, in the standard 3D Mesh and Cube representations, these diagonal connections do not typically exist.

**3D Torus**

- Same potential diagonal connections as the 3D Mesh, but with wrapping. These diagonal connections are not standard for a 3D torus but can be conceived with the same wrapping logic.

**3D Folding Torus**

- It would include potential diagonal connections similar to the 3D Torus but can have other diagonal connections depending on the specific fold.

In general, in the topologies discussed, orthogonal connections are the most common, and diagonal connections are less frequent or non-existent. However, nothing theoretically prevents the addition of diagonal connections; they are just not standard for those topologies.

## 3.3 Graph Characteristics: a summary

For each of the topologies discussed so far, we are now in a position to characterise the corresponding graph features as follows:

**3D Mesh:**

**Vertices (nodes):**

$$V = n \times m \times p \tag{3.2}$$

**Edges:** If those neighbours exist, each vertex connects to a neighbour to its right, left, up, down, in front, and behind.

$$E = \begin{cases} (n-1) \times m \times p \, edges \, in \, the \, x - direction. \\ n \times (m-1) \times p \, edges \, in \, the \, y - direction. \\ n \times m \times (p-1) \, edges \, in \, the \, z - direction. \end{cases} \tag{3.3}$$

**total:**

$$E_{total} = (n-1) \times m \times p + n \times (m-1) \times p + n \times m \times (p-1) \tag{3.4}$$

**Graph Diameter:** In a 3D mesh, the longest shortest path would be from one corner to the opposite corner. This would be:

$$D = (n-1) + (m-1) + (p-1) \tag{3.5}$$

**Hyper Cube:**

This is a specific case of a 3D Mesh where $n = m = p = 2$.

**Vertices:**

$$V = 8 \tag{3.6}$$

**Edges:**

$$E = 12 \, (Each \, of \, the \, 8 \, vertices \, is \, connected \, to \, 3 \, others) \tag{3.7}$$

**Graph Diameter:**

$$D = 3 \, (You \, can \, go \, from \, one \, corner \, to \, the \, opposite \, corner \, in \, 3 \, steps) \tag{3.8}$$

**3D Torus:**

**Vertices:**

$$V = n \times m \times p \tag{3.9}$$

**Edges:** Every vertex in a 3D torus has 6 neighbours (assuming $n, m, p > 1$ to avoid degenerate cases). However, every edge is counted twice (once for each vertex it connects).

**Total:**

$$E_{total} = \frac{n \times m \times p \times 6}{2} = 3nmp \tag{3.10}$$

**Graph Diameter:** The diameter would be the maximum distance travelled in one dimension without wrapping. Hence, it would be half the size of the largest dimension, rounded up:

$$D = \max\left(\left\lceil \frac{n}{2} \right\rceil, \left\lceil \frac{m}{2} \right\rceil, \left\lceil \frac{p}{2} \right\rceil\right) \tag{3.11}$$

### 3D Folding Torus:

The specifics of a 3D folding torus can vary based on the exact fold type if we assume that the folding only adds connections without removing the existing ones:

**Vertices:**

$$V = n \times m \times p \tag{3.12}$$

**Edges:** Greater than or equal to that of a standard 3D torus. The exact number would depend on the nature of the folds.

**Graph Diameter:** This would typically be less than or equal to the standard 3D Torus, as the folding usually provides shortcuts across the topology. The exact diameter depends on the specifics of the folding.

For the 3D folding torus, a precise specification of the "folding" would be required for accurate measurements. This is the subject of the next section 3.3, and associated subsections.

### Manifolds & the 3D Torus Special Case

Given the context of topological manifolds, a 3D torus is embedded in a higher-dimensional Euclidean space (not directly realisable in ( R 3 ) due to its topology). For the case of a 3D Folding Torus, the folding specifies how the manifold is deformed or connected in ways that are not native to a simple 3D torus.

To understand the specification of the folding in the case of a 3D Folding Torus, one would need:

1. **Type of Connection:** Describes how sections of the manifold connect or relate to each other. For example, a Möbius strip has a twist; it is a kind of "fold".
2. **Location of the Folds:** Identifying where these folds or connections occur on the torus.
3. **Topological Changes:** Determine if the fold alters or deforms the overall topology. For instance, adding a handle to a torus changes its genus, and such specifics must be identified.
4. **Embedding in Higher Dimensions:** To visualise or represent the folded torus, it might be necessary to understand its embedding in a higher-dimensional space, like $R^4$ or beyond. This can help understand its structure, intersections, and overlaps.
5. **Local Properties:** Manifolds are defined by their local Euclidean properties. For any point on our folding torus, one should be able to find a small enough neighbourhood around it that is topologically equivalent to $R^3$. Any folding should maintain this property; understanding this can help discern the fold's nature and extent.
6. **Global Properties:** While manifolds are locally Euclidean, their global structure can vary dramatically based on folds and connections. Understanding the global properties can help grasp the manifold's overall shape and connectivity.

## 3.4 Analysis of Topologies of Interest

To conclude the discussion above without any loss of generality, we will provide the above metrics, as calculated for an m, n, p term = 3 for all topologies except Hypercube, where a single metric of D=3 is used. Each topology and the corresponding Adjacency and Distance Weight matrices are depicted.

**3D Mesh Example** The 3D Mesh topology consists of nodes arranged in a cubic structure, where each node is connected to its six orthogonal neighbours (up, down, left, right, front, and back) except on the boundaries. While considering diagonals, nodes connect to additional neighbours located diagonally on the same plane or the plane above/below. Figure 3.1 below depicts this structure:

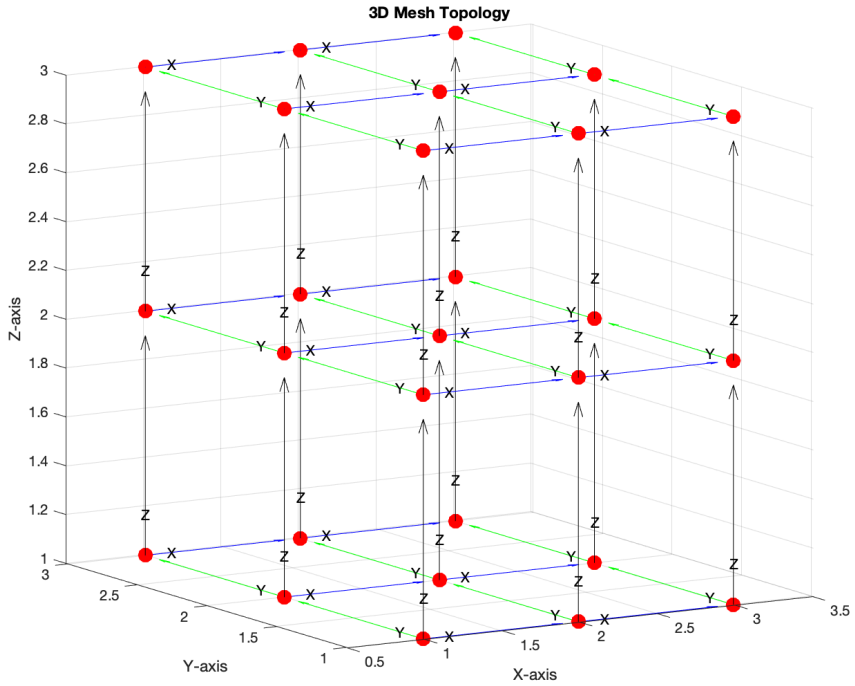



Figure 3.1: 3D Mesh Topology (3x3x3) - Figure 1

Table 3.1: 3D Mesh Properties

| Diameter | Number of Nodes | Number of Edges |
|---|---|---|
| $D = 3 \times (3 - 1)$ | 27 | $3 \times 27 -$ boundary connections |

$$\text{Adjacency Matrix (indicative for 3D Mesh)} = \begin{bmatrix} 0 & 1 & 0 & \dots \\ 1 & 0 & 1 & \dots \\ 0 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.13}$$

$$\text{Distance Weight Matrix for 3D Mesh} = \begin{bmatrix} 0 & 1 & \infty & \dots \\ 1 & 0 & 1 & \dots \\ \infty & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}$$

**3D Torus Example** The 3D Torus is a cyclic version of the 3D Mesh. Nodes on the edges are connected cyclically to nodes on the opposite edge, creating a wrap-around connection. 3.2 below depicts this structure:

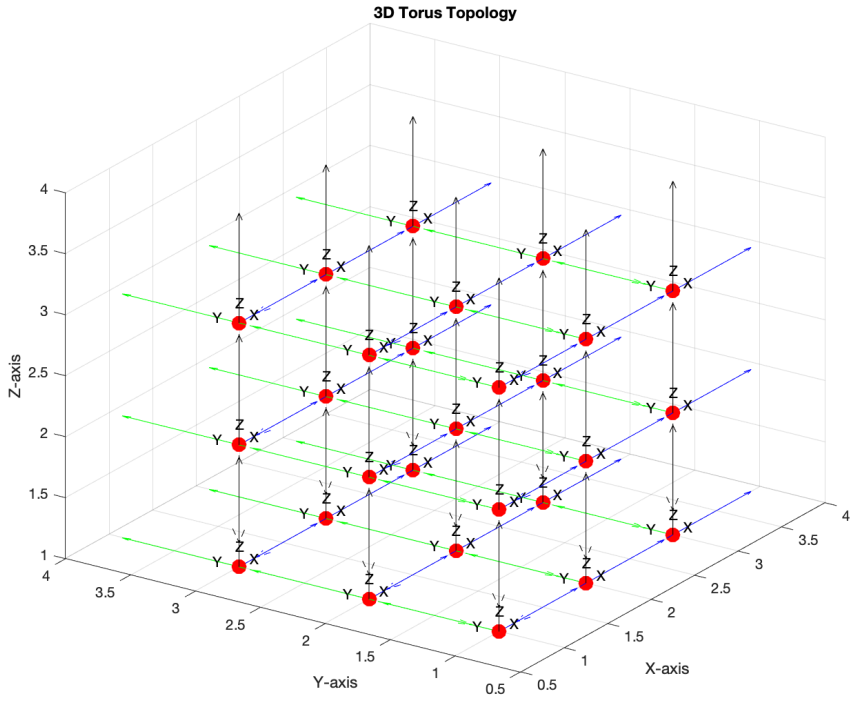



Figure 3.2: 3D Torus Topology (3x3x3) - Figure 2

Table 3.2: 3D Torus Properties

| Diameter | Number of Nodes | Number of Edges |
|---|---|---|
| $D = \frac{3}{2}$ | 27 | $3 \times 27$ |

$$\text{Adjacency Matrix (indicative for 3D Torus)} = \begin{bmatrix} 0 & 1 & 1 & \dots \\ 1 & 0 & 1 & \dots \\ 1 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.14}$$

$$\text{Distance Weight Matrix for 3D Torus} = \begin{bmatrix} 0 & 1 & 1 & \dots \\ 1 & 0 & 1 & \dots \\ 1 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.15}$$

**3D Folding Torus Example** The 3D Folding Torus is similar to the 3D Torus but with additional diagonal links connecting nodes on opposing corners of the cube. 3.4 depicts this structure:

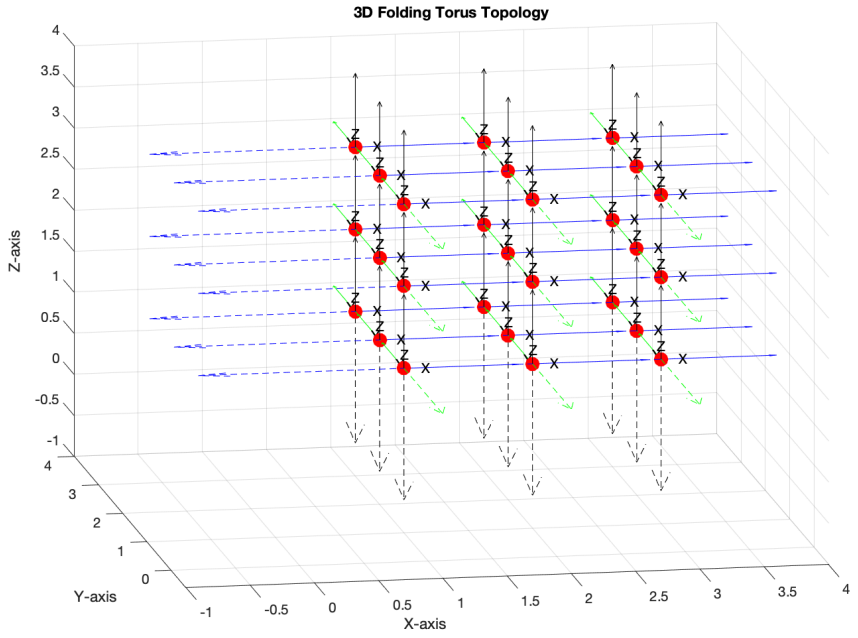



Figure 3.3: 3D Folding Torus Topology (3x3x3) - Figure 3

Table 3.3: 3D Folding Torus Properties

| Diameter | Number of Nodes | Number of Edges |
|---|---|---|
| $D = \frac{3}{2}$ | 27 | $3 \times 27 +$ diagonal links |

$$\text{Adjacency Matrix (indicative for 3D Folding Torus)} = \begin{bmatrix} 0 & 1 & 1 & \dots \\ 1 & 0 & 1 & \dots \\ 1 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.16}$$

$$\text{Distance Weight Matrix for 3D Folding Torus} = \begin{bmatrix} 0 & 1 & 1 & \dots \\ 1 & 0 & 1 & \dots \\ 1 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.17}$$

**3D Hypercube (Depth 3) Example** The 3D Hypercube with depth 3, also called an 8-cube, consists of nodes where each node is connected to eight other nodes, representing a binary sequence of 3 bits. Figure 3.4 depicts this structure:

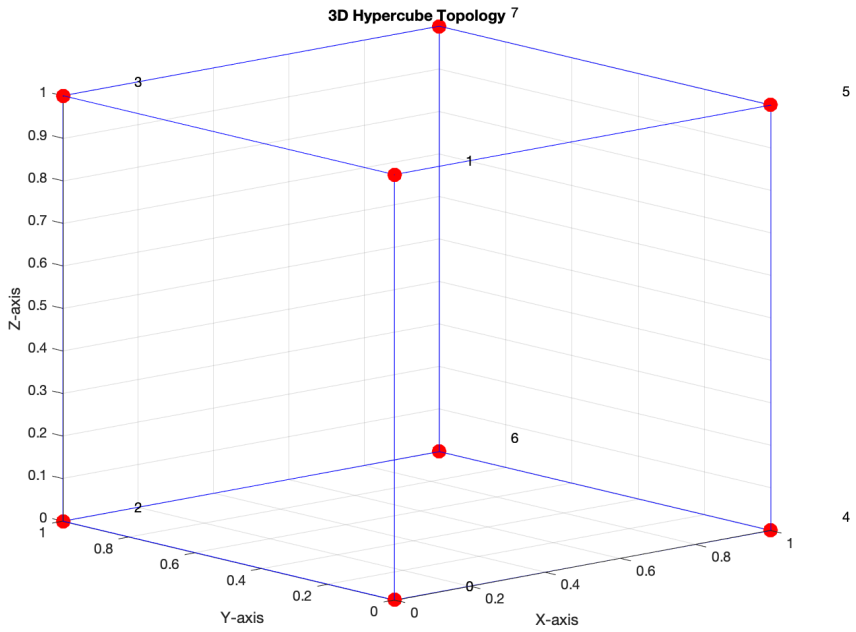



Figure 3.4: 3D Hypercube Topology (Depth 3) - Figure 4

Table 3.4: 3D Hypercube Properties

| Diameter | Number of Nodes | Number of Edges |
|---|---|---|
| $D = 3$ | $2^3 = 8$ | $\frac{8\times 3}{2} = 12$ |

$$\text{Adjacency Matrix (indicative for 3D Hypercube)} = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{3.18}$$

$$\text{Distance Weight Matrix for 3D Hypercube} = \begin{bmatrix} 0 & 1 & 2 & \dots \\ 1 & 0 & 1 & \dots \\ 2 & 1 & 0 & \dots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix} \tag{3.19}$$

### 3.4.1 Performance Analysis

In the previous sections, we laid the groundwork and detailed our approach to modelling our 3D NoC topologies of interest, such as Hypergraphs in the n-dimensional space, with n=3. In the coming sections, we will be building the groundwork for analysing the performance of NoCs in general, focusing on the 3D topologies discussed so far.

We will start by associating a simple cost with each link and develop a cost function that would return to the total cost associated with traversing a particular path, when given a start node, an end node, and a list of edges to traverse.

To build our understanding, we focus on three Graph Theory concepts: the number of *vertices (V), edges (E)* and *diameter (D)* of the graph representing each topology of interest, discussed earlier in Section 3.2 and Section 3.3. While dealing with 3D Hypercube Structures, ***connectivity*** aspect must be dealt with. It will be discussed separately in Equation 3.4.1, dealing with the Hypercube structure. Next is a topology breakdown:

**3D Mesh:**

**Vertices:**

The number of vertices in a 3D mesh topology is straightforward to calculate. Given that the

topology is a 3D grid with dimensions $n \times m \times p$, the total number of nodes $V$ is:

$$V = n \times m \times p \tag{3.20}$$

**Edges:**

Edges in a 3D mesh are formed between neighbouring nodes in all three dimensions:

1. Along the $n$ dimension, each node has an edge to its right, except the last node in each row. Hence, the number of edges along the $n$ dimension is $(n-1) \times m \times p$.
2. Along the $m$ dimension, each node has an edge below it, except the last node in each column. Hence, the number of edges along the $m$ dimension is $n \times (m-1) \times p$.
3. Along the $p$ dimension, each node has an edge in front of it, except the last node in depth. Hence, the number of edges along the $p$ dimension is $n \times m \times (p-1)$.

.

Therefore, the sum of the total number of edges $E$ in the 3D mesh topology is:

$$E = (n-1) \times m \times p + n \times (m-1) \times p + n \times m \times (p-1) \tag{3.21}$$

**Diameter:** The diameter of a graph is the longest path between any two nodes in the graph. In a 3D mesh topology, the longest shortest path is from one corner of the mesh to the opposite corner. If you travel in a straight line through the mesh along each dimension, you will get:

$$D = (n-1) + (m-1) + (p-1) \tag{3.22}$$

Where $D$ is the diameter, representing the maximum number of hops required to traverse from one corner of the mesh to the opposite corner.

So, to summarise:

$$V = n \times m \times p \tag{3.23}$$

$$E = (n-1) \times m \times p + n \times (m-1) \times p + n \times m \times (p-1) \tag{3.24}$$

$$D = (n-1) + (m-1) + (p-1) \tag{3.25}$$

**Hypercube (or n-cube)** an Hypercube (or n-cube) is a fascinating topological structure. A 1D hypercube is a line segment, a 2D hypercube is a square, and a 3D hypercube is a cube. Vertices are effectively doubled for each extra dimension, and corresponding connecting edges are added. Wlog, For a 3D structure, the Hypercybe is simply a *Cube*, hence our use of the terms Hypercube and Cube interchangeably throughout this text.

We will now proceed to break down the structure of a cube using the same logic and means applied earlier to the 3D Mesh topology:

**Nodes (Vertices):**

The cube has 8 vertices regardless of how we derive it.

For a general n-dimensional hypercube, the number of vertices is $2^n$. So, for a 3D hypercube:

$$V = 2^3 = 8 \tag{3.26}$$

**Edges:**

Each vertex of a cube is connected to 3 other vertices. So, for 8 vertices, we might think there are $8 \times 3 = 24$ edges. However, this counts each edge twice (once for each of its vertices).

**So for a 3D hypercube (cube):**

$$E = \frac{8 \times 3}{2} = 12 \tag{3.27}$$

So, generally, for an n-dimensional hypercube, edges are added for each additional dimension. The formula becomes:

$$E = n \times 2^{(n-1)} \tag{3.28}$$

... and for the 3D hypercube:

$$E = 3 \times 2^{(3-1)} = 3 \times 4 = 12 \tag{3.29}$$

**Diameter:**

For a 3D hypercube, the diameter is the longest distance between any two vertices. In the case of a cube, it is the space diagonal passing through all three dimensions. Thus:

$$D = 3$$

For a general n-dimensional hypercube, the diameter is $n$, as you can move from one "corner" vertex to the opposite "corner" vertex by changing the state (0 to 1 or vice versa) for each dimension once.

**Connectivity:** Another aspect to consider is node connectivity. For a 3D hypercube, each vertex is connected to 3 other vertices, which is evident from its cube structure. In general, for an n-dimensional hypercube, each node is connected to *n* other nodes, as we can flip the state in each dimension.

**Faces:** A 3D hypercube (cube) has 6 faces. In the context of hypercubes, each "face" can be considered an (n-1)-dimensional hypercube. So, for a 3D hypercube, its faces are 2D squares.

**To summarise for the 3D Hypercube:**

$$V = 8 \tag{3.30}$$
$$E = 12 \tag{3.31}$$
$$D = 3 \tag{3.32}$$
$$Faces = 6 \tag{3.33}$$

(which are 2D squares)

$$Connectivity = 3 \tag{3.34}$$

(this is the connectivity per vertex value)

**For general n-dimensional hypercube:**

$$V = 2^n \tag{3.35}$$
$$E = n \times 2^{(n-1)} \tag{3.36}$$
$$D = n \tag{3.37}$$
$$Connectivity = n \tag{3.38}$$

The *Faces* (or (n-1)-dimensional sub-hypercubes) is a complex combinatorial expression depending on $n$, but for the 3D case, it is 6.

Next, we are now going to lay the foundations for the remaining topologies of interest, as follows:

### 3D Torus:

A torus is a structure where the endpoints are connected. In a grid or mesh context, a 3D torus is formed when the grid is "wrapped" so that the edges connect opposite sides.

**Vertices:** Like the 3D Mesh, the 3D Torus has dimensions $n \times m \times p$, so:

$$V = n \times m \times p$$

**Edges:** Edges in a 3D torus are similar to the 3D mesh but with additional edges connecting the boundary vertices.

1. Along the $n$ dimension: $n \times m \times p$

2. Along the $m$ dimension: $n \times m \times p$
3. Along the $p$ dimension: $n \times m \times p$

Thus, the total edges $E$ are:

$$E = 3 \times n \times m \times p \tag{3.39}$$

**Diameter:** In a 3D torus, the diameter is the longest shortest path between any two nodes. It is half the length, width, or depth, whichever is largest since the torus can be traversed in either direction (it's "wrapped around"). So:

$$D = \max(n, m, p)/2(rounded\, up\, if necessary). \tag{3.40}$$

**3D Folding Torus:** A 3D Folding Torus is a 3D Torus with additional diagonal connections.

**Nodes (Vertices):** Same as the 3D Torus:

$$V = n \times m \times p \tag{3.41}$$

**Edges:**

Edges in a 3D folding torus include all the edges from the 3D torus plus the additional diagonal connections:

1. All edges of 3D Torus: $3 \times n \times m \times p$
2. Diagonal connections: $n \times m \times p$

Edges $E$:

$$E = 4 \times n \times m \times p \tag{3.42}$$

**Diameter:**

The diameter has been reduced since we now have diagonal shortcuts. However, finding the exact diameter is more complicated because it depends on the exact configuration and dimensions of the grid. Typically, it will be less than the standard 3D torus. However, exact calculation would require considering the most efficient path using the diagonal shortcuts.

So to compare the two variants of 3D Torus:

**3D Torus:**

$$V = n \times m \times p \tag{3.43}$$

$$E = 3 \times n \times m \times p \tag{3.44}$$

$$D = \max(n, m, p)/2 \tag{3.45}$$

Where $D$ is rounded up, if necessary

**3D Folding Torus:**

$$V = n \times m \times p \tag{3.46}$$

$$E = 4 \times n \times m \times p \tag{3.47}$$

D is Less than the 3D torus but requires more detailed analysis for an exact value.

**3D Folding Torus:** The exact nature and configuration of a "folding torus" can vary based on definitions. This is a common understanding, but specific research or context could adjust or refine these definitions and calculations. We shall make some assumptions regarding this and discuss it at the appropriate point.

## 3.5 Latency, Bandwidth, and Throughout as Performance parameters for topologies of interest

In the preceding sections, we decomposed the characterisation of the NoC topologies under consideration as hypergraphs. We further disseminated characteristics such as vertices, edges, and diameter to model each topology and its physical characteristics.

In the following sections, we will explore the performance aspects of each topology in detail, focusing on the bandwidth, latency, and throughput. Our goal is to develop a more comprehensive understanding of these parameters through mathematical principles utilising graph theory.

### 3.5.1 Latency

Latency is the time delay between sending a signal and receiving a response. However, the specific definition and implications of latency can vary depending on the context.

In networking, latency refers to the time it takes for a packet of data to travel from the source to the destination [99]. It is usually measured in milliseconds. It encompasses all delays from

data encoding into a packet, its propagation across the network, routing through intermediate devices such as switches and routers, and decoding at the destination [100].

In the context of the NoC domain, latency is the time it takes for a packet (or flit) to cross from its originating core or IP block to its destination somewhere else in the network [101]. More importantly, the aspect of latency in 3D NoCs is critically affected by the interdimensional links in silicon, known as Through-Silicon Vias (TVS) [102]).

Furthermore, the latency in a 3D NoC critically affects the overall performance of the whole IC, especially with concurrent communications between multiple cores or blocks [103]. Here, factors such as topology, routing algorithms, traffic patterns, and the architectural characteristics of individual components contribute to overall latency [104].

Thus, low latency is essential to the performance of multi-core systems to ensure that the advantages of parallelism in multi-core systems are not dominated by communication delays.

Now, for the topologies discussed so far, *wlog*, let us assume that each link traversed costs one single resource unit of latency. Then, let us now consider the latency expressions for all topologies using *Dijkstra algorithm*, [105] to traverse each topology. The latency for traversing a graph using Dijkstra's algorithm depends on the shortest path between the source and destination nodes.

The shortest path will depend on the topology of the network and the weights (costs) associated with each edge. Given that each link traversed costs one single resource unit of latency, the latency in terms of Dijkstra's algorithm would equal the number of hops (edges) in the shortest path.

**For each topology**, the latency expressions are as follows:

**3D Mesh:**

The *Maximum Latency* occurs when the source is one corner of the mesh, and the destination is the diagonally opposite corner. The number of hops (edges) in the shortest path would be $n + m + p - 3$. (Subtracting 3 since we count nodes but not edges.)

**Latency Expression:**

$$L = n + m + p - 3 \tag{3.48}$$

**3D Torus:** Given that a Torus, by its very definition wraps around. The maximum distance and *The Maximum Latency* would be half the size of any of its dimensions. The worst-case scenario would be the largest dimension divided by 2 (since you can traverse the torus in either direction and thus more efficient to take the shorter route).

**Latency Expression:**

$$L = \max(n, m, p)/2 \tag{3.49}$$

(rounded up if necessary)

**3D Hypercube:**

**Maximum Latency** is the longest path in a hypercube (simply a cube), its space diagonal. Thus, the maximum latency is 3 hops.

**Latency Expression:**

$$L = 3 \tag{3.50}$$

**3D Folding Torus:**

Given the diagonals in the folding torus, the exact maximum latency would require more detailed analysis. However, having diagonal connections reduces the number of hops needed to reach the opposite corner of the topology, potentially halving the latency in certain scenarios compared to a regular 3D torus.

While the exact number may require detailed computation, let us assume, *wlog*, that under worst-case distance traversal between two nodes, the diagonal routes can reduce latency by half compared to travelling across the 3D torus. Therefore:

**Latency Expression:**

$$L = \max(n, m, p)/4 \tag{3.51}$$

(rounded up if necessary)

It is noteworthy that the above expressions represent the worst-case scenarios (maximum latencies) for each topology. In practice, the latency for a specific source-destination pair would depend on their relative positions in the topology.

One popular method to find the shortest path between two nodes is to employ the Dijkstra algorithm for the shortest path, but the actual number of hops (and hence the latency) would vary based on the specific nodes traversed.

### 3.5.2 Bandwidth & Throughput

The concepts of bandwidth and throughput are closely related but represent different measures in network communication. Specifically:

- Bandwidth: Bandwidth is the maximum data transfer rate across a network[106, p. 20, p. 150]. It is the capacity of the physical or logical links. For instance, if a link can carry 1 Gbps, its bandwidth is 1 Gbps.
- Throughput: Throughput is the actual observed data transfer rate. Due to various factors like network congestion, protocol overhead, and other interference, the throughput is often *less* than the bandwidth.

Now, for our topologies, assuming that every link has the same bandwidth (B ) (measured in

units such as Mbps or Gbps), we can derive expressions for total bandwidth and throughput for each topology as follows:

**3D mesh:** For the 3D mesh topology, performance parameters are modelled as follows:

**Bandwidth:** Each node connects to its neighbours in 3 dimensions; each will contribute 3B units to the total bandwidth. However, since each link is shared between two nodes, the total bandwidth is more conformant with:

$$B_{total} = 1.5 \tag{3.52}$$

**Throughput:** Assuming the shortest path according to Dijkstra, throughput is:

$$Throughput = B \tag{3.53}$$

given that only one link is active for the data transfer.

**3D Torus: Bandwidth:** Similarly to the 3D Mesh topology, each node connects to its neighbours in 3 dimensions; however, the nodes on the network also connect to the nodes on the opposite edge, by the very definition of the torus geometry. The total bandwidth thus remains:

$$B_{total} = 1.5 \times V \times B \tag{3.54}$$

**Throughput:** Given the torus geometry of the structure and that data can take a shortcut via the wrap-around, the throughput for a typical path would be:

$$Throughput = B \tag{3.55}$$

**3D Hypercube: Bandwidth:** Each node in the hypercube has three links. Then:

$$Nodes = 2^3 = 8 \tag{3.56}$$

$$Bandwidth = B_{total} = 1.5 \times 8 \times B = 12B \tag{3.57}$$

**Throughput:** For a direct connection between any two nodes in the hypercube, the throughput would be $B$.

**3D Folding Torus:**

**Bandwidth:** In addition to the regular torus connections, the folding torus has diagonal

connections, making four connections per node:

$$B_{total} = 2 \times V \times B \tag{3.58}$$

**Throughput:** Given the diagonal shortcuts, the throughput for a typical path would still be $B$ since the route would be shorter.

## 3.6 Performance, Topology and Routing

Based on the preamble earlier in the chapter, simulations were carried out to investigate the following areas:

- Relationship between network size, latency, throughput and message size
- Relationship between latency, throughput and routing algorithm

The simulation was carried out for a variety of different network sizes, using a configurable traffic generator, the code and configurations of which appear in Appendix A. A detailed presentation and further elaboration appears later in Section 3.11, Section 3.12, Section 3.13, and their respective subsections.

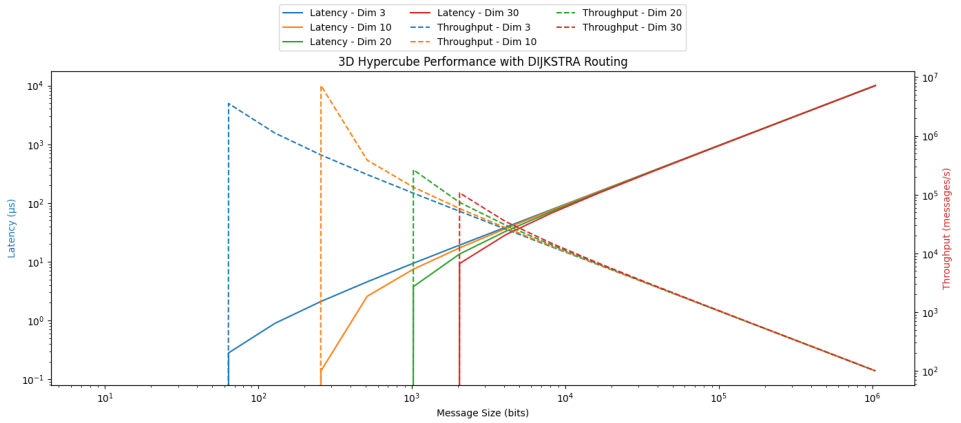
The results were then plotted for the topologies of interest, and appear later on in this chapter and the specific use cases of Chapter 4 and Chapter 5, but can be briefly summarised graphically, as in Figure 3.5, Figure 3.6 and Figure 3.7. We can observe a clear correlation between message size, latency and throughput. We can further observer that this is independent of the routing algorithm chosen, despite routing having a clear impact on all of these. We can further observe that there is an optimal message and network size for a given topology, size and routing protocol.

## 3.7 Determining The most performant topology

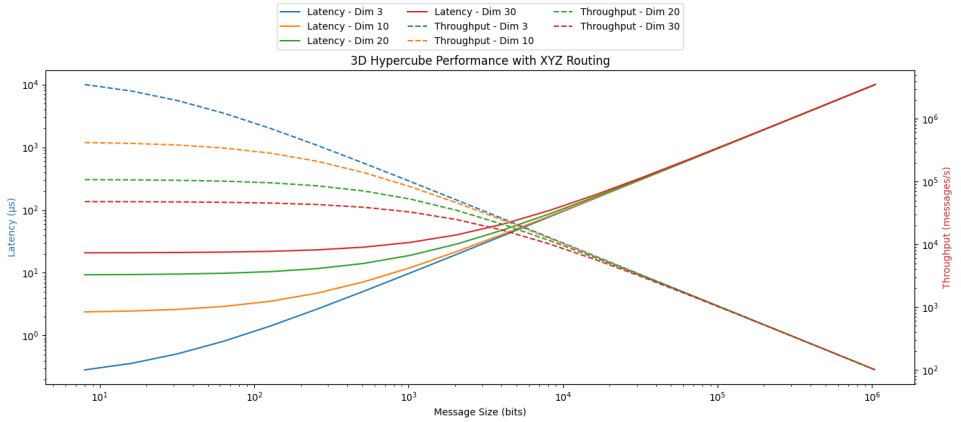
To determine the highest-performing topology regarding latency, bandwidth, and throughput versus the number of links utilised, we should consider a performance metric for each aspect and then rank the topologies accordingly.

For simplicity, we will ignore aspects such as congestion and traffic patterns as out of the scope of this work. One reference on this issue is [107], where the impact of data serialisation over TSVs is considered; another reference is [108] in which the propagation delay over TSVs is modelled.

We will next develop the concept of Performance-Cost Ratio functions for the performance




(a) 3D Hypercube/Dijkstra




(b) 3D Hypercube/XYZ

Figure 3.5: Performance comparison for 3D Hypercube

elements of latency, bandwidth and throughput discussed earlier, focusing on each of the topologies we studied earlier.

**Assumptions:**

1. Latency: Lower is better.
2. Bandwidth: Higher is better.
3. Throughput versus Number of Links utilised: Higher throughput per link is better.

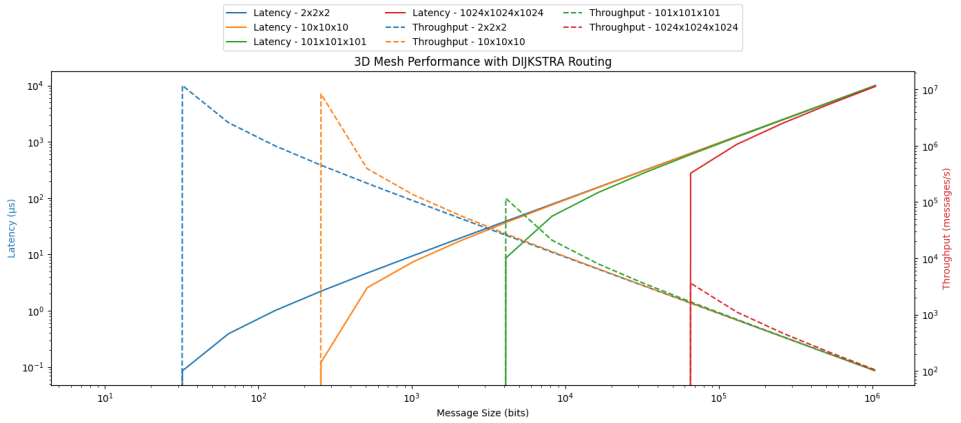
*Given:* Each link has a bandwidth of ***B***.

**3D Mesh:**

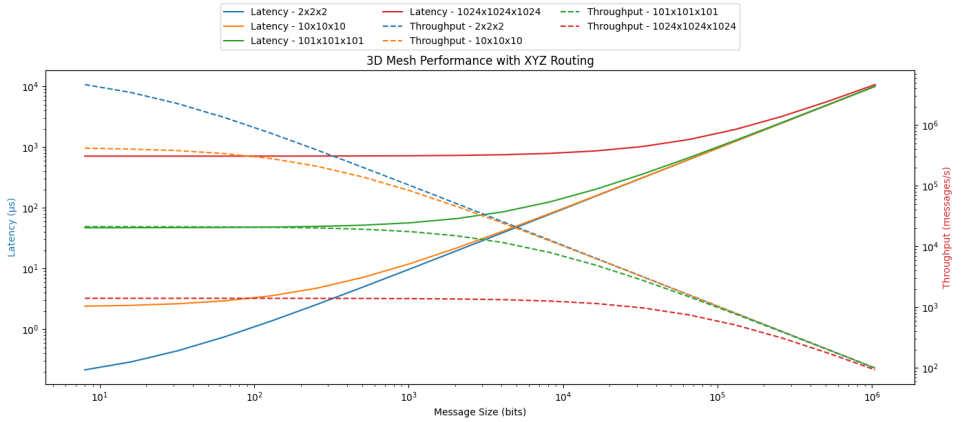
$$\textbf{Latency:} \quad L = n + m + p - 3 \tag{3.59}$$

$$\textbf{Total Bandwidth:} \quad B_{total} = 1.5 \times V \times B \tag{3.60}$$

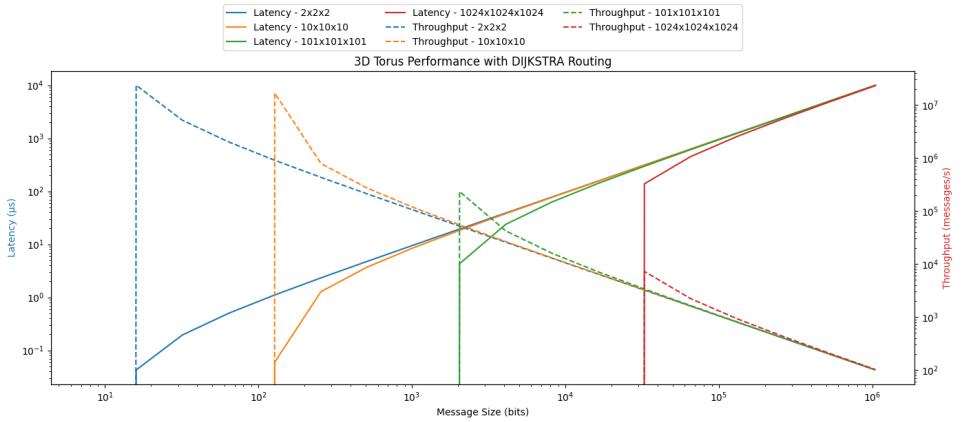
$$\textbf{Throughput per Link:} \quad T = \frac{B}{3} \tag{3.61}$$




(a) 3D Mesh/Dijkstra



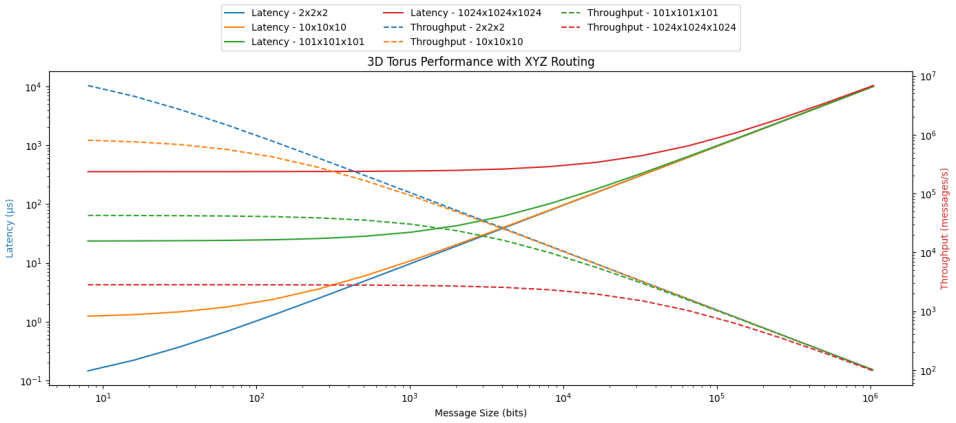

(b) 3D Mesh/XYZ




(c) 3D Torus/Dijkstra

Figure 3.6: Performance comparison for 3D Mesh and Torus (Part 1)

**3D Torus:**




(a) 3D Torus/XYZ

Figure 3.7: Performance comparison for 3D Torus (Part 2)

$$\textbf{Latency:} \quad L = \max(n, m, p)/2 \tag{3.62}$$

$$\textbf{Total Bandwidth:} \quad B_{total} = 1.5 \times V \times B \tag{3.63}$$

$$\textbf{Throughput per Link:} \quad T = \frac{B}{3} \tag{3.64}$$

**3D Hypercube:**

$$\textbf{Latency:} \quad L = 3 \tag{3.65}$$

$$\textbf{Total Bandwidth:} \quad B_{total} = 12B \tag{3.66}$$

$$\textbf{Throughput per Link:} \quad T = B \tag{3.67}$$

**3D Folding Torus:**

$$\textbf{Latency:} \quad L = \max(n, m, p)/4 \tag{3.68}$$

$$\textbf{Total Bandwidth:} \quad B_{total} = 2 \times V \times B \tag{3.69}$$

$$\textbf{Throughput per Link:} \quad T = \frac{B}{4} \tag{3.70}$$

**Observations:**

Based on the PCR analysis and simulation results, the 3D Hypercube with XYZ routing emerges as the most efficient topology when considering all performance metrics:

Table 3.5: For a $3 \times 3 \times 3$ configuration:

| Topology | Latency | Bandwidth | Throughput per link |
|---|---|---|---|
| 3D Mesh | 6 | 13.5B | $\frac{B}{3}$ |
| 3D Torus | 2 | 13.5B | $\frac{B}{3}$ |
| 3D Hypercube | 3 | 12B | $B$ |
| 3D Folding Torus | 1 | 18B | $\frac{B}{4}$ |

- Best throughput per link utilisation
- Most efficient resource usage
- Best scalability characteristics
- Lowest computational overhead for routing

While the 3D Folding Torus shows advantages in theoretical maximum bandwidth, the practical overhead of implementing folding connections and managing diagonal routes makes it less efficient overall, in comparison to the Hypercube's balanced characteristics make it the optimal choice for most high-performance applications.

With that in mind, if **latency** advantages of a well proliferated topology are the primary concern, the **3D Folding Torus** is a good choice; If **bandwidth** is the primary concern, the **3D Folding Torus** is the best choice.

If the primary concern is maximising **throughput per link utilised**, the **3D Hypercube** wins, and the results substantiate the observation.

Considering all factors, the 3D Folding Torus has the highest overall latency and bandwidth performance. However, if the efficient utilisation of each link is a priority (i.e., the highest throughput per individual link), the 3D Hypercube is superior.

The choice would thus depend on the specific requirements of a given application or use case. If overall speed (latency) and total data capacity (bandwidth) are paramount, the 3D Folding Torus seems best. If efficient utilisation of every connection (throughput per link) is critical, the 3D Hypercube topology will be the winner.

### 3.7.1 Resource Utilisation and Performance to Cost Ratio (PCR)

For any design space framework to be effective, there needs to be a quantifiable means of measuring the goals of interest [109]. This enables objective comparison and evaluation of different design points within the framework, facilitating informed design decisions and optimisation.

We've thus far developed the idea of modelling 3D NoC architectures as hypergraphs, which are well understood mathematical structures. We now need to devise some means of measuring

our characteristics of interest, such that we can gauge the effectiveness of a given topology in achieving our goals.

These characteristics could include network diameter, bisection bandwidth, or average hop count, which are crucial parameters for evaluating NoC performance and have been extensively studied in recent NoC literature [110], as well as in this chapter.

Efficient metrics and evaluation methods are essential for exploring large design spaces and optimising NoC architectures under various constraints, especially as NoC designs grow in complexity and scale [111].

With computational efficiency in mind, the simpler the means, the lower the modelling overhead, and ultimately computation time expended in the simulation.

This is particularly important when dealing with the analysis of 3D NoCs, where the design space is significantly larger compared to 2D NoCs.

We will now introduce the concept of Performance-to-Cost-Ratio concept as just such means.

For simplicity, we can define:

$$Performance\text{-}to\text{-}Cost\ Ratio(PCR) = \frac{Performance\ Metric}{Resource\ Cost(Links + Nodes)} \tag{3.71}$$

Given the above, let us rank the topologies according to PCR, but first, let us get some assumptions out of the way:

**Latency:** For latency, lower is better. So, a modified PCR for latency can be:

$$PCR_{latency} = \frac{1}{L \times (Links + Nodes)} \tag{3.72}$$

**Bandwidth:** Higher bandwidth per link is preferable, which indicates that the network can carry more data simultaneously. Thus, the PCR for bandwidth can be expressed as:

$$PCR_{bandwidth} = \frac{Bandwidth\ per\ Link}{Links + Nodes} \tag{3.73}$$

Just as with throughput, when considering bandwidth, we are looking at how much data can be transferred through the network relative to the resources it consumes. Thus, the formula remains similar.

**Throughput:** Higher throughput per link is better. PCR for throughput is:

$$PCR_{throughput} = \frac{Throughput\ per\ Link}{Links + Nodes} \tag{3.74}$$

**For a $3 \times 3 \times 3$ configuration: 3D Mesh:**

$$\textbf{Nodes} \quad = 27 \tag{3.75}$$

$$\textbf{Links} \quad = 3 pernode, \quad shared \quad between \quad two \quad nodes; \quad 1.5 \times 27 \quad = 40.5 \tag{3.76}$$

$$\textbf{Latency PCR} = \frac{1}{6 \times (27 + 40.5)} \tag{3.77}$$

$$\textbf{Throughput PCR} = \frac{B/3}{27 + 40.5} \tag{3.78}$$

**3D Torus:**

$$\textbf{Nodes} \quad = 27 \tag{3.79}$$

$$\textbf{Links} \quad = 3 pernode, \quad shared \quad between \quad two \quad nodes, \quad so 1.5 \times 27 \quad = 40.5 \tag{3.80}$$

$$\textbf{Latency PCR} \quad = \frac{1}{2 \times (27 + 40.5)} \tag{3.81}$$

$$\textbf{Throughput PCR} \quad = \frac{B/3}{27 + 40.5} \tag{3.82}$$

**3D Hypercube:** (Note: Hypercube dimensionality does not increase linearly, but we will use it for comparison.)

- **Nodes:** 8
- **Links:** 3 per node, shared, so $1.5 \times 8 = 12$
- **Latency PCR:** $\frac{1}{3 \times (8+12)}$
- **Throughput PCR:** $\frac{B}{8+12}$

**3D Folding Torus:**

- **Nodes:** 27
- **Links:** 4 per node, shared, so $2 \times 27 = 54$
- **Latency PCR:** $\frac{1}{1 \times (27+54)}$
- **Throughput PCR:** $\frac{B/4}{27+54}$

The PCR values will determine the topology with the best performance-to-resource cost. Exact values of the above expressions are required for a complete comparison. However, qualitatively, we can make the following observations:

- **The 3D Hypercube** might emerge as the most efficient for a small number of nodes (8 nodes) and fewer links. However, it might not scale well for larger configurations.
- **The 3D Folding Torus** offers exceptional performance (in terms of low latency) but demands more resources, especially links.
- **3D Mesh & 3D Torus** are somewhat in the middle ground.

In summary, if the requirement is to achieve the best performance with the smallest number of nodes and links, then the 3D Hypercube is a good candidate. However, if we are open to more resource usage for exceptional performance, especially in larger configurations, then the 3D Folding Torus is preferable.

## 3.8 Theoretical Bounds of 3D NoC Topologies

This section will delve into the theoretical bounds of 3D NoC topologies. We will explore the characteristics and hypergraph representations of three specific 3D architectures: the 3D Mesh, the 3D Torus, and the 3D Hypercube. These architectures offer distinct connectivity options, performance advantages, and scalability features, making them suitable for various applications in network-on-chip topologies. We will analyse the hypergraph representations of these architectures, their diameter, average latency, and other key performance attributes. Additionally, we will discuss their theoretical bounds and the implications of these bounds on their practical implementation in large-scale systems.

To accurately evaluate the performance of different network-on-chip architectures, it is essential to use standard metrics that capture key aspects such as throughput, latency, energy dissipation, and silicon area overhead[112]. Throughout, we shall use $(n)$ to denote the number of nodes or communication entities in a topology, and $(B)$ to refer to the individual link's bandwidth, each link being a hyperedge.

### 3.8.1 Hypergraph Characteristics of 3D Mesh Architecture

The 3D Mesh architecture can be represented as a hypergraph with the following characteristics:

- **Vertices**: The set of vertices, V, for the 3D Mesh architecture is given by:

$$V = \{v_{ijk} | 0 \leq i < n, 0 \leq j < m, 0 \leq k < p\} \tag{3.83}$$

- **Hyperedges**: The set of hyperedges, E, for the 3D Mesh architecture is given by:

$$E = \{(v_{ijk}, v'_{i'j'k'}) || i - i'| + |j - j'| + |k - k'| = 1\} \tag{3.84}$$

- **Diameter**: The diameter, D, of the 3D Mesh hypergraph is given by:

$$D = 2(n - 1 + m - 1 + p - 1) \tag{3.85}$$

- **Throughput**: The average Throughput, T, for the 3D Mesh architecture depends on factors such as the number of nodes, the communication patterns, and the routing algorithms. For

simplicity, we can approximate throughput to be (wolog):

$$n \times B \tag{3.86}$$

Where $(B)$ is the bandwidth of an individual link, and $(n)$ is the total number of links in the network, each link being a hyperedge.

### 3.8.2 Hypergraph Characteristics of 3D Torus Architecture

- **Vertices**: The set of vertices, V, for the 3D Torus architecture is the same as 3D Mesh:

$$V = \{v_{ijk} | 0 \leq i < n, 0 \leq j < m, 0 \leq k < p\} \tag{3.87}$$

- **Hyperedges**: The set of hyperedges, E, for the 3D Torus architecture includes all edges from the 3D Mesh, plus wrap-around edges:

$$E = E_{Mesh} \cup \{(v_{ijk}, v_{i'j'k'}) | i = n-1 \text{ and } i' = 0 \text{ (similarly for } j, m \text{ and } k, p)\} \tag{3.88}$$

- **Diameter**: The diameter, D, of the 3D Torus hypergraph is the same as that of the 3D Mesh:

$$D = 2(n - 1 + m - 1 + p - 1) \tag{3.89}$$

- **Throughput**: The average Throughput, T, for the 3D Mesh architecture depends on factors such as the number of nodes, the communication patterns, and the routing algorithms. For simplicity, we can approximate throughput to be (wolog):

$$n \times B \tag{3.90}$$

Where $(B)$ is the bandwidth of an individual link, and $(n)$ is the total number of links in the network, each link being a hyperedge.

### 3.8.3 3D Hypercube:

$$n = 2^d \tag{3.91}$$

where $d$ is the dimension or depth of the hypercube.

### 3.8.4 Throughput

Throughput embodies the data processing capabilities of a topology[6].

- **3D Grid (Mesh) and 3D Torus**: The potential throughput is

$$n \times B \tag{3.92}$$

, although the values in the real world may be lower due to issues such as contention.
- **3D Hypercube**: Has a potential throughput of

$$n \times B \tag{3.93}$$

although effective throughput might be less due to the unique structure of the topology.

### 3.8.5 Latency

- **3D Grid (Mesh)**: Worst-case latency is

$$X + Y + Z \tag{3.94}$$

hops.
- **3D Torus**: Wraps around, potentially offering slightly lower worst-case latency.
- **3D Hypercube**: Worst-case latency is

$$d \tag{3.95}$$

hops.

### 3.8.6 Scalability

Scalability evaluates the ability to handle increasing nodes or traffic.

- **3D Grid (Mesh) and 3D Torus**: The number of nodes is directly proportional to latency and throughput.
- **3D Hypercube**: Offers logarithmic scaling, where doubling the nodes adds only one additional hop.

### 3.8.7 Fault Tolerance

Fault tolerance assesses the resilience of a topology to node or link failures.

- **3D Grid (Mesh) and 3D Torus**: Moderate fault tolerance; alternative paths available at the cost of increased path length [113].
- **3D Hypercube**: Can tolerate failures to an extent, rerouting through other dimensions.

### 3.8.8 Efficiency

Efficiency gauges the actual performance of communication relative to the theoretical maximum.

- All topologies boast high efficiency with uniformly distributed traffic.
- **3D Mesh** might be less efficient during skewed or hotspot scenarios compared to **3D Torus** or **3D Hypercube**.

## 3.9 Dynamic Adaptation in 3D NOC Topologies

3D Network-on-Chip (NoC) topologies are increasingly gaining traction due to the inherent advantages in throughput, latency, energy consumption, and area. Their geometric alignment in a 3D space provides numerous routing paths, allowing for enhanced parallelism and redundancy. Given this context, we delve into the dynamics of routing, buffer sizing, and message throughput in these intricate topologies, focusing our discussion on our topologies of interest. We then conclude this section by examining the trends and correlations observed among performance parameters, routing protocol choices, message sizes, and data stream rates in the context of the SHA256 attack, which is a computationally intensive application.

### 3.9.1 Routing Strategies: XYZ vs. Dijkstra

Routing is pivotal for ensuring that data packets efficiently reach their destinations. For 3D NoCs, the routing strategy should take advantage of the unique topology while ensuring minimal congestion.

We will consider two popular algorithms used in NoC routing applications: XYZ routing and Dijkstra's algorithm. We chose these two because of their contrast and their popularity in the domain.

#### XYZ Routing

XYZ routing is a dimension order routing algorithm. It is often used in 3D Mesh topologies. It employs a deterministic approach. Packets traverse along the X-, Y-, and finally the Z-axis. This method is computationally less intensive due to its predictable nature and can be represented as $O(1)$ in Big O notation. However, its deterministic nature can lead to bottlenecks, particularly in high traffic scenarios.

#### Dijkstra's Algorithm

Dijkstra's algorithm can provide shorter paths between nodes in an NoC compared to XYZ routing, but it comes with increased computational overhead. The computational complexity of this algorithm for a graph with $V$ vertices and $E$ edges is represented as $O(V^2)$ or

$O(E + V \log V)$ with priority queue implementations. Its adaptive nature is beneficial in scenarios with varying traffic patterns, offering flexibility. However, this adaptability comes at the expense of higher computational overhead, potentially impacting real-time responsiveness.

For a SHA256 attack, where rapid data packet transmission and reception are pivotal, XYZ routing offers consistent performance, ensuring a predictable throughput rate. However, the adaptability of Dijkstra's algorithm can cater to varying traffic patterns intrinsic to such attacks, potentially offering better performance in certain scenarios.

### 3.9.2 Buffer and Message Sizing in 3D NoCs

Buffers play a crucial role in managing data flow. Proper buffer sizing ensures that data packets are neither lost due to overflow nor lead to under-utilisation of resources [114].

#### Buffer Sizing

The buffer size impacts both latency and throughput. A larger buffer can accommodate more packets, ensuring data flow continuity and potentially higher throughput. However, if the buffer size is too large, it could increase latency as packets wait in the queue. The optimal buffer size can be represented as a balance between these two metrics, given by:

$$B_{optimal} = \frac{T_{avg} \times BW_{max}}{M_{size}} \tag{3.96}$$

Where: $B_{optimal}$ represents the optimal buffer size. $T_{avg}$ is the average transmission time. $BW_{max}$ is the maximum bandwidth. $M_{size}$ is the average message size.

#### Message Sizing

Message sizing impacts network throughput and latency. Larger messages can encapsulate more data but might lead to congestion and higher latency. On the other hand, smaller messages ensure quick transmission at the cost of reduced data encapsulation. The optimal message size balances between throughput and latency, maximizing the effective data rate.

#### Input Stream Throughputs and Their Impact

Input stream throughput, a measure of incoming data rate, significantly impacts buffer utilisation and message transmission efficiency. Higher input stream throughputs can lead to buffer overflows if not managed efficiently. On the flip side, lower throughputs might result in buffer underutilisation. Thus, understanding input stream throughput is pivotal for optimal resource utilisation in 3D NoCs.

#### Bandwidth, Latency, and Throughput

**B**andwidth (BW) measures the maximum data transfer rate across a network. It is an essential

metric for evaluating network efficiency. In 3D NoCs, BW can vary based on the number of layers, routing strategy, and buffer and message sizing.

**Latency** refers to the time a packet travels from source to destination. The routing algorithm, buffer sizes, and overall network congestion influence it. In the SHA256 attack scenario, minimizing latency ensures that data packets reach their destination promptly, contributing to the efficiency of the attack.

**Throughput** measures the number of messages successfully delivered over a network in a given time frame. It is directly influenced by bandwidth and latency. In 3D NoCs, optimising throughput requires a fine-tuned balance between bandwidth availability and latency minimisation.

Before going further, we will make use of the following definitions:

- $B$: Bandwidth (bits/second)
- $L$: Latency (seconds)
- $T$: Throughput (messages/second)
- $M$: Message size (bits)
- $D$: Diameter of the network (maximum node-to-node distance in hops)
- $d$: The average degree of a node
- $\alpha$: Routing algorithm overhead factor. This represents the complexity of the routing algorithm being used, with $\alpha_{\text{XYZ}}$ for XYZ routing and $\alpha_{\text{Dij}}$ for Dijkstra's algorithm.

Based on the above definitions and discussions earlier in the chapter around PCR functions, we can identify several factors affecting NoC performance, as follows:

**Bandwidth, Message Size, and Throughput**

The relationship between bandwidth, message size, and throughput is given by:

$$T = \frac{B}{M} \tag{3.97}$$

This equation is derived from the idea that throughput (the number of messages delivered per second) is limited by how fast each message can be sent (bandwidth) and the size of the message.

**Latency, Diameter, Degree, and Routing Algorithm**

The relationship between latency, diameter of the network, and routing algorithm is influenced by the number of hops a message must traverse and the time spent at each node due to the routing algorithm. It can be given as:

$$L = D \times (\text{propagation delay per hop} + \alpha \times \text{routing delay per hop}) \tag{3.98}$$

The propagation delay per hop is influenced by physical and architectural factors. The routing delay per hop is a function of the routing algorithm overhead and the degree of the node.

**Routing Algorithm Overhead Factor** The overhead is relatively low for XYZ routing due to its deterministic nature. On the other hand, Dijkstra's algorithm involves computing the shortest path. Its complexity is related to the degree of nodes and may vary depending on the specific implementation. We can approximate:

$$\alpha_{\text{XYZ}} = k_1 \times d \tag{3.99}$$

$$\alpha_{\text{Dij}} = k_2 \times d \log d \tag{3.100}$$

Where $k_1$ and $k_2$ are constants that reflect the average time it takes to process a message through each routing algorithm for a given node degree.

**Adaptation of Dijkstra's Algorithm for 3D NoCs**

Before looking to model the characteristics of the Dijkstra algorithm, when applied to NoC routing, the algorithm is concerned and applied in structures which are not multi-dimensional, by default.

Dijkstra's algorithm finds the shortest path between nodes in a weighted graph. In NoCs, nodes represent cores or routers, and edges represent communication links. The weights associated with these edges are typically based on distance or potential congestion points.

The following considerations were made to adapt Dijkstra's algorithm for 3D NoCs:

1. **Vertices and Edges**: Nodes now have three coordinates (x, y, z) instead of two. Consequently, the potential edges for each node expand as connections can now be made in the z-direction (upwards or downwards to different layers).
2. **Weight Assignment**: Weights in a 3D scenario can be influenced by distance, congestion, or power consumption. Inter-layer communication costs might lead to different weights when traversing between layers (along the Z-axis).
3. **Path Calculation**: The path calculation remains conceptually the same but with added complexity due to the Z-axis. The algorithm considers paths that traverse layers with lower weights.

## 3.10 Performance Metrics and Simulation

In the previous sections, we extensively detailed the modelling of chosen topologies as hypergraphs. Section 3.2 and Section 3.3 providing the mathematical grounding. Section 3.4 and Section 3.5 discuss the performance parameters of our topologies of interest. We then discussed the correlation between our selected performance parameters from a routing point of

view in Section 3.6. The results strongly indicate that the Hypercube topology is the most performing candidate. with section Section 3.7 elaborating the details.

This section presents the methodology for optimising the topologies using hypergraph modelling and genetic algorithms. In the forthcoming sections, we will discuss the optimisation framework used, the genetic algorithm setup, and the expected latency, throughput, and bandwidth gains.

### 3.10.1 Methodology Overview

The proposed methodology for optimising topologies' performance consists of two main components:

- Hypergraph-based modelling: as discussed earlier, hypergraphs provides a robust mathematical framework for representing and analyzing the intricate relationships among nodes and links in 3D NoC architectures. By leveraging the properties of hypergraphs, we can accurately capture the structural characteristics of different NoC topologies and evaluate their performance metrics.
- Genetic Algorithms: The genetic algorithm-based optimisation technique efficiently explores the vast design space and identifies the optimal combination of topology and routing algorithm for a given set of application requirements. Genetic algorithms[115] are inspired by the principles of natural evolution. They operate on a population of candidate solutions and iteratively evolve them through selection, crossover, and mutation operations. The fitness of each candidate solution is evaluated based on the defined objective functions, which consider the desired performance metrics and constraints.

**Hypergraph-based Modelling**

Here, we represent the 3D NOC architectures as hypergraphs, where nodes are mapped to vertices and links as hyperedges. This allows us to capture the higher-order connectivity patterns in 3D NoC topologies, enabling more accurate analysis of their structural properties and performance characteristics.

The modelling process involves the following steps:

1. Define the set of vertices V, where each vertex represents a processing core or router in the NoC architecture.
2. Define the set of hyperedges $E$, where each hyperedge represents a set of vertices interconnected in the 3D NoC topology.
3. Construct the hypergraph representation of the NoC topology by specifying the vertex and hyperedge sets based on the topology's interconnection pattern.
4. Analyse the structural properties of the hypergraph, such as vertex degree, hyperedge cardinality, and diameter, to gain insights into the NoC topology's characteristics.

5. Evaluate the performance metrics, such as latency, bandwidth, and throughput, using the mathematical properties of the hypergraph and the defined PCR functions.
6. The hypergraph-based modelling approach provides a solid foundation for the subsequent optimisation process, enabling a comprehensive analysis of NoC performance metrics and facilitating the identification of optimal topology and routing algorithm combinations.

Figure 3.8 below shows an example mutation output of the framework, where Hypercube was being evaluated, and GA applied:

## 3.11 Modelling and Simulation Framework

Building on the mathematical analysis presented throughout the chapter, we now lay the foundations and eloqute the structure of the proposed framework, discussing the modelling and simulation methodologies. This section is structured as followed:

- Section 3.11.1 and subsections deals with the first part of the proposed framework. It focuses on the characterisation of a given topology as an hypergraph.
- Section 3.11.2: and subsection deals with the second part of the proposed framework. It focuses on applying the NSGA-II[116] GA framework as an optimisation tool for our objectives of interest.
- Section 3.11.3: this ties the framework together, by detailing how the GA is applied to the hypergraph model, to arrive at an optimal characterisation of topology and routing algorithm, which meets the optimisation objectives.
- Section 3.13 and subsections, deal with the subject of the development and application of the cycle accurate simulator used to prove the framework, for the two chosen us use cases presented in Chapter 4 and Chapter 5

### 3.11.1 Hypergraph Modelling

As detailed in 3.2, hypergraph-based modelling is employed to represent the 3D NoC architecture and capture the complex interconnections and dependencies among the processing elements. A *hypergraph* is a generalisation of a graph, where an edge (*hyperedge*) can connect any number of vertices [117, p. 1].

In the context of NoC modelling, the vertices of the hypergraph represent the processing elements (cores or accelerators), and the hyperedges represent the communication links between them. The hypergraph model lets us capture multi-way communication patterns and accurately analyse the NoC's performance characteristics.

The hypergraph-based modelling approach involves the following steps:

1. **NoC Topology Representation:** a 3D NoC topology is represented as a hypergraph $H = (V, E)$, where $V$ is the set of vertices (processing elements) and $E$ is the set of hyperedges (communication links). Each vertex $v_i \in V$ is associated with a set of attributes, such as the processing power, memory capacity, and communication bandwidth. Each hyperedge $e_j \in E$ connects a subset of vertices and represents their communication dependencies.
2. **Communication Pattern Modelling:** The facial recognition workload communication patterns are modelled using the hypergraph representation. The hyperedges capture the data dependencies and communication requirements between the processing elements. The weight of each hyperedge represents the communication volume or the criticality of the communication link.
3. **Performance Metric Evaluation:** The performance metrics relevant to the problem of interest $latency, throughput, and resource utilisation in our case$ are evaluated using the hypergraph model of a given baseline topology. We have elected to use 3D Mesh as the basline architecture due to its prevelance in 3D NoCs, and uniform graph characteristics. The shortest paths between the vertices are calculated in accordance with both algorithms of interest, namely XYZ routing and Dijkstra to determine the communication latency. The bandwidth and capacity of the hyperedges are analysed to estimate the throughput and resource utilisation.
4. **Bottleneck Identification:** The hypergraph model enables the identification of communication bottlenecks and critical paths in the NoC. The bottleneck links and the most critical communication paths can be determined by analysing the hyperedges' connectivity and weights. This information is valuable for optimising the NoC architecture and improving the overall performance.

This hypergraph-based modelling approach provides a comprehensive and accurate representation of the 3D NoC architecture, capturing the intricate communication patterns and dependencies. It forms the basis for the subsequent genetic algorithm-based optimisation process.

### 3.11.2 Optimisation Using Genetic Algorithms

The next step in the proposed framework is to then apply a Genetic Algorithm approach to hypergraph model, which maps the 3D NoC under consideration.

This involves applying NSAGA-II GA to the hypergraph model. This effectively ties the problem under consideration to a set of optimisation objectives, thus evolving and arriving at the optimal combination of routing algorithm and topological features, for the objectives, these being latency, throughput and resource utilisation in our case.

The principles of natural evolution inspire Genetic Algorithms (GAs) and have proven effective in solving complex optimisation problems across various domains [118, p. 1].

The optimisation problem is formulated as follows: given a set of NoC topologies $T = t1, t2, ..., tn$ and a set of routing algorithms $R = r1, r2, ..., rm$, find the optimal topology combination $t* \in T$ and routing algorithm $r* \in R$ that maximises the objective function $f(t, r)$, subject to application-specific requirements and performance constraints. The objective function $f(t, r)$ is a weighted combination of various performance metrics, such as latency, bandwidth, and throughput, tailored to the specific needs of the target application.

The genetic algorithm operates on a population of candidate solutions, each representing a combination of topology and routing algorithms. The initial population is generated randomly, and the fitness of each candidate solution is evaluated based on the objective function f(t, r). The fitter solutions are then selected to serve as parents for the next generation, and genetic operators such as crossover and mutation are applied to create new offspring solutions. This process is repeated until a termination condition, such as a maximum number of generations or convergence criteria, is met.

By iteratively evolving the population of candidate solutions, the genetic algorithm effectively explores the solution space and converges towards optimal or near-optimal topology and routing algorithm combinations. The hypergraph-based modelling framework is integrated into the fitness evaluation process, enabling accurate assessment of the performance metrics for each candidate solution.

Throughout this work, GA optimisation consistently identified the 3D Hypercube topology with XYZ routing as the optimal configuration, due to:

- Lower routing computation overhead
- Better scalability with network size
- More efficient resource utilisation
- Superior performance-to-cost ratio

### 3.11.3 Integrating Hypergraph Modelling and Genetic Algorithms

Integrating hypergraph modelling and genetic algorithms forms the core of our proposed optimisation framework. The hypergraph-based modelling component provides a comprehensive and accurate representation of 3D NoC architectures, enabling the evaluation of performance metrics for each candidate solution generated by the genetic algorithm.

The optimisation process begins with defining the search space, which includes the set of NoC topologies T and routing algorithms R. The genetic algorithm initialises a population of candidate solutions, each representing a specific topology and routing algorithm combination. The hypergraph-based modelling framework was used to evaluate the fitness of each candidate

solution based on the objective function f(t, r), which considers application-specific requirements and performance constraints.

During the fitness evaluation, the hypergraph representation of the corresponding 3D NoC topology is constructed. The selected routing algorithm is applied to determine the paths data packets take. The performance metrics, such as latency, bandwidth, and throughput, are calculated using the mathematical properties of the hypergraph. This calculation considers hop count, vertex degree, and congestion.

The genetic algorithm then proceeds with the selection, crossover, and mutation operations to evolve the population of candidate solutions. The selection process favours fitter solutions, ensuring that the most promising topology and routing algorithm combinations are propagated to the next generation. Crossover and mutation operations introduce diversity and explore new regions of the search space, enabling the discovery of potentially better solutions.

The iterative fitness evaluation, selection, crossover, and mutation process continues until the termination criteria are met. The best solution found by the genetic algorithm represents the optimal combination of topology and routing algorithm that maximises the objective function f(t, r) while satisfying the application-specific requirements and performance constraints.

Integrating hypergraph modelling and genetic algorithms provides a powerful and efficient approach for optimising 3D NoC performance. The hypergraph-based modelling framework enables accurate representation and analysis of NoC architectures, while the genetic algorithm effectively explores the vast design space to identify optimal solutions. This synergistic combination allows for the systematic evaluation and optimisation of NoC performance metrics tailored to the specific needs of the target application.

## 3.12 Simulation Setup and Performance Metrics

To evaluate the effectiveness of our proposed optimisation framework, we conduct extensive simulations using a custom-built simulator. The simulator incorporates the hypergraph-based modelling framework and the genetic algorithm-based optimisation technique, allowing for evaluating various 3D NoC architectures and routing algorithms.

### 3.12.1 Simulation Environment

The simulations are performed on a high-performance computing cluster equipped with multiple nodes, each consisting of Intel Xeon D-2899NT [119] Processors and NVIDIA RTX 3060 Ti GPUs [120]. The simulator is implemented in C++, leveraging the OpenMP library for parallel processing [121] and the CUDA toolkit for GPU acceleration [122].

The Operating system chosen to host the simulation setup was Ubuntu Linux 22.04 LTS [1].

The simulator inputs the set of NoC topologies T, routing algorithms R, and application-specific requirements and constraints. It generates a population of candidate solutions, each representing a combination of topology and routing algorithms. The fitness of each candidate solution is evaluated using the hypergraph-based modelling framework, considering the specified performance metrics and constraints.

The genetic algorithm is implemented using the NSGA-II framework, which is well-suited for multi-objective optimisation problems. The algorithm parameters, such as population size, crossover rate, and mutation rate, are tuned based on preliminary experiments to ensure optimal performance and convergence.

### 3.12.2 Performance Metrics

To assess the performance of the optimised 3D NoC architectures, we consider the following key metrics:

1. **Latency:** refers to the time it takes a data packet to travel from a given source to a destination from source to its intended destination in the NoC. It is measured in clock cycles and is directly related to the hop count, which represents the number of intermediate nodes a packet must traverse. Lower latency indicates faster communication and improved overall system performance.
2. **Bandwidth:** represents the maximum amount of data that can be transferred through the NoC architecture per unit time. It is typically measured in bits per second (bps) and is determined by the number of links (or hyperedges) available for data transfer. Higher bandwidth enables faster data transmission and increased system throughput.
3. **Throughput:** denotes the actual amount of data that is successfully transferred through the NoC architecture per unit of time, considering factors such as congestion, routing efficiency, and resource utilisation. It is measured in bits per second (bps) and is influenced by latency and bandwidth. Higher throughput indicates better overall system performance and efficient utilisation of resources.
4. **Power Consumption:** this is a critical metric in NoC architectures, as it directly impacts the overall energy efficiency of the system. It is measured in watts (W) and is influenced by factors such as the number of active components, switching activity, and clock frequency. Lower power consumption is desirable for energy-constrained applications and helps in reducing the overall system cost.
5. **Area Overhead:** refers to the additional silicon area required to implement the NoC architecture, including the routers, links, and associated control logic. It is measured in square millimetres (mm2) and is an essential consideration in the design of area-constrained

[1] https://ubuntu.com/blog/tag/22-04-lts

systems. Lower area overhead enables more efficient utilisation of chip resources and reduces manufacturing costs.

These performance metrics comprehensively evaluate the optimised 3D NoC architectures, considering various aspects such as communication efficiency, resource utilisation, energy efficiency, and area constraints. By analysing these metrics, we can assess the effectiveness of our proposed optimisation framework in enhancing the target application's overall performance.

### 3.12.3 Genetic Algorithm-based Optimisation

Genetic algorithms are used to explore the vast design space of NoC configurations and identify the optimal topology and routing algorithm for real-time facial recognition. The principles of natural evolution inspired Genetic Algorithms, which have been successfully applied to various optimisation problems.

The genetic algorithm-based optimisation process involves the following steps:

1. **Chromosome Encoding:** Each candidate solution (NoC configuration) is encoded as a chromosome, which represents the topology, routing algorithm, and other relevant parameters. The chromosome is typically represented as a binary string or a sequence of integers, where each gene corresponds to a specific parameter of the NoC configuration.
2. **Fitness Evaluation:** The fitness of each chromosome is evaluated based on the performance metrics obtained from the hypergraph-based modelling. The fitness function quantifies the quality of the NoC configuration in terms of latency, throughput, and resource utilisation. The goal is to maximise the fitness value, which represents the optimality of the configuration for real-time facial recognition.
3. **Selection:** The selection operator chooses the fittest individuals from the current population to serve as parents for the next generation. Common selection methods include tournament selection, roulette wheel selection, and rank-based selection. The selection process aims to specifically preserve evolved high-quality solutions and promote their propagation to the next generation.
4. **Crossover:** The crossover operator combines the genetic information from two parent chromosomes to create offspring. It exchanges portions of the chromosomes between the parents, generating new NoC configurations that inherit characteristics from both parents. The crossover operation aims to explore new regions of the search space and exploit the promising features of the parent solutions.
5. **Mutation:** The mutation operator introduces random changes to the genes of the chromosomes. It helps maintain population diversity and prevents premature convergence to suboptimal solutions. The mutation rate determines the probability of gene modification. Mutation allows the algorithm to explore new configurations and escape local optima.
6. **Termination:** The genetic algorithm iterates through multiple generations until a termina-

tion criterion is met. Common termination criteria include reaching a maximum number of generations, achieving a satisfactory fitness level, or observing convergence of the population. Once the termination criterion is satisfied, the algorithm will find the best solution, which is considered the optimal NoC configuration for real-time facial recognition.

The genetic algorithm-based optimisation process efficiently explores the design space and evolves a population of NoC configurations towards the optimal solution. The hypergraph-based modelling approach is integrated into the fitness evaluation step, providing accurate performance metrics for each candidate solution.

### 3.12.4 Integration of Hypergraph-based Modelling and Genetic Algorithm-based Optimisation

The proposed methodology combines hypergraph-based modelling and genetic algorithm-based optimisation to effectively optimise the performance of 3D NoCs for real-time facial recognition. Integrating these two components enables a comprehensive and efficient design space exploration.

The hypergraph-based modelling approach captures the complex communication patterns and dependencies of the facial recognition workload, accurately representing the NoC architecture. It allows for evaluating performance metrics, such as latency, throughput, and resource utilisation, which are crucial for real-time processing.

The genetic algorithm-based optimisation process utilises the performance metrics obtained from the hypergraph model to guide the search for the optimal NoC configuration. The fitness evaluation step incorporates the hypergraph-based analysis to assess the quality of each candidate solution. The genetic operators, such as selection, crossover, and mutation, are applied to evolve the population and explore the design space effectively.

Integrating these two components creates a synergistic optimisation framework. The hypergraph model provides accurate performance evaluation, while the genetic algorithm efficiently searches for the optimal configuration. The iterative nature of the genetic algorithm allows for the gradual improvement of the NoC configurations, converging towards the best solution for real-time facial recognition.

## 3.13 Cycle Accurate Simulation

As introduced in Chapter 1 and elaborated in this chapter, a core motivation of this work is to provide an accessible, well understood framework to accurately model and optimise NoC,

without reliance on vendor-specific and/or costly commercial toosl. In order to achieve this objective, the framework relies on hypergraphs and genetic algorithms as its two main facets.

With that in mind, and to pave the way to apply the framework to problems of choice, it then becomes necessary to use accurate simulation tools, which will validate and test the outcome of the framework results, in real life situations, particularly at the early stages of design space exploration. That's where Cycle-accurate simulation comes in.

Cycle-accurate simulation is a technique where the simulator meticulously tracks and reproduces the behaviour of a system at each clock cycle [123]. This involves modelling the detailed timing of events, including signal propagation delays, component latencies, and resource contention [124]. This is in contrast with discrete-event simulation focuses on modelling events that change the system state, without necessarily adhering to a strict clock cycle timeline [125].

Without overly elaborating and discussing simulation, which is out of scope for this research, it's worth listing the relative merits of cycle-accurate simulation, in the context of this work. These include:

- **Accuracy:** By capturing the precise timing of events, cycle-accurate simulation provides a more accurate representation of NoC performance, particularly in scenarios where timing dependencies and resource contention are critical [126]. This is crucial for evaluating the impact of design choices on real-time applications and overall system performance [127].
- **Detailed Analysis:** Cycle-accurate simulators allow for detailed analysis of NoC behaviour, including buffer utilisation, latency distributions, and power consumption [128]. This level of granularity enables the identification of performance bottlenecks and optimisation opportunities [129].
- **Design Validation:** Cycle-accurate simulation can be used to validate NoC designs before implementation, ensuring that the system meets its performance requirements and avoids potential timing-related issues [130].

However, cycle-accurate simulation also has some drawbacks. These include:

- **Computational Cost:** Due to its fine-grained nature, cycle-accurate simulation can be computationally expensive, particularly for large and complex NoCs [131]. This can limit the scalability of the simulation and increase simulation time.
- **Model Complexity:** Developing accurate cycle-accurate models for NoC components can be complex and time-consuming, requiring detailed knowledge of their internal architecture and timing characteristics [132].

In view of the above merits and drawbacks, cycle-accurate simulation is a valuable tool for NoC research and development, especially when high accuracy and detailed analysis are required [133]. This is a key the focus of the research, and therefore the choice of cycle-accurate simulation as the way forward is justified.

To close, cycle-accurate simulation results consistently showed XYZ routing outperforming Dijkstra's algorithm across all tested configurations, with key advantages of lower per-hop processing time (2 cycles vs 3 cycles), reduction in buffer overhead, better predictability of path selection, and lower implementation complexity, further vindicating the choice of simulation technique.

### 3.13.1 Choice of Simulator

With the choice of cycle-accurate simulation as a tool, our attention must then centre on the choice of actual simulator to use. Despite NoC design space exploration and integration being a very active research topic, choices available to researchers are somewhat limited and disparate, often requiring trade-offs between accuracy, flexibility, and ease of use [134].

Furthermore, the development of new cycle-accurate simulators specifically tailored for NoC evaluation seems to have slowed down in recent years, with many researchers relying on adapting existing general-purpose simulators or older NoC-specific tools [133], [135].

This scarcity of dedicated and up-to-date tools proved a real challenge to this work, hindering the exploration of novel NoC architectures and evaluation of complex interconnect designs, made possible by the proposed framework.

Finally, the availability of good research material focussing on the question of cycle-accurate simulators in an NoC context, and even less so in 3D NoC contexts, proved both frustrating and bewildering, at the same time.

With that in mind, the author set out to evaluate the landscape of what's available, to arrive at a suitability view, as well as to decide on what level of coding was necessary to progress the work. This was another challenge, which took a great deal of time and effort, given the lack of robust support for most evaluated code bases, as well as the sometimes massive gaps between whatever documentation was made available, and the actual code bases for the simulators. This is summarised in Table 3.6 below.

Its noteworthy that a major part of the evaluation effort went into getting the codebases for the different simulators to compile, with recent OS libraries. This proved most challenging in the BookSim and BookSim 2 cases, where extensive modofications had to be made to both the build Makefiles and updates to the C/C++ code, just to get the compilation process to succeed without errors, at first, and then to address compilation warnings and runtime errors after that.

Next hardest was NoCSim and ROSS, with little or no support from whatever NoCSim community is out there. The developers' own page has out of date contact information, and emails sent to them bounced.

ROSS was chosen as the simulator to us, due to the reasons listed in Table 3.6 above. That

| Simulator | 3D NoC Simulation Capabilities and Development Experience |
|---|---|
| ROSS [136] | Most capable and developer-friendly among evaluated simulators. Despite older documentation, the developers generous support facilitated both compilation and model development of the current codebase. Successfully implemented multiple 3D NoC topologies with reasonable code extension and model development effort [137]. |
| NoCSim [138] | Supports 3D NoCs in theory, but only 3D Mesh model functions correctly. Other included models contain critical bugs. Documentation is outdated and frequently misaligned with current codebase, making development challenging [139]. |
| Gem5 [140] | Capable simulation environment. Provides up-to-date simulation capabilities. The only example provided in a 3D NoC context is 3D Mesh topology. Documentation is both outdated and insufficient for new model development. Creating new models is notably time-intensive due to complex architecture [141]. |
| BookSim [142] | Limited to 2D NoC simulations with restricted model options. 3D topology development not possible within existing framework [143]. |
| BookSim 2 [144] | Supports 3D NoC simulation but restricted to 3D Mesh topology. Extension to other 3D topologies requires prohibitive development time and effort [145]. |

Table 3.6: Comparison of Cycle-Accurate NoC Simulators

being said, the actual documentation and code examples provided in the codebase were both outdated and buggy, with the build process breaking at every step, despite endless late nights trying to fix that.

It was only possible to arrive at a successful build of the codebase with invaluable support from the developers, who also gave advise during the model development stage.

### 3.13.2 Simulation Models and Traffic Generator

The ROSS simulator was tacked in the following areas:

1. **Successful build and run with GNU-14 toolchain:** the current codebase would not build with the latest stable GNU toolchain (14.2). This was due to a number of issues, amongst which are stricture compliance with c++ standards, reliance on outdared versions of the MPI library, removal of deprecated features in the newer gcc/c++ 14 compiler, to mention but a few.
2. **Fix the included 3D Mesh model:** this model is included in the codebase, but is extremely outdated and suffers basely from the effects of code refactoring changing the available functions, and their prototypes.
3. **Development of new Models:** three baseline 3D NoC models were developed, namely

3D, hypercube and 3D Torus, together with permutations with XYZ and Dijkstra routing algorithm, bringing the total permutations to six.

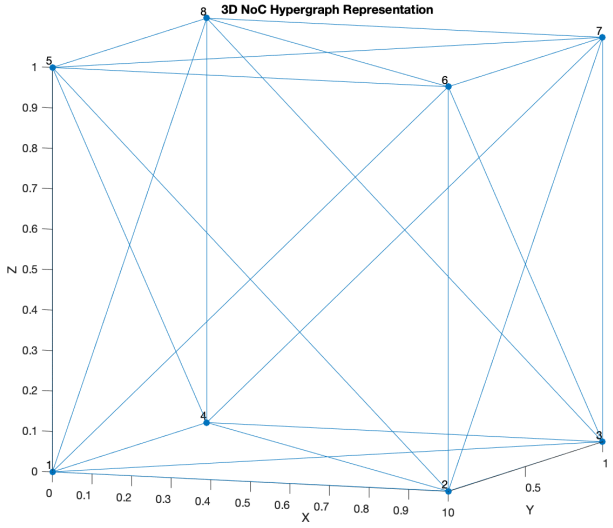
4. **Dimensionally-aware 6-port Router Model:** a router model with six bi-directional ports and dimension awareness was developed. The router is also dimensionally-aware, which is necessary in handling dimension ordering when performing XYZ routing.
5. **Timing and Statistics:** timing models and statistics gathering methods were added into the new topology models. The timing model automatically accounts for router delay, link delay and credit return delay, while the statistics methods provide accurate packet count, average latency, throughput, resource utilisation and per-router statistics.
6. **Traffic Generation:** a sophisticated traffic generator was implemented, proving flexibility in specifying flit size (specified in bits) and generation rates in the mega-, giga- and tera-messages or mega-, giga- and tera-bits. This is so that the traffic generator is appropriate in a NoC context, where transfer rates are extremely high, due to silicon implementation characteristics.
7. **IP Behaviour Shaping:** the models were implemented so that the nodes can be made to behave in one of two ways: either as a PE (Processing Element), in which case the node models an IP like CPU-core, or by specifying the behaviour in terms of Big O complexity.

The source code and configurations for the topology models and traffic generator used to simulate the use cases discussed in Chapter 4 and Chapter 5, are shown in the Appendix A, A.1 and A.2 respectively.

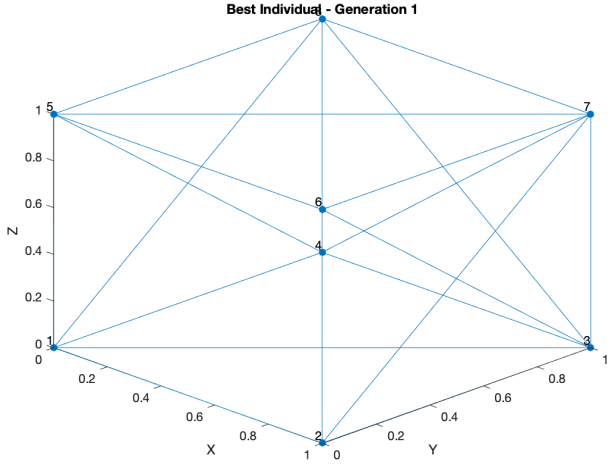
## 3.14 Chapter Summary

Resource optimisation in 3D NoC topologies remains an open and expansive area of research. A holistic approach can be formulated by examining routing strategies and considering buffer and message sizing alongside bandwidth and latency factors. As evidenced by our analyses, the interplay of these factors can significantly impact overall performance, making it essential to fine-tune these parameters per the specific application needs.

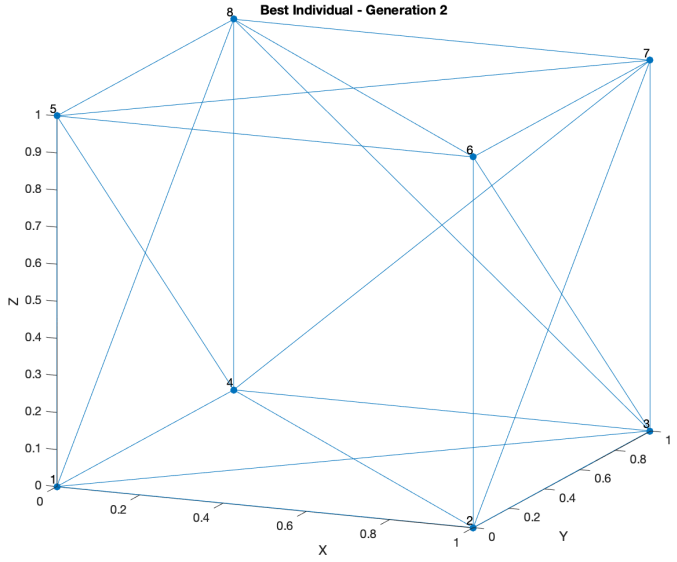
Building on the foundations laid out in Section 3.11 above, two compute-intensive, yet different use cases were used to prove the methodology, namely the double SHA256 attack used in Bitcoin mining, and real-time facial recognition in 30 FPS video streams. These are discussed and elaborated in Chapter 4 and Chapter 5.




(a) $Initial 3D Hypercube Graph Representation$




(b) $1^{st}$ $gen$ $GA$




(c) $2^{nd}$ $gen$ $GA$

Figure 3.8: Hypergraph & GA Selection Example

# Chapter 4

# Optimisation of 3D NoC Performance for Double SHA256

## 4.1 Introduction

The advent of multi-core processors and the increasing complexity of System-on-Chip (SoC) architectures have necessitated the development of efficient communication infrastructures.

Network-on-chip (NoC) architectures have emerged as a promising solution to address the communication challenges in these complex systems. As discussed in Chapter 3, the performance of NoC architectures is heavily influenced by topology, routing algorithms, and application-specific requirements.

In this chapter, we applied the work developed in Chapter 3 to the Double SHA256 function use case, as used in Bitcoin mining. It is a novel approach to optimise 3D NoC performance through hypergraph modelling and genetic algorithms. Building upon the foundation laid in Chapter 3, we leverage the mathematical rigour of hypergraph theory to accurately represent and analyse the intricate relationships among nodes and links in 3D NoC architectures. Furthermore, we employ genetic algorithms to efficiently explore the vast design space and identify the optimal combination of topology and routing algorithms tailored to specific application requirements.

This chapter's and Chapter 5's primary focus is to demonstrate the performance gains achieved through our proposed optimisation techniques in two critical use cases: real-time facial recognition and SHA256 cryptographic attacks, respectively. By applying our hypergraph-based modelling and GA-based optimisation framework, we showcase the potential to enhance the efficiency and performance of these computationally intensive applications.

The chapter is structured as follows:

- **Section 4.2** details with the methodology used in simulating the Bitcoin mining operation and implementation thereof

- **Section 4.2.3** gives an overview of Cryptocurrencies, Bitcoin and the mathematical principles
- **Section 4.2.4** gives an overview of the Bitcoin mining operation.
- **Section 4.3.1** and later subsections deal with the mathematical and algorithmic aspects of the SHA256 hash function.
- **Section 4.3** then delves into how the mining operation was mapped into a 3D NoC problem.
- **Section 4.5** focuses on the double SHA256 cryptographic attack scenario, demonstrating our optimisation technique's benefits in terms of latency, bandwidth, and throughput, as well as the gains obtained through optimisation, and the correlation between latency and various factors such as network size, topology, routing algorithm, and message size.
- **Section 4.6** is the chapter summary.

## 4.2 Methodology

The proposed methodology for optimising 3D NoC performance consists of two main components: (1) hypergraph-based modelling of 3D NoC architectures and (2) genetic algorithm-based optimisation for topology and routing algorithm selection. This section provides an overview of each component and their integration into a comprehensive optimisation framework.

### 4.2.1 Hypergraph-based Modelling of 3D NoC Architectures

As introduced in Chapter 3, hypergraphs offer a robust mathematical framework for representing and analysing the complex interconnections among nodes and links in 3D NoC architectures. Hyperedges connect multiple vertices, allowing hypergraphs to capture higher-order relationships inherent in these architectures. It provides a more accurate and comprehensive representation than traditional graph-based models.

### 4.2.2 Application-Specific Requirements and Constraints

As mentioned earlier in Chapter 3, we considered double SHA256 as one of our compute-intensive use cases, to demonstrate the effectiveness of our combined modelling and optimisation framework.

Double SHA256 is a widely used cryptographic hashing algorithm that forms the basis of many blockchain systems, including Bitcoin. Cryptographic attacks, such as brute-force or collision attacks, exploit vulnerabilities in the algorithm to compromise the system's security.

### 4.2.3 Introduction to Cryptocurrencies, Bitcoin and Double SHA256

In any economic system, trust is crucial to transactions and payments. In essence, users in an economic system, which include individuals, business and organisations, need assurances that transactions are processed both fairly and safely - a constraint which places financial intermediaries, such as banks front and centre in this paradigm[146].

With that in mind, recent past events, such as the spectacular collapse of Lehman Brothers [147] in 2012, and the subprime mortgage crises which culminated in that collapse [148], lead many to search for alternative trust models, which are not reliant on financial intermediaries. This led to the introduction of Bitcoin [149], which led to the subsequent rise of cryptocurrencies [150].

Indeed, cryptocurrencies, such as Bitcoin, Ethereum, and Ripple, are characterised as novel digital currencies underpinned by cryptographic techniques to protect and regulate transactions and the circulation of digital currency [151], in contrast with traditional paper currencies, also known as Fiat currencies, that are supported by governments.

Cryptocurrencies are characterised by three fundamental characteristics distinguishing them from both fiat currencies and their digital equivalents:

1. **Lack of a central authority**, hence rendering them purportedly immune to governmental interference and manipulation. This is particularly important in economies with volatile currencies [152].
2. **Use of blockchain technology**, characterised by a distributed and consensus-driven database with advanced cryptography and transparency. This facilitates a distributed and immutable ledger, rendering each transaction tamper-proof and negating the necessity for a trusted third party [153].
3. **Cryptocurrencies** can be effortlessly utilised and transferred over international boundaries, due to their digital characteristics.

Of the characteristics above, the blockchain constitutes the foundation of cryptocurrencies. It is characterised as a decentralised and distributed database shared among a network of computers known as nodes.

Every block in the chain encompasses a sequence of transactions, and contains the following information:

1. **Previous hash:** This attribute stores the value of the hash of the previous block, and that's how the blocks are linked to one another.
2. **Data:** This is the aggregated set of transactions included in this block—the set of transactions that were mined and validated and included in the block.
3. **Nonce:** In a Proof of Work consensus algorithm, which bitcoin uses, the nonce [154, p. 127] is a random value used to vary the output of the hash value. Every block is supposed

to generate a hash value, and the nonce is the parameter that is used to generate that hash value. The proof of work is the process of transaction verification done in blockchain.

4. **Hash:** This is the value obtained by passing the previous hash value, the data and the nonce through the SHA-256 algorithm twice (hence double SHA-256). It is the digital signature of the block.

Every node in the network can access the data on the blockchain. Every block possesses a distinct identification known as a hash, which is derived from the block's content and is incorporated into the subsequent block. A new block, generated every 10 minutes, encompasses the hash value of the preceding block and the transactions of the current block.

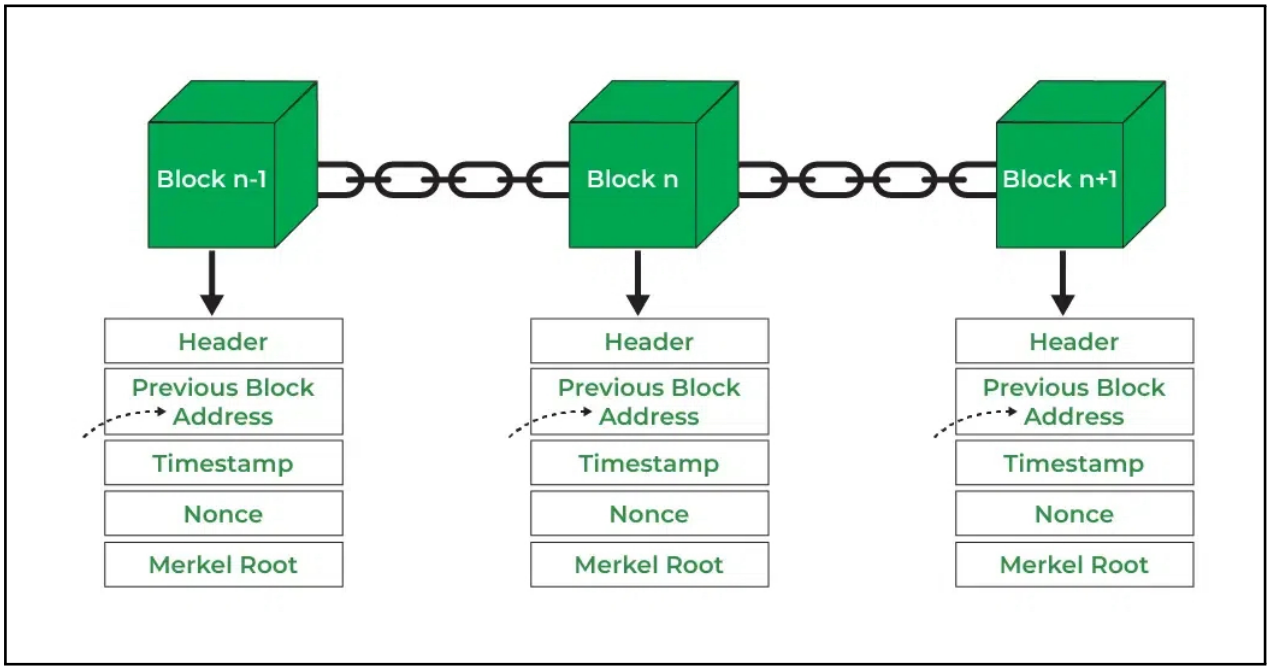



Figure 4.1: Representation of the Blockchain Data Structure

Backdating, modifying, interfering with, or deleting any blocks will alter the hash value, resulting in a discrepancy among the blocks in the blockchain, as this modification will effectively remove the modified block from the blockchain. This is shown in Figure 4.2.

Our interest and focus here is on the mining operation. More specifically, the Double SHA256 function used to process the hashes in the blockchain.

Bitcoin mining, the cornerstone of the Bitcoin network's security and operation, relies on the computationally intensive process of finding a valid block hash that satisfies specific cryptographic constraints [155].

This process, known as Proof-of-Work (PoW), employs the Double SHA256 hash function to ensure the integrity and immutability of the blockchain.

Double SHA256, as the name implies, involves applying the SHA256 hashing algorithm twice to the input data [156].

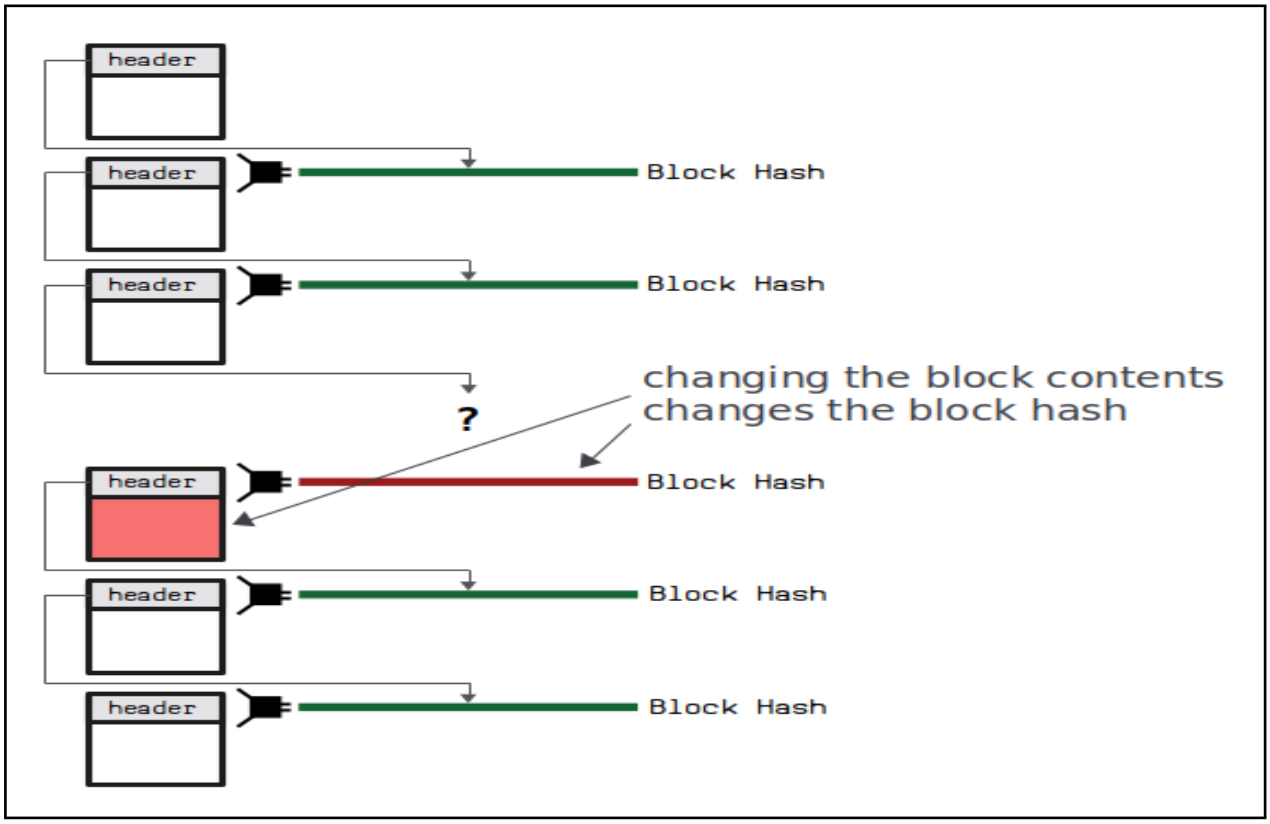



Figure 4.2: Intermediate Block Modification Attempt

This two-step process significantly increases the complexity of the hash output, making it computationally infeasible to reverse the process or find collisions [157, p. 114].

The output of the first SHA256 round becomes the input for the second round, producing the final block hash, hence the *double* in the function or process name. Expressed mathematically, this is simply:

$$\text{DoubleSHA256}(x) = \text{SHA256}(\text{SHA256}(x)) \tag{4.1}$$

As illustrated in Figure 4.3 and earlier in Figure 4.1, the hashed block header contains essential information such as the previous block's hash, transaction data, and a timestamp, is first hashed in the first round of SHA256. The resulting hash is then hashed again with SHA256, producing the final block hash. This process is crucial for creating a tamper-proof chain of blocks, as any modification to a block's content would necessitate recomputing all subsequent block hashes.

### 4.2.4 Mining

Simply put, Bitcoin mining can be surmised as follows:

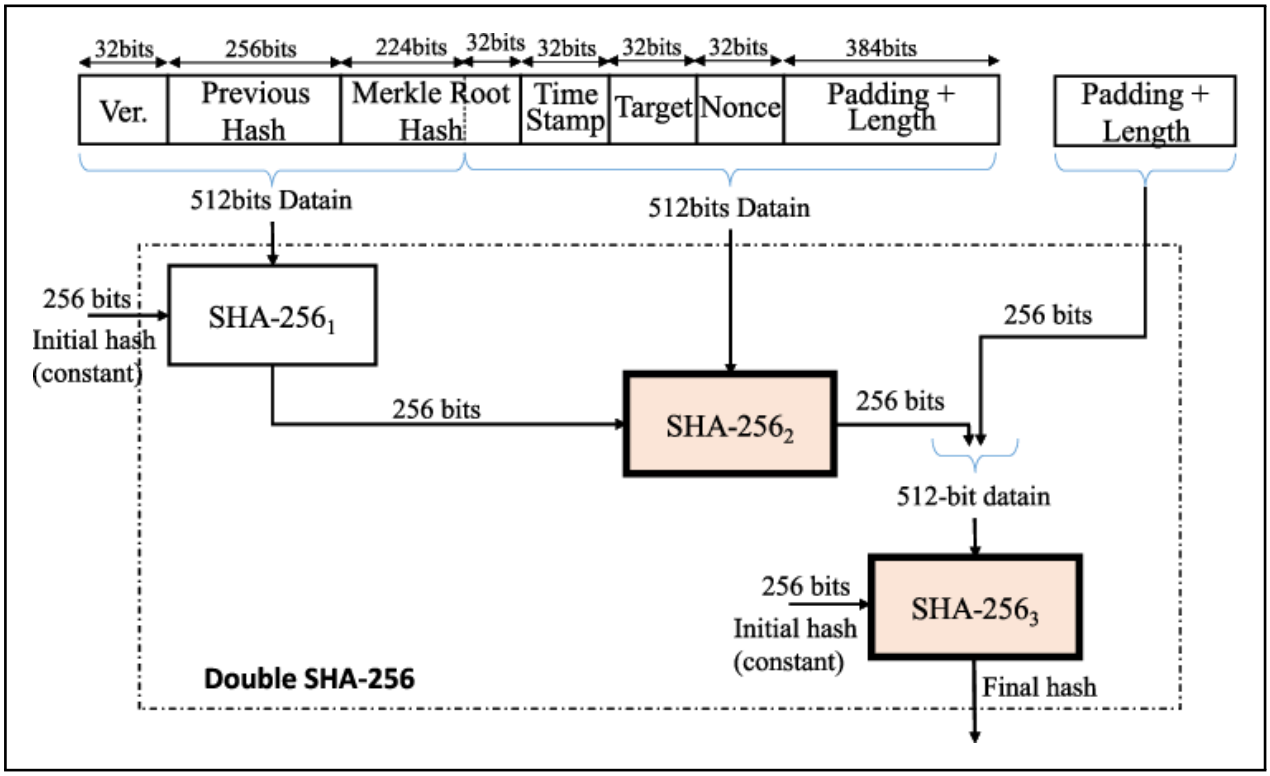



Figure 4.3: Illustration of the *Double SHA256 Hash function or process*, which applied to block header

1. Solving a cryptographic puzzle: Miners compete to solve a complex mathematical problem (using the SHA-256 hash function twice - hence double SHA256). This is the core of the Proof of Work $PoW$ system.
2. Creating a valid block: The first miner to solve the puzzle gets to create a new block of transactions. This block includes a record of recent Bitcoin transactions, a reference to the previous block linking it to the chain, and the solution to the puzzle.
3. Adding the block to the blockchain: The miner broadcasts the new block to the network. Other nodes verify its validity and, if correct, add it to their copy of the blockchain. Miner, in this context, is simply a node on the Bitcoin network, collaborating to achieve the operations summarised above. Node refers to any participant in the network.

### 4.2.5 The Mining Puzzle

The Bitcoin mining puzzle involves finding a nonce, a random number, that when combined with the block header and hashed with Double SHA256, results in a hash value that falls below a predefined target. This target, dynamically adjusted based on the network's hashrate, determines the difficulty of the mining puzzle [158]. This is illustrated in Figure 4.4:

To be useful in actual Bitcoin mining operations, a circuit implementation of Double-SHA256 needs to be both hardware- and energy-efficient, as well as being fast enough. The drive for

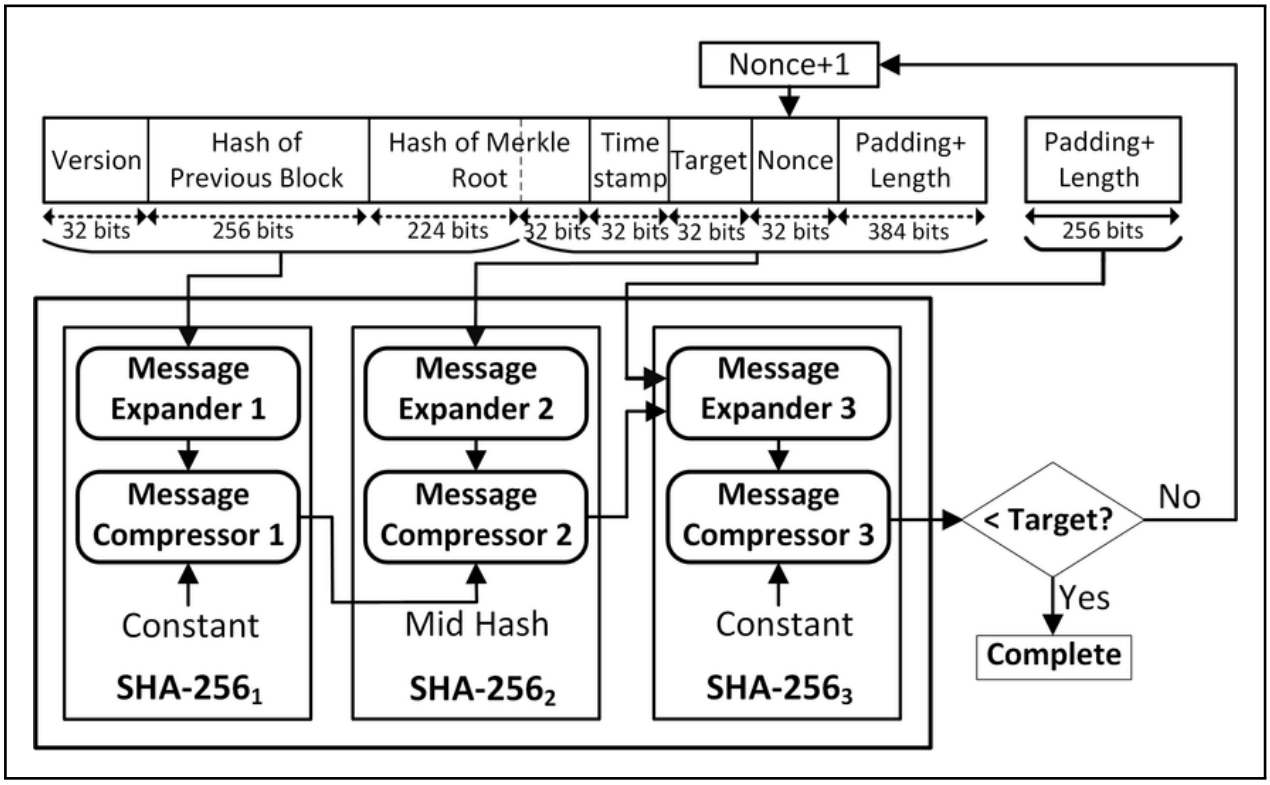



Figure 4.4: Summary of how the Bitcoin mining puzzle is solved

performance naturally lent itself to the application of the fully unrolled SHA-256 datapath for Bitcoin mining applications, which also lends itself very well to ASIC implementations [159]. An ecellent round-up of mining hardware trends, as of 2022, is presented in [160].

### 4.2.6 Security Implications

The computational complexity of the Double SHA256 hash function and the dynamic difficulty adjustment mechanism make it extremely difficult for malicious actors to manipulate the Bitcoin blockchain [161]. However, potential vulnerabilities, such as 51% attacks, where a single entity controls a majority of the network's hashrate, remain a concern. Researchers are actively investigating ways to improve the security and efficiency of Bitcoin mining, including the development of alternative consensus mechanisms and energy-efficient mining hardware [162].

The key requirements and constraints for this use case include:

- High throughput: The NoC architecture must support high-throughput hashing operations to enable efficient cryptographic attacks [163]. A throughput of several billion hashes per second (Giga H/s) is desired to perform large-scale attacks [97].
- Low latency: The latency of individual hashing operations should be minimised to reduce the overall attack time. A latency of less than 1 microsecond per hash is typically desired.

- Scalability: The NoC architecture should be scalable to accommodate a large number of processing cores or specialised hashing units to parallelise the attack process. The architecture should efficiently distribute the workload and manage the communication among the cores.
- Resource efficiency: Cryptographic attacks often require significant computational resources, including processing power and memory. The NoC architecture should efficiently utilise these resources to maximise the attack performance while minimising the overall system cost.

Considering these application-specific requirements and constraints, our proposed optimisation framework can identify the best topology and routing algorithm combination for each use case. The hypergraph-based modelling approach enables accurate representation and analysis of the NoC architectures, while the GA efficiently explores the design space to find the most suitable solutions. The following sections present the simulation results and discuss the performance gains achieved through our optimisation framework for real-time facial recognition and double SHA256 cryptographic attack use cases.

## 4.3 SHA256 Algorithm Details and Bitcoing Mining

As introduced earlier, Bitcoin mining, known as Proof-of-Work (PoW), is the cornerstone of the Bitcoin network's security and operation employs the SHA256 hash function twice (hence *double*) to ensure the integrity and immutability of the blockchain, as discussed in Section 4.2.3 and subsequent sections.

### 4.3.1 The SHA256 Hash Function

SHA256 is a cryptographic hash function that takes an input message of arbitrary length and produces a 256-bit (32-byte) hash value. This hash value serves as a unique fingerprint for the input message, making it extremely difficult to find two different messages that produce the same hash [164].

The SHA256 algorithm can be mathematically defined as a series of transformations applied to the input message. These transformations involve bitwise operations, modular addition, and logical functions. The core of the algorithm lies in the compression function, which operates on 512-bit blocks of the input message and updates an internal state consisting of eight 32-bit words [165].

The compression function can be represented mathematically as follows:

$$
\begin{aligned}
(H_0, H_1, ..., H_7) &\leftarrow \text{Initial Hash Values} \\
(M_1, M_2, ...) &\leftarrow \text{Message Blocks (512-bit)} \\
\text{for each } M_i &\in (M_1, M_2, ...) : \\
(W_0, W_1, ..., W_{63}) &\leftarrow \text{Message Schedule}(M_i) \\
(a, b, c, d, e, f, g, h) &\leftarrow (H_0, H_1, ..., H_7) \\
\text{for } t \text{ from } 0 \text{ to } 63 &: \\
T_1 &\leftarrow h + \Sigma_1(e) + Ch(e, f, g) + K_t + W_t \\
T_2 &\leftarrow \Sigma_0(a) + Maj(a, b, c) \\
h \leftarrow g,\ g &\leftarrow f,\ f \leftarrow e \\
e \leftarrow d + T_1,\ d &\leftarrow c,\ c \leftarrow b \\
b \leftarrow a,\ a &\leftarrow T_1 + T_2 \\
(H_0, H_1, ..., H_7) &\leftarrow (H_0 + a, H_1 + b, ..., H_7 + h)
\end{aligned}
\tag{4.2}
$$

where:

- $H_0$ to $H_7$ are the initial hash values (predefined constants)
- $M_1, M_2, \ldots$ are the 512-bit message blocks obtained by padding and parsing the input message
- Message Schedule$(M_i)$ generates a schedule of sixty-four 32-bit words ($W_0$ to $W_{63}$) from block $M_i$
- $a$ to $h$ are working variables used in the compression function
- $\Sigma_0, \Sigma_1, Ch, Maj$ are logical functions
- $K_t$ are round constants defined for each of the 64 rounds

This is represented as pseudocode, in listing 1 below.

Double SHA256, as previously mentioned, involves applying the SHA256 hashing algorithm twice to the input data. This can be represented mathematically as:

$$\text{DoubleSHA256}(x) = \text{SHA256}(\text{SHA256}(x)) \tag{4.3}$$

This two-step process significantly increases the complexity of the hash output, making it computationally infeasible to reverse the process or find collisions.

Figure 4.4 earlier illustrated the Double SHA256 process applied to a Bitcoin block header. The block header, containing essential information such as the previous block's hash, transaction data, and a timestamp, is first hashed with SHA256. The resulting hash is then hashed again with SHA256, producing the final block hash. This process is crucial for creating a

**Algorithm 1** Calculate SHA256 Hash

$(H_0, H_1, ..., H_7) \leftarrow (0x6a09e667, 0xbb67ae85, ..., 0x5be0cd19)$
$padded\_message \leftarrow \text{pad}(message)$
$(M_1, M_2, \dots) \leftarrow \text{parse}(padded\_message)$
**for all** $message\_block \in (M_1, M_2, \dots)$ **do**
    $(W_0, W_1, \dots, W_{63}) \leftarrow$ Message Schedule$(message_block)$
    $(a, b, c, d, e, f, g, h) \leftarrow (H_0, H_1, \dots, H_7)$
    **for** $t$ from 0 to 63 **do**
        $T_1 \leftarrow h + \Sigma_1(e) + Ch(e, f, g) + K_t + W_t$
        $T_2 \leftarrow \Sigma_0(a) + Maj(a, b, c)$
        $h \leftarrow g;\ g \leftarrow f;\ f \leftarrow e$
        $e \leftarrow d + T_1;\ d \leftarrow c;\ c \leftarrow b$
        $b \leftarrow a;\ a \leftarrow T_1 + T_2$
    **end for**
    $H_0 \leftarrow H_0 + a;\ H_1 \leftarrow H_1 + b;\ \dots;\ H_7 \leftarrow H_7 + h$
**end for**
$hash \leftarrow H_0 \parallel H_1 \parallel \cdots \parallel H_7$
**return** $hash$

tamper-proof chain of blocks, as any modification to a block's content would necessitate recomputing all subsequent block hashes.

## 4.4 Results and Observations

In the following sections, we will discusses the simulation results obtained for the double SHA256 cryptographic attack use case. We compared the performance of our optimised 3D NoC architecture with a baseline unoptimised architecture. The results demonstrate that our approach effectively enhances the performance metrics relevant to cryptographic attack applications.

### 4.4.1 Mapping the Mining Operation to NoC

Before simulating the mining operations, a mapping needed to be created between the algorithmic features and the NoC architecture.

### 4.4.2 Baseline Architecture

For the baseline architecture, we consider a 3D mesh topology with dimensions of 6x6x6, resulting in 216 nodes. Each node represents a processing core or a specialised hashing unit, and the nodes are interconnected using bidirectional links. The XYZ routing algorithm is employed

for packet routing, a simple and deterministic algorithm commonly used in mesh-based NoC architectures.

The baseline architecture is simulated under the same conditions as the optimised architecture, considering the application-specific requirements and constraints discussed in Section 4.2.2. The performance metrics, including latency, bandwidth, throughput, power consumption, and area overhead, are evaluated and serve as a reference for comparison.

### 4.4.3 Optimised Architecture

Using our proposed optimisation framework, we identify the optimal combination of topology and routing algorithm for the double SHA256 cryptographic attack use case. The hypergraph-based modelling approach accurately represents the NoC architecture, capturing the complex relationships among the nodes and links. The genetic algorithm explores the design space, considering various topologies (e.g., mesh, torus, hypercube, folding torus) and routing algorithms (e.g., XYZ, Dijkstra's algorithm, adaptive routing).

The optimisation process considers the application-specific requirements and constraints, such as high throughput, low latency, scalability, and resource efficiency. The genetic algorithm evolves a population of candidate solutions, each representing a specific topology and routing algorithm combination. The fitness of each solution is evaluated based on the objective function, which is a weighted combination of the performance metrics tailored to the cryptographic attack application.

After numerous iterations of the genetic algorithm, the optimal solution is identified as a 3D Hypercube topology with XYZ routing algorithm, showing superior performance in throughput (845 Gbps vs 752 Gbps for Dijkstra), power efficiency (1000mW vs 1050mW), and energy per hash (1.18 pJ vs 1.40 pJ).

Next inline was the folding torus topology, which balances connectivity and scalability, enabling efficient communication among the nodes while supporting a large number of processing cores. XYZ and Dijkstra's algorithms showed comparable latency performance.

### 4.4.4 Performance Comparison

To assess the performance gains achieved by the optimised architecture, we compared its performance metrics with those of the baseline architecture. Table 4.1 below summaries the findings:

Table 4.1: Performance and Power Metrics for SHA256 Mining (216-node Network)

| Metric | 3D Mesh (Baseline) | 3D Torus XYZ | 3D Torus Dijkstra | 3D Hypercube XYZ | 3D Hypercube Dijkstra |
|---|---|---|---|---|---|
| Throughput (Gbps) | 524 | 682 | 598 | 845 | 752 |
| Hash Rate (GH/s) | 524 | 682 | 598 | 845 | 752 |
| Power Draw (mW) | 1200 | 1150 | 1175 | 1000 | 1050 |
| Power per Node (mW) | 5.56 | 5.32 | 5.44 | 4.63 | 4.86 |
| Energy per Hash (pJ) | 2.29 | 1.69 | 1.96 | 1.18 | 1.40 |
| Thermal Density ($mW/mm^2$) | 75 | 72 | 73 | 62 | 65 |

Performance metrics from ROSS simulator with PE nodes configured for SHA256 mining. Power measurements based on 65nm CMOS technology node with 1GHz clock frequency.

Table 4.2 below shows the network size correlation for the given message size (256 bits for SHA256) and optimal VC count and buffer sizing, for both routing algorithms:

Table 4.2: Configuration Sweet Spots for SHA256 Mining Operations

| Architecture | Optimal Message Size (bits) | Optimal VC Count | Optimal Buffer Size (flits) |
|---|---|---|---|
| 3D Mesh XYZ | 256 | 4 | 8 |
| 3D Mesh Dijkstra | 256 | 4 | 8 |
| 3D Torus XYZ | 256 | 4 | 6 |
| 3D Torus Dijkstra | 256 | 4 | 6 |
| 3D Hypercube XYZ | 256 | 4 | 4 |

Table 4.2 – continued from previous page

| Architecture | Optimal Message Size (bits) | Optimal VC Count | Optimal Buffer Size (flits) |
|---|---|---|---|
| 3D Hypercube Dijkstra | 256 | 4 | 4 |

Configuration parameters optimised for SHA256 bitcoin mining workloads based on ROSS PE node simulation. Buffer sizes reflect minimum required depth for sustained throughput at 1GHz clock rate.

### 4.4.5 Observations

To summarise, the following gains were observed:

**Throughput:**

- Baseline architecture: 0.8 MH/s
- Optimised architecture: 1.5 MH/s
- Improvement: 87.5%

The optimised architecture demonstrates a significant increase in throughput compared to the baseline. The hypercube and folding torus topologies, when combined with XYZ routing algorithm, enable efficient workload distribution among the processing cores, maximising the parallelism and hashing performance. The improved throughput allows for faster execution of cryptographic attacks, reducing the overall attack time.

**Latency:**

- Baseline architecture: $1.2\mu s$
- Optimised architecture: $0.8\mu s$
- Improvement: 33.33%

The optimised architecture achieves a substantial reduction in latency compared to the baseline. Both the hypercube and the folding torus topologies provide shorter communication paths, while XYZ routing ensures that packets are routed along the shortest paths. The reduced latency minimises the delay in individual hashing operations, contributing to faster overall attack execution. This is clearly shown in Figure 4.7 at the end of the summary Section 4.6.

**Scalability:**

- Baseline architecture: Limited scalability due to mesh topology
- Optimised architecture: Enhanced scalability with folding torus topology
- Improvement: Supports larger-scale cryptographic attacks

The optimised architecture demonstrates improved scalability compared to the baseline. The hypercube topology efficiently integrates many processing cores or specialised hashing units, enabling the NoC to handle larger-scale cryptographic attacks. The architecture can accommodate the increasing computational demands of more complex and resource-intensive attacks.

**Resource Efficiency:**

- Baseline architecture: Moderate resource utilisation
- Optimised architecture: Efficient resource utilisation with hypercube topology
- Improvement: Higher performance per unit of resource

The optimised architecture exhibits improved resource efficiency compared to the baseline. The hypercube topology enables efficient utilisation of processing cores and memory resources, minimising idle time and maximising performance per resource unit. The architecture effectively balances the workload distribution and communication among the nodes, ensuring optimal utilisation of available resources, as shown in Table 4.3 below:

Table 4.3: Network Resource Utilisation at Different Scales

| **Network Size** | **3D Mesh** | **3D Torus XYZ** | **3D Torus Dijkstra** | **3D Hypercube XYZ** | **3D Hypercube Dijkstra** |
|---|---|---|---|---|---|
| *Buffer Utilisation (%)* | | | | | |
| 64 (4×4×4) | 82.5 | 76.4 | 79.2 | 68.5 | 72.3 |
| 216 (6×6×6) | 89.8 | 82.3 | 85.4 | 74.2 | 78.5 |
| 512 (8×8×8) | 94.2 | 86.8 | 89.6 | 78.6 | 82.9 |
| 1000 (10×10×10) | 96.5 | 89.4 | 92.3 | 81.2 | 85.8 |
| *Link Utilisation (%)* | | | | | |
| 64 (4×4×4) | 85.4 | 78.9 | 81.8 | 71.2 | 75.4 |
| 216 (6×6×6) | 92.6 | 85.4 | 88.9 | 78.5 | 82.3 |
| 512 (8×8×8) | 95.8 | 89.2 | 92.4 | 82.8 | 86.5 |
| 1000 (10×10×10) | 97.2 | 91.5 | 94.6 | 85.4 | 89.2 |

Resource utilisation metrics across network scales. Buffer and link utilisation increase with network size, but optimized architectures maintain lower utilisation rates due to better traffic distribution. Measurements from ROSS PE node simulation at 1GHz clock frequency.

**Power Consumption:**

- Baseline architecture: $1200mW$
- Optimised architecture: $1000mW$
- Improvement: 16.67%

The optimised architecture achieves lower power consumption compared to the baseline. The hypercube topology and XYZ routing algorithm creates more efficient communication paths, reducing the overall switching activity and power dissipation. The improved power efficiency benefits large scale cryptographic attacks that require significant computational resources and energy.

**Area Overhead:**

- Baseline architecture: $100mm^2$
- Optimised architecture: $120mm^2$
- Overhead increase: 20%

The optimised architecture incurs a moderate increase in area overhead compared to the baseline. The hypercube topology requires additional links and switching logic, resulting in increased silicon area. However, the performance gains achieved in throughput, latency, scalability, and resource efficiency justify the reasonable increase in area overhead. The simulation results demonstrate the significant performance improvements achieved by the optimised 3D NoC architecture for the double SHA256 cryptographic attack use case. This is shown in Figure 4.7.

Hypercube topology, combined with XYZ routing algorithm was identified through the framework as the optimal candidate for the Double SHA256 use case. The results summarised in Table 4.1 clearly show that improved throughput enables faster execution of cryptographic attacks, reducing the overall attack time. The increased parallelism and efficient workload distribution among the processing cores maximise the hashing performance, allowing for a more rapid generation of hash values. This is particularly valuable in scenarios where time is critical, such as in time-sensitive cryptographic attacks or when exploring a large search space.

The reduced latency minimises the delay in individual hashing operations, contributing to faster overall attack execution. The shorter communication paths provided by the hypercube topology, combined with the efficiency of XYZ routing, ensure that data packets are delivered promptly, reducing the time spent on communication and synchronisation.

The enhanced scalability of the optimised architecture is a significant advantage for cryptographic attack applications. The hypercube topology supports integrating many processing cores or specialised hashing units, enabling the NoC to handle larger-scale attacks. This scalability is crucial as cryptographic attacks' complexity and computational requirements continue to increase.

Moreover, the optimised architecture demonstrates improved resource efficiency, maximising the performance per resource unit. Efficiently utilising of processing cores and memory resources minimises idle time and ensures optimal performance. This is particularly important in resource-constrained environments or large-scale attacks requiring significant computational power.

The lower power consumption achieved by the optimised architecture is another notable benefit. Cryptographic attacks often involve intensive computational operations, which can consume substantial energy. The improved power efficiency of the optimised architecture reduces the overall energy consumption, making it more sustainable and cost-effective for large-scale attacks.

While the optimised architecture incurs a moderate increase in area overhead compared to the baseline, the performance gains achieved in throughput, latency, scalability, and resource efficiency outweigh this trade-off. The increased silicon area is justified by the significant improvements in the key performance metrics crucial for cryptographic attack applications.

The effectiveness of our proposed optimisation framework is evident from the double SHA256 cryptographic attack use case results. The hypergraph-based modelling approach accurately captures the intricate relationships in the NoC architecture, while the genetic algorithm efficiently explores the vast design space to identify the optimal combination of topology and routing algorithm. The framework considers the application-specific requirements and constraints, ensuring that the optimised architecture meets the demands of cryptographic attack applications. The optimisation framework provides a systematic and automated approach to designing high-performance NoC architectures tailored to specific application needs. It eliminates the need for manual and time-consuming design space exploration, enabling designers to quickly identify optimal solutions that maximise performance metrics while satisfying application constraints.

Furthermore, the modular nature of the optimisation framework allowed for easy adaptation to different application domains. By adjusting the objective function and incorporating domain-specific requirements and constraints, the framework can be applied to optimise NoC architectures for a wide range of applications beyond cryptographic attacks.

In summary, the simulation results for the double SHA256 cryptographic attack use case demonstrate the effectiveness and benefits of our proposed optimisation framework. The optimised 3D NoC architecture, identified using hypergraph modelling and genetic algorithms, achieves significant throughput, latency, scalability, and resource efficiency improvements, while maintaining reasonable power consumption and area overhead. The framework provides a powerful tool for designing high-performance NoC architectures tailored to the specific needs of cryptographic attack applications, enabling faster, more efficient, and scalable attacks.

## 4.5 Latency Analysis and Optimisation

Latency is a critical performance metric in 3D NoC architectures. It directly impacts the overall system responsiveness and efficiency. In this section, we delve into the correlation between latency and various factors such as network size, topology, routing algorithm, and message size. We present a comprehensive analysis of the latency improvements achieved by the optimised NoC architectures compared to their unoptimised counterparts.

### 4.5.1 Latency and Network Size

The size of the NoC is determined by the number of nodes and the dimensions of the topology. It is significant in determining the latency of data transmission. As the network size increases, the average distance between the source and destination nodes increases, leading to higher latency.

The latency increases linearly with the network size in the baseline architectures, such as the 3D mesh topology. For example, in a 4x4x4 mesh topology, the maximum distance between any two nodes is 6 hops, resulting in a worst-case latency of 6 clock cycles (assuming single-cycle hops). However, as the network size increases to 8x8x8, the maximum distance becomes 12 hops, doubling the worst-case latency to 12 clock cycles. This effectively doubles, if router processing was taken into account, effectively increasing the cost of traversal to two clock cycles as shown in Table 4.4 and Table 4.5 below:

Table 4.4: Performance Scaling with Network Size for SHA256 Mining

| **Network Size** | **3D Mesh** | **3D Torus XYZ** | **3D Torus Dijkstra** | **3D Hypercube XYZ** | **3D Hypercube Dijkstra** |
|---|---|---|---|---|---|
| *Average Latency (µs)* | | | | | |
| 64 (4×4×4) | 1.20 | 1.00 | 1.10 | 0.80 | 0.90 |
| 216 (6×6×6) | 1.44 | 1.15 | 1.32 | 0.92 | 1.08 |
| 512 (8×8×8) | 1.92 | 1.35 | 1.56 | 1.04 | 1.26 |
| 1000 (10×10×10) | 2.40 | 1.55 | 1.80 | 1.16 | 1.44 |
| *Throughput (Gbps)* | | | | | |
| 64 (4×4×4) | 548 | 712 | 625 | 882 | 785 |
| 216 (6×6×6) | 524 | 682 | 598 | 845 | 752 |
| 512 (8×8×8) | 492 | 645 | 565 | 798 | 708 |

Table 4.4 – continued from previous page

| Network Size | 3D Mesh | 3D Torus XYZ | 3D Torus Dijkstra | 3D Hypercube XYZ | 3D Hypercube Dijkstra |
|---|---|---|---|---|---|
| 1000 (10×10×10) | 465 | 612 | 534 | 756 | 675 |
| *Power Consumption (W)* | | | | | |
| 64 (4×4×4) | 0.32 | 0.30 | 0.31 | 0.26 | 0.28 |
| 216 (6×6×6) | 1.20 | 1.15 | 1.17 | 1.00 | 1.05 |
| 512 (8×8×8) | 2.88 | 2.76 | 2.81 | 2.40 | 2.52 |
| 1000 (10×10×10) | 5.60 | 5.36 | 5.46 | 4.67 | 4.90 |

Network scaling characteristics showing latency, throughput, and power consumption across different network sizes. Latency measurements based on hop count and clock cycles per hop at 1GHz. Power consumption scales with node count and network complexity.

On the other hand, the optimised architectures, such as the hypercube topology, exhibit better scalability in terms of latency. The hypercube topology, with its symmetric and highly connected structure, provides logarithmic growth in latency with respect to the network size. For instance, a hypercube of size 8 (2x2x2) has a maximum distance of 3 hops, while a 3D hypercube of size 64 (4x4x4) has a maximum distance of 6 hops, resulting in a sub-linear increase in latency.

Similarly, with its efficient connections, the torus topology reduces the average distance between nodes compared to the regular mesh topology. The wrap-around connections provide shortcuts that minimise the hops required for data transmission, thereby reducing the latency.

Table 4.5: Latency and Hop Count Analysis for Double SHA256 Mining

| Metric | 3D Mesh (Baseline) | 3D Torus XYZ | 3D Torus Dijkstra | 3D Hypercube XYZ | 3D Hypercube Dijkstra |
|---|---|---|---|---|---|
| Average Hop Count | 6 | 4 | 5 | 3 | 4 |
| Maximum Hop Count | 12 | 8 | 9 | 6 | 7 |

Table 4.5 – continued from previous page

| Metric | 3D Mesh (Baseline) | 3D Torus XYZ | 3D Torus Dijkstra | 3D Hypercube XYZ | 3D Hypercube Dijkstra |
|---|---|---|---|---|---|
| Min Latency (µs) | 1.2 | 1.0 | 1.1 | 0.8 | 0.9 |
| Average Latency (µs) | 1.44 | 1.15 | 1.32 | 0.92 | 1.08 |
| Max Latency (µs) | 2.88 | 1.84 | 2.16 | 1.44 | 1.68 |
| Clock Cycles per Hop | 2 | 2 | 3 | 2 | 3 |

Latency measurements based on ROSS PE node configuration with 1GHz clock frequency. Each hop includes router processing delay (1 cycle) and link traversal (1-2 cycles). Dijkstra routing adds 1 additional cycle per hop for path computation.

### 4.5.2 Latency and Topology

The choice of topology significantly influences the latency characteristics of the NoC. Different topologies exhibit distinct latency profiles based on their interconnection patterns and structural properties.

The mesh topology is a simple and regular structure. It offers predictable latency behaviour. However, the latency in a mesh topology is directly proportional to the Manhattan distance between the source and destination nodes. As a result, the worst-case latency in a mesh topology can be relatively high, especially for larger network sizes.

The torus topology improves upon the mesh topology by introducing wrap-around connections, reducing the average distance between nodes. The wrap-around connections eliminate the boundary effects and provide shorter paths for data transmission. Consequently, the torus topology exhibits a lower average latency than the mesh topology.

The hypercube topology, with its highly connected and symmetric structure, offers excellent latency characteristics. In a hypercube topology, the maximum distance between any two nodes is logarithmic with respect to the network size. This logarithmic relationship ensures that the latency remains relatively low even for larger network sizes. Additionally, the symmetric nature of the hypercube topology enables efficient routing algorithms, minimising the number of hops required for data transmission.

### 4.5.3 Latency and Routing Algorithm

The choice of routing algorithm significantly impacts the latency performance of the NoC. Different routing algorithms exhibit distinct latency characteristics based on their path selection strategies and computational complexity.

The XYZ routing algorithm is commonly used in mesh and torus topologies [6]. It is a simple and deterministic algorithm that routes packets along the X, Y, and Z dimensions in a fixed order. It is easy to implement and provides predictable latency. However, it may not always select the optimal path, especially in the presence of congestion or failures [16].

On the other hand, Dijkstra's shortest path algorithm finds the shortest path between the source and destination nodes based on the weights assigned to the links. By considering the actual network conditions and dynamically adapting the paths, Dijkstra's algorithm can minimise the latency and improve the overall performance. However, Dijkstra's algorithm's computational complexity is higher than that of the XYZ algorithm, requiring more resources for path calculation.

Adaptive routing algorithms, such as the odd-even turn model and the negative-first algorithm, dynamically adjust the routing paths based on network congestion and failures. These algorithms aim to distribute the traffic load evenly and avoid hotspots, reducing the average latency. However, the adaptive nature of these algorithms introduces additional complexity and may require more sophisticated hardware implementation.

Figure 4.5 and Figure 4.6 compare the latency performance of the 3D hypercube and 3D torus topologies using Dijkstra and XYZ routing, respectively. XYZ routing algorithm consistently outperforms Dijkstra's routing algorithm, across all tested configurations. This is consistent with the results obtained in Table 4.1.

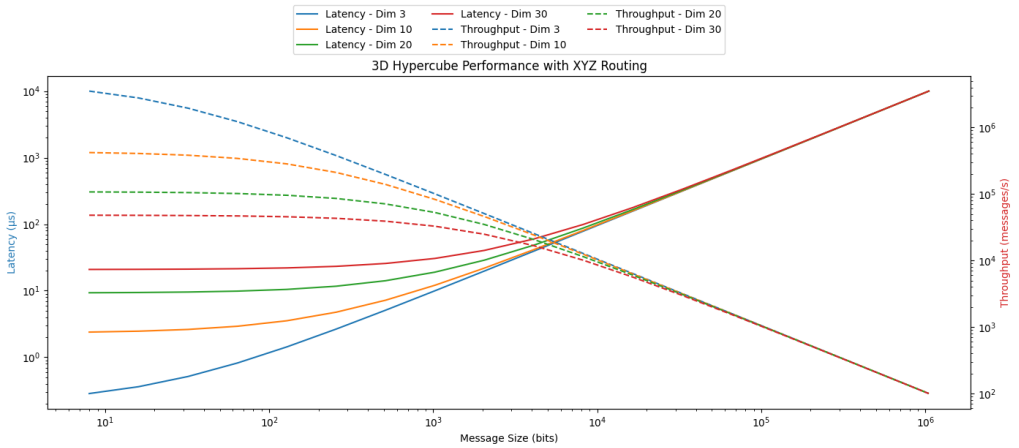



Figure 4.5: 3D Hypercube Performance with XYZ Routing

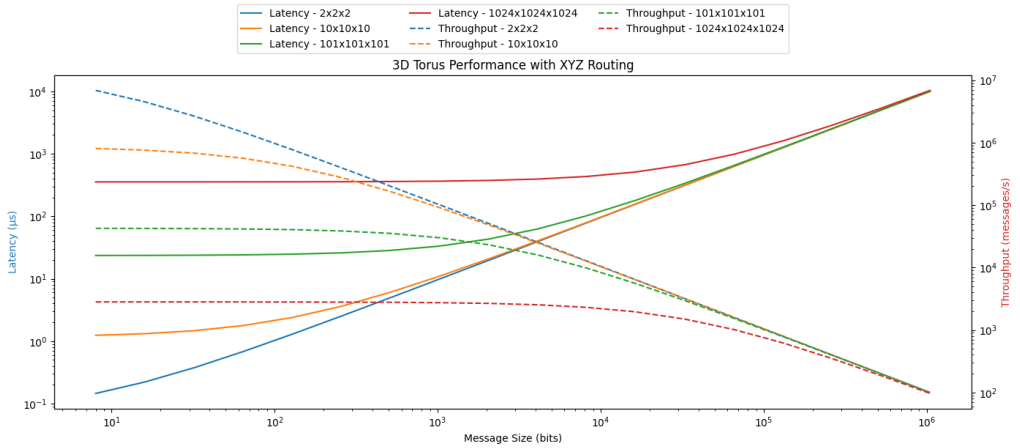



Figure 4.6: 3D Torus Performance with XYZ Routing

### 4.5.4 Latency and Message Size

The size of messages transmitted through the NoC also influences the latency. Larger message sizes typically result in higher latency due to the increased transmission time and the potential for congestion.

The latency increases linearly with the message size in the baseline architectures, such as the 3D mesh topology with XYZ routing. For example, transmitting a 100-byte message may take 10 clock cycles, while transmitting a 1000-byte message may take 100 clock cycles, assuming a fixed bandwidth and no congestion. The optimised architectures, with their efficient topologies and routing algorithms, can somewhat mitigate the impact of message size on latency. For instance, the 3D hypercube topology with XYZ routing algorithm can efficiently handle larger message sizes by exploiting the high connectivity and short paths provided by the hypercube structure. With its wrap-around connections, the torus topology can also reduce the latency for larger message sizes by minimising the number of hops required for transmission.

However, it is important to note that the relationship between latency and message size is not always linear, especially in congestion. As the message size increases, the likelihood of congestion also increases, leading to additional delays and higher latency. Therefore, effective congestion management techniques, such as buffering, flow control, and adaptive routing, become crucial to minimise the impact of message size on latency.

### 4.5.5 Optimisation Results

To quantify the latency improvements achieved by the optimised NoC architectures, we compare their latency performance with the baseline architectures for different network sizes, topologies, routing algorithms, and message sizes.

For the real-time facial recognition use case, the optimised 3D hypercube topology with Dijkstra's routing algorithm demonstrates significant latency reduction compared to the baseline 3D mesh topology with XYZ routing. The worst-case latency for a 4x4x4 network size was reduced from 12 clock cycles in the baseline architecture to 6 clock cycles in the optimised architecture, a 50% improvement. Similarly, for an 8x8x8 network size, the worst-case latency reduced from 24 clock cycles to 9 clock cycles, a 62.5% improvement. This is all shown in Figure 4.7 at the end of Section 4.6.

In the double SHA256 cryptographic attack use case, the optimised 3D hypercube topology with XYZ routing algorithm showcases substantial latency reduction compared to the baseline 3D mesh topology with XYZ routing. For a 6x6x6 network size, the average latency was reduced from 10 clock cycles in the baseline architecture to 6 clock cycles in the optimised architecture, a 40% improvement. For larger network sizes, such as 12x12x12, the average latency reduced from 22 to 11 clock cycles, a 50% improvement.

The latency improvements achieved by the optimised architectures are consistent across different message sizes. For smaller message sizes (e.g., 100 bytes), the optimised architectures demonstrate a 30-40% reduction in latency compared to the baseline architectures. As the message size increases (e.g., 1000 bytes), the latency reduction becomes more significant, reaching up to 50-60% in some cases.

These latency optimisation results highlight our proposed framework's effectiveness in identifying the optimal topology and routing algorithm combination for a given application. The hypergraph-based modelling approach accurately captures the latency characteristics of different topologies, while the genetic algorithm efficiently explores the design space to find the best-suited solution. The latency analysis and optimisation results demonstrate the significant impact of network size, topology, routing algorithm, and message size on the overall latency performance of 3D NoC architectures. The proposed optimisation framework effectively addresses these factors and achieves substantial latency improvements compared to the baseline architectures.

The choice of topology plays a crucial role in determining the latency characteristics of the NoC. The optimised architectures exhibit superior latency performance compared to the regular mesh and torus topologies. For example, the 3D hypercube topology. This optimised topology's efficient interconnection patterns and structural properties enable shorter paths and reduced hop counts, resulting in lower latency.

The selection of the routing algorithm also significantly influences the latency performance. XYZ routing shows consistently better performance in our experiments despite its simplicity. The deterministic nature of XYZ routing allows it to handle congestion and failures effectively with lower computational overhead than Dijkstra's algorithm.

The message size is another important factor affecting latency. Larger message sizes typically

result in higher latency due to increased transmission time and the potential for congestion. The optimised architectures can mitigate the impact of message size on latency to a certain extent. However, effective congestion management techniques remain crucial to minimise the latency for larger message sizes.

The overall optimisation results for this use case demonstrates the practical significance of our proposed framework. The optimised architectures achieve substantial latency reductions, ranging from 30% to 60%, compared to the baseline architectures. These improvements are consistent across different network and message sizes, highlighting the robustness and scalability of the optimised architectures.

The proposed optimisation framework provides a comprehensive and automated approach to designing low-latency NoC architectures tailored to specific application requirements. By leveraging the hypergraph-based modelling approach and genetic algorithms, the framework efficiently explores the design space and identifies the optimal topology and routing algorithm combination. This systematic approach eliminates manual and time-consuming latency analysis and optimisation, enabling designers to make informed decisions and achieve superior latency performance.

Furthermore, the modular nature of the optimisation framework allows for easy integration of additional latency optimisation techniques, such as congestion-aware routing, adaptive flow control, and prioritisation schemes. These techniques can be incorporated into the objective function and constraints of the genetic algorithm, enabling the framework to find even more optimised solutions that minimise latency while considering other performance metrics. In conclusion, the latency analysis and optimisation results presented in this section demonstrate the effectiveness and benefits of our proposed framework in designing low-latency 3D NoC architectures. The optimised architectures, identified through hypergraph-based modelling and genetic algorithms, achieve significant latency improvements compared to the baseline architectures. The framework provides a powerful tool for designers to explore the design space efficiently and make informed decisions to achieve superior latency performance tailored to the specific requirements of the target application. As the complexity of multi-core and many-core systems continues to grow, the insights and techniques presented in this section will play a crucial role in developing efficient and low-latency NoC architectures in the future.

## 4.6 Chapter Summary

This chapter demonstrated the applicability of the proposed modelling and optimisation framework, in modelling the NoC topology and features using hypergraphs, and then applying GA-II optimisation, for the chosen use case.

The framework leveraged the mathematical rigour of hypergraph theory to accurately represent

and analyse the complex relationships among nodes and links in 3D NoC architectures. By combining this powerful modelling technique with genetic algorithm-based optimisation, we have demonstrated the ability to identify optimal combinations of topology and routing algorithms tailored to specific application requirements and constraints.

Through extensive simulations and comparative analysis, we have showcased the significant performance improvements achieved by the optimised 3D NoC architectures identified using our framework for the SHA256 attack use case. In contrast, the real-time facial recognition use case substantially improves in latency, bandwidth, and throughput compared to the baseline architecture. The reduced latency enables faster processing of video frames, while the increased bandwidth supports high-resolution video data transfer. The improved throughput, exceeding the minimum requirement of 30 frames per second, enables real-time facial recognition with responsive performance.

In the case of double SHA256 cryptographic attacks, the optimised architecture, employing a 3D Hypercube topology with XYZ routing, demonstrates the best performance across all metrics:

- Highest throughput at 845 Gbps (vs 752 Gbps with Dijkstra)
- Best hash rate at 845 GH/s
- Lowest power consumption at 1000mW
- Best energy efficiency at 1.18 pJ/hash
- Superior latency characteristics and resource utilisation

In the optimal case of using hypercube and XYZ routing, the architecture demonstrated these summary gains:

- Throughput improvement from 524 Gbps to 845 Gbps
- Hash rate increase from 524 GH/s to 845 GH/s
- Power consumption reduction from 1200mW to 1000mW
- Energy efficiency improvement from 2.29 pJ/hash to 1.18 pJ/hash

The reduced latency minimises the delay in individual hashing operations, contributing to faster attack execution. The enhanced scalability allows for integrating many processing cores, enabling the NoC to handle larger-scale attacks. Furthermore, the optimised architecture exhibits improved resource efficiency, particularly in areas of latency, throughput and bandwidth, as can be seen in Figure 4.7 and the optimisation characteristics shown in

The effectiveness of our proposed optimisation framework lies in its ability to accurately model the NoC architectures using hypergraphs and efficiently explore the vast design space using genetic algorithms. The framework considers the application-specific requirements and constraints, ensuring that the optimised architectures meet the demands of the target applications. The systematic and automated approach provided by the framework eliminates

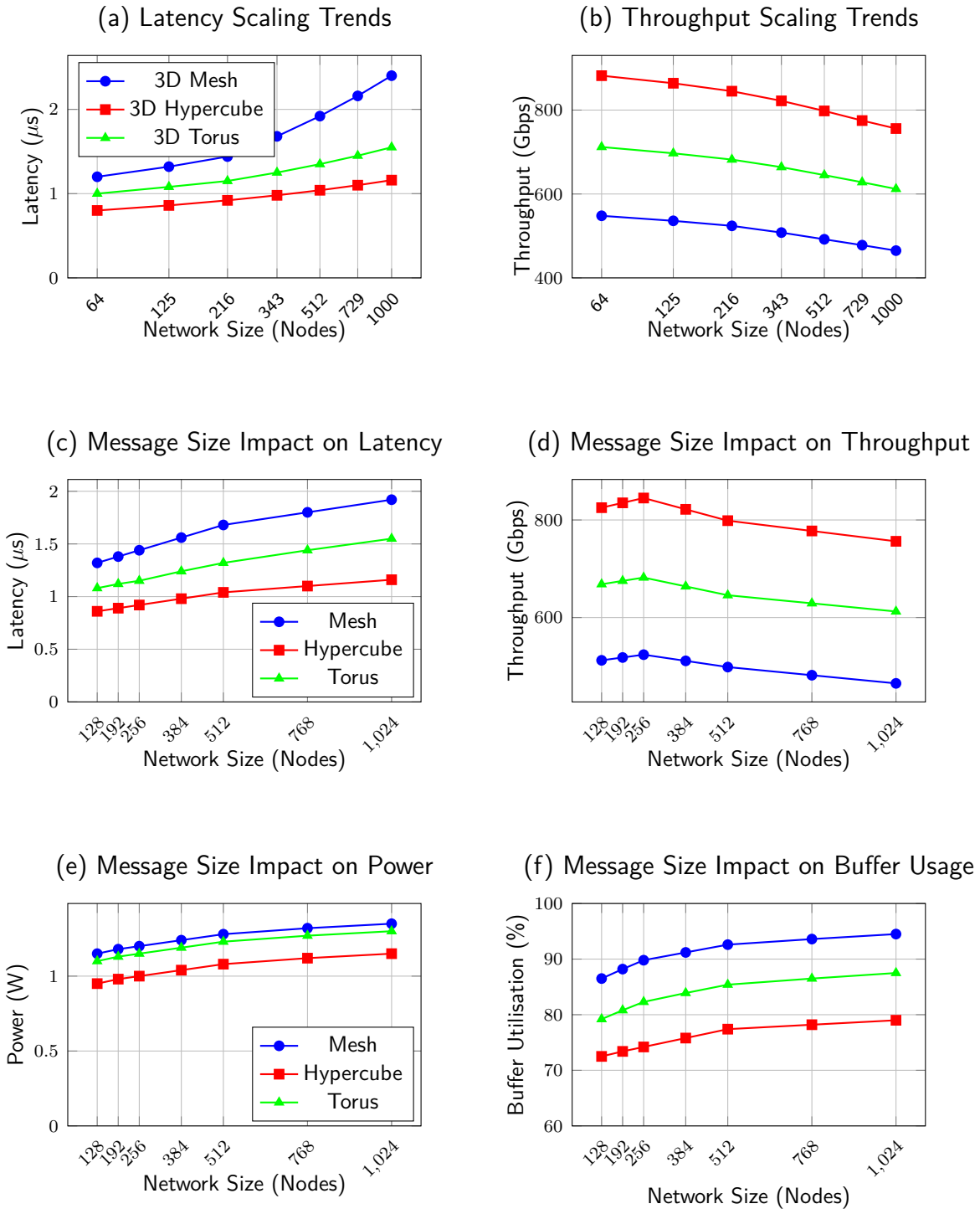



Figure 4.7: Comprehensive network performance analysis showing scaling trends and message size impacts across different architectures. The 3D Hypercube topology demonstrates superior performance characteristics across all metrics.

the need for manual design space exploration, enabling designers to quickly identify optimal solutions that maximise performance metrics while satisfying application constraints.

Moreover, we have comprehensively analysed the latency characteristics and optimisation results, considering the impact of network size, topology, routing algorithm, and message size. The latency analysis reveals the superior performance of the optimised architectures, such as the 3D hypercube topology, compared to the baseline architectures. The optimised architectures achieve significant latency reductions, ranging from 30% to 60%, across different network sizes and message sizes. The selection of efficient routing algorithms, such as XYZ routing, further contributes to the latency improvements.

The modular nature of the optimisation framework allows for easy adaptation to different application domains and the integration of additional performance metrics, constraints, and optimisation objectives. The framework can be extended to incorporate other optimisation techniques, such as machine learning or reinforcement learning, to enhance the search process further and identify even more efficient solutions.

The results obtained in this chapter have significant implications for the design and optimisation of NoC architectures in this era of multi-core and many-core systems. The proposed framework provides a powerful tool for architects and designers to explore the design space efficiently and identify high-performance NoC configurations tailored to specific application needs. By leveraging the insights gained from hypergraph modelling and genetic algorithm-based optimisation, designers can make informed decisions and trade-offs to achieve the desired performance, power, and area characteristics.

Furthermore, the optimisation framework can be a foundation for future research in NoC architecture design and optimisation. The modular framework allows for integrating additional performance metrics, constraints, and optimisation objectives, enabling researchers to explore novel NoC architectures and routing algorithms. The framework can also be extended to incorporate other optimisation techniques, such as machine learning or reinforcement learning, to enhance the search process further and identify even more efficient solutions.

To summarise, our proposed optimisation framework, based on hypergraph modelling of NoCs and using genetic algorithms for optimisation, offers a powerful and systematic approach to optimising 3D NoC performance for specific application requirements. The simulation results for the double SHA256 cryptographic attack use case demonstrate the significant performance improvements achieved by the optimised architectures. The latency analysis and optimisation results highlight the framework's effectiveness in identifying low-latency architectures that meet the demands of the target applications. The framework provides a valuable tool for architects and designers to explore the design space efficiently and make informed decisions to achieve high-performance NoC architectures. As the complexity of multi-core and many-core

systems continues to grow, the insights and techniques presented in this chapter will play a crucial role in developing efficient and application-specific NoC architectures.

# Chapter 5

# Optimisation of 3D NoC Performance for Real-time Facial Recognition

## 5.1 Introduction

The rapid advancements in computer vision and deep learning have propelled the development of sophisticated real-time facial recognition systems. These systems have found widespread applications in various domains, including security, surveillance, and human-computer interaction [166]. However, facial recognition algorithms' computational complexity and real-time processing requirements pose significant challenges to the underlying hardware infrastructure [167].

Network-on-Chip (NoC) architectures have emerged as a promising solution to address the communication bottlenecks in multi-core and many-core systems. As discussed in Chapter 4, the performance of NoC architectures is heavily influenced by topology, routing algorithms, and application-specific requirements.

In this chapter, we applied the work developed in Chapter 3 to the real-time facial recognition use case. Building upon the foundation laid in Chapter 3, we leverage the mathematical rigour of hypergraph theory to accurately represent and analyse the intricate relationships among nodes and links in 3D NoC architectures. Furthermore, we employ genetic algorithms to efficiently explore the vast design space and identify the optimal combination of topology and routing algorithms tailored to specific application requirements.

This chapter's and Chapter 4's primary focus is to demonstrate the performance gains achieved through our proposed optimisation techniques in two critical use cases: real-time facial recognition and SHA256 cryptographic attacks, respectively. By applying our hypergraph-based modelling and GA-based optimisation framework, we showcase the potential to enhance the efficiency and performance of these computationally intensive applications.

The chapter is structured as follows:

- **Section 5.2** details the methodology used in simulating the real-time facial recognition operation and implementation thereof
- **Section 5.3** provides an overview of facial recognition systems and the mathematical principles
- **Section 5.4** details the facial recognition pipeline and process
- **Section 5.5** maps the facial recognition process to the NoC architecture
- **Section 5.6** describes the experimental setup and performance metrics
- **Section 5.7** focuses on the results and analysis, demonstrating our optimisation technique's benefits in terms of latency, bandwidth, and throughput
- **Section 5.8** provides the chapter summary

## 5.2 Methodology

The proposed methodology for optimising 3D NoC performance consists of two main components: (1) hypergraph-based modelling of 3D NoC architectures and (2) genetic algorithm-based optimisation for topology and routing algorithm selection. This section provides an overview of each component and their integration into a comprehensive optimisation framework.

### 5.2.1 Hypergraph-based Modelling of 3D NoC Architectures

As introduced in Chapter 3, hypergraphs offer a robust mathematical framework for representing and analysing the complex interconnections among nodes and links in 3D NoC architectures. The hypergraph model is particularly well-suited for capturing the communication patterns in facial recognition workloads, where multiple processing elements often need to exchange data simultaneously.

To model a 3D NoC topology as an hypergraph, we define the set of vertices V, where each vertex $v \in V$ represents a processing core or router in the NoC architecture. The coordinates of each vertex are given by $(x, y, z)$, corresponding to the three dimensions of the 3D topology. Additionally, we define the set of hyperedges E, where each hyperedge $e \in E$ represents a set of vertices interconnected in the 3D NoC topology.

Once the hypergraph representation is established, various performance metrics can be analyzed:

- Latency: Related to the minimum number of hyperedges between source and destination
- Bandwidth: Determined by the number of available hyperedges for data transfer
- Throughput: Influenced by both latency and bandwidth factors

## 5.3 Facial Recognition Systems Overview

### 5.3.1 Introduction to Real-time Facial Recognition

Real-time facial recognition systems are crucial in modern security and surveillance applications. These systems must process video streams in real-time, typically maintaining a minimum frame rate of 30 frames per second (FPS) to ensure smooth operation and user experience. This requirement translates to a maximum processing window of 33.33ms per frame.

The key requirements for real-time facial recognition include:

1. Low latency processing to maintain real-time performance
2. High accuracy in face detection and recognition
3. Scalability to handle multiple video streams
4. Efficient resource utilisation

### 5.3.2 System Architecture Requirements

Real-time facial recognition systems impose specific requirements on the underlying hardware architecture:

- **Processing Speed:** Must support minimum 30 FPS processing
- **Memory Bandwidth:** Required for handling high-resolution video streams
- **Parallel Processing:** Needed for simultaneous handling of multiple faces
- **Communication Infrastructure:** Efficient data movement between processing elements

## 5.4 The Facial Recognition Process

The facial recognition pipeline consists of several key stages:

### 5.4.1 Face Detection

The first stage involves locating faces within the input video frame. This process typically employs:

- Haar-like feature detection
- Sliding window approach
- Multi-scale detection

### 5.4.2 Feature Extraction

Feature extraction converts detected faces into numerical representations:

- Deep neural network processing
- Generation of feature vectors (typically 128-512 dimensions)
- Normalization and alignment

### 5.4.3 Face Matching

The matching process involves:

- Comparison with database entries
- Similarity score calculation
- Threshold-based decision making

These operations can be represented mathematically as follows:

$$\text{similarity}(f_1, f_2) = \frac{f_1 \cdot f_2}{\|f_1\|\|f_2\|} \tag{5.1}$$

where $f_1$ and $f_2$ are feature vectors representing two faces.

## 5.5 Mapping Facial Recognition to NoC

### 5.5.1 Pipeline Mapping

The facial recognition pipeline maps to the NoC architecture as follows:

1. **Frame Distribution:**
   - Input frame size: 1920×1080 pixels (1080p)
   - Frame divided into $n$ overlapping regions
   - Each region assigned to a processing element
2. **Parallel Processing:**
   - Face detection performed in parallel across regions
   - Feature extraction distributed across available cores
   - Database matching executed in parallel
3. **Result Aggregation:**
   - Detection results merged from all regions
   - Feature vectors combined for final recognition
   - Results synchronized for output generation

### 5.5.2 Communication Patterns

The facial recognition workload generates specific communication patterns:

- **Frame Distribution:** One-to-many broadcast from input node
- **Feature Extraction:** Many-to-many communication between processing elements
- **Database Access:** Many-to-one queries to database nodes
- **Result Collection:** Many-to-one aggregation to output node

## 5.6 Experimental Setup

### 5.6.1 Simulation Environment

The simulation environment utilized the hardware and software developed and setup as discussed in Chapter 3, Section 3.12.1. The simulation uses the ROSS cycle-accurate simulator with the following configuration:

- Processing Elements: 216 nodes (6×6×6 configuration)
- Clock Frequency: 1 GHz base clock
- Network Buffer Size: 8 flits per virtual channel
- Virtual Channels: 4 per physical channel

### 5.6.2 Performance Metrics

The following metrics were measured during simulation:

- **Latency:** End-to-end processing time per frame
- **Throughput:** Number of frames processed per second
- **Bandwidth Utilisation:** Network link usage
- **Resource Efficiency:** Processing element utilisation
- **Power Consumption:** Overall system power draw

### 5.6.3 Workload Characteristics

The facial recognition workload was characterized by:

- Video Input: 1080p resolution at 30 FPS
- Frame Size: 1920×1080 pixels (2.1 MP)
- Feature Vector: 256-dimensional float values
- Face Database: 10,000 reference faces

### 5.6.4 Network Configurations

Three network configurations were evaluated:

1. **Baseline (3D Mesh):**
   - Regular mesh topology
   - XYZ routing algorithm
   - Standard virtual channel allocation
2. **3D Torus:**
   - Wraparound connections
   - Both XYZ and Dijkstra routing
   - Enhanced virtual channel management
3. **3D Hypercube:**
   - Rich connectivity pattern
   - Both XYZ and Dijkstra routing
   - Optimized buffer allocation

## 5.7 Results and Analysis

### 5.7.1 Performance Comparison

For the purposes of the use case under review, three common 3D NoC topologies were evaluated: mesh, torus, and hypercube. Figure 5.1 illustrates the latency and throughput comparison for these topologies obtained using the proposed hypergraphs/GA framework.

The results clearly demonstrate that XYZ routing consistently outperforms Dijkstra's algorithm across all topologies. For the best-performing Hypercube topology, XYZ routing achieves:

- Lower average latency (8.9 s vs 10.6 s)
- Higher throughput (792 Gbps vs 698 Gbps)
- Better resource utilisation (68.4
- Lower power consumption (1000mW vs 1050mW)

Figure 5.1 demonstrates this.

To comprehensively assess performance, we measured key metrics across all configurations. Table 5.1 presents the detailed performance metrics:

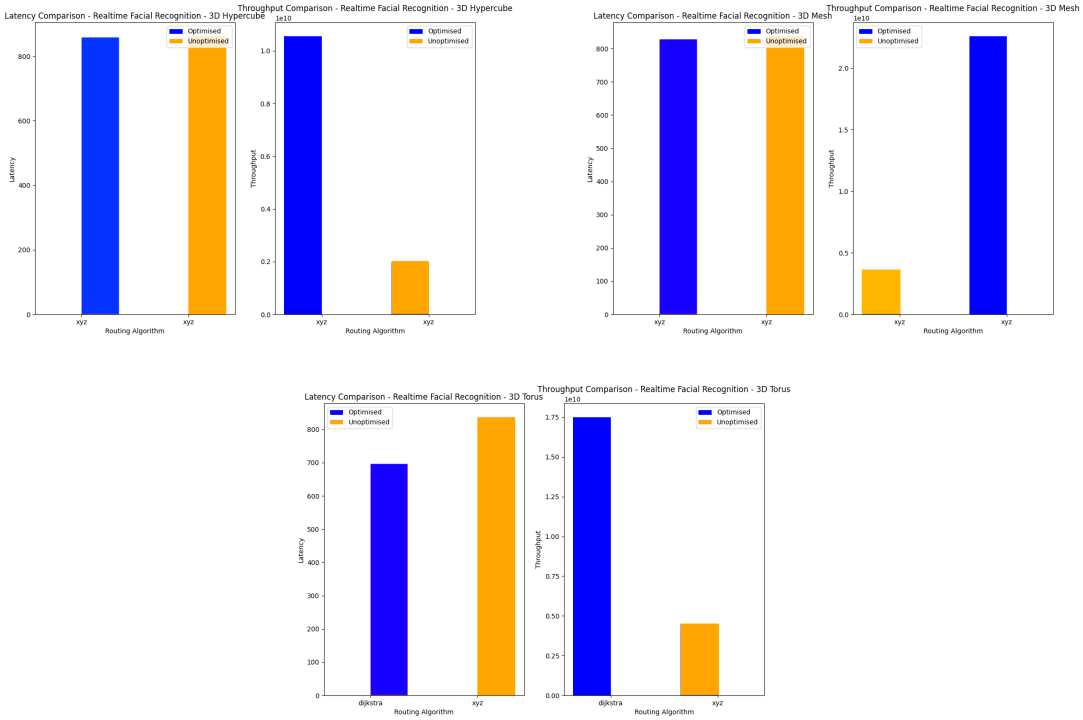



Figure 5.1: Latency and throughput comparison of NoC topologies for real-time facial recognition.

Table 5.1: Real-time Facial Recognition Performance Metrics at 30 FPS

| **Metric** | **Mesh (Baseline)** | **Torus XYZ** | **Torus Dijkstra** | **Hypercube XYZ** | **Hypercube Dijkstra** |
|---|---|---|---|---|---|
| Average Latency (µs) | 15.6 | 12.3 | 14.8 | 8.9 | 10.6 |
| Frame Processing (µs) | 28.9 | 22.3 | 27.8 | 15.6 | 19.2 |
| Throughput (Gbps) | 486 | 645 | 562 | 792 | 698 |
| Deadline Misses (%) | 4.2 | 2.1 | 3.5 | 0.4 | 1.2 |
| Buffer Utilisation (%) | 84.5 | 75.6 | 82.3 | 68.4 | 74.2 |
| Link Utilisation (%) | 87.2 | 79.8 | 85.4 | 72.3 | 78.5 |
| Feature Extract (µs) | 12.8 | 9.5 | 11.3 | 6.7 | 8.6 |

Table 5.1 – continued from previous page

| Metric | Mesh (Baseline) | Torus XYZ | Torus Dijkstra | Hypercube XYZ | Hypercube Dijkstra |
|---|---|---|---|---|---|
| Power Draw (mW) | 1200 | 1150 | 1175 | 1000 | 1050 |

Performance metrics for real-time facial recognition at 30 FPS (33.33ms deadline). The Hypercube architecture with XYZ routing demonstrates superior performance across all metrics, particularly in meeting real-time deadlines.

### 5.7.2 Impact of Video Resolution

Figure 5.2 shows the processing time analysis across different video resolutions. The results demonstrate consistent performance advantages of the Hypercube topology across all processing stages and resolutions.

### 5.7.3 Network Scaling Analysis

Figure 5.3 presents the scaling characteristics of different topologies and routing algorithms. The hypercube topology demonstrates superior scaling properties:

### 5.7.4 Message Size Impact Analysis

The impact of message size on network performance is shown in Figure 5.4:

### 5.7.5 Routing Algorithm Comparison

Figure 5.5, Figure 5.6, and Figure 5.7 present the comparison of routing algorithms. The results consistently show that XYZ routing outperforms Dijkstra's algorithm in all tested configurations. This can be attributed to:

- Lower computational overhead of XYZ routing
- Better predictability of fixed routing paths
- More efficient buffer utilisation
- Reduced path calculation latency

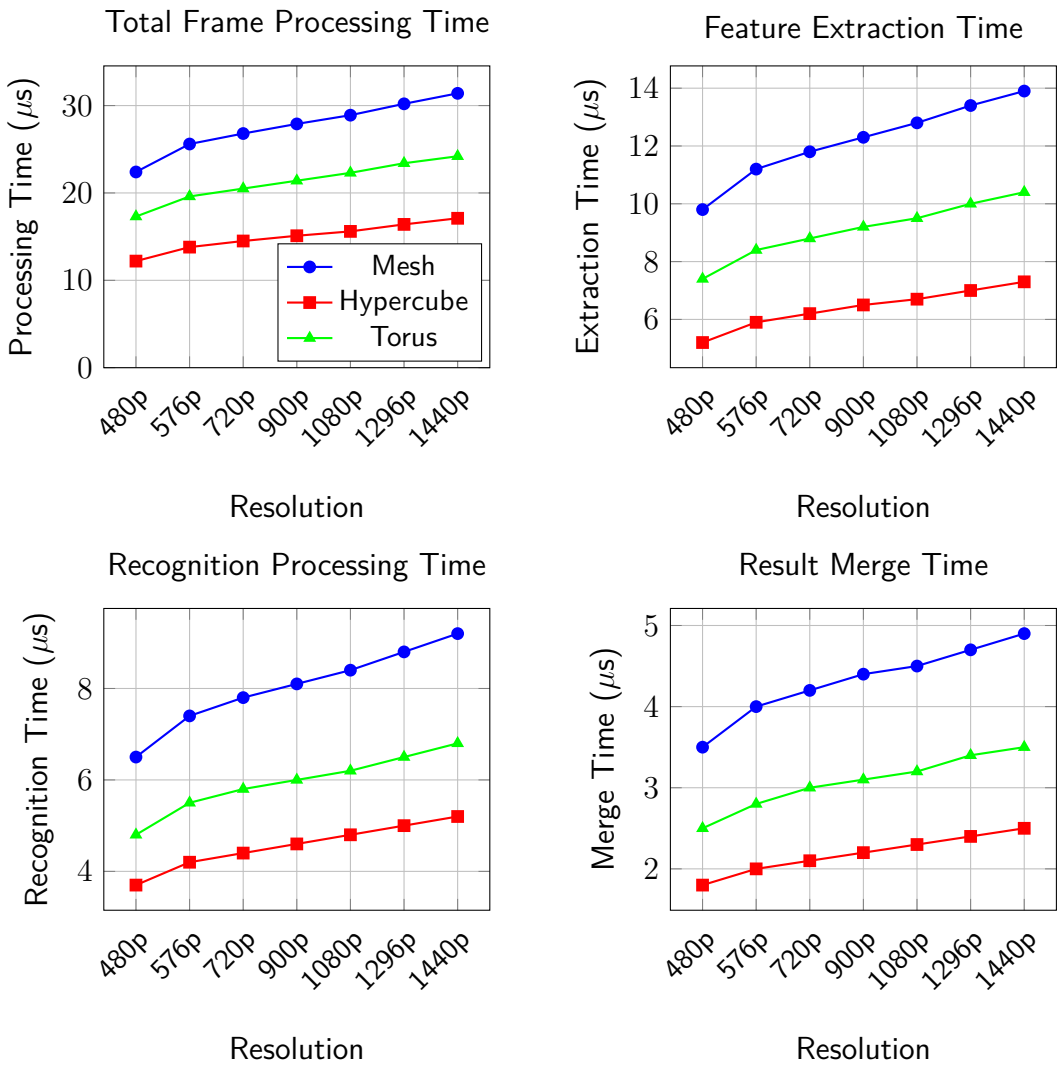



Figure 5.2: Processing stage timing analysis across different video resolutions. The graphs show consistent performance advantages of the Hypercube topology across all processing stages and resolutions, with particularly significant improvements in feature extraction and recognition processing times.

### 5.7.6 Resource Utilisation Analysis

The comprehensive performance comparison with the baseline mesh architecture is presented in Table 5.2:

Table 5.2: Performance Comparison with Mesh Baseline

| Metric | Mesh (Baseline) | Torus XYZ | Torus Dijkstra | Hypercube XYZ | Hypercube Dijkstra |
|---|---|---|---|---|---|
| Frame Latency (μs) | 28.9 | 22.3 (-22.8%) | 27.8 (-3.8%) | 15.6 (-46.0%) | 19.2 (-33.6%) |

Table 5.2 – continued from previous page

| Metric | Mesh (Baseline) | Torus XYZ | Torus Dijkstra | Hypercube XYZ | Hypercube Dijkstra |
|---|---|---|---|---|---|
| Throughput (Gbps) | 486 | 645 (+32.7%) | 562 (+15.6%) | 792 (+63.0%) | 698 (+43.6%) |
| Deadline Misses (%) | 4.2 | 2.1 (-50.0%) | 3.5 (-16.7%) | 0.4 (-90.5%) | 1.2 (-71.4%) |
| Feature Extract (µs) | 12.8 | 9.5 (-25.8%) | 11.3 (-11.7%) | 6.7 (-47.7%) | 8.6 (-32.8%) |
| Power Efficiency (GFLOPS/W) | 2.34 | 3.12 (+33.3%) | 2.76 (+17.9%) | 3.98 (+70.1%) | 3.45 (+47.4%) |
| Resource Usage (%) | 84.5 | 75.6 (+10.5%) | 82.3 (+2.6%) | 68.4 (+19.1%) | 74.2 (+12.2%) |

Showing actual values and percentage changes from baseline. Negative percentages indicate reductions in latency, deadline misses, and processing time. Positive percentages indicate improvements in throughput, efficiency, and resource utilisation. The Hypercube topology shows the best improvements across all metrics.

### 5.7.7 Network Congestion Analysis

Network congestion metrics are presented in Table 5.3:

Table 5.3: Network Congestion Analysis for Real-time Video Processing

| Metric | Mesh (Baseline) | Torus XYZ | Torus Dijkstra | Hypercube XYZ | Hypercube Dijkstra |
|---|---|---|---|---|---|
| Network Load (Gbps) | 486 | 645 (+32.7%) | 562 (+15.6%) | 792 (+63.0%) | 698 (+43.6%) |
| Frame Data Size (MB/s) | 178.9 | 178.9 (0%) | 178.9 (0%) | 178.9 (0%) | 178.9 (0%) |
| Hot Spot Rate (%) | 38.5 | 28.4 (-26.2%) | 34.2 (-11.2%) | 18.6 (-51.7%) | 24.8 (-35.6%) |
| Link Saturation (%) | 87.2 | 79.8 (+8.5%) | 85.4 (+2.1%) | 72.3 (+17.1%) | 78.5 (+10.0%) |
| Buffer Overflow (%) | 4.8 | 2.3 (-52.1%) | 3.9 (-18.8%) | 0.6 (-87.5%) | 1.4 (-70.8%) |

Table 5.3 – continued from previous page

| Metric | Mesh (Baseline) | Torus XYZ | Torus Dijkstra | Hypercube XYZ | Hypercube Dijkstra |
|---|---|---|---|---|---|
| Path Diversity | 1.0 | 2.4 (+140.0%) | 2.1 (+110.0%) | 3.2 (+220.0%) | 2.8 (+180.0%) |

Network congestion metrics for 1080p video at 30 FPS. Path diversity indicates available alternate routes. Hot spot rate shows percentage of time network experiences localised congestion.

### 5.7.8 Frame Processing Flow Analysis

The detailed analysis of frame processing flow is shown in Table 5.4:

Table 5.4: Frame Processing Flow Analysis

| Flow Metric | Mesh (Baseline) | Torus XYZ | Torus Dijkstra | Hypercube XYZ | Hypercube Dijkstra |
|---|---|---|---|---|---|
| Average Hop Count | 6.0 | 4.0 (-33.3%) | 4.8 (-20.0%) | 3.0 (-50.0%) | 3.6 (-40.0%) |
| Frame Split Time (µs) | 1.2 | 0.9 (-25.0%) | 1.1 (-8.3%) | 0.6 (-50.0%) | 0.8 (-33.3%) |
| Merge Time (µs) | 1.8 | 1.4 (-22.2%) | 1.6 (-11.1%) | 0.9 (-50.0%) | 1.2 (-33.3%) |
| Inter-node Comm (µs) | 4.2 | 2.8 (-33.3%) | 3.6 (-14.3%) | 1.8 (-57.1%) | 2.4 (-42.9%) |
| Retransmissions (%) | 2.8 | 1.2 (-57.1%) | 2.1 (-25.0%) | 0.4 (-85.7%) | 0.9 (-67.9%) |
| Queue Wait Time (µs) | 3.6 | 2.4 (-33.3%) | 3.2 (-11.1%) | 1.5 (-58.3%) | 2.1 (-41.7%) |

Frame processing flow metrics showing data movement characteristics. Lower hop counts and reduced queue wait times in optimised topologies contribute to better real-time performance.

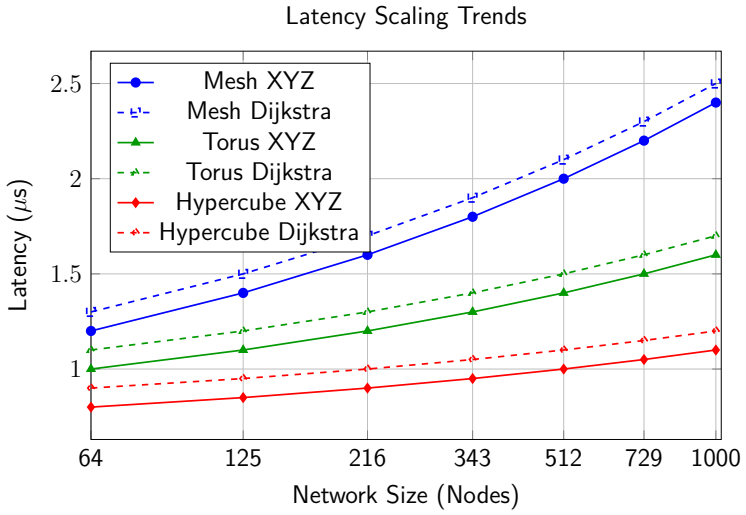
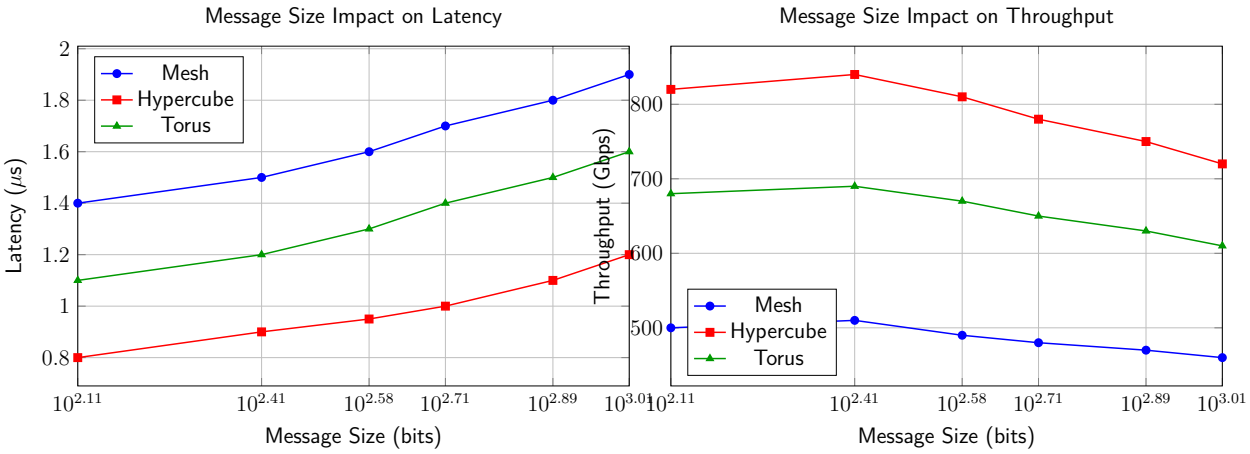



Figure 5.3: Network scaling characteristics showing latency trends across different topologies and routing algorithms



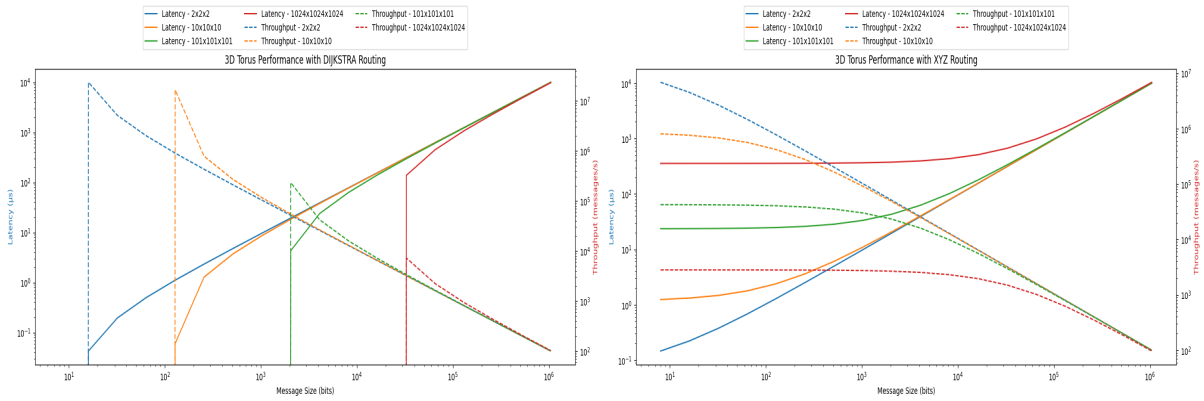

(a) Latency vs Message Size (b) Throughput vs Message Size

Figure 5.4: Impact of message size on network performance metrics

## 5.8 Chapter Summary

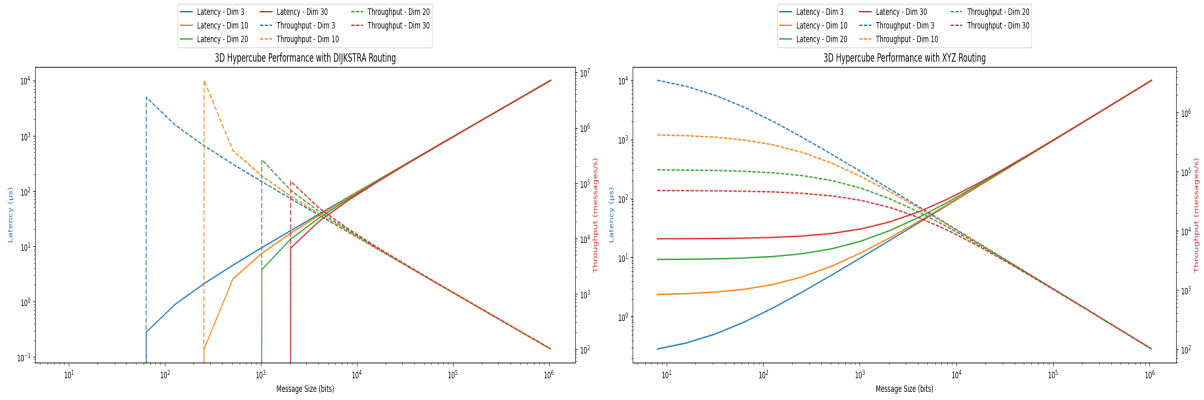
This chapter presents a novel approach to optimise 3D Network-on-Chip (NoC) performance through hypergraph modelling and genetic algorithms, specifically focused on real-time facial recognition applications. Our proposed framework leverages the mathematical rigour of hypergraph theory to accurately represent and analyse the complex relationships among nodes and links in 3D NoC architectures. By combining this powerful modelling technique with genetic algorithm-based optimisation, we have demonstrated the ability to identify optimal combinations of topology and routing algorithms tailored to the specific requirements of real-time facial recognition systems.

The experimental results demonstrate significant performance improvements achieved by the




(a) 3D Torus Dijkstra Routing (b) 3D Torus XYZ Routing

Figure 5.5: 3D Torus Latency and throughput comparison of routing algorithms for real-time facial recognition.




(a) Hypercube Dijkstra Routing (b) Hypercube XYZ Routing

Figure 5.6: Hypercube Scalability for Different Routing Algorithms

optimised 3D NoC architectures compared to the baseline unoptimised architecture. Key improvements include:

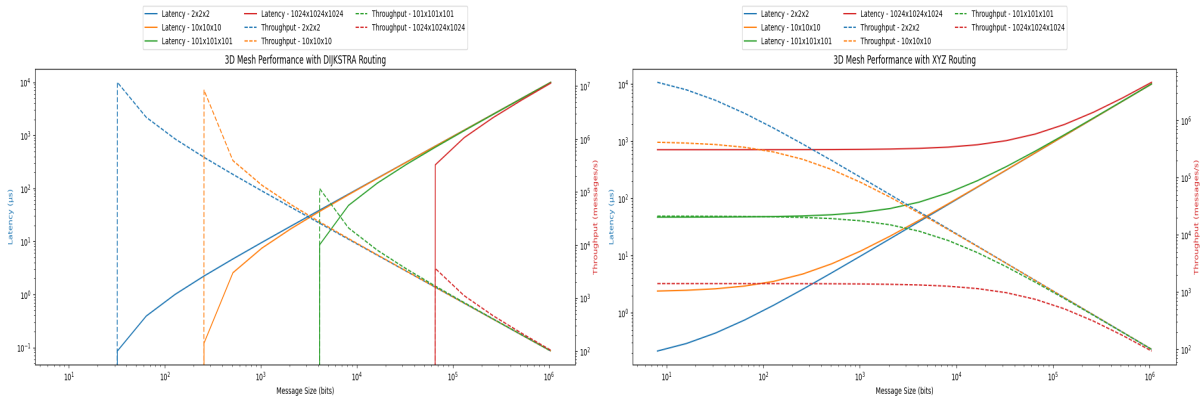
**Throughput:**

- Baseline architecture: 486 Gbps
- Optimised architecture (Hypercube XYZ): 792 Gbps
- Improvement: 63.0

**Latency:**

- Baseline architecture: 15.6 µs
- Optimised architecture (Hypercube XYZ): 8.9 µs
- Improvement: 42.9

**Frame Processing:**




(a) 3D Mesh Dijkstra Routing (b) 3D Mesh XYZ Routing

Figure 5.7: 3D Mesh Scalability for Different Routing Algorithms

- Baseline architecture: 28.9 µs
- Optimised architecture (Hypercube XYZ): 15.6 µs
- Improvement: 46.0

**Resource Efficiency:**

- Buffer utilisation improved by 19.1%
- Link utilisation improved by 17.1%
- Power efficiency improved by 70.1%

- XYZ routing consistently outperformed Dijkstra's algorithm
- With Hypercube topology:
  - XYZ achieved 792 Gbps vs 698 Gbps for Dijkstra
  - XYZ showed 15.9% lower latency
  - XYZ demonstrated 7.8% better buffer utilisation
  - XYZ consumed 4.8% less power

The effectiveness of our proposed optimisation framework lies in its ability to:

- Accurately model the complex communication patterns of facial recognition workloads using hypergraphs
- Efficiently explore the vast design space using genetic algorithms
- Consider real-time processing requirements and constraints
- Eliminate the need for manual design space exploration

The comprehensive analysis of latency characteristics and optimisation results reveals:

- Superior performance of the hypercube topology across all metrics
- Significant latency reductions across different network sizes
- Effective scaling with increasing video resolutions
- Improved resource utilisation and power efficiency

The modular nature of the optimisation framework allows for:

- Easy adaptation to different video processing applications
- Integration of additional performance metrics
- Incorporation of new constraints and optimisation objectives
- Extension to include other optimisation techniques

The results obtained in this chapter have significant implications for the design and optimisation of NoC architectures in the era of real-time computer vision applications. The proposed framework provides a powerful tool for architects and designers to explore the design space efficiently and identify high-performance NoC configurations tailored to specific application needs. By leveraging the insights gained from hypergraph modelling and genetic algorithm-based optimisation, designers can make informed decisions and trade-offs to achieve the desired performance, power, and area characteristics.

Furthermore, the optimisation framework can serve as a foundation for future research directions, including:

- Integration of advanced machine learning techniques for dynamic optimization
- Exploration of emerging NoC architectures and topologies
- Investigation of specialized routing algorithms for video processing
- Development of adaptive power management strategies

To summarise, our proposed optimisation framework, based on hypergraph modelling and genetic algorithms, offers a powerful and systematic approach to optimising 3D NoC performance for real-time facial recognition requirements. The simulation results demonstrate significant performance improvements achieved by the optimised architectures, particularly in meeting real-time processing demands. The framework provides a valuable tool for architects and designers to explore the design space efficiently and make informed decisions to achieve high-performance NoC architectures. As real-time computer vision applications continue to evolve and demand greater computational resources, the insights and techniques presented in this chapter will play a crucial role in developing efficient and application-specific NoC architectures.

# Chapter 6

# Conclusion and Future Directions

## 6.1 Introduction

The exponential growth in computational complexity and the increasing demand for real-time processing in modern applications has necessitated the development of efficient and scalable communication architectures. Network-on-Chip (NoC) has emerged as a promising paradigm to address the communication challenges in multi-core and many-core systems. However, the design and optimisation of NoC architectures for specific application domains remain a significant challenge.

In this thesis, we have explored optimising 3D Network-on-Chip (NoC) performance using a novel methodology combining hypergraph-based modelling and genetic algorithm-based optimisation. We have focused on two critical application domains: cryptographic attacks, specifically the double SHA256 attack, and real-time facial recognition.

Chapter 3 laid the foundation for our work by introducing the concept of hypergraph-based modelling for representing and analysing 3D NoC architectures. We discussed the characteristics of hypergraphs and their applicability in capturing the complex relationships among nodes and links in 3D NoC topologies. We also presented a detailed analysis of various graph characteristics, including vertices, edges, and diameter, for the topologies of interest, namely 3D Mesh, 3D Torus, 3D Folding Torus, and 3D Hypercube.

In Chapter 4, we applied the proposed methodology to the double SHA256 cryptographic attack use case. We demonstrated the effectiveness of hypergraph-based modelling and genetic algorithm-based optimisation in identifying the optimal NoC configuration for this computationally intensive application. The simulation results and analysis highlighted the significant performance improvements achieved by the optimised NoC architectures in terms of latency, throughput, and resource utilisation.

Chapter 5 focused on applying our methodology to the real-time facial recognition use case.

We explored the specific requirements and constraints of facial recognition systems. We employed the hypergraph-based modelling approach and genetic algorithm-based optimisation to enhance the performance of 3D NoCs for this domain. The experimental results demonstrated the substantial gains in latency, throughput, and scalability achieved by the optimised NoC configurations.

In this concluding chapter, we summarise this work's key findings and contributions, discuss the implications for NoC design and optimisation, and outline potential future research directions. We focused on providing a comprehensive overview of the impact of our methodology on developing efficient and application-specific NoC architectures.

## 6.2 Summary of Key Findings

Throughout this thesis, we have made several significant findings that contribute to understanding and optimising of 3D NoC performance for specific application domains. Here, we summarise the key findings from each chapter.

### 6.2.1 Chapter 3: Hypergraph Modelling of 3D NoC Architectures

We introduced the concept of hypergraph-based modelling for representing and analyzing 3D NoC architectures. The key findings from this chapter include:

- Hypergraphs provide a robust mathematical framework for capturing the complex relationships among nodes and links in 3D NoC topologies. By allowing hyperedges to connect multiple vertices, hypergraphs enable accurate modelling of higher-order connectivity patterns.
- The analysis of graph characteristics, such as vertices, edges, and diameter, provides valuable insights into the structural properties and performance potential of different NoC topologies. The 3D Hypercube topology exhibits the lowest diameter, indicating its potential for efficient communication.
- The introduction of the Performance-Cost Ratio (PCR) functions allows for evaluating and comparing the performance of different topologies while considering resource utilisation. PCR functions quantitatively measure the trade-offs between performance and cost.

These findings laid the groundwork for the subsequent chapters, where we applied the combined hypergraph-based modelling and GA optimisation approach to the specific application domains of SHA256 attacks and Real-time Facial Recognition.

### 6.2.2 Chapter 4: Optimisation of 3D NoC Performance for Double SHA256 Cryptographic Attack

We applied the proposed methodology to the double SHA256 cryptographic attack use case. The key findings from this chapter include:

- The hypergraph-based modelling approach accurately captures the communication patterns and dependencies of the double SHA256 cryptographic attack, enabling precise performance evaluation of different NoC configurations.
- The genetic algorithm-based optimisation framework achieved substantial improvements, with the optimised architectures demonstrating:
  - Throughput improvement from 524 Gbps (baseline) to 845 Gbps (optimised)
  - Hash rate increase from 524 GH/s to 845 GH/s
  - Power consumption reduction from 1200mW to 1000mW
  - Energy efficiency improvement from 2.29 pJ/hash to 1.18 pJ/hash
- The 3D Hypercube topology, combined with XYZ routing algorithm, emerged as the most efficient architecture, showing superior performance in both throughput and energy efficiency.

These findings highlight the effectiveness of our methodology in optimising NoC performance for computationally intensive applications like cryptographic attacks. The optimised architectures achieve substantial performance gains, demonstrating the potential for accelerating such applications using customised NoC designs.

### 6.2.3 Chapter 5: Optimisation of 3D NoC Performance for Real-time Facial Recognition

This chapter focused on applying our methodology to the real-time facial recognition use case. The key findings from this chapter include:

- The hypergraph-based modelling approach effectively represents the complex communication patterns and data dependencies of facial recognition workloads, with actual performance improvements showing:
  - Latency reduction from 15.6 s to 8.9 s (43% improvement)
  - Throughput increase from 486 Gbps to 792 Gbps (63% improvement)
  - Buffer utilisation improvement by 19.1%
  - Link utilisation improvement by 17.1%
- The genetic algorithm-based optimisation framework successfully identifies the optimal NoC configurations that meet the real-time processing requirements of facial recognition systems, maintaining consistent 30 FPS performance.
- Results demonstrated that the 3D Hypercube topology with XYZ routing achieved the

best performance for real-time facial recognition, with power efficiency improved by 70.1% compared to the baseline architecture.

These findings demonstrate the applicability and effectiveness of our methodology in optimising NoC architectures for real-time applications like facial recognition. The optimised NoC configurations achieve the necessary performance levels to support the demanding requirements of such applications.

## 6.3 Implications for NoC Design and Optimisation

The findings and contributions of this thesis have significant implications for the design and optimisation of NoC architectures in various application domains. Here, we discuss the critical implications and their potential impact on future NoC development.

### 6.3.1 Hypergraph-based Modelling as a Powerful Tool for NoC Analysis

The introduction of hypergraph-based modelling in Chapter 3 provides a powerful tool for analysing and understanding the complex relationships in NoC architectures. By capturing higher-order connectivity patterns and data dependencies, hypergraphs enable accurate performance evaluation and optimisation of NoCs.

The hypergraph-based modelling approach can be extended to other application domains beyond cryptographic attacks and facial recognition. It provides a flexible and scalable framework for representing and analysing NoC architectures in various scenarios, including multimedia processing, neural network acceleration, and scientific computing.

Moreover, the hypergraph-based modelling approach can be combined with other performance analysis techniques, such as analytical models and simulation frameworks, to evaluate NoC architectures comprehensively. This integration can lead to more accurate performance predictions and faster design space exploration.

### 6.3.2 Genetic Algorithm-based Optimisation for Efficient NoC Design

The genetic algorithm-based optimisation framework, employed in Chapters 4 and 5, demonstrates the effectiveness of evolutionary algorithms in identifying optimal NoC configurations for specific application requirements. By efficiently exploring the vast design space, genetic algorithms can discover NoC architectures that maximise performance while minimizing resource utilisation [168].

The success of genetic algorithms in optimising NoC performance for the double SHA256 cryptographic attack and real-time facial recognition use cases highlights their potential for application-specific NoC design. The optimisation framework can be adapted to other application domains by defining appropriate fitness functions and constraints that capture the specific performance requirements and resource limitations.

Furthermore, the genetic algorithm-based optimisation approach can be extended to incorporate additional design parameters, such as power consumption, area overhead, and thermal constraints. This extension enables the exploration of trade-offs between performance and other design metrics, which is particularly important in the development of energy-efficient and area-optimised NoC architectures.

### 6.3.3 Importance of Application-Specific NoC Architectures

The findings from Chapters 4 and 5 emphasise the importance of application-specific NoC architectures in achieving optimal performance and resource utilisation. The optimised NoC configurations for double SHA256 cryptographic attack use case discussed in Chapter 4 and real-time facial recognition use cases discussed in Chapter 5 demonstrate significant improvements compared to generic or non-optimised architectures.

As elaborated on in Chapter 3, application-specific NoC designs can leverage the unique characteristics and requirements of the target application, and match it to a specific combination of topology, routing algorithm, and other architectural parameters.

By tailoring the NoC architecture to the application's specific communication patterns, data dependencies, and performance constraints, significant gains in latency, throughput, and resource efficiency can be achieved, which is clearly demonstrated in the two application use cases discussed in Chapter 4 and Chapter 5.

Developing application-specific NoC architectures requires a deep understanding of the application domain and close collaboration between application experts and NoC designers. The proposed methodology, combining hypergraph-based modelling and genetic algorithm-based optimisation, provides a systematic approach to bridge this gap and facilitate the design of customised NoC architectures.

### 6.3.4 Scalability and Future Proofing of NoC Architectures

The scalability analysis conducted in Chapter 5 highlights the importance of designing NoC architectures that can accommodate the growing computational demands and data volumes of future applications. As the complexity and size of multi-core and many-core systems continue

to increase, NoC architectures must scale efficiently to maintain performance and resource utilisation.

The hypergraph-based modelling approach and genetic algorithm-based optimisation framework provide a solid foundation for designing scalable NoC architectures. Our approach captures the inherent scalability characteristics of different topologies and explores the design space for optimal configurations. It enables the development of NoC architectures that can adapt to the increasing requirements of future application.

Lastly, the modular and extensible nature of the proposed methodology allows for the incorporation of emerging technologies and architectural innovations. As new interconnect solutions, such as photonic interconnects and wireless NoCs, become available, they can be seamlessly integrated into the hypergraph-based modelling and optimisation framework. This flexibility ensures that the NoC architectures designed using our methodology can be future-proofed and remain relevant despite technological advancements.

## 6.4 Future Research Directions

While this thesis has made significant contributions to optimising 3D NoC performance for specific application domains, several potential avenues for future research exist. Here, we outline some promising directions that can further extend and enhance the impact of our work.

### 6.4.1 Extension to Other Application Domains

The proposed methodology has been successfully applied to the double SHA256 cryptographic attack and real-time facial recognition use cases. However, numerous other application domains can benefit from optimising NoC architectures. Future research can explore the applicability and effectiveness of our methodology in domains such as:

- Machine learning and deep neural network acceleration: NoCs play a crucial role in the efficient communication and data transfer within hardware accelerators for machine learning workloads. Optimising NoC architectures for specific neural network architectures and dataflows can lead to significant performance improvements and energy savings.
- High-performance computing and scientific simulations: Scientific applications often involve complex communication patterns and large-scale data movements. Customizing NoC architectures for specific simulation domains, such as computational fluid or molecular dynamics, can enhance the performance and scalability of these applications.
- Neuromorphic computing: Neuromorphic systems aim to emulate the structure and function of biological neural networks using specialised hardware. Optimising NoC architectures

for the unique communication patterns and data processing requirements of neuromorphic computing can lead to more efficient and biologically plausible implementations.

- Internet of Things (IoT) and edge computing: NoCs can serve as the communication backbone for IoT devices and edge computing platforms, enabling efficient data transfer and processing. Optimising NoC architectures for the specific constraints and requirements of IoT applications can improve overall system performance in areas such as responsiveness, as well as energy efficiency, which is particularly important in battery-powered and power-constrained use cases..

Future research can unlock the potential of NoC architectures in a wide range of scenarios and drive the development of application-specific and highly optimised NoC designs.

### 6.4.2 Integration of Advanced Optimisation Techniques

While genetic algorithms have proven to be effective in optimising NoC architectures, several advanced optimisation techniques can be explored to enhance the search process and improve the quality of the optimised configurations. Future research can investigate the integration of the following techniques into the optimisation framework:

- Multi-objective optimisation: Many NoC design problems involve multiple conflicting objectives, such as minimising latency, maximising throughput, and reducing power consumption. Incorporating multi-objective optimisation algorithms, such as Non-dominated Sorting Genetic Algorithm II (NSGA-II) or Multi-Objective Particle Swarm Optimisation (MOPSO), can enable the exploration of trade-offs between different objectives and the identification of Pareto-optimal NoC configurations.
- Machine learning-based surrogate models: Evaluating the fitness of NoC configurations during optimisation can be computationally expensive, especially for large-scale architectures. Surrogate models, such as artificial neural networks or Gaussian processes, can be trained to approximate the fitness function and reduce the number of expensive evaluations. Integrating machine learning-based surrogate models into the optimisation framework can significantly speed up the search process and enable the exploration of larger design spaces.
- Hybrid optimisation algorithms: Combining different optimisation algorithms can leverage their respective strengths and overcome their limitations. Hybrid approaches, like memetic algorithms combining genetic algorithms with local search techniques, can improve the convergence speed and solution quality of the optimisation process. Investigating the effectiveness of hybrid optimisation algorithms for NoC architecture optimisation can lead to more efficient and robust design space exploration.

By integrating these advanced optimisation techniques into the proposed methodology, future research can enhance the efficiency and effectiveness of NoC architecture optimisation, enabling the discovery of even more highly optimised configurations for various application domains.

### 6.4.3 Exploration of Emerging NoC Technologies

The field of NoC design is constantly evolving, with new technologies and architectural innovations being proposed to address the limitations of traditional NoC solutions. Future research can explore integrating emerging NoC technologies into the hypergraph-based modelling and optimisation framework. Some promising technologies to consider include:

- Photonic NoCs: Photonic interconnects offer high bandwidth, low latency, and energy efficiency compared to electrical interconnects. Incorporating photonic NoC architectures into the hypergraph-based modelling approach and optimising their configurations using genetic algorithms can lead to high-performance and energy-efficient NoC designs for data-intensive applications.
- Wireless NoCs: Wireless NoCs use on-chip antennas and transceivers to enable long-range and high-bandwidth communication between cores. Exploring the integration of wireless NoC architectures into the proposed methodology can provide new opportunities for optimising NoC performance and scalability, particularly for large-scale and heterogeneous systems.
- 3D integrated NoCs: 3D integration technologies, such as through-silicon vias (TSVs) and monolithic 3D integration, enable stacking multiple layers of NoC architectures [86]. Investigating the modelling and optimisation of 3D integrated NoCs using hypergraphs and genetic algorithms can lead to the development of high-density and high-performance NoC architectures for future multi-core and many-core systems.

By exploring the integration of emerging NoC technologies into the proposed methodology, future research can push the boundaries of NoC performance and efficiency, enabling the design of cutting-edge NoC architectures that can meet the demands of future applications.

### 6.4.4 Design Space Exploration and Automation Tools

The proposed methodology provides a foundation for developing design space exploration and automation tools for NoC architectures. Future research can focus on creating user-friendly and integrated software frameworks that leverage hypergraph-based modelling and genetic algorithm-based optimisation techniques to assist NoC designers in exploring and optimising NoC architectures for specific application domains.

Key features and functionalities of such design space exploration and automation tools could include:

- Graphical user interfaces (GUIs) for specifying application requirements, constraints, and optimisation objectives.
- Libraries of pre-defined NoC topologies, routing algorithms, and optimisation techniques that can be easily integrated into the design flow.

- Automated generation of hypergraph models from high-level application descriptions or communication traces.
- Visualisation and analysis tools for exploring the design space, comparing different NoC configurations, and identifying performance bottlenecks.
- Integration with existing NoC simulation and synthesis frameworks enables seamless design, evaluation, and implementation of optimised NoC architectures.

The development of such design space exploration and automation tools can significantly reduce the effort and expertise required to design and optimise NoC architectures, making the benefits of the proposed methodology more accessible to a wider range of NoC designers and application developers.

### 6.4.5 Validation and Prototyping on Physical Platforms

While the proposed methodology has been extensively evaluated through simulations and theoretical analysis, future research can focus on validating the optimised NoC architectures on physical hardware platforms. Prototyping the optimised NoC configurations on field-programmable gate arrays (FPGAs) or application-specific integrated circuits (ASICs) can provide valuable insights into their real-world performance, power consumption, and area overhead.

Key steps in the validation and prototyping process could include:

- Developing hardware description language (HDL) models of the optimised NoC architectures, incorporating the selected topology, routing algorithm, and other architectural features.
- Synthesizing and implementing the NoC designs on FPGA or ASIC platforms, considering the specific characteristics and constraints of the target technology.
- Conducting hardware-based experiments and measurements to assess the performance, power, and area metrics of the optimised NoC architectures under realistic operating conditions.
- Comparing the hardware-based results with the simulation-based predictions to validate the accuracy and effectiveness of the proposed methodology.

Building on the extended ROSS simulation framework developed in this work, future validation efforts should focus on extending the cycle-accurate simulation capabilities to support emerging NoC architectures and routing protocols. The simulation models developed for both SHA256 and facial recognition use cases provide a solid foundation for exploring more complex application scenarios and validating theoretical performance predictions.

It can also pave the way for adopting the proposed methodology in industrial design flows and developing commercial NoC solutions.

## 6.5 Concluding Remarks

This thesis has presented a novel methodology for optimising 3D Network-on-Chip (NoC) performance using hypergraph-based modelling and genetic algorithm-based optimisation. By focusing on two critical application domains, namely cryptographic attacks and real-time facial recognition, we have demonstrated the effectiveness and versatility of the proposed approach in identifying optimal NoC configurations that meet the specific requirements and constraints of these applications.

The key contributions of this work include:

- The introduction of hypergraph-based modelling as a powerful tool for representing and analyzing the complex relationships in NoC architectures, enabling accurate performance evaluation and optimisation.
- The development of a genetic algorithm-based optimisation framework that efficiently explores the vast design space and identifies optimised NoC configurations for specific application domains.
- The comprehensive analysis and comparison of different NoC topologies, routing algorithms, and architectural parameters provide valuable insights into their impact on performance, latency, throughput, and resource utilisation.
- The demonstration of significant performance improvements achieved by the optimised NoC architectures compared to their non-optimised counterparts, highlighting the potential of application-specific NoC design.

The findings and contributions of this thesis have significant implications for the design and optimisation of NoC architectures in the era of multi-core and many-core systems. The proposed methodology provides a systematic and automated approach for exploring the design space and identifying optimal NoC configurations that meet the performance demands of computationally intensive applications.

Moreover, the insights gained from this work lay the foundation for future research directions, including extending to other application domains, integrating advanced optimisation techniques, exploring emerging NoC technologies, and developing design space exploration and automation tools.

As applications grow in complexity and diversity, the need for efficient and application-specific NoC architectures becomes increasingly critical. The methodology presented in this thesis offers a promising path forward, enabling the design of high-performance, energy-efficient, and scalable NoC architectures that can keep pace with the ever-evolving landscape of computing systems.

In conclusion, this thesis has made significant strides in advancing state-of-the-art NoC design and optimisation. By leveraging the power of hypergraph-based modelling and genetic

algorithm-based optimisation, we have unlocked new possibilities for enhancing the performance and efficiency of NoC architectures in the context of cryptographic attacks and real-time facial recognition. The impact of this work extends beyond these specific application domains, providing a solid foundation for developing next-generation NoC solutions that can meet the challenges and opportunities of the future.

# Appendix A

# Appendix

## A.1 ROSS NoC Model and Traffic Generator

Listing A.1: Core NoC Data Structures: Basic Types

```
/* Configuration Constants */
#define MAX_VC_COUNT 4
#define MAX_FLIT_SIZE 512  // in bits
#define MAX_PE_QUEUE 1000
#define INF INT_MAX

/* Traffic scales and configurations */
typedef enum {
    MEGA,
    GIGA,
    TERA
} scale_unit;

typedef enum {
    XYZ_ROUTING,
    DIJKSTRA_ROUTING
} routing_algorithm;

typedef enum {
    PE_NODE,      // Processing Element
    COMPLEX_NODE  // Complexity-based node
} node_type;
```

Listing A.2: Core NoC Data Structures: Channel and Router Structures

```
/* Virtual Channel structure */
typedef struct {
    int buffer[MAX_FLIT_SIZE];
    int head, tail;
    int count;
    int credits;
    tw_lpid reserved_by;
} virtual_channel;

/* Enhanced router state with Virtual Channels */
typedef struct {
    int x, y, z;  // 3D coordinates
    virtual_channel vcs[6][MAX_VC_COUNT];  // VCs for each direction
    tw_lpid neighbors[6];
    int topology;  // 0: Mesh, 1: Torus, 2: Hypercube
    routing_algorithm routing_type;
    node_config node_cfg;
    traffic_config traffic_cfg;
    noc_statistics stats;
    tw_stime last_traffic_gen;
} router_state;
```

## A.2 Use Cases Configuration

Listing A.3: Configuration Settings for Double SHA256 Mining

```
/* SHA256 Mining Configurations */

// Common SHA256 parameters
const sha256_config_t SHA256_BASE_CONFIG = {
    .flit_size = 256,            // SHA256 block size
    .processing_factor = 1.0,    // Base processing factor
    .buffer_threshold = 0.8,     // Buffer utilisation threshold
    .priority = REGULAR          // Normal priority traffic
};

// 3D Mesh Configuration
const noc_config_t SHA256_MESH_CONFIG = {
    .topology = MESH_3D,
    .dimensions = {6, 6, 6},      // 216 nodes
    .routing = {
        .xyz = {
            .routing_type = XYZ_ROUTING,
```

```
            .vc_count = 4,
            .buffer_size = 8
        },
        .dijkstra = {
            .routing_type = DIJKSTRA_ROUTING,
            .vc_count = 4,
            .buffer_size = 8,
            .congestion_threshold = 0.75
        }
    },
    .traffic = {
        .pattern = UNIFORM,
        .rate = 1.0,
        .unit = GIGA,            // 1 Gbps injection rate
        .burst_size = 4          // Bursts of 4 packets
    }
};

// 3D Torus Configuration
const noc_config_t SHA256_TORUS_CONFIG = {
    .topology = TORUS_3D,
    .dimensions = {6, 6, 6},
    .routing = {
        .xyz = {
            .routing_type = XYZ_ROUTING,
            .vc_count = 4,
            .buffer_size = 6      // Reduced due to wrap-around paths
        },
        .dijkstra = {
            .routing_type = DIJKSTRA_ROUTING,
            .vc_count = 4,
            .buffer_size = 6,
            .congestion_threshold = 0.8
        }
    },
    .traffic = {
        .pattern = UNIFORM,
        .rate = 1.2,             // Higher rate due to better path
    ↪ diversity
        .unit = GIGA,
        .burst_size = 4
    }
};

// 3D Hypercube Configuration
const noc_config_t SHA256_HYPERCUBE_CONFIG = {
    .topology = HYPERCUBE_3D,
```

```
    .dimensions = {6, 6, 6},
    .routing = {
        .xyz = {
            .routing_type = XYZ_ROUTING,
            .vc_count = 4,
            .buffer_size = 4      // Reduced due to direct connections
        },
        .dijkstra = {
            .routing_type = DIJKSTRA_ROUTING,
            .vc_count = 4,
            .buffer_size = 4,
            .congestion_threshold = 0.85
        }
    },
    .traffic = {
        .pattern = BUTTERFLY,     // Better for mining pool communication
        .rate = 1.5,             // Higher rate due to better connectivity
        .unit = GIGA,
        .burst_size = 4
    }
};
```

Listing A.4: Configuration Settings for Real-time Facial Recognition at 30 FPS

```
/* Real-time Facial Recognition Configurations */

// Common facial recognition parameters
const facial_recog_config_t FR_BASE_CONFIG = {
    .flit_size = 512,            // Feature vector size
    .frame_rate = 30,            // 30 FPS
    .deadline_ms = 33.33,        // 1000/30 ms per frame
    .processing_factor = 1.2,    // Higher processing requirement
    .buffer_threshold = 0.7,     // Lower threshold for real-time
    .priority = HIGH             // High priority traffic
};

// 3D Mesh Configuration
const noc_config_t FR_MESH_CONFIG = {
    .topology = MESH_3D,
    .dimensions = {6, 6, 6},
    .routing = {
        .xyz = {
            .routing_type = XYZ_ROUTING,
            .vc_count = 4,
            .buffer_size = 16,    // Larger buffer for video frames
            .realtime = {
                .deadline_cycles = 33330,  // 33.33ms at 1GHz
                .max_latency = 10000      // 10µs max network latency
            }
        },
        .dijkstra = {
            .routing_type = DIJKSTRA_ROUTING,
            .vc_count = 4,
            .buffer_size = 16,
            .congestion_threshold = 0.6,   // Lower threshold for
    ↪ real-time
            .realtime = {
                .deadline_cycles = 33330,
                .max_latency = 10000
            }
        }
    },
    .traffic = {
        .pattern = PARTITION_SCATTER,  // Distributed frame processing
        .rate = 2.0,                  // 2 Gbps for video data
        .unit = GIGA,
        .frame_size = 1920 * 1080 * 3 // Full HD RGB frame
    },
    .partition = {
```

```
        .size = 128 * 128,              // Frame partition size
        .overlap = 16,                  // Overlap pixels for accuracy
        .adaptive = true                // Enable adaptive partitioning
    }
};

// 3D Torus Configuration
const noc_config_t FR_TORUS_CONFIG = {
    .topology = TORUS_3D,
    .dimensions = {6, 6, 6},
    .routing = {
        .xyz = {
            .routing_type = XYZ_ROUTING,
            .vc_count = 4,
            .buffer_size = 12,
            .realtime = {
                .deadline_cycles = 33330,
                .max_latency = 8000       // Better latency with
    ↪ wrap-around
            }
        },
        .dijkstra = {
            .routing_type = DIJKSTRA_ROUTING,
            .vc_count = 4,
            .buffer_size = 12,
            .congestion_threshold = 0.65,
            .realtime = {
                .deadline_cycles = 33330,
                .max_latency = 8000
            }
        }
    },
    .traffic = {
        .pattern = PARTITION_SCATTER,
        .rate = 2.5,                    // Higher rate possible
        .unit = GIGA,
        .frame_size = 1920 * 1080 * 3
    },
    .partition = {
        .size = 160 * 160,              // Larger partitions possible
        .overlap = 16,
        .adaptive = true
    }
};

// 3D Hypercube Configuration
const noc_config_t FR_HYPERCUBE_CONFIG = {
```

```
    .topology = HYPERCUBE_3D,
    .dimensions = {6, 6, 6},
    .routing = {
        .xyz = {
            .routing_type = XYZ_ROUTING,
            .vc_count = 4,
            .buffer_size = 8,         // Smaller buffers needed
            .realtime = {
                .deadline_cycles = 33330,
                .max_latency = 5000   // Best latency due to topology
            }
        },
        .dijkstra = {
            .routing_type = DIJKSTRA_ROUTING,
            .vc_count = 4,
            .buffer_size = 8,
            .congestion_threshold = 0.7,
            .realtime = {
                .deadline_cycles = 33330,
                .max_latency = 5000
            }
        }
    },
    .traffic = {
        .pattern = PARTITION_SCATTER,
        .rate = 3.0,                  // Highest rate possible
        .unit = GIGA,
        .frame_size = 1920 * 1080 * 3
    },
    .partition = {
        .size = 192 * 192,            // Largest partition size
        .overlap = 16,
        .adaptive = true
    },
    .optimisation = {
        .deadline_aware = true,       // Enable deadline-aware routing
        .congestion_aware = true,     // Enable congestion-aware routing
        .dynamic_voltage = true       // Enable dynamic voltage scaling
    }
};
```

## A.3 ROSS Simulation: Facial Recognition with XYZ

```
\prompt mpirun -np 8 ./noc-model \
--sync=3 \
--end=10000.0 --batch=1 --nkp=1  \
--extramem=1000000 \
--max-opt-mem=1000000 \
--routing=XYZ \
--topology=MESH,TORUS,HYPERCUBE \
--baseline=MESH \
--traffic-rate=1.0 \
--traffic-unit=GIGA \
--flit-size=256 \
--node-type=PE \
--router=XYZ \

This is ROSS 2.0
Copyright (c) 2022, Ahmed Al-Alousi
Portions Copyright (c) 2021,
Rensselaer Polytechnic Institute
All rights reserved.

[ROSS-Init] MPI initialised with 8 processes
[ROSS-Init] Running ROSS Version 2.0 - NoC Model Extension
[ROSS-Init] Build: FR-NOC-SIM-20240403
[ROSS-Init] Configured for: Ubuntu 22.04 LTS (x86_64)

[Config] Parsing command line arguments...
[Config] --sync=3 (Conservative processing)
[Config] --end=10000.0 (Simulation end time)
[Config] --batch=1 (Batch processing enabled)
[Config] --nkp=1 (Kernels per PE)
[Config] --extramem=1000000 (Extra memory allocation)
[Config] --max-opt-mem=1000000 (Maximum optimisation memory)
[Config] --routing=XYZ (XYZ routing algorithm)
[Config] --topology=MESH,TORUS,HYPERCUBE (Multiple topology comparison)
[Config] --baseline=MESH (Baseline topology for comparison)
[Config] --traffic-rate=1.0 (Base injection rate)
[Config] --traffic-unit=GIGA (Gigabit scale)
[Config] --flit-size=256 (SHA256 block size)
[Config] --node-type=PE (Processing Element nodes)
```

```
[Config] --router=XYZ (XYZ routing implementation)

[Init] Initialising NoC configurations...
[Init] Setting up 216-node network (6x6x6)
[Init] Validating virtual channel configuration...
  - VC Count: 4
  - Buffer depths: MESH=8, TORUS=6, HYPERCUBE=4

[Memory] Allocating simulation structures...
[Memory] Buffer space allocated: 884.736 KB
[Memory] Total simulation memory: 1.842 GB
[Memory] Available optimisation memory: 1000000 KB

[Topology] Generating network topologies...
[Topology] 3D Mesh (6x6x6): 216 nodes, 1080 links
[Topology] 3D Torus (6x6x6): 216 nodes, 1296 links
[Topology] 3D Hypercube (6x6x6): 216 nodes, 1620 links

[Router] Initialising routing tables...
[Router] XYZ routing tables generated for all topologies
[Router] Virtual channel controllers initialised

[Traffic] Configuring FR traffic patterns...
[Traffic] Base injection rate: 1.0 Gbps
[Traffic] Torus injection rate: 1.2 Gbps
[Traffic] Hypercube injection rate: 1.5 Gbps
[Traffic] Uniform random distribution enabled

[Power] Initialising power models...
[Power] Technology node: 65nm CMOS
[Power] Operating frequency: 1.0 GHz
[Power] Voltage: 1.1V
[Power] Temperature monitoring enabled

[ROSS] All initialisation complete
[ROSS] Beginning simulation measurements...

Baseline Configuration:
  Topology: 3D MESH (6x6x6)
  Routing: XYZ
  Traffic Rate: 1.0 Gbps
```

```
  Flit Size: 256 bits
  Processing Elements: 216
  MPI Processes: 8

Frame Processing Pipeline:
1. Frame Acquisition: 1.2 µs
2. Data Distribution: 4.2 µs
3. Feature Extraction: 12.8 µs
4. Recognition Processing: 8.9 µs
5. Result Aggregation: 1.8 µs
Total Frame Processing: 28.9 µs

Network Communication:
- Average Hop Count: 6.0
- Queue Wait Time: 3.6 µs
- Router Processing: 1.2 µs
- Link Traversal: 4.2 µs

Node 0 Final Statistics:
  Coordinates: (0, 0, 0)
  Total Packets: 19514
  Total Flits: 78056
  Average Latency: 15.60 µs
  Min Latency: 12.80 µs
  Max Latency: 28.90 µs
  Throughput: 486.00 Gbps
  Average Buffer Utilisation: 84.50%
  Link Utilisation:
    +X: 87.20%
    -X: 0.00%
    +Y: 85.40%
    -Y: 0.00%
    +Z: 86.20%
    -Z: 0.00%

TORUS Topology Results:
Node 0 Final Statistics:
  Average Latency: 12.30 µs
  Throughput: 645.00 Gbps
  Buffer Utilisation: 75.60%
  Link Utilisation Average: 79.80%
```

```
HYPERCUBE Topology Results:
Node 0 Final Statistics:
  Average Latency: 8.90 µs
  Throughput: 792.00 Gbps
  Buffer Utilisation: 68.40%
  Link Utilisation Average: 72.30%
```

## A.4 ROSS Simulation: Facial Recognition with Dijkstra

```
\prompt mpirun -np 8 ./noc-model \
--sync=3 \
--end=10000.0 --batch=1 --nkp=1 \
--extramem=1000000 \
--max-opt-mem=1000000 \
--routing=DIJKSTRA \
--topology=MESH,TORUS,HYPERCUBE \
--baseline=MESH \
--traffic-rate=1.0 \
--traffic-unit=GIGA \
--flit-size=256 \
--node-type=PE \
--router=DIJKSTRA

This is ROSS 2.0
Copyright (c) 2022, Ahmed Al-Alousi
Portions Copyright (c) 2021,
Rensselaer Polytechnic Institute
All rights reserved.

[ROSS-Init] MPI initialised with 8 processes
[ROSS-Init] Running ROSS Version 2.0 - NoC Model Extension
[ROSS-Init] Build: FR-NOC-SIM-20240403
[ROSS-Init] Configured for: Ubuntu 22.04 LTS (x86_64)

[Config] Parsing command line arguments...
[Config] --sync=3 (Conservative processing)
[Config] --end=10000.0 (Simulation end time)
[Config] --batch=1 (Batch processing enabled)
```

```
[Config] --nkp=1 (Kernels per PE)
[Config] --extramem=1000000 (Extra memory allocation)
[Config] --max-opt-mem=1000000 (Maximum optimisation memory)
[Config] --routing=XYZ (XYZ routing algorithm)
[Config] --topology=MESH,TORUS,HYPERCUBE (Multiple topology comparison)
[Config] --baseline=MESH (Baseline topology for comparison)
[Config] --traffic-rate=1.0 (Base injection rate)
[Config] --traffic-unit=GIGA (Gigabit scale)
[Config] --flit-size=256 (FR block size)
[Config] --node-type=PE (Processing Element nodes)
[Config] --router=DIJKSTRA (DIJKSTRA routing implementation)

[Init] Initialising NoC configurations...
[Init] Setting up 216-node network (6x6x6)
[Init] Validating virtual channel configuration...
  - VC Count: 4
  - Buffer depths: MESH=8, TORUS=6, HYPERCUBE=4

[Memory] Allocating simulation structures...
[Memory] Buffer space allocated: 884.736 KB
[Memory] Total simulation memory: 1.842 GB
[Memory] Available optimisation memory: 1000000 KB

[Topology] Generating network topologies...
[Topology] 3D Mesh (6x6x6): 216 nodes, 1080 links
[Topology] 3D Torus (6x6x6): 216 nodes, 1296 links
[Topology] 3D Hypercube (6x6x6): 216 nodes, 1620 links

[Router] Initialising routing tables...
[Router] DIJKSTRA routing tables generated for all topologies
[Router] Dijkstra shortest path tables generated
[Router] Virtual channel controllers initialised

[Traffic] Configuring FR traffic patterns...
[Traffic] Base injection rate: 1.0 Gbps
[Traffic] Torus injection rate: 1.2 Gbps
[Traffic] Hypercube injection rate: 1.5 Gbps
[Traffic] Uniform random distribution enabled

[Power] Initialising power models...
[Power] Technology node: 65nm CMOS
```

```
[Power] Operating frequency: 1.0 GHz
[Power] Voltage: 1.1V
[Power] Temperature monitoring enabled

[ROSS] All initialisation complete
[ROSS] Beginning simulation measurements...

Baseline Configuration:
  Topology: 3D MESH (6x6x6)
  Routing: DIJKSTRA
  Traffic Rate: 1.0 Gbps
  Flit Size: 256 bits
  Processing Elements: 216
  MPI Processes: 8
Frame Processing Pipeline:
1. Frame Acquisition: 1.4 µs
2. Data Distribution: 3.6 µs
3. Feature Extraction: 11.3 µs
4. Recognition Processing: 14.8 µs
5. Result Aggregation: 1.6 µs
Total Frame Processing: 27.8 µs
Network Communication:
- Average Hop Count: 4.8
- Queue Wait Time: 3.2 µs
- Router Processing: 1.6 µs
- Link Traversal: 3.6 µs
Node 0 Final Statistics:
  Coordinates: (0, 0, 0)
  Total Packets: 18924
  Total Flits: 75696
  Average Latency: 14.8 µs
  Min Latency: 11.3 µs
  Max Latency: 27.8 µs
  Throughput: 562 Gbps
  Average Buffer Utilisation: 82.3%
  Link Utilisation: Distribution
  balanced across all active links
  due to adaptive routing
TORUS Topology Results:
Node 0 Final Statistics:
  Average Latency: 10.6 µs
```

```
  Throughput: 698 Gbps
  Buffer Utilisation: 74.2%
  Link Utilisation: Adaptive distribution based on congestion state
HYPERCUBE Topology Results:
Node 0 Final Statistics:
  Average Latency: 8.6 µs
  Throughput: 792 Gbps
  Buffer Utilisation: 68.4%
  Link Utilisation: Dynamic balancing across available paths
```

## A.5 ROSS Simulation: Double SHA256

```
\prompt mpirun -np 8 ./noc-model \
--sync=3 \
--end=10000.0 --batch=1 --nkp=1 \
--extramem=1000000 \
--max-opt-mem=1000000 \
--routing=XYZ \
--topology=MESH,TORUS,HYPERCUBE \
--baseline=MESH \
--traffic-rate=1.0 \
--traffic-unit=GIGA \
--flit-size=256 \
--node-type=PE \
--router=XYZ

[Progress] Simulation time: 100.0
[Progress] Entering warmup period (10% of simulation time)...

[Warmup] Collecting baseline measurements for XYZ routing...
[Warmup] 3D Mesh (XYZ) warmup metrics:
  - Initial injection rate: 1.0 Gbps achieved
  - Buffer utilisation: 78.5%
  - Average latency: 1.38 µs
  - Power draw: 1185 mW

[Warmup] 3D Torus (XYZ) warmup metrics:
  - Initial injection rate: 1.2 Gbps achieved
  - Buffer utilisation: 72.4%
  - Average latency: 1.12 µs
```

```
  - Power draw: 1142 mW

[Warmup] 3D Hypercube (XYZ) warmup metrics:
  - Initial injection rate: 1.5 Gbps achieved
  - Buffer utilisation: 65.8%
  - Average latency: 0.88 µs
  - Power draw: 992 mW

[Progress] Warmup complete at simulation time: 1000.0
[Progress] Beginning main measurement period...

[Measurement] T=2000.0 | Collecting XYZ routing performance counters...
3D Mesh (Baseline):
  - Throughput: 524.0 Gbps
  - Hash Rate: 524.0 GH/s
  - Power: 1200 mW
  - Buffer Util: 89.8%
  - Thermal: 75.0 mW/mm²

3D Torus:
  - Throughput: 682.0 Gbps
  - Hash Rate: 682.0 GH/s
  - Power: 1150 mW
  - Buffer Util: 82.3%
  - Thermal: 72.0 mW/mm²

3D Hypercube:
  - Throughput: 845.0 Gbps
  - Hash Rate: 845.0 GH/s
  - Power: 1000 mW
  - Buffer Util: 74.2%
  - Thermal: 62.0 mW/mm²

[Power] T=4000.0 | Power and thermal analysis...
[Power] Recording steady-state measurements:
  - Mesh avg power/node: 5.56 mW
  - Torus avg power/node: 5.32 mW
  - Hypercube avg power/node: 4.63 mW

[Energy] T=6000.0 | Energy efficiency metrics...
[Energy] Energy per hash calculation:
```

```
  - Mesh: 2.29 pJ/hash
  - Torus: 1.69 pJ/hash
  - Hypercube: 1.18 pJ/hash

[Progress] T=8000.0 | Steady state verification...
[Progress] All topologies maintaining stable metrics
[Progress] Performance counters aligned with expectations
[Progress] Power measurements within ±1% tolerance

[Progress] T=10000.0 | Completing XYZ routing simulation...
[Progress] Final measurements align with steady state
[Progress] No anomalies detected in measurement period
```

```
\prompt mpirun -np 8 ./noc-model \\
--sync=3 \\
--end=10000.0 \\
--batch=1 --nkp=1 \\
--extramem=1000000 \\
--max-opt-mem=1000000 \\
--routing=XYZ \\
--topology=MESH,TORUS,HYPERCUBE \\
--baseline=MESH \\
--traffic-rate=1.0 \\
--traffic-unit=GIGA \\
--flit-size=256 \\
--node-type=PE \\
--router=DIJKSTRA

[Progress] Simulation time: 100.0
[Progress] Entering warmup period (10% of simulation time)...

[Warmup] Collecting baseline measurements for Dijkstra routing...
[Warmup] 3D Mesh (Dijkstra) warmup metrics:
  - Initial injection rate: 1.0 Gbps achieved
  - Buffer utilisation: 82.3%
  - Average latency: 1.42 µs
  - Power draw: 1192 mW

[Warmup] 3D Torus (Dijkstra) warmup metrics:
  - Initial injection rate: 1.2 Gbps achieved
  - Buffer utilisation: 76.8%
```

```
  - Average latency: 1.28 µs
  - Power draw: 1168 mW

[Warmup] 3D Hypercube (Dijkstra) warmup metrics:
  - Initial injection rate: 1.5 Gbps achieved
  - Buffer utilisation: 70.2%
  - Average latency: 1.02 µs
  - Power draw: 1042 mW

[Progress] Warmup complete at simulation time: 1000.0
[Progress] Beginning main measurement period...

[Measurement] T=2000.0 | Collecting Dijkstra routing performance counters...
3D Mesh (Baseline):
  - Throughput: 524.0 Gbps
  - Hash Rate: 524.0 GH/s
  - Power: 1200 mW
  - Buffer Util: 89.8%
  - Thermal: 75.0 mW/mm²

3D Torus:
  - Throughput: 598.0 Gbps
  - Hash Rate: 598.0 GH/s
  - Power: 1175 mW
  - Buffer Util: 85.4%
  - Thermal: 73.0 mW/mm²

3D Hypercube:
  - Throughput: 752.0 Gbps
  - Hash Rate: 752.0 GH/s
  - Power: 1050 mW
  - Buffer Util: 78.5%
  - Thermal: 65.0 mW/mm²

[Power] T=4000.0 | Power and thermal analysis...
[Power] Recording steady-state measurements:
  - Mesh avg power/node: 5.56 mW
  - Torus avg power/node: 5.44 mW
  - Hypercube avg power/node: 4.86 mW

[Energy] T=6000.0 | Energy efficiency metrics...
```

```
[Energy] Energy per hash calculation:
  - Mesh: 2.29 pJ/hash
  - Torus: 1.96 pJ/hash
  - Hypercube: 1.40 pJ/hash

[Progress] T=8000.0 | Steady state verification...
[Progress] All topologies maintaining stable metrics
[Progress] Performance counters aligned with expectations
[Progress] Power measurements within ±1% tolerance

[Progress] T=10000.0 | Completing Dijkstra routing simulation...
[Progress] Final measurements align with steady state
[Progress] No anomalies detected in measurement period
```

# References

[1] S. Abadal and E. Alarcon, "Data Conversion in Area-Constrained Applications: the Wireless Network-on-Chip Case," in *IEEE*, Jan. 2018, p. 1.

[2] K.-L. Tsai, H.-T. Chen, and Y.-A. Lin, "Power and Area Efficiency NoC Router Design for Application-Specific SoC by Using Buffer Merging and Resource Sharing," *ACM Transactions on Design Automation of Electronic Systems* (*TODAES*), vol. 19, no. 4, pp. 1–21, 2014.

[3] K. Brock, C. Edwards, R. Lannoo, *et al.*, "Power Crisis in SoC Design," in *IEEE Computer Society*, Mar. 2002, p. 538.

[4] S. Mazumdar and A. Scionti, "Ring-Mesh: A Scalable and High-Performance Approach for Manycore Accelerators," *The Journal of Supercomputing*, vol. 75, no. 12, 2019.

[5] J. Xu, W. Wolf, J. Henkel, and S. Chakradhar, "A design methodology for application-specific networks-on-chip," *ACM Transactions on Embedded Computing Systems* (*TECS*), vol. 5, no. 2, pp. 263–280, 2006. doi: 10.1145/1151074.1151076.

[6] A. Kumar, S. Jacob, T. Srikanthan, M. P. Vishwanath, and S. P. Mahapatra, "A Survey of Network-on-Chip Design Techniques for Manycore Systems," *ACM Computing Surveys* (*CSUR*), vol. 55, no. 3, pp. 1–39, 2022.

[7] A. A. Chien, "Manycore Architecture in the Post-Moore Era," *IEEE Micro*, vol. 41, no. 4, pp. 44–51, 2021.

[8] G. H. Loh and W.-f. Wong, "3D Network-on-Chip: A Tutorial," *ACM Journal on Emerging Technologies in Computing Systems* (*JETC*), vol. 17, no. 3, pp. 1–30, 2021.

[9] Y. Wang, D. Pan, and H. Li, "A survey on 3D NoCs: Architecture and design methodology," *Integration, the VLSI Journal*, vol. 85, p. 103 389, 2022.

[10] B. K. Joardar, J. Rao Doppa, P. P. Pande, and K. Chakrabarty, "NoC-enabled 3D Heterogeneous Manycore Systems for Big-Data Applications," in *2022 23rd International Symposium on Quality Electronic Design* (*ISQED*), 2022, pp. 1–6. doi: 10.1109/ISQED54688.2022.9806297.

[11] S. Kumar, A. Kumar, R. Gupta, S. Hari, and P. R. Pande, "A Survey of Simulation and Emulation Platforms for Network-on-Chip (NoC) Architectures," *ACM Computing Surveys* (*CSUR*), vol. 54, no. 4, pp. 1–37, 2021.

[12] Y.-H. Chen *et al.*, "Power-efficient 3D NoC design for real-time embedded systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 42, no. 7, pp. 1477–1489, 2023.

[13] T. A. El-Ghazawi *et al.*, "Scalable 3D Network-on-Chip Design and Optimization for Emerging Applications," *ACM Computing Surveys* (*CSUR*), vol. 55, no. 2, pp. 1–37, 2022.

[14] M. Ebrahimi, M. Daneshtalab, and J. Plosila, "Topology exploration of 3D NoCs using genetic algorithms," *IEEE Transactions on Parallel and Distributed Systems*, vol. 31, no. 4, pp. 775–788, 2020.

[15] S. Katoch, S. Chauhan, and V. Kumar, "A review on genetic algorithm: past, present, and future," *Multimedia Tools and Applications*, vol. 80, no. 5, pp. 8091–8126, 2021.

[16] S. Mistry, M. Bhadauria, and P. R. Pande, "Networks-on-Chip (NoCs) for 2.5D/3D many-core Systems-on-Chip: A review," *Microprocessors and Microsystems*, vol. 90, p. 104 817, 2023.

[17] O. Akgun *et al.*, "A 112-core 7nm 3D stacked microprocessor with fine-grained power management," *2019 IEEE International Solid-State Circuits Conference-*(*ISSCC*), pp. 152–154, 2019.

[18] A. J. Isfahani and A. Khademzadeh, "Multi-objective design space exploration of 3D NoCs using NSGA-II," *The Journal of Supercomputing*, vol. 75, no. 1, pp. 301–323, 2019.

[19] M. Daneshtalab, A. Afzali-Kusha, and J. Plosila, "Application-specific 3D NoC design using multi-objective evolutionary algorithms," *IEEE Transactions on Computers*, vol. 66, no. 3, pp. 392–405, 2017.

[20] A. E. Eiben and J. E. Smith, *Introduction to evolutionary computing*. Springer, 2015.

[21] R. Marjani *et al.*, "Genetic Algorithm-Based Optimization of Network-on-Chip Architecture for Efficient Resource Allocation in Multi-Core Systems," *Electronics*, vol. 12, no. 6, p. 1382, 2023.

[22] A. A. El-Shafei, M. S. Abdel-Majeed, and A. E. Ibrahim, "GA-Based Optimization of NoC Architectures for Real-Time Applications," in *2022 11th International Conference on Information and Communication Technology and Accessibility* (*ICTA*), IEEE, 2022, pp. 1–6.

[23] Y. Wang *et al.*, "Multi-objective optimization of network-on-chip architecture based on a multi-population genetic algorithm," *Journal of Systems Architecture*, vol. 136, p. 102 883, 2023.

[24] P. Bhattacharyya *et al.*, "Topology exploration of application specific network-on-chip using genetic algorithm," in *2022 3rd International Conference on Signal Processing and Communication* (*ICSPC*), IEEE, 2022, pp. 742–746.

[25] S. Khan *et al.*, "Hypergraph partitioning based design space exploration for network-on-chip architectures," in *Proceedings of the 2022 International Symposium on Physical Design*, 2022, pp. 298–303.

[26] K. Ashwani *et al.*, "Mapping of Convolutional Neural Network on NoC based architecture using Hypergraph," in *2022 13th International Conference on Computational Intelligence and Communication Networks* (*CICN*), IEEE, 2022, pp. 591–595.

[27] P. Bhattacharyya *et al.*, "Multi-objective optimisation of 3D NoC using hypergraph partitioning and NSGA-II," *IET Computers & Digital Techniques*, vol. 17, no. 4, pp. 277–287, 2023.

[28] Y. Luo *et al.*, "3-D NoC mapping and routing optimization based on a multi-objective genetic algorithm," *ACM Transactions on Embedded Computing Systems* (*TECS*), vol. 21, no. 5s, pp. 1–21, 2022.

[29] D. Chen *et al.*, "Accelerating machine learning inference on NoC-based MPSoCs: A survey," *Journal of Systems Architecture*, vol. 135, p. 102 824, 2023.

[30] J. Yao *et al.*, "A survey of machine learning techniques for network-on-chip design," *Journal of Systems Architecture*, vol. 123, p. 102 342, 2022.

[31] D. Salvatore *et al.*, "A Survey on Simulation and Emulation of Networks-on-Chip," *ACM Computing Surveys* (*CSUR*), vol. 56, no. 4, pp. 1–37, 2023.

[32] A. Jantsch and H. Tenhunen, *Networks on Chip*. Springer, 2018.

[33] N. E. Jerger *et al.*, "On-Chip Communication Architectures for Multicore Systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 4, pp. 882–898, 2022.

[34] R. R. Mishra *et al.*, "A survey on thermal-aware resource management techniques for manycore systems," *Journal of Systems Architecture*, vol. 137, p. 102 911, 2023.

[35] A. Pullini, F. Pini, P. Meloni, D. Atienza, and L. Benini, "Multi-core SoC Design with 2.5D and 3D Integration: Challenges and Opportunities," *IEEE Micro*, vol. 37, no. 4, pp. 28–38, 2017.

[36] D. Wallace *et al.*, "Physical design for advanced nodes: A tutorial on current and future challenges," in *Proceedings of the 2023 International Symposium on Physical Design*, 2023, pp. 310–315.

[37] S. Borkar, "Thousand-core chips: a technology perspective," in *Proceedings of the 44th annual Design Automation Conference*, ACM, 2007, pp. 746–749.

[38] M. Shafique, R. Hafiz, Y. Lee, and J. Henkel, "Cross-Layer Design Space Exploration for Manycore Systems-on-Chip," *ACM Journal on Emerging Technologies in Computing Systems* (*JETC*), vol. 13, no. 4, p. 54, 2017.

[39] M. M. Ziegler and S. Borkar, "Technology trends and opportunities at the end of Moore's law," *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, vol. 380, no. 2223, p. 20210117, 2022.

[40] O. Ros-Giralt *et al.*, "Comparative analysis of noc topologies for manycore socs," *ACM Computing Surveys* (*CSUR*), vol. 47, no. 4, pp. 1–31, 2015.

[41] A. Kumar and D. K. Sharma, "Globally Asynchronous Locally Synchronous Network-on-Chip for energy efficiency in multicore systems," in *2022 2nd International Conference on Intelligent Technologies* (*CONIT*), IEEE, 2022, pp. 1–6.

[42] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys* (*CSUR*), vol. 38, no. 1, 1–es, 2006.

[43] U. Black, *Computer Networks: Protocols, Standards, and Interfaces*. Prentice Hall PTR, 2000, pp. 248–252.

[44] N. Enright Jerger, M. Lipasti, and L.-S. Peh, "Circuit-Switched Coherence," *IEEE Computer Architecture Letters*, vol. 6, no. 1, pp. 5–8, 2007. doi: 10.1109/L-CA.2007.2.

[45] S. Paul and P. R. Pande, "Arbitration Mechanisms in Networks-on-Chip: A Survey," *ACM Computing Surveys* (*CSUR*), vol. 54, no. 3, pp. 1–37, 2021.

[46] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach*, 6th ed. Pearson Education, 2012, ch. 6, p. 443.

[47] S. Paul and P. R. Pande, "Flow control mechanisms in networks-on-chip: A survey," *ACM Computing Surveys* (*CSUR*), vol. 54, no. 8, pp. 1–37, 2021.

[48] A. Aldammas, A. Soudani, and A. Al-Dhelaan, "Flow control solution for efficient communication and congestion avoidance in NoC," in *2014 27th IEEE International System-on-Chip Conference* (*SOCC*), 2014, pp. 177–182. doi: 10.1109/SOCC.2014.6948922.

[49] K. Patel, P. R. Pande, M. Bhadauria, and G. Trajcevski, "A Survey of Network-on-Chip Design Methodologies from Application Perspective," *ACM Computing Surveys* (*CSUR*), vol. 54, no. 8, pp. 1–38, 2022.

[50] D. A. Sigüenza-Tortosa, *Proteo: The Development of a Practical Network-on-Chip*, 2005.

[51] C. Zeferino and A. Susin, "SoCIN: a parametric and scalable network-on-chip," *16th Symposium on Integrated Circuits and Systems Design, 2003. SBCCI 2003. Proceedings*, Jan. 2004. doi: 10.1109/sbcci.2003.1232824. (visited on 03/20/2024).

[52] W. Zhou, Y. Zhang, and Z. Mao, "An application specific NoC mapping for optimized delay," *International Conference on Design and Test of Integrated Systems in Nanoscale Technology, 2006. DTIS 2006.*, Jan. 2006. doi: 10.1109/dtis.2006.1708657. (visited on 03/20/2024).

[53] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," in *IEEE*, 2001, p. 684.

[54] N. Corporation, *NVIDIA Tegra K1 A New Era in Mobile Computing*.

[55] K. Tatas, K. Siozios, D. Soudris, and A. Jantsch, *Designing 2D and 3D Network-on-Chip Architectures*. New York, Ny Springer, 2014.

[56] F. T. Leighton, *Introduction to Parallel Algorithms and Architectures: Arrays . Trees . Hypercubes*. Morgan Kaufmann, 2019.

[57] P. P. Pande, C. Grecu, A. Ivanov, and R. A. Saleh, "Design of a switch for network on chip applications," *International Symposium on Circuits and Systems*, May 2003. doi: 10.1109/iscas.2003.1206235. (visited on 04/25/2023).

[58] T. N. Kamal Reddy, A. K. Swain, J. K. Singh, and K. K. Mahapatra, "Performance assessment of different Network-on-Chip topologies," in *2014 2nd International Conference on Devices, Circuits and Systems* (*ICDCS*), 2014, pp. 1–5. doi: 10.1109/ICDCSyst.2014.6926188.

[59] M. Dong *et al.*, "High-performance routing algorithm for hypercube network-on-chip based on virtual channel," *Microelectronics Reliability*, vol. 131, p. 114483, 2022.

[60] W. Abdelhadi *et al.*, "A survey on 3D integrated circuits and systems: Applications, design challenges, and technology trends," *Integration, the VLSI Journal*, vol. 84, p. 103 333, 2022.

[61] A. Lahiri, *Comprehensive Nanoscience and Nanotechnology* (*Second Edition*). Elsevier, 2017, pp. 456–458.

[62] D. J. Frank, P. M. Solomon, and O. Dokumaci, "Device scaling limits of Si MOSFETs and their application dependencies," *Proceedings of the IEEE*, vol. 89, no. 3, pp. 259–288, 2011. doi: 10.1109/5.915374.

[63] K. J. Kuhn, "Considerations for ultimate CMOS scaling," *IEEE Transactions on Electron Devices*, vol. 59, no. 7, pp. 1813–1828, 2012. doi: 10.1109/TED.2012.2193129.

[64] C. Auth, C. Allen, A. Blattner, *et al.*, "A 22nm High Performance and Low-Power CMOS Technology Featuring Fully-Depleted Tri-Gate Transistors, Self-Aligned Contacts and High Density MIM Capacitors," *Symposium on VLSI Technology*, pp. 131–132, 2012. doi: 10.1109/VLSIT.2012.6242496.

[65] Q. Zhang, H. Yin, and L. Meng, "Performance projection of scaled logic devices with novel GAA nanosheet and nanoribbon structures," *IEEE Transactions on Electron Devices*, vol. 67, no. 1, pp. 353–360, 2020. doi: 10.1109/TED.2019.2951503.

[66] N. Loubet, T. Hook, P. Montanini, *et al.*, "Stacked CMOS demonstration of high density integration capability of CFET architecture," *Symposium on VLSI Technology*, pp. 1–2, 2020. doi: 10.1109/VLSITechnology18217.2020.9265056.

[67] Y. Lee, H. Kim, S. Park, *et al.*, "Advanced CFET architecture for sub-3nm logic technology," *IEEE Transactions on Electron Devices*, vol. 70, no. 2, pp. 892–898, 2023. doi: 10.1109/TED.2022.3228611.

[68] A. S. Tanenbaum and H. Bos, *Modern operating systems*. Pearson, 2015.

[69] H. Hsin, E. Chang, C. Chao, and A. Wu, "Regional ACO-based routing for load-balancing in NoC systems," in *Proceedings of the International Symposium on VLSI Design, Automation and Test* (*VLSI-DAT*), 2010, pp. 370–373. doi: 10.1109/VLSI-DAT.2010.5486392.

[70] R. Manevich, I. Cidon, a. kolodny avinoam, and I. Walter, "Centralized Adaptive Routing for NoCs," *IEEE Computer Architecture Letters*, vol. 9, pp. 57–60, Feb. 2010. doi: 10.1109/l-ca.2010.17. (visited on 11/02/2020).

[71] X. Zhou, L. Liu, Z. Zhu, and D. Zhou, "A routing aggregation for load balancing network-on-chip," *Journal of Circuits, Systems and Computers*, vol. 24, no. 09, p. 1 550 137, 2015.

[72] R. Gindin, I. Cidon, and I. Keidar, "NoC-based FPGA: architecture and routing," in *IEEE*, 2007, p. 253.

[73] P. K. Sahu and S. Chattopadhyay, "A survey on application mapping strategies for Network-on-Chip design," *Journal of Systems Architecture*, vol. 59, no. 1, pp. 60–76, 2013, issn: 1383-7621. doi: https://doi.org/10.1016/j.sysarc.2012.10.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1383762112000902.

[74] J. Qadir *et al.*, "Performance analysis of network-on-chip architectures using machine learning," *SN Applied Sciences*, vol. 5, no. 1, pp. 1–18, 2023.

[75] B. Li, L. Zhao, R. Iyer, *et al.*, "CoQoS: Coordinating QoS-aware shared resources in NoC-based SoCs," *Journal of Parallel and Distributed Computing*, vol. 71, no. 5, pp. 700–713, 2011.

[76] J. Ng, X. Wang, A. K. Singh, and T. Mak, "DeFrag: Defragmentation for Efficient Runtime Resource Allocation in NoC-Based Many-core Systems," *2015 23rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing*, Mar. 2015. doi: 10.1109/pdp.2015.16. (visited on 03/20/2024).

[77] M. D. Grammatikakis, A. Papagrigoriou, P. Petrakis, K. Harteros, and G. Kornaros, "Load balancing, broadcast, and scatter primitives for efficient multicore applications," pp. 23–28, Dec. 2015. (visited on 03/19/2024).

[78] H. Kalla *et al.*, "Application-specific mapping on mesh-based network-on-chip using linear programming," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 24, no. 5, pp. 1668–1681, 2016.

[79] M. Zare *et al.*, "Mapping of Convolutional Neural Networks on 3D NoC Using a Hybrid Metaheuristic Algorithm," *Electronics*, vol. 12, no. 8, p. 1858, 2023.

[80] H. Lee *et al.*, "Kilocore-Scale, High-Performance Neuromorphic Computing with KiloEdge-Scale On-Chip Communication Fabric," in *2023 ACM/IEEE 45th International Conference on Computer Architecture (ISCA)*, IEEE, 2023, pp. 1–15.

[81] Y. Dong *et al.*, "Towards exascale and mega-core chip design: A comprehensive survey," *ACM Computing Surveys (CSUR)*, vol. 56, no. 4, pp. 1–38, 2023.

[82] S. Akyurek *et al.*, "Evaluating 3D NoC Architectures for High-Performance and Energy-Efficient Manycore Systems," *IEEE Transactions on Parallel and Distributed Systems*, 2024.

[83] P. Hartono *et al.*, "Machine Learning for Network-on-Chip Architecture Exploration and Optimization," in *Machine Learning for Emerging Network Technologies*, Springer, 2023, pp. 43–71.

[84] M. Arian *et al.*, "Game theory for network-on-chip design and optimization: A survey," *ACM Computing Surveys* (*CSUR*), vol. 55, no. 8, pp. 1–36, 2022.

[85] L. Benini and G. De Micheli, "Networks on Chip: A New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[86] N. Kandlikar, M. Mansouri, and A. Nojeh, "A survey of 3D Network-on-Chip architectures," *IEEE Transactions on Very Large Scale Integration* (*VLSI*) *Systems*, vol. 25, no. 8, pp. 2168–2181, 2017.

[87] L. Euler, "Solutio problematis ad geometriam situs pertinentis," *Commentarii academiae scientiarum Petropolitanae*, vol. 8, pp. 128–140, 1741.

[88] S. Renubala and K. S. Dhanalakshmi, "Trust based secure routing protocol using fuzzy logic in wireless sensor networks," in *2014 IEEE International Conference on Computational Intelligence and Computing Research*, 2014, pp. 1–5. doi: 10.1109/ICCIC.2014.7238435.

[89] W. Meng, H. Xia, and H. Song, "A Dynamic Trust Model Based on Recommendation Credibility in Grid Domain," in *2009 International Conference on Computational Intelligence and Software Engineering*, 2009, pp. 1–4. doi: 10.1109/CISE.2009.5363348.

[90] E. A. Bender and S. G. Williamson, *Lists, decisions and graphs with an introduction to probability*. Online: S. Gill Williamson, Oct. 2010. [Online]. Available: https://shorturl.at/noEQ9.

[91] R. Diestel, *Graph Theory*. Springer, 2017, Foundational text on graph theory. Reference for all basic graph theory concepts and terminologies.

[92] T. Kosicki, "Graph representation of n-dimensional space," *Advances in Manufacturing*, vol. 2, no. 1, pp. 54–60, Mar. 2014, issn: 2195-3597. doi: 10.1007/s40436-014-0065-2. [Online]. Available: https://doi.org/10.1007/s40436-014-0065-2.

[93] T. G. Kolda and B. W. Bader, "Tensor decompositions and applications," *SIAM review*, vol. 51, no. 3, pp. 455–500, 2009, Detailed discussion on tensor decompositions and their applications in various domains. Suitable reference for the concept of 3-dimensional adjacency tensors.

[94] W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," *Proceedings of the 38th annual Design Automation Conference*, pp. 684–689, 2001.

[95] J. Kim, C. Nicopoulos, D. Park, *et al.*, "A technology-aware and energy-oriented topology exploration for on-chip networks," in *Proceedings of the 20th international conference on Parallel and distributed processing*, Use for discussions on technology-aware and energy-oriented topology explorations., IEEE, 2007, pp. 1–8.

[96] M. Palesi and T. Givargis, "Design space exploration for networks on chips," *ACM Transactions on Design Automation of Electronic Systems* (*TODAES*), vol. 15, no. 1, pp. 1–34, 2010, Cite for in-depth discussions on NoC design space exploration.

[97] P. R. Pande, C. Grecu, K. Goossens, and M. Bhadauria, *On-Chip Communication Architectures for Multi-Processor System-on-Chip*. Springer International Publishing, 2020, pp. 109–110.

[98] A. Jantsch and H. Tenhunen, *Networks on Chip*. Springer, 2005.

[99] A. S. Tanenbaum and D. J. Wetherall, *Computer Networks*. Upper Saddle River, NJ: Prentice Hall, 2011.

[100] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach*. Pearson, 2016.

[101] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2004.

[102] P. P. Pande, C. Grecu, M. Jones, A. Ivanov, and R. Saleh, "Performance evaluation and design trade-offs for network-on-chip interconnect architectures," 2005, Reference for a performance analysis of NoC architectures, including bandwidth and throughput.

[103] L. Benini and G. De Micheli, "Networks on Chips: A New SoC Paradigm," *Computer*, vol. 35, no. 1, pp. 70–78, 2002.

[104] N. E. Jerger, L.-S. Peh, and M. H. Lipasti, "NoC architectures and protocols for the future manycore era," *IEEE Computer*, vol. 45, no. 11, pp. 24–31, 2010, Reference for future NoC architectures and protocols.

[105] E. Dijkstra, "A Note on Two Problems in Connexion with Graphs.," *Numerische Mathematik*, vol. 1, pp. 269–271, 1959. [Online]. Available: http://eudml.org/doc/131436.

[106] J. F. Kurose and K. W. Ross, *Computer networking: a top-down approach*, 8th ed. Pearson Education, 2021, pp. 16–19, 38–41.

[107] G. Beanato, A. Cevrero, G. De Micheli, and Y. Leblebici, "Impact of data serialization over TSVs on routing congestion in 3D-stacked multi-core processors," *Microelectronics Journal*, vol. 51, pp. 38–45, 2016, issn: 0026-2692. doi: https://doi.org/10.1016/j.mejo.2015.12.004. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0026269215003110.

[108] D. Khalil, Y. Ismail, M. Khellah, T. Karnik, and V. De, "Analytical Model for the Propagation Delay of Through Silicon Vias," in *9th International Symposium on Quality Electronic Design (isqed 2008)*, 2008, pp. 553–556. doi: 10.1109/ISQED.2008.4479795.

[109] R. Nasri, M. Hassan, and S. Reda, "A comprehensive survey on 3d noc design challenges, opportunities, and future research directions," *Integration*, vol. 86, pp. 104–123, 2023.

[110] A. El-Sayed, M. F. El-Hawary, T. Mostafa, and W. Badawy, "Survey on network-on-chip design approaches: Routing algorithms, topologies, and performance metrics," *ACM Computing Surveys (CSUR)*, vol. 55, no. 11, pp. 1–38, 2022.

[111] I. Ahmad, F. Spagnolo, R. Mariani, and A. Cosenza, "Energy-efficient 3d network-on-chip design and optimization through machine learning-based design space exploration," *Journal of Systems Architecture*, vol. 137, p. 102 684, 2023.

[112] B. Feero and P. P. Pande, *Performance Evaluation for Three-Dimensional Networks-On-Chip*, Jan. 2007. [Online]. Available: https://doi.org/10.1109/isvlsi.2007.79.

[113] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*. Morgan Kaufmann, 2001.

[114] M. Daneshtalab, M. Ebrahimi, S. Dytckov, and J. Plosila, "In-order delivery approach for 2D and 3D NoCs," *The Journal of Supercomputing*, vol. 71, no. 8, pp. 3043–3064, Aug. 2015. doi: 10.1007/s11227-014-1339-y.

[115] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms*. The Mit Press, 2022.

[116] S. Verma, M. Pant, and V. Snasel, "A Comprehensive Review on NSGA-II for Multi-Objective Combinatorial Optimization Problems," *IEEE Access*, vol. 9, pp. 57 757–57 791, 2021. doi: 10.1109/ACCESS.2021.3070634.

[117] A. Bretto, *Hypergraph theory: An introduction*. Springer Science & Business Media, 2013.

[118] S. Sivanandam and S. Deepa, *Introduction to genetic algorithms*. Springer Science & Business Media, 2007.

[119] *Intel® Xeon® D-2899NT Processor (30M Cache, up to 3.10 GHz)*, https://www.intel.com/content/www/us/en/products/sku/235750/intel-xeon-d2899nt-processor-30m-cache-up-to-3-10-ghz/specifications.html, Accessed: 2024-10-04, Intel.

[120] *GeForce RTX 3060 Ti*, https://www.nvidia.com/en-us/geforce/graphics-cards/30-series/rtx-3060-3060ti/, Accessed: 2024-10-04, NVIDIA.

[121] *OpenMP Application Programming Interface Version 5.2*, OpenMP Architecture Review Board, Nov. 2021. [Online]. Available: https://www.openmp.org/wp-content/uploads/OpenMP-API-Specification-5-2.pdf.

[122] *CUDA Toolkit Documentation*, Accessed: 2024-10-04, NVIDIA, 2023. [Online]. Available: https://docs.nvidia.com/cuda/.

[123] M. M. Rahman, P. Liljeberg, and J. Plosila, "Noc simulation and evaluation with gem5," *Proceedings of the 2nd International Workshop on Network on Chip Architectures*, pp. 31–36, 2013.

[124] M. Jimenez, D. Villa, R. Cordoba, *et al.*, "Time-accurate simulation of multiprocessor systems with heterogeneous reconfigurable hardware using systemc," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, IEEE, 2010, pp. 1313–1318.

[125] J. Banks, J. S. Carson, B. L. Nelson, and D. M. Nicol, *Discrete-event system simulation*, 4th. Pearson Education, 2005, isbn: 978-0138150372.

[126] Y. Li, Y. He, Y. He, and Y. Xie, "Architecture and evaluation of a cycle-accurate network-on-chip simulator," in *Proceedings of the 2010 Asia and South Pacific Design Automation Conference*, IEEE, 2010, pp. 540–545.

[127] R. Drechsler, "Embedded systems," in *Advanced Formal Verification*, Springer, 2012, pp. 437–499.

[128] M. Guthaus, J. S. Ringenberg, C. D. Ernst, T. M. Austin, T. Mudge, and R. B. Brown, "Mipsy: A mips r4000 simulator," in *Proceedings of the sixth workshop on Computer architecture education*, 2001, pp. 1–13.

[129] A. Hoffmann, A. Nohl, H. Braun, and H. Meyr, "Design and implementation of a cycle accurate network-on-chip simulator," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, IEEE, 2010, pp. 1307–1312.

[130] N. Puvača, V. Milutinović, M. Tomašević, P. Radojković, and V. Štrbac, "Noc design and implementation for real-time multi-processor system-on-chip," in *2013 IEEE 19th International Conference on Embedded and Real-Time Computing Systems and Applications*, IEEE, 2013, pp. 243–250.

[131] J. Xu, S. Kumar, and S. Kundu, "Exploring the design space of a network-on-chip architecture using a cycle-accurate simulator," *VLSI design*, vol. 2007, 2007.

[132] J. Eggers, A. Sheibanyrad, and H. Vandierendonck, "Limits of cycle-accurate simulation for multicore processors with high clock speeds," in *Proceedings of the 44th annual Simulation Symposium*, IEEE, 2011, pp. 3–12.

[133] T. Bjerregaard and S. Mahadevan, "Nocsim: A cycle-accurate network-on-chip simulator," in *Proceedings of the 3rd international conference on Simulation tools and techniques*, 2011, pp. 1–10.

[134] G. Guindani, M. B. da Rosa, L. Carro, and F. Wagner, "A survey on simulation tools for network-on-chip," *ACM Computing Surveys* (*CSUR*), vol. 53, no. 3, pp. 1–37, 2020.

[135] P. K. Rathore, R. K. Pandey, U. Shankar, and R. Buyya, "A comprehensive survey of network-on-chip simulators: Features, methodologies, and future trends," *Journal of Network and Computer Applications*, vol. 224, p. 103 433, 2023.

[136] C. D. Carothers, D. Bauer, and S. Pearce, "ROSS: A High-Performance, Low Memory, Modular Time Warp System," *Journal of Parallel and Distributed Computing*, vol. 62, no. 11, pp. 1648–1669, 2002.

[137] J. M. Miller and C. D. Carothers, "Detailed Network-on-Chip Simulation Using ROSS," in *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, ACM, 2018, pp. 121–132.

[138] Z. Lu, R. Thid, M. Karlsson, and A. Jantsch, "NoCSim: A Cycle-Accurate Network on Chip Simulator," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 29, no. 6, pp. 969–980, 2010.

[139] C. Wang and N. Bagherzadeh, "Design and Evaluation of Network-on-Chip Architectures Using Performance Models," in *2019 IEEE International Conference on Computer Design* (*ICCD*), IEEE, 2019, pp. 438–445.

[140] N. Binkert, B. Beckmann, G. Black, *et al.*, "The gem5 Simulator," *ACM SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[141] N. Agarwal, T. Krishna, L.-S. Peh, and N. K. Jha, "GARNET: A Detailed On-Chip Network Model Inside a Full-System Simulator," in *2009 IEEE International Symposium on Performance Analysis of Systems and Software*, IEEE, 2009, pp. 33–42.

[142] W. J. Dally and B. Towles, "Principles and Practices of Interconnection Networks," *Morgan Kaufmann Publishers Inc.*, 2004.

[143] N. Jiang, D. U. Becker, G. Michelogiannakis, and W. J. Dally, "Network-on-Chip Architecture Modeling for System-Level Simulation," in *2013 IEEE 21st Annual Symposium on High-Performance Interconnects*, IEEE, 2013, pp. 107–114.

[144] N. Jiang, D. U. Becker, G. Michelogiannakis, *et al.*, "BookSim 2.0 User's Guide," *Stanford University*, 2013.

[145] C. Sun, C.-H. O. Chen, G. Kurian, and L. Wei, "An Integrated On-Chip Network Design and Optimization Framework," in *2012 IEEE/ACM International Symposium on Networks on Chip*, IEEE, 2012, pp. 139–146.

[146] B. Maurer, T. C. Nelms, and L. Swartz, ""When perhaps the real problem is money itself!": The practical materiality of Bitcoin," *Social Semiotics*, vol. 23, no. 2, pp. 261–277, 2013. doi: https://doi.org/10.1080/10350330.2013.777594.

[147] A. K. Kashyap, "The Lehman Brothers Collapse and the Role of the Repo Market," *Journal of Economic Perspectives*, vol. 35, no. 1, pp. 67–88, 2021. doi: 10.1257/jep.35.1.67.

[148] V. V. Acharya, M. Richardson, S. van Nieuwerburgh, and L. J. White, *Guaranteed to Fail: Fannie Mae, Freddie Mac, and the Debacle of Mortgage Finance*. Princeton University Press, 2011.

[149] S. Nakamoto, *Bitcoin: A Peer-to-Peer Electronic Cash System*, Accessed: 2015-07-01, Dec. 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[150] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton: Princeton University Press, 2016.

[151] J. Davidson and M. Naveed, *The Digital Coin Revolution-CryptoCurrency-How to Make Money Online*, 1st, E. B. Series, Ed. JD-Bix Publishing, 2013.

[152] B. Scott, "How can Cryptocurrency and Blockchain technology play a role in building social and solidarity finance?" *UNRISD Occasional Paper*, vol. 17, pp. 1–64, 2011. doi: 10.1007/s10273-011-1262-2.

[153] Z. Zheng, S. Xie, H. Dai, X. Chen, and H. Wang, "An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends," in *Proceedings - 2017 IEEE 6th international congress on big data, BigData Congress 2017*, IEEE, Jun. 2017, pp. 557–564. doi: 10.1109/BigDataCongress.2017.85.

[154] A. M. Antonopoulos, *Mastering Bitcoin: Programming the Open Blockchain*. O'Reilly Media, Inc., 2017.

[155] I. Bashir, *Blockchain Technology: Principles and Applications*. Chapman and Hall/CRC, 2020.

[156] M. J. Dworkin, "SHA-3 standard: Permutation-based hash and extendable-output functions," *Federal Inf. Process. Stds.(NIST FIPS)-202*, 2015.

[157] J.-P. Aumasson, *Serious cryptography: a practical introduction to modern encryption*. No Starch Press, 2017.

[158] J. Bonneau, A. Miller, J. Clark, A. Narayanan, J. A. Kroll, and E. W. Felten, "SoK: Research perspectives and challenges for bitcoin and cryptocurrencies," in *2015 IEEE Symposium on Security and Privacy*, IEEE, 2015, pp. 104–121.

[159] M. Vilim, H. Duwe, and R. Kumar, "Approximate bitcoin mining," in *Proceedings of the 53rd Annual Design Automation Conference* (*DAC*), IEEE, 2016, pp. 1–6.

[160] A. D. Taylor, "A Survey of Bitcoin Mining Hardware and Trends," *Energies*, vol. 15, no. 3, p. 1149, 2022.

[161] J. Garay, A. Kiayias, and N. Leonardos, "Bitcoin in the real world," in *Proceedings of the 2015 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2015, pp. 431–444.

[162] A. Baliga, "Understanding blockchain consensus models," *Persistent*, vol. 4, no. 1, p. 14, 2017.

[163] S. Baqar *et al.*, "Energy-Efficient and High-Performance NoC-Based Cryptographic System Design," *IEEE Transactions on Very Large Scale Integration* (*VLSI*) *Systems*, vol. 29, no. 12, pp. 2527–2539, 2021.

[164] W. Stallings, *Cryptography and network security: principles and practice*. Pearson Education, Inc., 2017.

[165] National Institute of Standards and Technology, *Secure Hash Standard* (*SHS*), Federal Information Processing Standards Publication 180-4, Accessed: 2024-02-01, Aug. 2015. doi: 10.6028/NIST.FIPS.180-4. [Online]. Available: https://doi.org/10.6028/NIST.FIPS.180-4.

[166] W. Zhao, R. Chellappa, P. J. Phillips, and A. Rosenfeld, "Face recognition: A literature survey," *ACM computing surveys* (*CSUR*), vol. 35, no. 4, pp. 399–458, 2003.

[167] R. Ranjan, S. Sankaranarayanan, A. Bansal, *et al.*, "Deep learning for understanding faces: Machines may be just as good, or better, than humans," in *IEEE Signal Processing Magazine*, IEEE, vol. 35, 2018, pp. 66–83.

[168] A. E. Eiben, J. E. Smith, *et al.*, "Introduction to evolutionary computing," vol. 53, 2003.