USING NETWORK-ON-CHIP STRUCTURE IN DEEP NEURAL NETWORK ACCELERATOR DESIGN

DERİN SİNİR AĞI HIZLANDIRICI TASARIMLARINDA YONGA-ÜSTÜ-AĞ YAPISININ KULLANIMI

FURKAN NACAR

PROF. DR. SÜLEYMAN TOSUN
Supervisor

Submitted to
Graduate School of Science and Engineering of Hacettepe University
as a Partial Fulfillment to the Requirements
for the Award of the Degree of Master of Science
in Computer Engineering

January 2024

ABSTRACT

USING NETWORK-ON-CHIP STRUCTURE IN DEEP NEURAL NETWORK ACCELERATOR DESIGN

Furkan Nacar
Master of Science , Computer Engineering
Supervisor: Prof. Dr. Süleyman TOSUN
January 2024, 102 pages

The widespread adoption of Deep Neural Networks (DNNs) in various fields, such as image and speech recognition, natural language processing (NLP), and autonomous systems, has been noted. However, the computational cost of these networks is often prohibitively high due to the large number of communicating layers and neurons and the significant amount of energy consumed. To address these challenges, developing new architectures to accelerate DNNs is necessary. In this thesis, a Network-on-Chip (NoC)-based DNN accelerator is proposed, taking into consideration both fully connected and partially connected DNN models. Heuristic methods, including Integer Linear Programming (ILP) and Simulated Annealing (SA), are utilized to group the neurons, to minimize the total volume of data among the groups. The neurons are then mapped onto a 2D mesh NoC fabric, utilizing ILP and SA, to minimize the system's total communication cost. The proposed design is novel in that it addresses the issue of high data communication in DNNs by utilizing the scalable, low-overhead, and energy-efficient NoC communication structure. Through extensive experimentation on various benchmarks and DNN models, an average improvement of 40% in communication cost has been observed. The proposed design targets low-overhead inferencing and training DNNs on edge devices in the Internet-of-Things (IoT) era, with a combination with cloud computing. The results of this thesis provide a new approach for the acceleration of DNNs and can be applied to various fields, such as edge computing, IoT, autonomous systems, computer vision, natural language processing, speech recognition, and cloud computing.

Keywords: Deep Neural Network (DNN), Accelerators, Network-on-Chip (NoC), Mapping Techniques, Integer Linear Programming (ILP), Simulated Annealing (SA), Optimization, Comparative Study

ÖZET

DERİN SİNİR AĞI HIZLANDIRICI TASARIMLARINDA YONGA-ÜSTÜ-AĞ YAPISININ KULLANIMI

Furkan Nacar
Yüksek Lisans, Bilgisayar Mühendisliği
Danışman: Prof. Dr. Süleyman TOSUN
Ocak 2024, 102 sayfa

Son yıllarda görüntü ve konuşma tanıma, doğal dil işleme (NLP) ve otonom sistemler gibi çeşitli alanlarda kaydedilen ilerlemelerde Derin Sinir Ağları (DNN'ler) yaygın olarak benimseniyor. Bu alanlardaki güncel problemlerin giderek karmaşık hale gelmesi, sinir ağlarının iletişim kuran katmanlarının ve bu katmanlarda bulunan nöronların sayısının artmasına neden olmuştur. Bu nedenle kullanılan sinir ağlarının enerji tüketimi ve çalışma süresi gibi maliyetleri artırmıştır. Bu maliyetleri karşılamak için sinir ağlarının çalışmasını hızlandıracak yeni mimarilerin geliştirilmesi gerekmektedir. Bilgisayarların çalıştırıldığı uygulamaya bağlı olarak işlem birimlerinin işlevi için en uygun olan birim üzerinde çalıştırılması ve bu şekilde donanımın uygulamaya özelleşmesi heterojen mimari adı altında giderek yaygınlaşmaktadır. Bu tezde, sinir ağı katmanlarının hem tam bağlı hem de kısmen bağlı sinir ağı modelleri dikkate alınarak Yonga-Üstü-Ağ (NoC) tabanlı bir hızlandırıcı tasarımı önerilmiştir. Yonga-Üstü-Ağ yapısının sunduğu çip üzerindeki her bir işlem elemanının kendi yönlendiricisine sahip olması ve işlem elemanlarının düzenli bir yapıya sahip olması, çip üzerindeki veri iletişimini muadillerinden ileri bir seviyeye taşımaktadır. Sinir ağındaki nöronların gruplandırılması ve bu işlem elemanları üzerinde çalıştırılması sağlanan bu iletişim altyapısı, iyi bir seçenek haline gelmektedir. İşlem elemanlarında yapılan hesaplamanın, o işlem elemanına iletilen veri kadar olacağından, veri iletişimi yoğunluğu Yonga-Üstü-Ağ yapısı üzerinde yapılacak olan hızlandırma modellerinin karşılaştırmasında bir kıstas olarak kullanılabilir. Neuron grupları arasındaki veri hacmini en aza indirmek amacıyla nöronları gruplandırmada tamsayılı doğrusal programlama (ILP) ve simüle tavlama (SA) gibi sezgisel yöntemler analiz edilmiştir. Daha sonra nöronlar, sistemin toplam iletişim maliyetini en aza indirmek için ILP ve SA kullanılarak 2 boyutlu Yonga-Üstü-Ağ yapısına eşlenecektir. Böylece görülecektir ki Yonga-Üstü-Ağ yapısı kullanılarak sinir ağı hızlandırıcısı tasarımı iki aşamalı bir problemdir: Nöron gruplandırma ve grupların Yonga-Üstü-Ağ üzerine eşlenmesi. Bu tezde önerilen tasarım, Yonga-Üstü-Ağ yapısı üzerindeki sinir ağları hızlandırıcılarında yüksek veri iletişimi sorununu ele alması bakımından yenidir. Çeşitli DNN modelleri üzerinde yapılan kapsamlı deneyler, önerilen tasarım ile iletişim maliyetinde ortalama %40'lık bir iyileşme göstermektedir. Önerilen tasarım, DNN'lerin hızlandırılması için yeni bir yaklaşım sunmakta olup IoT, otonom sistemler, doğal dil işleme, konuşma tanıma ve bulut bilişim gibi çeşitli alanlara uygulanabilir.

Anahtar Kelimeler: Derin Sinir Ağlar, Hızlandırıcılar, Yonga-üstü-Ağ, Tamsayılı Doğrusal Programlama, Optimizasyon

ACKNOWLEDGEMENTS

First and foremost I am immensely grateful to my supervisor, Prof. Dr. Süleyman Tosun, whose unwavering support, guidance, and dedication were instrumental in this research. His tireless efforts and commitment to my academic growth have been invaluable. I appreciate all of his lasting patience and neverending positivity throughout my study. Without his persistent encouragement and endless support, it would have been impossible to finish this research.

I extend my sincere appreciation to Dr. Selma Dilek for generously granting me access to her ILP formulations and research materials. Her willingness to share her expertise and positive attitude has been beneficial in this research. I would also like to thank Alperen Çakın, my esteemed colleague, for his willingness to share his extensive knowledge and experience in the field of optimization. His insights and collaboration have been invaluable throughout this journey.

I owe my gratitude to Dr. İsmail Uyanık, for his encouragement in pursuing academic purposes.

I am thankful to my superiors at work, Akın Yılmaz and Fatih İleri, for their understanding and tolerance during this academic pursuit.

I want to thank my sister, Emine, for being a lifelong ally. Her influence and presence have contributed to all the qualities I cherish in myself. To my mother, Fatma, I extend my deepest gratitude for the love she has given me. It is her love that has enabled me to extend love to the world around me. I also want to express my thanks to my father, Hasan, even though it made some things impossible for him, he made everything possible for me.

I wish to thank Sultan for making everything in my life easier. Recognizing that the choices we make define our identity, I appreciate your decision to choose us. I dedicate this work to you.

CONTENTS

TABLES

FIGURES

ABBREVIATIONS

AI	:	**A**rtificial **I**ntelligence
ANN	:	**A**rtifical **N**eural **N**etwork
ASIC	:	**A**pplication **S**pecific **I**ntegrated **C**ircuit
CNN	:	**C**onvolutional **N**eural **N**etwork
CPU	:	**C**entral **P**rocessing **U**nit
DNN	:	**D**eep **N**eural **N**etwork
DRAM	:	**D**ynamic **R**andom **A**ccess **M**emory
FPGA	:	**F**ield **P**rogrammable **G**ate **A**rray
GPU	:	**G**raphical **P**rocessing **U**nit
HM	:	**H**euristic **M**ethod
ILP	:	**I**nteger **L**inear **P**rogramming
IoT	:	**I**nternet **O**f **T**hings
NLP	:	**N**atural **L**anguage **P**rocessing
NoC	:	**N**etwork **O**n **C**hip
NRE	:	**N**onrecurring **E**ngineering
OSI	:	**O**pen **S**ystems **I**nterconnection
PE	:	**P**rocessing **E**lement
SA	:	**S**imulated **A**nnealing
SNN	:	**S**piking **N**eural **N**etwork
SRAM	:	**S**tatic **R**andom **A**ccess **M**emory

1. INTRODUCTION

Deep Neural Networks (DNNs) [1] are computational architectures that simulate the tangled neural connections of the human brain to process and understand complex data. DNNs have emerged as a pivotal technology with a transformative impact across various domains, owing to their ability to unravel complex patterns and perform complex tasks. These networks have become instrumental in revolutionizing computer vision, Natural Language Processing (NLP), healthcare, finance, autonomous systems, speech recognition, industrial processes, and entertainment. In the realm of computer vision, DNNs excel in object detection and recognition, enabling machines to precisely identify objects in images and videos [2]. Moreover, they play a crucial role in image segmentation, a critical task in many vision applications.

In the context of NLP, DNNs have achieved remarkable milestones, powering machine translation [3] for seamless communication across languages. They excel in sentiment analysis, accurately discerning emotions from textual data [4], and contribute to language generation tasks, enhancing applications like chatbots and content creation. Healthcare leverages DNNs for medical diagnosis, especially in the analysis of medical images like MRI scans and CT scans [5]. In finance, DNNs have proven their utility in algorithmic trading, aiding in predicting market trends and optimizing trading strategies [6]. They also find relevance in credit scoring applications [7].

In the domain of autonomous systems, DNNs are a cornerstone for self-driving cars [8, 9]. Speech recognition applications [10] greatly benefit from DNNs, accurately converting spoken language into text[11]. Industries adopt DNNs for predictive maintenance, anticipating machinery failures through analysis of sensor data [12], and for enhancing quality control processes [13]. Furthermore, DNNs play a pivotal role in entertainment, contributing to content recommendation systems tailored to user preferences [14] and even generating artistic content. These diverse applications of DNNs underscore their transformative potential, shaping innovation and redefining industries across the globe.

In numerous instances, the computational requirements of Deep Neural Networks surpass the capacities of conventional processors like Central Processing Units (CPUs) and Graphics Processing Units (GPUs). So, the computational complexity of DNNs demands efficient hardware accelerators to achieve real-time performance.

Hardware accelerators are specialized components designed to enhance the efficiency and speed of specific computational tasks. Among these, DNN accelerators constitute a subset tailored for the efficient processing of DNNs. These dedicated hardware units are intricately crafted to optimize the operational efficiency and overall performance of Deep Neural Network [15].

Primarily, DNN accelerators offer a substantial boost to DNN performance, enabling efficient real-time processing of substantial data volumes and swift execution of complex tasks. Such acceleration proves particularly critical in applications where rapid response times are important, such as in scenarios involving mobile autonomous vehicles [9] or virtual assistants [10]. Furthermore, DNN accelerators exhibit pronounced energy efficiency compared to traditional processors, a virtue derived from their meticulous design tailored to the specific computational demands of DNNs. This feature holds particular significance in the context of mobile and IoT devices, which deal with demanding power constraints [16]. Finally, as DNNs continually expand in both complexity and scale, the conventional processors may present a performance bottleneck. To mitigate this concern, DNN accelerators are crafted to seamlessly accommodate more sophisticated models, ensuring their sustained enhancement in performance as DNN technology evolves [17].

Network-on-Chip (NoC) architectures have emerged as a promising solution for designing efficient DNN accelerators, providing high-bandwidth communication and parallel processing capabilities. One critical aspect of designing NoC-based DNN accelerators is the mapping of neural network layers onto the NoC Processing Elements (PEs). The mapping scheme can significantly impact the performance and energy efficiency of the accelerator. In particular, the grouping and mapping of neurons within a layer can greatly influence data communication and processing load balancing.

In this study, the focus is on investigating various neuron grouping and mapping methods for NoC-based DNN accelerators. The aim is to compare and evaluate different approaches to identify the most effective method for optimizing communication and computational efficiency while maintaining high accuracy.

1.1. Scope Of The Thesis

This thesis mainly focuses on the proposed Network-on-Chip-based Deep Neural Network accelerator, which has the potential to impact the field of Artificial Intelligence (AI) by improving the processing speed and energy efficiency of DNNs, which are widely used in various AI applications such as image and speech recognition, natural language processing, and autonomous vehicles. By optimizing the communication cost between neuron groups, the proposed solution can reduce the time required for DNN computations, enabling faster and more efficient inference and training. Furthermore, the use of Integer Linear Programming (ILP) and Simulated Annealing (SA) algorithms also provides a more systematic and effective approach to optimizing DNN architectures. Overall, this work has the potential to advance the state-of-the-art in DNN acceleration and contribute to the development of more powerful and efficient AI systems.

1.2. Contributions

The main contributions of this work can be summarized as follows:

- A novel approach to designing DNN accelerators based on 2D mesh NoC architectures, accommodating both fully connected and partially connected DNNs, is presented in this study. To this end, the neuron grouping and node mapping problems are formally defined within the context of this approach. This approach offers the advantage of reducing the communication overhead between the processing elements, which directly impacts the system's performance and energy efficiency.

- A metaheuristic SA algorithm is introduced for solving the neuron grouping problem. This method, faster than the ILP-based approach, exploits multi-threading to explore multiple local minima in parallel, thereby finding optimal or near-optimal solutions.
- SA-based mapping algorithm is proposed to map the graph representation of grouped neurons onto the 2D mesh NoC architecture. The objective of this mapping is to minimize the total communication cost, which is directly proportional to the dynamic energy consumption and performance of the system.
- Extensive evaluations of the acceleration on several DNN applications, implemented on both fully connected and partially connected models, are performed. The results demonstrate that the SA-based neuron grouping and SA-based mapping solutions achieve the best performance in terms of inference accuracy, overall performance, and energy consumption for practical NoC-based DNN accelerator designs. Furthermore, the approach can be utilized as a general framework to design efficient DNN accelerators for various neural network architectures.
- Utilizing the existing Integer Linear Programming (ILP) method, the thesis applied and compared it to the proposed heuristics for solving the neuron grouping problem. While ILP, known for its extended computation times in scenarios involving DNNs with numerous neurons, was not developed as part of this thesis, its optimal outcomes serve as a benchmark to assess the effectiveness of the proposed heuristics. Specifically, ILP is employed to establish the optimal grouping of neurons for smaller DNN models, and the obtained results are systematically compared with those derived from the proposed heuristics

In summary, this work addresses the fundamental challenge of designing efficient and scalable DNN accelerators based on NoC architectures. A set of algorithms is provided that can be used to optimally or heuristically group neurons and map them onto the NoC architecture, with an emphasis on minimizing communication costs. The proposed approach is adaptable to different types of neural network architectures and has the potential to lead to more energy-efficient and high-performance DNN accelerators.

1.3. Organization

The organization of the thesis is as follows:

- Chapter 1 presents the motivation, contributions, and the scope of the thesis.
- Chapter 2 provides information about basic concepts essential for a comprehensive understanding of the work outlined.
- Chapter 3 discusses previous studies and approaches related to the subject.
- Chapter 4 clearly defines the problem and includes detailed explanations of the solution methods offered.
- Chapter 5 demonstrates experimental results obtained from the implemented methods.
- Chapter 6 states the summary of the thesis and possible future directions.

2. BACKGROUND OVERVIEW

To comprehend the work presented in this thesis, it is essential to be informed some fundamental concepts. These basic concepts will be discussed from an accelerator perspective in this section. Following an overview of hardware accelerators, the NoC will be defined, and more specific details about its types and hardware topology will be presented. This concludes the hardware aspect of the concepts that need familiarity. Subsequently, information about neural networks will be provided, focusing on DNNs, along with insights into the challenges encountered in accelerating them and the commonly employed acceleration methods. Based on these concepts and previous findings, the usage of NoC on DNN acceleration is justified.

2.1. Hardware Accelerators

In modern chip designs, the majority of performance gains are achieved through parallelism, which is made possible by specialization. In the past, when processor performance scaled according to Moore's Law, high overhead was acceptable as existing applications would automatically run faster with time. However, Moore's Law has largely ended in recent times, forcing the exploration of alternative architectures, such as domain-specific accelerators, to achieve performance scalability.

A domain-specific accelerator refers to specialized hardware designed for running specific applications. There are four main techniques employed to improve performance in these accelerators:

- Data specialization: Specialized logic can be implemented to enhance inner-loop function gains by considering the specialized data types used in the target applications.
- Parallelism: Parallelism can be exploited at multiple layers, where parallel units utilize local memory and minimize global references to achieve performance gains.

- Local and optimized memory: Key data structures can be stored in small local memories, such as caches, to achieve high bandwidth and reduce memory access latency.
- Reduced overhead: Hardware specialization minimizes the overhead associated with program interpretation, resulting in improved performance.

Domain-specific accelerators can be implemented using various technologies, such as ASICs, FPGAs, or GPUs, each offering different trade-offs regarding development cost, programmability, and efficiency. ASICs typically provide the highest efficiency but have high nonrecurring engineering (NRE) costs and limited programmability.

Designing domain-specific accelerators involves striking a balance between generality and efficiency. Building an architecture that is specialized for a single application can yield high efficiency, but may have limited usability. On the other hand, building a completely general-purpose computer may result in poor efficiency. Therefore, the key effort lies in crafting an architecture that combines high parallelism with a small local memory footprint and low global memory access requests.

In the future, domain-specific accelerators will likely be designed by writing parallel programs and specifying the mapping of these programs to hardware resources in terms of time and space. This approach holds promise for achieving optimized performance in domain-specific applications [18].

2.2. Network-on-Chip

Network-on-chip (NoC) has emerged as a promising paradigm for addressing the communication challenges in modern computing systems. It provides a scalable and efficient on-chip communication infrastructure for connecting various components, such as processors, memories, and accelerators, in a system-on-chip (SoC) design. NoC offers several advantages over traditional bus-based or point-to-point interconnect architectures, including improved scalability, higher bandwidth, lower power consumption, and better fault tolerance [19].

At its core, a NoC consists of a network of interconnected routers that facilitate the exchange of data between the components in the system. The routers are responsible for routing packets of information across the network using efficient algorithms, ensuring reliable and low-latency communication [20]. The routers in a NoC are commonly interconnected using either a mesh or a grid topology, ensuring that each router is connected to its adjacent routers in a structured manner. This regular arrangement facilitates the implementation of efficient routing algorithms and simplifies the overall design and layout of the Network-on-Chip.

Extensive research has been conducted on NoC architectures, leading to the development of numerous design considerations and optimization techniques. Among these considerations, the choice of routing algorithm implemented by the routers is crucial. Various routing algorithms, including XY-routing, source routing, and adaptive routing, have been proposed to ensure efficient navigation through the network and mitigate congestion-related issues. These algorithms aim to minimize communication latency and maximize the utilization of network resources.

Another essential factor to consider in NoC design is the selection of an appropriate topology. Researchers have investigated different topologies, including mesh, torus, and tree-based structures, to cater to the specific needs of diverse applications. Each topology offers distinct advantages and trade-offs in terms of scalability, fault tolerance, and performance. Evaluating these factors is crucial for determining the most suitable topology for a given application or system.

By embracing the concept of reuse, system designers can harness the potential of NoC architectures to address the communication challenges posed by contemporary applications effectively. This approach enables the construction of efficient and adaptable systems that can meet the increasing demands of diverse and evolving application domains.

The Network-on-Chip (NoC) architecture serves as the on-chip communication infrastructure, encompassing the physical layer, the data link layer, and the network layer of the OSI protocol stack. It provides a crucial framework for efficient and scalable communication among various components within a system.

To meet the demands of modern applications, system designs must leverage the reuse of components, architectures, applications, and implementations. This approach enables the development of cost-effective and flexible solutions that can adapt to diverse application requirements. By capitalizing on reusable elements, system designers can streamline the design process, optimize resource utilization, and expedite time-to-market for new applications. The primary goal of the Network-on-Chip (NoC) development environment is to facilitate the separation of various concerns and activities while providing a layer of abstraction that shields specific tools and design tasks from the intricacies of others.

By establishing a modular and well-defined development environment, system designers can focus on specific aspects of NoC design without being burdened by the complexities of unrelated tools or tasks. This separation of concerns allows for parallel development and specialization, enabling experts in different domains to collaborate effectively while maintaining a clear boundary between their respective areas of expertise.

Furthermore, the abstraction provided by the NoC development environment allows designers to work at higher levels of abstraction, promoting design reuse and modularity. This abstraction shields designers from the underlying implementation details, enabling them to focus on higher-level design considerations and optimization strategies.

The separation of concerns and the shielding of tools and design tasks within the NoC development environment contribute to improved productivity, enhanced collaboration, and increased design efficiency. By decoupling different aspects of NoC design, such as communication protocols, routing algorithms, and performance analysis, designers can leverage specialized tools and methodologies tailored to each specific task. This enables them to explore design alternatives, analyze system behavior, and optimize performance without being overwhelmed by the complexity of the entire NoC design process.

In summary, the NoC development environment aims to facilitate the division of concerns and shield specific tools and design tasks from unnecessary details. This approach promotes effective collaboration, modularity, and productivity, ultimately leading to the development of efficient and scalable NoC architectures. [21]

Research efforts have also focused on power-aware NoC design to address the increasing energy consumption in modern computing systems. Techniques such as power gating, dynamic voltage and frequency scaling (DVFS), and clock gating have been investigated to optimize power consumption in NoCs [22].

In recent times, there has been a significant uptake of NoC-based architectures across multiple domains, encompassing high-performance computing, embedded systems, and emerging technologies like the Internet of Things (IoT) and edge computing. These architectures provide an efficient communication infrastructure for facilitating parallel and distributed processing, thereby leading to enhanced performance and scalability in diverse applications. Furthermore, NoCs have been utilized in accelerating specific applications, such as DNNs, by leveraging their parallelism and communication capabilities [23].

2.2.1. NoC Topology Alternatives

Several design parameters are considered to evaluate the performance of NoC topology, including hop count, path diversity, router complexity (ports), and bisection bandwidth. The topology should be easy to lay out on-chip substrate and meet the wire budget of the NoC.

2D-Mesh Regular and equal-length linkages define the 2D-Mesh topology, which makes it simple to lay out on-chip. High path diversity is provided, allowing for a variety of shortest pathways to connect source and destination pairs. The 2D-Mesh features a simple layout on the substrate, is fault-tolerant, and is load-balanced. However, because different parts of the mesh may have varying levels of connection, the location of neurons can have an impact on how well the mesh NoC performs. To maximize performance, it is crucial to take the placement strategy into account.

Concentrated-Mesh A centralized mesh topology (CMesh) contains multiple cores per router, thus requiring fewer routers and interconnects compared to a 2D mesh with the same number of cores. This resource reduction enables cost and energy savings. However, CMesh requires additional ports on routers to connect to centralized local cores, which can introduce buffer queuing and alignment services. To maintain effective communication, this increased complexity must be carefully managed.

Crossbar Crossbar topologies are an effective communication pattern for neural networks because any neuron can reach any other neuron in a single hop. However, crossbars impose high hardware overhead due to the large number of crosspoints required. It does not scale to large neural networks due to the quadratic increase in intersections and consequent connection length. When considering his crossbar-based NoC, it is important to address the challenges of hardware complexity and scalability.

Ring Ring network topology requires only three ports per router (local port, left and right ports), thus reducing router complexity compared to 2D mesh NoC. However, depending on the number of cores or routers in your network, the number of hops and latency will increase significantly. Ring topologies lack path diversity, which can limit performance for certain communication patterns. The network size and communication requirements of the neural network must be carefully considered when using ring topologies.

Torus Torus NoC topology addresses the port asymmetry problem of 2D mesh NoC and provides higher path diversity due to router and port symmetry. A torus network effectively distributes the traffic balance across the NoC. However, scalability becomes an issue as the length of long links increases with the size of the NoC. Longer links also increase line latency, making it difficult to maintain performance in large-scale environments. Mitigating these scalability limitations requires careful design and optimization techniques.

Flattened Butterfly A flattened butterfly topology uses high cardinality routers to reduce the hop count in the network, resulting in shorter delays. It also offers high pass versatility. However, like traditional butterfly topologies, flattened butterfly topologies require long cables and can pose challenges in terms of physical implementation and signal integrity. Due to the complexity of the routers and the increased number of channels required, this topology may not be suitable for large neural networks.

Fat Tree A fat tree topology resembles a binary tree structure, with resources arranged in leaf nodes and intermediate nodes that act as routers. It offers recursive scalability and easy partitioning. However, the upper levels of the fat tree can become bottlenecks, limiting the scalability of large neural network systems. Path diversity can also be an issue and affect overall network performance.

Star In a star network setup, there is a big central point, and all the other points are connected to it. However, star topologies suffer from a single point of failure. If the central hub fails, the entire network will be affected. This vulnerability should be considered when deploying star topologies in NoC-based systems.

Express Channel Express Channel is an enabler for NoC topologies that reduce the number of multi-hops and reduce latency and power consumption. This is accomplished by inserting long cable links between remote routers, usually one or more hops away. However, using long wire connections increases wire delay, which can affect the practicality of using such connections in practice. When considering his NoC topology for Express Channel, the trade-off between a reduced number of multi-hops and increased line delay should be carefully analyzed.

3D-Mesh A 3D mesh topology stacks multiple dies in a NoC, resulting in fewer network hops, lower line latency, and lower power consumption compared to traditional 2D integration. This integrated technology provides thermal and performance benefits.

However, it also poses challenges such as thermal hotspots and manufacturing difficulties associated with the lamination process. Effective thermal management and managing manufacturing complexity are critical to the successful implementation of 3D mesh topologies.

2.3. Neural Networks

Neural Networks are widely used in solving many complex real-world problems such as computer vision, speech recognition, natural language processing, finance, and weather prediction due to their remarkable success in classification and prediction problems. NNs consist of a large number of simple processing units called neurons, which are connected to form a network. An NN includes input, output, and hidden neuron layers. The confluence of substantial datasets and powerful computing resources has sparked a renaissance in machine learning, with a specific emphasis on NNs.
NNs aim to accomplish a brain-like functionality and are based on the analysis of neurons as a nonlinear function of the weighted sum of the inputs. These representative neurons are organized into layers, where the outputs of one layer serve as inputs to the next. The term ”deep” is attributed to neural networks with multiple layers. [24]

Three kinds of NNs are popular today:

1. Multi-layer perceptrons (MLP)
   Each new layer is a set of nonlinear functions of the weighted sum of all outputs from the prior one.

2. Convolutional Neural Networks (CNN)
   In this case, each layer is a set of nonlinear functions of weighted sums at different coordinates of the spatially nearby subset of outputs from the previous layer. This method allows weights to be reused.

3. Recurrent Neural Networks (RNN)
   Each layer is a collection of nonlinear functions of weighted sums of outputs from the

previous state. LSTM (Long Short-Term Memory) is a popular RNN. The main idea of LSTM is deciding what to forget and what to pass on as the state of the next layer.

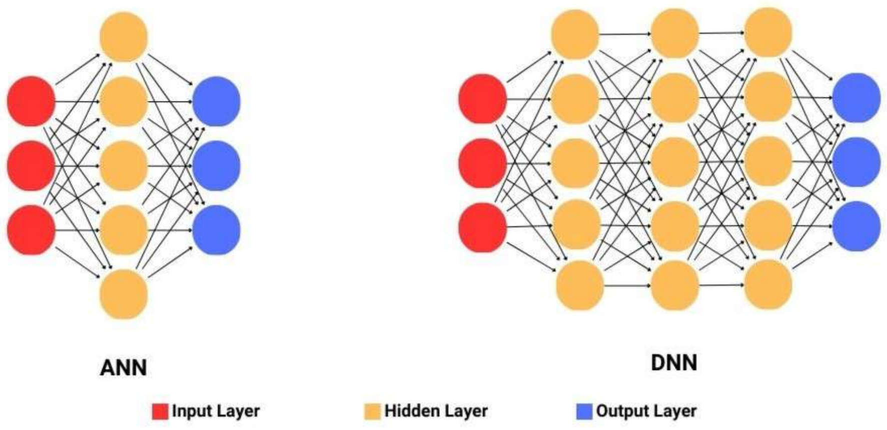

Figure 2.1 Example ANN and DNN structures.

Figure 2.1 provides examples of ANN and DNN structures. DNN is highly successful in solving complex problems because it can process large, high-dimensional data sets, model complex non-linear relationships, and discover hidden structures in unlabeled and unstructured data. For example, a DNN can take thousands of images and cluster them based on similarities, such as human, cat, and dog images. However, NNs require a high level of computational power due to a large number of layers and neurons. One way to meet this high computational power requirement is to parallelize operations, that is, distribute the DNN neurons to different processing nodes (cores) in a multi-core architecture. However, since computations are linked by communication between neurons, and the interconnection between neurons in a DNN is high, an efficient communication architecture is required to achieve the necessary parallelism.

2.4. Challenges in Accelerating DNNs

Accelerating deep neural networks (DNNs) poses several challenges that need to be addressed to achieve efficient and high-performance solutions. This subsection discusses some of the key challenges in accelerating DNNs:

1. Computational Complexity:
   DNNs often consist of multiple layers with a large number of neurons, resulting in a high computational workload. The complex computations involved in forward and backward propagation, weight updates, and activation functions demand significant computational resources [25, 26].

2. Communication Overhead:
   DNNs demand intensive neuronal communication, which includes synchronization between parallel processing units and data transit between layers. Large communication overhead and latency might result from the number of data transmissions. [27]

3. Energy Efficiency:
   DNNs consume a lot of power due to massive parallelism and the intensive computation that comes with it. The energy efficiency of DNN accelerators is a key concern in achieving sustainable and energy-efficient solutions. [15]

4. Memory Bandwidth:
   DNN models often exhibit high memory access requirements due to a large number of parameters and intermediate feature maps. Efficient utilization of memory bandwidth is essential to avoid memory bottlenecks and maximize performance [28]

5. Scalability:
   Scalability becomes a major challenge as DNN models grow in size and complexity. Accelerator designs must scale efficiently to accommodate increasing model sizes and computational demands. [29] Different applications with varying DNN layer

shapes and sizes brought difficulties since they directly affect data communication and processing. [30]

6. Flexibility and Programmability:
   DNN accelerators should support a wide range of network architectures and provide flexibility and programmability to optimize performance for various applications. The ability to adapt to evolving DNN models and algorithms is critical.

Addressing these challenges requires innovative hardware architectures, efficient algorithms, and system-level optimizations. Researchers have proposed various techniques such as specialized hardware accelerators, NoC designs, algorithmic optimizations, and quantization methods to overcome these challenges and improve the performance and efficiency of DNN accelerators.

2.5. Previous Approaches to DNN Acceleration

Accelerating DNNs has been an active area of research, with several traditional approaches being explored. These approaches aim to improve the computational efficiency and speed of DNNs using various techniques. In this subsection, some of the commonly employed traditional approaches and their characteristics are discussed.

1. Parallel Processing:
   Parallel processing techniques involve distributing the computation of DNNs across multiple processing units, such as CPUs or GPUs. This approach harnesses the power of parallelism, allowing operations to be performed simultaneously on different data segments. By leveraging parallel processing, the overall training or inference time of DNNs can be significantly reduced. Numerous studies in the field of deep learning have highlighted the effectiveness of parallel processing techniques in accelerating the training and inference processes of DNN models.

2. Model Compression:
   Model compression techniques focus on reducing the computational and memory

requirements of DNNs. These techniques aim to compress the size of the model while preserving its performance. Methods such as pruning, quantization, and knowledge distillation have been widely studied and applied to compress DNNs [31, 32].

Quantization is a powerful technique in deep neural network (DNN) acceleration that aims to reduce the computational complexity and memory footprint of network models. By reducing the precision of weights, biases, and activations, typically from floating-point to lower-bit fixed-point representations, quantization effectively reduces the number of computations required during inference. This not only leads to significant savings in memory and storage requirements but also enables more efficient utilization of hardware resources, such as CPUs or GPUs. Various quantization methods have been proposed, with 8-bit integer quantization being the most common approach. However, recent advancements have explored even lower bit-width implementations, including binary neural networks, which achieve further compression while maintaining acceptable accuracy levels. The combination of quantization with other optimization techniques, such as pruning, offers the potential for even greater acceleration and resource efficiency.

Pruning is another effective technique in DNN acceleration that focuses on reducing the network's complexity by removing redundant computations and parameters. It can be performed either statically, offline during the training phase, or dynamically, at runtime. Statically pruning techniques identify and eliminate redundant connections, filters, or entire layers based on predefined criteria, such as weight magnitude or importance. On the other hand, dynamic pruning adapts the network during runtime by dynamically activating or deactivating specific neurons or connections based on the input data. Pruning operates on different granularities, ranging from element-wise pruning to channel-wise, shape-wise, filter-wise, layer-wise, and even network-wise pruning. By reducing the network's size and computational requirements, pruning enables faster inference and reduces the energy costs associated with DNN deployment. By combining pruning with quantization techniques, further improvements in speed and efficiency can be achieved, making it a promising approach for DNN acceleration. [33]

As exemplified in a study [28], a notable approach to model compression involves pruning weights and neurons in a DNN based on a minimal distance error (MDE) criterion within a range of safety margin error (SME). The pruning process ensues when the condition MDE $<$ SME (where SME is set at 10^{-6}) is satisfied. This binding criterion helps prevent overfitting. To evaluate the efficacy of weight and neuron pruning, the authors evaluate the Error Before (EB) and Error After (EA) of the pruning process and subsequently derive the MDE through the equation MDE = EB - EA. This pre-strategy ensures a careful reduction in model complexity without compromising performance.

The term ”one-shot learning” signifies the classification of tasks through the utilization of a limited number of examples. The capability of a memory-augmented neural network to swiftly incorporate new data is underscored in the literature [34]. The approach outlined in the literature encompasses two key facets: the gradual acquisition of an abstract technique for deriving valuable representations of raw data, achieved through gradient descent, and the rapid integration of previously unencountered information after a singular exposure, facilitated by an external memory module. This synthesis of strategies empowers the neural network to efficiently process novel information while building upon its existing knowledge base.

3. Hardware Accelerators:

   Hardware accelerators are specialized hardware components designed to accelerate the computation of DNNs. These accelerators can be implemented using application-specific integrated circuits (ASICs), field-programmable gate arrays (FPGAs), or dedicated PEs.

4. Circuit Optimization:

   Circuit optimization techniques focus on improving the efficiency of hardware implementations by reducing power consumption and increasing computational throughput. These techniques involve architectural optimizations, such as low-power design, pipeline parallelism, and memory hierarchy optimization. [15]

Some DNN accelerators leverage on-chip memory resources [28], such as embedded dynamic RAM (eDRAM), static RAM (SRAM), and global buffers, to facilitate local data storage of trained models. This strategic use of on-chip memory aims to enhance memory access efficiency and optimize bandwidth requirements during DNN inference and training processes.

By employing on-chip memory for local data storage, DNN accelerators can minimize the need for frequent off-chip memory accesses, which tend to introduce latency and incur higher energy costs. Instead, critical data can be readily accessed from the on-chip memory, thereby significantly reducing data transfer time and improving overall performance.

It is important to note that the choice of on-chip memory technology, such as eDRAM, SRAM, or global buffers, can impact the trade-offs between memory access speed, memory capacity, and energy consumption. Each memory type offers distinct characteristics that must be considered based on the specific requirements of the DNN accelerator design.

Eyersis v2 [30] presents a novel DNN accelerator architectural design. Specifically tailored for compact and sparse DNN models. A key feature of this design is the utilization of a highly flexible on-chip network called the hierarchical mesh (HM-NoC).

Eyeriss v2 is constructed around an array of processing elements (PEs), each equipped with multiply-and-accumulate logic and local scratchpad memory to exploit data reuse. Additionally, global buffers (GLBs) are integrated into the design, serving as an intermediary level of memory hierarchy situated between the PEs and the off-chip DRAM. For enhanced flexibility, the PEs and GLBs are organized into clusters, enabling the implementation of a cost-effective on-chip network (NoC) that interconnects the GLBs to the PEs.

One noteworthy aspect of Eyeriss v2 is the deployment of separate NoCs for transferring three distinct data types: input activations, weights, and partial sums between the GLBs and PEs. The authors employ the Row-Stationary dataflow in this context. The memory hierarchy in Eyeriss v2 operates as follows:

- Input activations are read from the off-chip source into the GLB cluster. Depending on the configuration, these input activations can either be stored in the GLB memory or directly passed to the router cluster.
- Partial sums, once generated by the PE cluster, are consistently stored in the GLB memory.
- The final output activations bypass the GLB cluster and are directly transferred off-chip.
- Weights, on the other hand, are not stored in the GLB but are transmitted to the router clusters and directly stored in the local scratch pads within each PE.

Eyeriss v2 introduces the hierarchical mesh network (HM-NoC), which offers different configurable modes, thereby accommodating a wide range of bandwidth and data reuse requirements.

5. Software Optimization:

   Software optimization techniques aim to enhance the performance of DNNs by optimizing the software implementation. This includes algorithmic optimizations, memory management, and compiler optimizations.

   In the realm of DNN accelerators, data reuse is a crucial aspect of enhancing efficiency. However, the extent of data reuse for each data type (input activations, weights, partial sums) in a DNN layer depends on its shape and size. Diminished data dimensions pose challenges in exploiting data reuse effectively, leading to two key issues:

   - Array Utilization: When data dimensions are initially selected with high parallelism in mind, an array of processing elements can be utilized optimally.

- Processing Element Utilization: As data reuse decreases, higher data bandwidth is required to keep the processing elements efficiently utilized.

Moreover, as a software optimization technique, skipping cycles of processing results with zero weights or input activations is desirable to further enhance efficiency.

In the previous study made on Spiking Neural Network (SNN) implementation on [35], the processing elements, correspond to spiking neurons within the Network-on-Chip (NoC). In this context, NoC channels can be likened to synaptic connections. The paper under consideration introduces an advanced hierarchical NoC architecture tailored for SNN implementations. This architectural design capitalizes on a fusion of star and mesh topologies to optimize the one-to-many multicast communication paradigm that characterizes inter-neuronal interactions in the networks. The fundamental unit of this proposed architecture is the cluster facility, where an assembly of neurons is interconnected through a hierarchical framework. This framework employs an array of both low and high-level NoC routers to facilitate both local (within-cluster) and global (between-cluster) connections among neurons. This configuration enables efficient information exchange among neurons within the network, effectively enhancing the communication efficiency of the SNN architecture.

In the context of topology choices for neural network accelerators, mesh topology is an outstanding consideration. While shared bus topology offers simplicity and cost-effectiveness, its scalability in terms of performance is limited. In contrast, mesh topology, due to its inherent parallelism and communication pathways, has been favored as a solution to address performance demands. However, researchers caution against its unrestricted use, as larger mesh configurations can lead to heightened average latency resulting from the absence of direct connections between distant nodes. To navigate these challenges, innovative approaches like CuPAN [36] (Custom Parallel Architecture for Neural Networks) have been introduced. CuPAN distinguishes itself by adopting a multi-stage clos interconnection topology, a strategy designed to optimize inter-neuron connections while encouraging performance constraints associated with larger mesh topologies. This nuanced exploration underscores the iterative nature of architectural design in pursuing optimal neural network accelerator performance.

Some previous studies [37] delve into optimizing bit lengths in DNN accelerator operations. ”Bit Fusion” blocks are introduced for which dynamically adjusting bit lengths for efficient layer-specific operations, thereby enhancing computational efficiency and performance.

It is important to note that these traditional approaches have made significant contributions to DNN acceleration. However, they often face limitations in terms of scalability, energy efficiency, or hardware/software flexibility. Therefore, exploring novel approaches, such as Network-on-Chip (NoC)-based accelerators, is essential to overcome these challenges.

2.6. Justification for Using NoC to Accelerate DNNs

DNNs require efficient communication between the layers to reduce the bottleneck effect. As DNNs contain multiple hidden layers in addition to input and output layers, there is often one-to-many and many-to-one communication between layers and no communication within a single layer.

DNNs consist of convolution layers and fully connected layers. Convolution layers connect each hidden/output neuron to a small region of the input neurons, while fully connected layers use all input neurons for computation, leading to high and diverse communication between layers. Communication can consume a significant portion of overall energy in a deep learning accelerator.

In the context of accelerating DNNs, a comparison between CPU, GPU, ASIC, and FPGA implementations reveals various trade-offs. CPUs and GPUs, while offering general-purpose computing capabilities, often fall short in terms of performance when it comes to DNN computations. Additionally, all four options exhibit high power consumption, limiting their efficiency. Furthermore, their computational flexibility at runtime is limited, posing challenges for dynamic adaptation to different DNN applications.

Comparatively, FPGA-based DNN accelerators provide a flexible design space for a wide range of accelerator configurations. However, they still face constraints in terms of computational flexibility, hindering their ability to support runtime reconfigurability. To address these limitations and enhance design flexibility while reducing interconnection complexity, a modular structure inspired by NoC can be leveraged.

While CPUs and GPUs offer high reconfigurability at runtime, enabling support for diverse DNN applications, they suffer from issues such as high power consumption and data transfer latency. On the other hand, ASIC-based DNN accelerators are specifically tailored for optimal performance and efficiency in a particular DNN model but lack reconfigurability at runtime. Although FPGA-based solutions improve design flexibility, their computational flexibility is still insufficient to support runtime reconfiguration. It is important to note that in FPGA solutions, the data path is fixed for a specific RNN or DNN model at design time, limiting further reconfiguration.

Among the various options for DNN accelerator designs, the adoption of NoC-based architectures proves advantageous due to several factors, including power efficiency, computational flexibility, and reconfigurability. By leveraging flexible communication mechanisms, NoC-based designs reduce the need for frequent off-chip memory access by facilitating efficient on-chip data transfer between cores. This reduction in memory access translates into significant power savings. While GPUs exhibit efficient parallel execution of essential DNN operations, such as matrix multiplication and convolution, concerns arise regarding their increasing power consumption.

In summary, considering the trade-offs in terms of performance, power consumption, computational flexibility, and reconfigurability, the adoption of NoC-based DNN accelerators emerges as a compelling and promising choice for accelerating deep neural network workloads. The integration of NoC architectures addresses the limitations observed in other conventional options such as CPUs, GPUs, ASICs, and FPGAs.

One of the significant advantages of NoC-based DNN accelerators lies in their ability to efficiently manage communication between processing elements, reducing data transfer latencies and alleviating the bottleneck effect. This efficient communication is crucial in DNNs, which often consist of multiple hidden layers with one-to-many and many-to-one communication patterns between layers. Using NoC as the on-chip communication infrastructure, DNN accelerators can effectively handle these communication requirements and achieve better overall performance.

Furthermore, the modular design and power-gating capability of NoC architectures allow specific regions or components of the NoC to be deactivated when not in use. This power-saving feature contributes to reduced energy consumption and aligns with the growing need for energy-efficient computing solutions.

NoC allows multiple cores to communicate with each other without interference, and a pair of nodes have multiple paths to reach each other, ensuring high path diversity. The modular design of NoC enables regions or components of the NoC that are not performing any computation to be switched off through power-gating, without affecting other components, which reduces energy consumption. NoC is a promising solution for the interconnection between cores in DNNs due to its support for parallelism [30] and fault tolerance.

Moreover, NoC-based DNN accelerators offer enhanced computational flexibility, enabling them to adapt to different DNN models with varying complexities and sizes. Through proper mapping and routing algorithms, the computational power and performance can be dynamically adjusted based on the specific requirements of each DNN workload. This adaptability ensures that NoC-based accelerators can efficiently execute diverse DNN applications, making them versatile solutions for accelerating a wide range of tasks.

Given the complex communication patterns in DNNs, a NoC with less complexity and cost is necessary to accelerate DNNs. A solution addressing issues related to communication patterns and achieving high performance and energy efficiency in DNNs was proposed in a previous study [23]. A mapping algorithm was utilized by the authors to minimize communication overhead, enabling the efficient mapping of the DNN onto the NoC.

Compared to CPUs and GPUs, which may suffer from low performance or high power consumption, and ASIC-based accelerators, which lack reconfigurability, NoC-based designs provide a more balanced and favorable trade-off between these critical aspects. The incorporation of NoC architectures into DNN accelerators allows for a more efficient allocation of resources, supporting optimal communication and computation, and resulting in improved overall efficiency [28].

In conclusion, the adoption of NoC-based DNN accelerators offers an effective and efficient solution for addressing the challenges associated with accelerating DNN workloads. By leveraging the benefits of efficient communication, reduced power consumption, and enhanced computational flexibility, NoC-based designs provide a promising path towards achieving high-performance and energy-efficient deep neural network acceleration [38].

2.7. Optimization Techniques for NoC-based DNN Accelerators

In this section, a range of strategies aimed at enhancing the efficiency and performance of DNN accelerators built on NoC architectures are explored. These optimization techniques are crucial for effectively addressing the challenges posed by complex network architectures, computation-intensive tasks, and concerns related to energy consumption.

Different mapping algorithms for DNNs in NoC-based accelerators are partially explored in a previous study [23]. Two specific mapping algorithms are highlighted, each offering distinct advantages and trade-offs:

1. Shortest Path Algorithm:
   The shortest path algorithm is designed to minimize the communication distance between neurons within the neural networks on the NoC. By placing communicating neurons in close proximity, this algorithm significantly improves communication performance by reducing data transfer delays. However, it's important to note that a mapping solution solely based on communication distance can potentially lead to unbalanced load distribution across the NoC. Some regions of the NoC may experience

high utilization, while other parts remain under-utilized. This unbalanced load distribution can create hotspots, increasing contention among packets and negatively impacting performance.

2. Load-Balanced Algorithm:
   The load-balanced mapping solution, on the other hand, addresses the issue of hotspots caused by unbalanced load distribution. By employing load-balancing techniques, this algorithm ensures a more even distribution of computational workload and communication traffic throughout the NoC. As a result, it improves parallelism within the system and reduces the occurrence of hotspots. Lower hotspots lead to reduced congestion and queuing delay within the NoC, ultimately resulting in improved overall performance.

This study addressed the grouping and mapping problems using two optimization techniques, Simulated Annealing (SA) and Integer Linear Programming (ILP). A comparative evaluation was conducted to determine their effectiveness in optimizing the system's performance.

2.7.1. Simulated Annealing (SA)

Simulated Annealing, a widely adopted metaheuristic optimization technique, has found application in addressing intricate optimization problems across various domains [39]. Drawing inspiration from the annealing process employed in metallurgy to refine the structural integrity of metals, SA offers an algorithmic approach to solving complex optimization challenges.

In its computational incarnation, SA commences with an initial solution to a problem and embarks on an iterative journey through the solution space. During each iteration, the algorithm introduces random, incremental alterations to the current solution. These modifications may be accepted or rejected based on a probabilistic criterion, intricately intertwined with both the quality of the current solution and user-defined algorithmic parameters.

As the algorithm progresses, it gradually diminishes the likelihood of accepting deteriorating solutions. This adaptive approach serves as a mechanism for breaking free from local optima, enabling continued exploration of the expansive search space. Consequently, SA endeavors to discern the optimal or near-optimal solution within the vast expanse of a global search space [40]. This flexibility positions Simulated Annealing as a versatile tool applicable to a diverse array of optimization conundrums, encompassing combinatorial optimization, scheduling quandaries, network design conundrums, and parameter estimation dilemmas.

2.7.2. Integer Linear Programming (ILP)

ILP is a mathematical optimization technique that combines the concepts of linear programming and integer variables. It is used to solve problems where decision variables must take on integer values, rather than continuous values. This is particularly useful in cases where the decision variables represent quantities that cannot be divided or allocated in fractions. For example, ILP can be used to solve optimization problems involving resource allocation, scheduling, network flows, and production planning. Additionally, it finds applications in areas such as topology optimization [41], logistics, telecommunications, and finance.

By formulating the optimization problem with integer variables and linear constraints, ILP guarantees convergence to the globally optimal solution [42]. This approach has been successfully applied across various fields, such as production planning and vehicle scheduling, in addition to computational and systems biology. One of the advantages of ILP is that computationally it ensures convergence to the globally optimal solution. Due to its linear formulation of the problem, it can be efficient in practice, enabling application to large networks. Additionally, optimal solutions can be found using standard algorithms developed specifically for 0-1 integer programs, and there are many readily available automated packages to solve integer programs.

However, in some cases, the required computational effort to find optimal solutions for some problems can be prohibitive, making it necessary to explore alternative approaches, such as heuristic methods.

Overall, ILP is a powerful optimization technique that can be used in a variety of applications, including DNN acceleration. By formulating the problem as an ILP, it is possible to find an optimal solution that minimizes communication costs and improves processing speed.

3. RELATED WORK

In this section, an overview is provided of the related studies in the literature that focus on deep neural network (DNN) acceleration techniques and network-on-chip (NoC) architectures for DNN accelerators. Specific emphasis is placed on neuron grouping and mapping methods for NoC-based DNN accelerator designs.

Previous research on NoC-based DNN accelerators has primarily concentrated on designing the accelerators themselves [30, 35, 38, 43] as well as developing simulators tailored for these accelerators [44, 45].

The problem of mapping neurons on processing elements (PEs) of NoC architectures with 2-D topologies has been emphasized in several studies. Most of these studies have employed methods that cluster neurons into groups and then map the resulting connected groups to improve performance and efficiency. However, these approaches often rely on simplistic assumptions, considering only either the communication weights among neurons or the computation cost of each PE. Furthermore, these methods are typically limited to fully connected PEs, while today's DNN models often involve pruned (partially connected) neural networks.

Given the increasing popularity of pruned neural networks and the need for efficient acceleration of DNN models, there is a demand for more sophisticated methods that can optimally group neurons in a DNN model, taking into account both the communication and computation costs.

The grouping of neurons in a DNN model using a novel method that takes into account both communication and computation for optimal grouping is a crucial step. This process aims to create groups of neurons that are well-suited for subsequent mapping onto a NoC architecture. The grouped DNN neurons are then represented in an intermediary graph, which serves as a foundation for the subsequent mapping algorithm. The choice of the mapping algorithm is of utmost importance, as it should consider both communication and computation factors to ensure efficient and effective utilization of the NoC resources.

3.1. Neuron Grouping (Clustering)

The strategic grouping of neurons in DNNs using NoC architectures represents a critical juncture in enhancing computational efficiency and managing data flow within advanced neural network structures. This section elaborates on the frequent neuron grouping methodologies, emphasizing their significance in optimizing DNN accelerators.

1. Fixed-Number Inter-Layer Grouping [44]: This strategy involves grouping a predefined number of neurons across adjacent layers, which can facilitate parallel processing and reduce computational bottlenecks. While promising, its effectiveness is potentially limited in complex DNN architectures where layer connectivity and neuron density vary significantly. Understanding these dynamics is crucial for optimizing data flow and minimizing latency, especially in large-scale networks where computational demands are substantial. This approach is explained visually in the Figure 3.1.

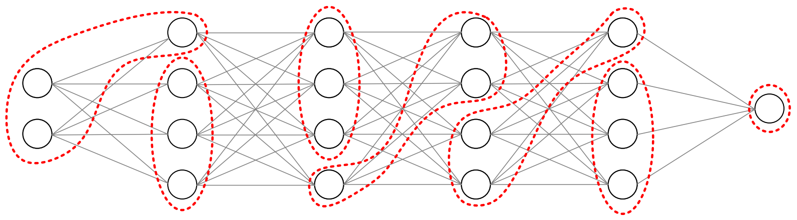

Figure 3.1 Fixed-Number Inter-Layer Grouping Demonstration.

2. Fixed-Number Intra-Layer Grouping [38] This method restricts neuron groups to a constant size within individual layers. This strategy is particularly beneficial in scenarios where maintaining a streamlined and efficient data flow within layers is crucial. As seen in Figure 3.2 grouping is made inside layers. Its application is most evident in smaller networks where NoC latency overshadows processing delays, underscoring the need for tailored approaches depending on network size and complexity.

3. Dynamic Intra-Layer Grouping with Optimization [46]: This comparatively advanced approach involves a dynamic adjustment in the number of neurons per group based

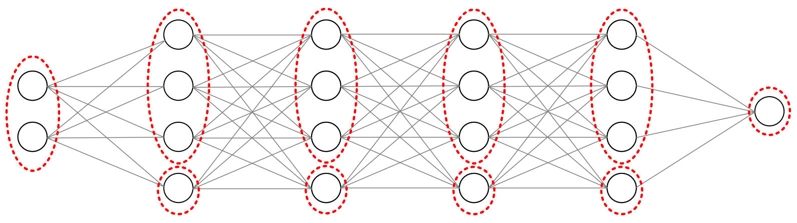

Figure 3.2 Fixed-Number Intra-Layer Grouping Demonstration.

on computational load requirements. This method optimizes processing power and resource allocation, ensuring that computational loads are evenly distributed and network efficiency is maximized. As represented in Figure 3.3 neurons are grouped evenly inside the layer due to consideration of processing power. The adaptability of this strategy makes it particularly attractive for diverse and evolving DNN configurations.

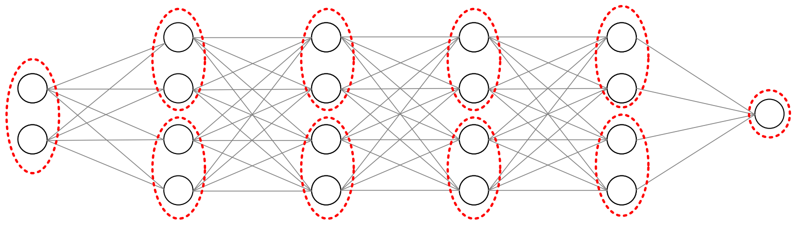

Figure 3.3 Dynamic Intra-Layer Grouping with Optimization Demonstration.

Recent years have witnessed the emergence of innovative methods for groping neurons within DNNs. Sequential layer mapping is advanced as a method [47] in NoC architectures, though it may lead to suboptimal utilization of processing elements in certain scenarios. This highlights the ongoing challenge of balancing computational efficiency with hardware resource allocation. Clustering methods that popular in other fields like k-means and network slicing have been explored [48, 49], respectively, each offering unique advantages tailored to specific network needs. These varied approaches underscore the importance of customizing neuron grouping strategies to the specific requirements of each DNN.

In the realm of CNNs, pioneering steps have been taken [50] by grouping neurons based on connection weights within channels or filters, a method particularly relevant for managing complex CNN architectures. This approach addresses both computational complexity and memory efficiency, key factors in the design of advanced DNN systems.

The Neu-NoC architecture [43], condensed a breakthrough in neuron grouping optimization for NoC-based DNN accelerators. By integrating a hybrid ring-mesh NoC structure, Neu-NoC effectively marries the benefits of local and global data transfer, significantly reducing redundant communications and enhancing bandwidth utilization. This architecture not only exemplifies innovation in NoC design but also serves as a benchmark for future developments in the field.

Despite the rich diversity of strategies in neuron grouping, the field has not yet converged on a universally optimal approach, particularly when considering the vast array of DNN architectures and NoC configurations. This thesis endeavors to fill this gap by presenting a detailed comparative analysis of these neuron grouping methodologies. By thoroughly evaluating each strategy against a spectrum of performance metrics and computational demands, this research aims to illuminate the path forward in NoC-based DNN accelerator design, contributing to both theoretical understanding and practical applications in this rapidly evolving field. In Table3.1 a summary of neuron grouping methods in the literature is presented by indicating the advantages and disadvantages of each of them. The heuristic method presented in this study is also added at the end.

Table 3.1 Summary of neuron grouping methods in the literature.

Study	**Grouping Approach**	**Advantages**	**Disadvantages**
[44]	Fixed-Number Inter-Layer	Reduced traffic load	Potential latency issues in complex architectures
[38]	Fixed-Number Intra-Layer	Efficient data routing within layers	Limited inter-layer data processing
[46]	Dynamic Intra-Layer with Optimization	Adaptive to computational loads	Requires additional computation for optimization
[47]	Sequential Layer Mapping	Structured approach to neuron grouping	Possible underutilization of processing elements
[48]	k-Means Clustering	Customizable to network needs	May not optimally reflect neuron connectivity
[49]	Network Slicing	Evenly distributed inter-neuron data traffic	Increased communication cost
[50]	Connection Weights Based Grouping	Prevents unnecessary memory overhead	Assumptions about load may not apply to all ANNs
This study	Novel Method	Optimal grouping based on communication load	Time and memory problems in large architectures

3.2. Neuron Mapping

The quest for efficient DNN accelerators in modern computational architectures necessitates innovative neuron mapping strategies, especially for 2D-mesh Network-on-Chip (NoC) systems. This subsection synthesizes a range of mapping methodologies, highlighting their evolution, applications, and implications in optimizing DNN accelerator performance.

Direct and layer-based mapping strategies [44] explored previously, notably the `Dir_X / Dir_Y` and `Lyr_X / Lyr_Y` approaches. These methods emphasize the balance between ease of implementation and maintaining the logical integrity of DNN layers.

Building upon this foundational work, sequential mapping methods [47] streamline the process but could potentially lead to underutilization of resources. Complementing these techniques, snake pattern mapping [50] is introduced to address memory footprint concerns, showcasing the diversity of approaches in the field.

Recent progress in the mapping of neurons has involved a transition toward more sophisticated strategies. The integration of clustering algorithms is proposed [28] with mapping solutions to simplify the complex interconnections in DNNs.

An NN-aware mapping algorithm is introduced [43], tailoring the mapping process to the specific communication flow and NoC topology. Similarly, linear programming is utilized in the mapping approach [23], aiming to reduce communication distances while balancing multiple performance factors.

The role of mesh topology in DNN traffic management has been increasingly recognized. Its scalability and efficacy in handling memory traffic within DNN accelerators are highlighted [51]. Further, in-depth studies are conducted on mesh networks [52, 53], emphasizing load-balanced mapping and efficient traffic distribution. These studies collectively underscore the importance of mesh topology in the context of NoC frameworks.

Comprehensive analysis of mapping within NoC is offered in the literature [54], considering crucial factors like Manhattan Distance and traffic volume. Their research underlines the significance of these elements in formulating effective mapping strategies. Such analytical perspectives are instrumental in devising mapping frameworks that are compatible with the architectural nuances of NoC topologies.

In Table 3.2 summary of neuron mapping approaches in the literature is presented with their advantages and disadvantages. The novel methods that are examined in this study are also added.

Table 3.2 Summary of neuron mapping approaches in the literature.

Study	**Mapping Approach**	**Advantages**	**Disadvantages**
[44]	Dir_X / Dir_Y	Direct, continuous mapping	May not suit complex network structures
[38]	Lyr_X / Lyr_Y	Layer-wise mapping enhances structure	Potential underutilization of PEs
[50]	Snake Pattern Mapping	Efficient memory utilization	Complexity in mapping process
[28]	Clustering-Integrated Mapping	Simplifies interconnections	Requires complex pre-mapping clustering
[43]	NN-Aware Mapping	Tailored to communication flow	Computationally intensive
[23]	Linear Programming-Based Mapping	Minimizes communication distances	May overlook some network nuances
[51]	Mesh Network Traffic Distribution	Scalable and efficient	Requires careful traffic management
[52]	Shortest Path Algorithm	Minimizes energy consumption	Risk of imbalanced load distribution
[53]	Load-Balanced Mapping	Enhances parallelism and TDP	Requires detailed network analysis
[54]	Manhattan Distance-Based Mapping	Optimizes data movement	Complex calculation of optimal paths
This study	SA Mapping	Minimizes communication costs	Computationally demanding in large networks

In this research, these diverse methodologies are built upon by employing ILP and SA for neuron mapping, aiming to minimize communication costs, a critical aspect of energy consumption and performance in NoC-based DNN accelerators. This unique approach leverages established theories while introducing novel insights to optimize router efficiency and enhance data exchange efficiency. The integration of these methodologies with the research goals aims to contribute a distinctive perspective to the field, paving the way for more efficient and effective DNN accelerator architectures.

4. PROPOSED METHOD

In this section, the problem is first defined precisely, building upon the foundational challenges and traditional approaches discussed in the '2..Background Overview' section. The background section provided detailed insights into the complexity of hardware accelerators, Network-on-Chip (NoC) technologies, and the specific complications faced in the field of Deep Neural Network (DNN) acceleration. The goal is to specify the problem to be addressed, particularly concentrating on optimizing DNN acceleration within NoC architectures.

In addition to defining the problem, this section is dedicated to presenting the methods that have been developed as solutions. These methodologies are designed to specifically target and address the critical aspects identified in the problem definition. The discussion will delve into how these solutions are tailored to enhance the performance and energy efficiency of 2D-mesh NoC architectures, a dominant topology in NoC designs known for their adaptability in various DNN applications.

The exploration includes an in-depth examination of computation and communication dynamics within these architectures. Optimal task distribution across PEs is crucial, and the approach aims to balance computational loads effectively while minimizing energy consumption, as supported by existing literature [55]. Moreover, the application of dynamic voltage scaling (DVS) is considered a viable strategy to adjust PE operations in response to real-time computational demands [56].

The pivotal role of data movement in determining the performance of DNNs cannot be overstated. Recognizing this, the primary objective in enhancing DNN accelerators revolves around minimizing the volume of data transit within the NoC network. This focus is rooted in the understanding that optimizing communication costs is directly proportional to improving both system performance and energy efficiency.

In pursuit of this goal, the work is dedicated to refining both the communication and computation aspects of DNN accelerator systems. A methodical approach is proposed to map neurons onto processing elements (PEs) in a manner that ensures an approximately equal distribution of computational loads. By doing so, the aim is not only to balance the workload across the PEs but also to significantly reduce the communication overhead within the network. This dual focus on computation and communication is intrinsic to the methodology, addressing the two-fold challenge of enhancing processing efficiency while concurrently reducing energy consumption.

This strategy recognizes that efficiency in DNN accelerators is not solely dependent on the computational capability of individual processing elements (PEs). Instead, it hinges on the holistic optimization of the entire system, where data movement and processing efficiency are inextricably linked. By adopting this comprehensive approach, the aim is to establish a new benchmark in the design and operation of NoC-based DNN accelerators, one that aligns computational power with communication efficacy to achieve optimal system performance.

In addressing the neuron mapping challenge within NoC architectures, a complex many-to-many relationship that is inherently NP-hard (no known algorithm that can find the optimal solution in polynomial time) is confronted. This complexity is particularly pronounced when considering a direct mapping of neurons to PEs. Conceptualizing this, let p represent the number of PEs and n the total number of neurons. To map these neurons equally across r neurons per PE, a computational complexity of $p!\, C(n, r)$ is faced. Given the scale of modern DNNs, which often comprise thousands of neurons, finding optimal mapping solutions within a reasonable timeframe becomes a formidable challenge.

To mitigate this complexity, the approach involves bifurcating the neuron-PE mapping problem into two distinct phases. The initial phase focuses on grouping the neurons, where neurons are aggregated into clusters, with each cluster containing several neurons less than or equal to the number of available PEs. This strategic grouping paves the way for a more manageable one-to-one mapping between neuron clusters and PEs. The outcome of this neuron grouping phase is represented as a connected graph.

Subsequently, in the second phase, the mapping of this connected graph onto the 2D-mesh NoC architecture is addressed. The guiding principle is to minimize communication costs, a factor intrinsically linked to the dynamic energy consumption of the system. By structuring the problem-solving approach into these two distinct yet interconnected phases, the complexity is effectively reduced to $C(n, r) + p!$. This division not only simplifies the computational process but also aligns closely with the goal of optimizing both performance and energy efficiency within NoC-based DNN accelerators.

As the intricate details of the proposed methods for neuron grouping and mapping in 2D-mesh NoC architectures are delved into, it becomes imperative to establish a clear understanding of the specific notations used throughout this discussion. These notations are fundamental to accurately grasping the mathematical and computational concepts underlying the approach. They serve as the building blocks for the complex models and algorithms developed to optimize DNN acceleration within NoC systems.

Table 4.1 provides a comprehensive list of these notations along with their descriptions. This reference is designed to aid readers in navigating through the technical aspects of the methodology, ensuring clarity and coherence in understanding the problem-solving strategies employed in this study.

By familiarizing themselves with these notations, readers can gain deeper insights into the logical framework and computational processes that are central to this research. Each notation plays a pivotal role in shaping the models and solutions presented, thereby contributing to the innovative advancements proposed in the field of DNN acceleration.

In the pursuit of optimizing DNN accelerators within 2D-mesh NoC architectures, the efficient grouping of neurons emerges as a fundamental task. This process is integral to enhancing both the performance and energy efficiency of the system, necessitating a strategic alignment of neurons to nodes within the network.

Central to this neuron grouping effort is the objective to map each neuron, denoted as $n_i \in N$ in the DNN graph $DNNG(N, A)$, to a distinct node $v_j \in V$ in the application graph $DAG(V, E)$. The primary aim here is to minimize the total communication weight, Ω, a critical factor for optimal system functionality.

Table 4.1 Notations used in problem definitions and methodology.

Notation	**Description**
N	Set of neurons in the DNN Graph
A	Set of connections (arcs) among neurons in the DNN Graph
$a_{i,j}$	Connection from neuron n_i to neuron n_j
$DNNG(N, A)$	Directed graph representing the DNN topology
V	Set of vertices (nodes) in the Application Graph
E	Set of directed edges in the Application Graph
$e_{i,j}$	Edge representing the connection from node v_i to node v_j
$w_{i,j}$	Communication weight of the edge $e_{i,j}$
$DAG(V, E)$	Directed acyclic graph representing the Application Graph
P	Set of processing elements (cores) in the 2D-mesh NoC
L	Set of bidirectional links connecting neighboring cores
$l_{i,j}$	Link connecting core p_i to core p_j
$TG(P, L)$	Bidirectional graph representing 2D-mesh NoC topology
Ω	Total communication weight in the neuron grouping problem
Ψ_{NoC}	Total communication cost in the NoC architecture
$d_{i,j}$	Hop distance between nodes v_i and v_j on the 2D-mesh
t_{avg}	Average computation cost for each processing element
t_{v_i}	Total computation time for neurons in node v_i
δ	Deviation parameter from the average computation

In this mapping, the total number of nodes in DAG, represented by $|V|$, is constrained not to exceed the available processing elements (PEs) in the NoC architecture, indicated by $|P|$. The average computation cost for each PE, t_{avg}, is also considered, ensuring that the total computation time for neurons in each node v_i, represented as t_{v_i}, adheres to a user-defined range. This range is determined by δ, a deviation parameter from t_{avg}, tailored to specific system needs.

The focus of the optimization is thus captured in the following equation(1), aiming to minimize the overall communication weight:

$$\min \Gamma\ \Omega = \sum_{\forall e_{i,j} \in E} w_{i,j} \tag{1}$$

Simultaneously, the node mapping challenge in 2D-mesh NoC systems for DNN accelerators is approached with equal significance. This involves establishing a one-to-one mapping between the nodes in DAG and the PEs in the NoC, pivotal for ensuring efficient data transfer and system performance.

This mapping involves assigning each node $v_i \in V$ of DAG to a unique PE $p \in P$, with the overarching goal of minimizing the total communication cost within the NoC, denoted as Ψ_{NoC}. The calculation of this cost includes aggregating the costs for each data transfer, determined by the product of data weight $w_{i,j}$ and the hop distance $d_{i,j}$ between nodes v_i and v_j on the 2D-mesh. This distance is calculated using the Manhattan metric as follows: $d_{i,j} = |x_{v_i} - x_{v_j}| + |y_{v_i} - y_{v_j}|$.

The function aiming to reduce the total communication cost is formulated in(2) as follows:

$$\min \Gamma\ \Psi_{NoC} = \sum_{\forall e_{i,j} \in E} d_{i,j} w_{i,j} \tag{2}$$

By adopting these strategies, both neuron grouping and node mapping challenges within NoC architectures are comprehensively addressed, with a dedicated focus on optimizing communication efficiency and enhancing the overall functionality of DNN accelerators.

As depicted in Figure 4.1, a visual representation of the challenges discussed earlier, specifically neuron grouping and node mapping within a DNN framework, is provided. This figure utilizes a representative DNN model composed of seventeen neurons, with twelve of these neurons evenly distributed across two hidden layers, exemplifying a common structural organization in DNNs. The model is mathematically depicted as $DNNG(V, A)$, aligning with the notations defined earlier.

For this illustration, a neuron grouping approach is used which was proposed previously [50], referenced in Figure 3.3, is adopted. This example serves to visually contextualize the abstract concepts of neuron grouping and mapping, providing a clearer understanding of how these processes are implemented in practical scenarios.

In the central part of Figure 4.1, the application graph, designated as $DAG(V, E)$, is presented. This graph effectively visualizes the groups formed from the neuron grouping process, with each node distinctly marked with a number that denotes its group. The numerical labels on the edges of this graph are indicative of the communication volume between groups, essentially quantifying the number of weights transferred from one neuron group to another. In the context of the sample DNN model, this grouping method resulted in a cumulative weight of 28, reflecting the total communication demand imposed by the chosen neuron grouping configuration.

Furthering the analysis, the spiral mapping technique for node mapping, a method detailed in a previous study [50], was applied. This approach involves arranging the nodes in a sequential, snake-like pattern, adhering to their numerical order, which facilitates a more organized and systematic mapping process. Employing this method yielded a total communication cost of 43.

Figure 4.2 in the study illustrates the impact of employing an alternative approach to neuron grouping. This variation involves a different arrangement of neurons within the layers compared to the initial example. Despite the altered grouping strategy, the total weight of the communication, interestingly, remains at 28, similar to the first example. However, it

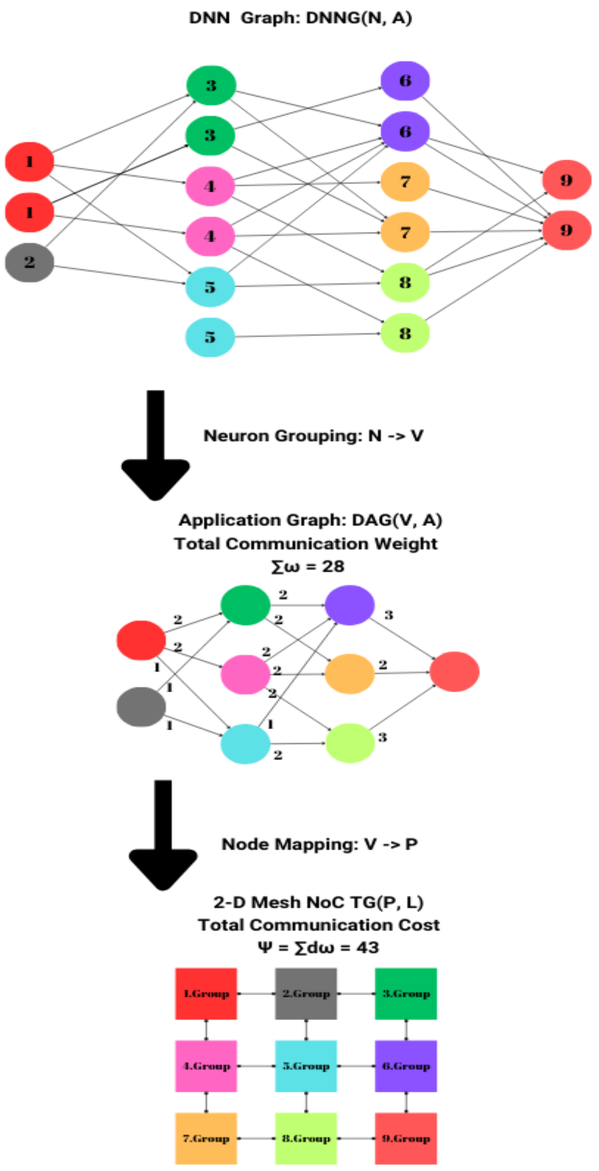

Figure 4.1 A representative instance of neuron clustering and node mapping problems.

is important to note that in this scenario, the mapping method remained consistent with the previous example.

A noteworthy observation from this approach is the subtle yet significant increase in the total communication cost, which rises to 45. This increment, despite the constant total weight, underscores the sensitivity of the communication cost to the specific neuron grouping method employed. It highlights that even with a similar level of communication weight, different grouping strategies can lead to varying efficiencies in communication cost, thereby influencing the overall effectiveness of the network design.

This outcome demonstrates the critical influence that neuron grouping methods exert on the final design objectives of NoC-based DNN accelerators. It underlines the importance of careful consideration and selection of neuron grouping strategies in optimizing network performance and achieving desired design goals.

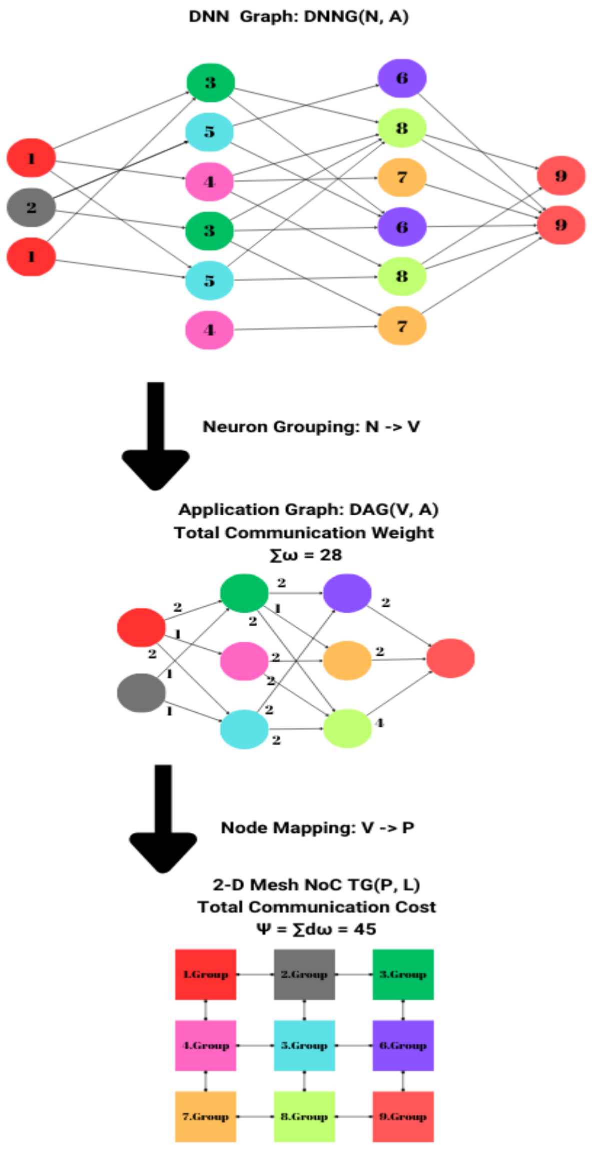

Figure 4.2 The impact of alternative neuron clustering techniques on overall communication cost.

In the continued exploration, Figure 4.3 illustrates the significant impact of varying the node mapping method. By applying an alternative mapping strategy to the application graph showcased in Figure 4.2, a notable reduction in the total communication cost is observed. This adjusted approach leads to a decreased communication cost of 40, as effectively depicted in Figure 4.3.

This result not only highlights the influence of the chosen node mapping method on communication efficiency but also underscores the importance of an integrated approach to both neuron grouping and node mapping in the system design. The variation in communication costs with different mapping techniques indicates that optimal solutions for both grouping and mapping are essential to achieving the most effective and efficient design outcomes in NoC-based DNN accelerators.

4.1. Existing ILP Approaches to Neuron Grouping and Mapping

ILP serves as a potent mathematical optimization tool, especially effective for problems characterized by discrete and integer variables. Its core function is to identify an optimal solution, either by maximizing or minimizing an objective function, within a defined framework of constraints. ILP's strength lies in its ability to provide precise and optimal solutions, making it highly valuable for specific problem-solving scenarios.

A previously existing ILP-based approach will be compared with the proposed SA and HM within the scope of this thesis. Essentially, ILP is a good option to compare proposed methods because it includes the mathematical representation of the problem and its optimal solution.

However, it is important to acknowledge that ILP can be resource-intensive, particularly in terms of computation, potentially limiting its scalability for larger problem sets. Given this, ILP has been strategically applied to smaller-scale DNNs in the study. This application is primarily aimed at achieving optimal solutions within feasible time frames. Furthermore, the use of ILP in this approach is instrumental in evaluating how closely the results derived from SA and heuristic methods align with these optimal solutions.

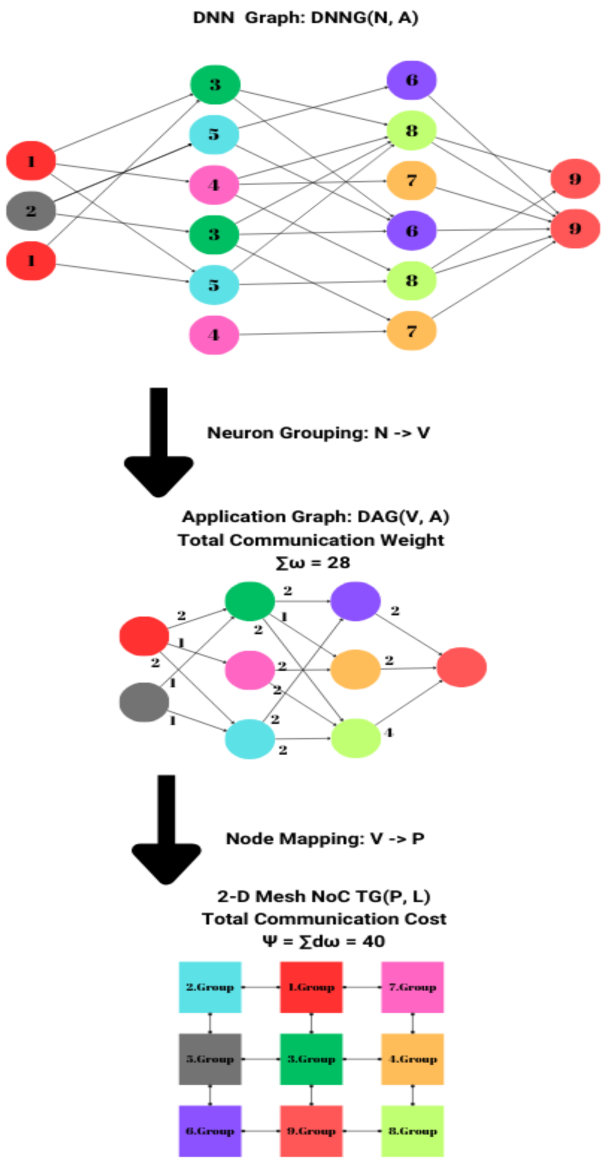

Figure 4.3 The impact of alternative node mapping methods on overall communication cost.

4.1.1. ILP-based Neuron Grouping Method

For clarity and ease of understanding, the notations employed in the neuron grouping ILP formulations are detailed in Table 4.2.

In this ILP model, $\Gamma_{i,j}$ is defined as a binary variable. It is assigned the value 1 (`TRUE`) if and only if the neuron $\mathcal{N}_i$ is a member of the group $\mathcal{G}_j$. Conversely, it is set to 0 (`FALSE`) if this

Table 4.2 Notations used in the ILP model of the neuron grouping problem.

Input Variable	**Description**
$\mathcal{N} = \{\mathcal{N}_i \,\Gamma\, i = 1, \ldots, N\}$	Represents the set of N neurons in the DNN.
$\mathcal{L} = \{\mathcal{L}_i \,\Gamma\, i = 1, \ldots, L\}$	Represents the set of L layers in the DNN.
$\mathcal{G} = \{\mathcal{G}_i \,\Gamma\, i = 1, \ldots, P\}$	Represents the set of P groups of neurons, where P is the number of available processing elements.
$NN_{i,j}$	$N \times L$ binary matrix representation of the DNN, where $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, L\}$.
$\mathcal{M}_{i,j}$	Adjacency matrix representation of the DNN, where $i, j \in \{1, \ldots, N\}$.
ϵ_i	Number of incoming edges to neuron $\mathcal{N}_i$.
t_{avg}	Average execution time of a processing element.
δ	Maximum acceptable deviation of a group's execution time from the average (e.g., 0.1, 0.2).

Decision Variable	**Description**
$\Gamma_{i,j}$	Binary variable that is TRUE if neuron $\mathcal{N}_i$ belongs to group $\mathcal{G}_j$, for $i \in \{1, \ldots, N\}$ and $j \in \{1, \ldots, P\}$.
$\sigma_{i,j,k}$	Binary variable that is TRUE if neurons $\mathcal{N}_i$ and $\mathcal{N}_j$ are in the same group $\mathcal{G}_k$, for $i, j \in \{1, \ldots, N\}$ and $k \in \{1, \ldots, P\}$.
$\kappa_{i,j,k,l}$	Binary variable that is TRUE if neuron $\mathcal{N}_i$ in $\mathcal{G}_k$ and neuron $\mathcal{N}_j$ in $\mathcal{G}_l$, for $i, j \in \{1, \ldots, N\}$ and $k, l \in \{1, \ldots, P\}$.
$\varphi_{i,l}$	Binary variable that is TRUE if neuron $\mathcal{N}_i$ communicates with at least one other neuron in group $\mathcal{G}_l$.
$t_{\mathcal{G}_i}$	Execution time of group $\mathcal{G}_i$, for $i \in \{1, \ldots, P\}$.
$\omega_{i,j}$	Communication weight between groups $\mathcal{G}_i$ and $\mathcal{G}_j$.
Ω	Total communication cost between all groups.

is not the case. The formulation of this binary variable is crucial for accurately representing the grouping of neurons in the model, as detailed in equations (3) and (4).

The binary nature of $\Gamma_{i,j}$ is formally expressed in (3) as:

$$\forall (i \in \mathcal{N}, j \in \mathcal{G}) \;\Gamma\; \Gamma_{i,j} \in \{0, 1\} \tag{3}$$

The value assignment for $\Gamma_{i,j}$ is based on the grouping condition is expressed in (4) as follows:

$$\Gamma_{i,j} = \begin{cases} 1, & \text{if } \mathcal{N}_i \text{ belongs to } \mathcal{G}_j \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

The ILP model necessitates that each neuron is allocated to one and only one group, ensuring an exclusive and clear assignment. This requirement is mathematically articulated in equation (5), ensuring a distinct and singular grouping for each neuron:

$$\forall(i \in \mathcal{N})\ \Gamma \sum_{j \in \mathcal{G}} \Gamma_{i,j} = 1 \tag{5}$$

Moreover, to optimize the use of available processing elements (PEs) and prevent any from remaining idle, the model includes a constraint mandating that each group must encompass at least one neuron. This condition is essential for the efficient utilization of computational resources and is formulated in equation (6):

$$\forall(j \in \mathcal{G})\ \Gamma \sum_{i \in \mathcal{N}} \Gamma_{i,j} \geq 1 \tag{6}$$

In this ILP framework, $\sigma_{i,j,k}$ is defined as a binary decision variable. It assumes the value 1 (`TRUE`) when both neurons $\mathcal{N}_i$ and $\mathcal{N}_j$ are assigned to the same group $\mathcal{G}_k$. Conversely, it is set to 0 (`FALSE`) if this condition does not hold. This binary variable is crucial for determining the group composition of neurons and is articulated through the following formulations:

The binary nature of $\sigma_{i,j,k}$ is specified in equation (7) as follows:

$$\forall(i, j \in \mathcal{N}, k \in \mathcal{G})\ \Gamma\ \sigma_{i,j,k} \in \{0, 1\} \tag{7}$$

The condition under which $\sigma_{i,j,k}$ is assigned a value is defined in equation (8) as:

$$\sigma_{i,j,k} = \begin{cases} 1, & \text{if } \mathcal{N}_i \text{ and } \mathcal{N}_j \text{ are both in group } \mathcal{G}_k \\ 0, & \text{otherwise} \end{cases} \tag{8}$$

Moreover, the relationship between $\sigma_{i,j,k}$ and the neuron-to-group assignment variables $\Gamma_{i,k}$ and $\Gamma_{j,k}$ is expressed in equation (9) as:

$$\forall(i, j \in \mathcal{N}, k \in \mathcal{G})\ \Gamma\ \sigma_{i,j,k} = \Gamma_{i,k} \times \Gamma_{j,k} \tag{9}$$

A crucial aspect of the ILP formulation involves the linearization of the product of the binary decision variables, as defined in Equation (9). This step is essential for ensuring the mathematical tractability of the model. To achieve this, $\sigma_{i,j,k}$ is reformulated using a set of linear inequalities, thereby enabling its integration into the ILP framework without compromising the model's integrity. These linear inequalities, detailed below, effectively encapsulate the interdependencies between the decision variables:

$$\begin{align} \forall(i, j \in \mathcal{N}, k \in \mathcal{G})\ \Gamma \tag{10}\\ \sigma_{i,j,k} \leq \Gamma_{i,k} \tag{11}\\ \sigma_{i,j,k} \leq \Gamma_{j,k} \tag{12}\\ \sigma_{i,j,k} \geq \Gamma_{i,k} + \Gamma_{j,k} - 1 \tag{13} \end{align}$$

These inequalities are designed to ensure that $\sigma_{i,j,k}$ accurately reflects the group composition as dictated by the binary variables $\Gamma_{i,k}$ and $\Gamma_{j,k}$. This linearization step is a key element in the optimization process, as it allows for the efficient solving of the ILP model while maintaining the essential relationships between neurons and their respective groups.

In the ILP model, the specific focus is on intra-layer neuron grouping. To enforce this, a constraint is introduced that prohibits neurons from being grouped if they are not in the same layer. This constraint is crucial for maintaining the integrity of layer-specific computations within the DNN. The formulation of this constraint is given in (14):

$$\forall(i,j \in \mathcal{N}, l \in \mathcal{L}) \ \Gamma \ \text{ If } NN_{i,l} \neq NN_{j,l} \sum_{k \in \mathcal{G}} \sigma_{i,j,k} = 0 \tag{14}$$

Additionally, $t_{\mathcal{G}_i}$ is defined as a decision variable representing the execution time of each group $\mathcal{G}_i$. This variable is computed by summing the execution times of all neurons in the group. As the execution times can vary based on the processing elements and their detailed calculation can be complex, a neuron's execution time is approximated using the number of its incoming edges, denoted as ϵ_j. This approximation aligns with the fact that neuron execution is proportionate to its incoming connections, as shown in Equation (15):

$$\forall(i \in \mathcal{G}) \ \Gamma \ t_{\mathcal{G}_i} = \sum_{j \in \mathcal{N}} \left(\Gamma_{j,i} \times \epsilon_j\right) \tag{15}$$

The number of incoming edges ϵ_j for each neuron $\mathcal{N}_j$ is calculated as per Equation (16):

$$\forall(j \in \mathcal{N}) \ \Gamma \ \epsilon_j = \sum_{i \in \mathcal{N}} \mathcal{M}_{i,j} \tag{16}$$

To prevent overloading the processing elements, a constraint is imposed on the execution time of each group. This constraint ensures that the execution time does not exceed the average execution time of all groups, denoted as t_{avg}, by more than a specified margin δ. This is expressed in the inequality (17).

$$\forall(i \in \mathcal{G}) \ \Gamma \ t_{\mathcal{G}_i} \leq (1+\delta) \times t_{avg} \tag{17}$$

Lastly, the average execution time t_{avg} is calculated, which is the sum of the execution times of all neurons divided by the number of available processing elements P, as defined in Equation (18):

$$t_{avg} = \frac{\sum_{i \in \mathcal{N}} \epsilon_i}{P} \tag{18}$$

In the ILP model, the computation of communication weight between neuron groups necessitates the introduction of additional decision variables: $\kappa_{i,j,k,l}$ and $\varphi_{i,l}$. The variable $\kappa_{i,j,k,l}$, a binary decision variable, is assigned the value 1 (`TRUE`) when neuron $\mathcal{N}_i$ in group $\mathcal{G}_k$ communicates with neuron $\mathcal{N}_j$ in group $\mathcal{G}_l$, and 0 (`FALSE`) otherwise. The formulation for $\kappa_{i,j,k,l}$ is given in (19).

$$\forall(i, j \in \mathcal{N}, k, l \in \mathcal{G}) \; \Gamma \; \kappa_{i,j,k,l} \in \{0, 1\} \tag{19}$$

The linearization of the product of two decision variables in Equation (20) is achieved using a similar approach as for $\sigma_{i,j,k}$.

$$\forall(i, j \in \mathcal{N}, k, l \in \mathcal{G}) \; \Gamma \; \kappa_{i,j,k,l} = \Gamma_{i,k} \times \Gamma_{j,l} \tag{20}$$

$\varphi_{i,l}$, another binary decision variable, is set to 1 (`TRUE`) if neuron $\mathcal{N}_i$ communicates with any neuron in group $\mathcal{G}_l$, and 0 (`FALSE`) otherwise. The formulation for $\varphi_{i,l}$ is given in (21) and (22).

$$\forall(i \in \mathcal{N}, l \in \mathcal{G}) \; \Gamma \; \varphi_{i,l} \in \{0, 1\} \tag{21}$$

$$\forall(i \in \mathcal{N}, l \in \mathcal{G}) \; \Gamma \; \varphi_{i,l} \geq \kappa_{i,j,k,l} \times \mathcal{M}_{i,j} \tag{22}$$

The communication weight between groups $\mathcal{G}_k$ and $\mathcal{G}_l$ is represented by the decision variable $\omega_{k,l}$. This weight is calculated by summing all communication links originating from group $\mathcal{G}_k$ intended for neurons in group $\mathcal{G}_l$. Each communication link, irrespective of the number of recipient neurons in group $\mathcal{G}_l$, is assumed to have a weight of 1. This assumption is based on the fact that once a neuron's output information is sent from its group's PE, it becomes available to all neurons in the recipient group via shared memory. The calculation of $\omega_{k,l}$, therefore, involves summing all instances of $\varphi_{i,l}$ where neuron $\mathcal{N}_i$ belongs to group $\mathcal{G}_k$. This is formulated in equation (23).

$$\forall(k, l \in \mathcal{G})\ \Gamma\ \omega_{k,l} = \sum_{i \in \mathcal{N}} (\varphi_{i,l} \times \Gamma_{i,k}) \tag{23}$$

Finally, the objective function seeks to minimize the total communication weight Ω across all neuron groups, formalized in equation (24):

$$\text{Minimize } \Omega = \sum_{k,l \in \mathcal{G}} \omega_{k,l} \tag{24}$$

4.1.2. ILP-based Node Mapping Method

The notations utilized in the mathematical formulations for node mapping are delineated in Table 4.3. This table provides a comprehensive overview of the input and decision variables that are integral to the model.

Table 4.3 Notations used in the mathematical model for node mapping.

Input Variable	**Description**
$\mathcal{G} = \{\mathcal{G}_i\ \Gamma\ i = 1, \ldots, P\}$	Set of P groups to be mapped.
$\mathcal{C} = \{\mathcal{C}_i\ \Gamma\ i = 1, \ldots, P\}$	Set of P cores in the topology.
WCTG	Communication weight matrix of the DAG.
Decision Variable	**Description**
$\psi_{i,j}$	Total communication cost between cores $\mathcal{C}_i$ and $\mathcal{C}_j$.
$\mu_{i,j}$	Indicates if group $\mathcal{G}_i$ is mapped to core $\mathcal{C}_j$.
$\chi_{i,j,r,s}$	True if group $\mathcal{G}_i$ is mapped to core $\mathcal{C}_r$ and group $\mathcal{G}_j$ is mapped to core $\mathcal{C}_s$.
Ψ_{NoC}	Total communication cost of the system.

The decision variable $\mu_{i,j}$ is defined as a binary indicator, true if group $\mathcal{G}_i$ is mapped to core $\mathcal{C}_j$ and false otherwise. This binary nature of $\mu_{i,j}$ is established in Equation (25), while its conditional assignment is formulated in Equation (26).

$$\forall (i \in \mathcal{G}, j \in \mathcal{C}) \; \Gamma \; \mu_{i,j} \quad is\_binary \tag{25}$$

$$\mu_{i,j} = \begin{cases} 1 & \text{if } \mathcal{G}_i \text{ is assigned to } \mathcal{C}_j \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

To ensure a unique mapping of each group to a specific core, a constraint is imposed, as delineated in Equation (27). This constraint ensures that each group $\mathcal{G}_i$ is mapped to exactly one core in $\mathcal{C}$.

$$\forall i \in \mathcal{G} \; \Gamma \sum_{j \in \mathcal{C}} \mu_{i,j} = 1 \tag{27}$$

The model also incorporates a constraint ensuring that each core in the set $\mathcal{C}$ is assigned to at most one group from the set $\mathcal{G}$. This constraint, applicable even in scenarios where the number of cores exceeds the number of groups, is articulated in Equation (28).

$$\forall j \in \mathcal{C} \; \Gamma \sum_{i \in \mathcal{G}} \mu_{i,j} \leq 1 \tag{28}$$

Furthermore, the binary variable $\chi_{i,j,r,s}$ is introduced to indicate the simultaneous mapping of group $\mathcal{G}_i$ to core $\mathcal{C}_r$ and group $\mathcal{G}_j$ to core $\mathcal{C}_s$. The binary nature and conditional logic of $\chi_{i,j,r,s}$ are defined in Equations (29) and (30).

$$\forall (i, j \in \mathcal{G}, r, s \in \mathcal{C}) \; \Gamma \; \chi_{i,j,r,s} \quad is\_binary \tag{29}$$

$$\chi_{i,j,r,s} = \begin{cases} 1 & \text{if } \mathcal{G}_i \text{ is assigned to } \mathcal{C}_r \\ & \text{and } \mathcal{G}_j \text{ is assigned to } \mathcal{C}_s \\ 0 & \text{otherwise} \end{cases} \tag{30}$$

The calculation of the binary variable $\chi_{i,j,r,s}$ involves the logical `AND` operation between $\mu_{i,r}$ and $\mu_{j,s}$, as delineated in Equation (31).

$$\forall (i, j \in \mathcal{G}, r, s \in \mathcal{C})\ \Gamma\ \chi_{i,j,r,s}\ = \mu_{i,r} \times \mu_{j,s} \tag{31}$$

To address the non-linear nature of Equation (31), a linearization process is employed, resulting in the establishment of the following set of inequalities (32).

$$\forall (i, j \in \mathcal{G}, r, s \in \mathcal{C})\ \Gamma\ \chi_{i,j,r,s} \leq \mu_{i,r} \tag{32a}$$

$$\forall (i, j \in \mathcal{G}, r, s \in \mathcal{C})\ \Gamma\ \chi_{i,j,r,s} \leq \mu_{j,s} \tag{32b}$$

$$\forall (i, j \in \mathcal{G}, r, s \in \mathcal{C})\ \Gamma \chi_{i,j,r,s} \geq \mu_{i,r} + \mu_{j,s} - 1 \tag{32c}$$

The total communication cost between any two cores, $\mathcal{C}_i$ and $\mathcal{C}_j$, is represented by the variable $\psi_{i,j}$. This cost is articulated in Equation (33). The formulation encapsulates the communication cost for each pair of communicating groups $\mathcal{G}_i$ and $\mathcal{G}_j$ within the group graph DAG, assigned respectively to cores $\mathcal{C}_r$ and $\mathcal{C}_s$. The cost computation involves the product of the communication weight and the Manhattan distance $MD_{r,s}$ between these cores.

$$\begin{aligned} &\forall (r, s \in \mathcal{C})\ \Gamma \\ &\psi_{r,s}\ = \sum_{i,j \in \mathcal{G}} MD_{r,s} \times \chi_{i,j,r,s} \times WCTG_{i,j} \end{aligned} \tag{33}$$

The total communication cost within the Network-on-Chip (NoC) architecture, denoted as Ψ_{NoC}, is determined as per Equation (34). This equation accumulates the individual communication costs across all pairs of cores within the system.

$$\Psi_{NoC} = \sum_{r,s \in \mathcal{C}} \psi_{r,s} \tag{34}$$

The ultimate goal of this mathematical model is to minimize the overall communication cost across the NoC-based system.

4.2. Proposed Methods For Neuron Grouping

In the ensuing discussion, the methodologies examined for neuron grouping within DNNs, in the context of 2D-mesh NoC architectures, are delved into.

4.2.1. Improved Heuristic Method

The approach to optimizing neural networks introduces a novel heuristic method that expands upon the established intra-layer neuron grouping, as delineated in Figure 3.2. At the heart of this methodological enhancement is the integration of a crucial parameter, denoted as δ. Far from being just an add-on, this δ parameter is instrumental in refining the neuron grouping process.

The introduction of δ serves a dual purpose: it not only guides the distribution of processing loads within a pre-specified margin but also sets the upper and lower limits for the processing load each mesh node can handle. This strategic implementation of δ is vital in ensuring that every node is engaged optimally, effectively opposing scenarios where nodes are either underutilized or burdened with overload.

This heuristic method represents a paradigm shift from traditional, uniform distribution strategies. By adopting a more nuanced and flexible approach to load distribution, it allows for a more effective utilization of network nodes. This adaptability in managing processing loads is essential in maximizing the efficiency and performance of the neural network within the 2D-mesh NoC architecture.

In the heuristic approach, each neuron threaded in the network as a distinct entity, precisely evaluating its contribution to the overall communication cost. This method involves a comprehensive calculation of the average communication weight across all neurons, establishing a pivotal criterion for the subsequent steps.

Central to this approach is the determination of a communication weight margin, delineating the maximum communication load that can be efficiently managed by each mesh core. Guided by this margin, a methodical grouping of neurons within each layer is executed. This sequential grouping, in line with the defined communication weight margin, ensures that the distribution of neurons across the network is both balanced and efficient.

This strategy, as illustrated in Figure 3.2, is integral to the heuristic method. It allows for a more nuanced and effective grouping of neurons, taking into account the individual impact of each neuron on the network's communication dynamics.

4.2.2. SA-based neuron grouping method

Metaheuristic algorithms represent a sophisticated class of optimization techniques designed to tackle complex, large-scale optimization problems. These algorithms are characterized by their strategic fusion of various heuristic methods and advanced techniques to traverse the solution space. This approach enables the identification of high-quality solutions that, while not guaranteed to be optimal, often represent significant improvements over local optima. A key feature of metaheuristics is the incorporation of randomization elements, which play a pivotal role in the expansive exploration of the solution space. This is achieved through a blend of deterministic and stochastic processes, with random walks ensuring a globally diversified set of candidate solutions [57, 58]. Furthermore, random walks serve as a critical mechanism in broadening the search horizon, enhancing the global reach of the solution exploration [59].

One of the notable strengths of metaheuristic algorithms lies in their efficiency and adaptability in managing large and intricate problem domains. Unlike Integer Linear Programming (ILP), which may face scalability issues, metaheuristics demonstrate a robust capacity to handle extensive, multifaceted problems. However, it is important to acknowledge that the solutions yielded by metaheuristic approaches may not always attain optimal status. Additionally, assessing the quality of these solutions can pose a challenge, given the inherent complexity and breadth of the solution landscape they explore.

SA stands as a prominent metaheuristic optimization technique, drawing inspiration from the physical process of annealing in metallurgy. This method is particularly adept at navigating complex optimization landscapes to identify optimal or near-optimal solutions. Its foundational principle lies in emulating the gradual cooling process of a material, a strategy aimed at minimizing structural defects and stabilizing the material in its lowest energy state [60].

The efficacy of SA is rooted in its inherent ability to circumvent the pitfalls of local optima, a common limitation in deterministic optimization methods. The technique operates by progressively lowering the system's 'temperature', effectively reducing the likelihood of accepting suboptimal solutions over time. Essential to this process are several key steps: initiating with an initial state, typically a randomly generated solution; determining an appropriate starting temperature; and methodically selecting neighboring candidate solutions for evaluation.

Each candidate is assessed based on its performance against the objective function, with the subsequent decision to accept or reject the candidate depending on a probability function that depends on the current temperature of the system. Critical to the success of the SA algorithm is the customization of implementation specifics, such as the selection of the initial state and the determination of an effective cooling schedule, tailored to the nuances of the given optimization problem.

Initial State Generation The initialization of an effective solution state is a critical step in the Simulated Annealing The initialization of an effective solution state is a critical step in the SA algorithm. For this purpose, a heuristic approach is adopted that aims to generate a preliminary random grouping of neurons. The fundamental objective of this approach is to establish $\mathcal{P}$ distinct groups, ensuring that each group comprises neurons from a single layer and adheres to principles of computational load balancing. The specifics of this method are outlined in Algorithm 1.

This heuristic strategy begins with the application of a greedy bin-packing algorithm, applied independently to each layer. This algorithm forms the basis of the initial grouping of neurons. The bin capacity in this context is defined as $(1 + \delta) \times t_{avg}$, aligning with the constraints to accommodate computational loads appropriately. Consequently, this step results in an initial set of neuron groups that conform to the defined load-balancing criteria.

Furthermore, to ensure the total number of groups does not exceed $\mathcal{P}$, the algorithm iteratively modifies the group structure. In cases where the number of groups is less than $\mathcal{P}$, the algorithm strategically splits the largest group into two, in each iteration. This process continues until the total number of groups precisely matches $\mathcal{P}$. Such a structured approach to initial state generation lays the groundwork for an efficient and balanced exploration of the solution space in the subsequent phases of the SA algorithm.

Neighbor State Generation The approach for generating a neighbor solution in the Simulated Annealing process is detailed in Algorithm 2. This algorithm strategically modifies the current state to explore the solution space, aiming to find an efficient grouping of neurons that optimizes communication within the NoC architecture.

In the Simulated Annealing process for neuron grouping, the strategy involves altering the group assignment of a randomly selected neuron, ensuring adherence to specific criteria for a viable solution. A solution is deemed acceptable when it forms exactly $\mathcal{P}$ groups, with each group comprising neurons exclusively from a single layer. Layers consisting of a solitary neuron are exempted from this process due to the immutability of their grouping.

Upon selecting a neuron at random, the algorithm contemplates two primary actions: either relocating the neuron to an alternate group within its current layer or exchanging it with another neuron from a different group in the same layer. This decision-making process encompasses two pivotal scenarios:

Algorithm 1 Generation of the Initial Solution

Inputs:

- $NN_{i,j}$: $N \times L$ binary matrix representing the Neural Network (NN)
- $\mathcal{M}_{i,j}$: Adjacency matrix representation of the NN
- $\mathcal{P}$: Number of processing elements
- ϵ: Number of incoming edges to neuron $\mathcal{N}_i$

Output: $\mathcal{G}$ - the randomly generated initial grouping state.

```
1:  for each layer $i \in L$ do
2:      $tmp\_groups \leftarrow$ Bin-pack($\mathcal{NL}_i, \epsilon, (1+\delta) \times t_{avg}$).
3:      Append $tmp\_groups$ to $\mathcal{G}$.
4:  end for
5:  $NumGroups \leftarrow$ length of $\mathcal{G}$.
6:  while $NumGroups < \mathcal{P}$ do
7:      Initialize a new group $\mathcal{G}_{NumGroups+1}$.
8:      $maxGroup \leftarrow$ getGroupIdxWithMaxNeurons().
9:      for each neuron $i$ from middle to end of $maxGroup.neurons$ do
10:         Add neuron to $\mathcal{G}_{NumGroups+1}$.
11:         Remove neuron from $\mathcal{G}_{maxGroup}$.
12:     end for
13:     Increment $NumGroups$.
14: end while
15: return $\mathcal{G}$.
```

- Empty Original Group Scenario: If transferring the neuron results in an empty original group, it becomes necessary to re-establish the prescribed number of groups. This is achieved by dividing an existing group, thereby relocating one of its neurons to form a new group.

- Single Group Layer Scenario: In cases where the selected neuron resides in a layer with a singular group, the algorithm identifies a layer with a minimum of two groups, one of which should ideally contain only one neuron. The lone neuron from this identified group is then moved to another group within the same layer, thus vacating its original group. Subsequently, the selected neuron is relocated to this now-available group.

Through these mechanisms, the algorithm dynamically restructures the neuron groups while maintaining the essential criteria of solution viability.

Algorithm 2 Generation of a Neighbor Solution

Inputs:

- $\mathcal{G}$: The current state
- $NN_{i,j}$: $N \times L$ binary matrix representing the Neural Network (NN)
- $\mathcal{M}_{i,j}$: Adjacency matrix representation of the NN
- $\mathcal{P}$: Number of processing elements

Output: $\mathcal{G}_n$ - A randomly generated neighbor state.

```
1: $\mathcal{N}_{rand} \leftarrow$ random$(0, N-1)$ {Select a random neuron}
2: $\mathcal{L}_{rand} \leftarrow \mathcal{N}_{rand}.layer$
3: assert $\mathcal{L}_{rand}.size > 1$ {Ensure layer has more than one neuron}
4: $\mathcal{G}_{rand} \leftarrow \mathcal{N}_{rand}.group$
5: if other groups exist in layer $\mathcal{L}_{rand}$ then
6:     $\mathcal{G}_{trg} \leftarrow$ random group in $\mathcal{L}_{rand}$ different from $\mathcal{G}_{rand}$
7:     $swap \leftarrow$ random$(0, 1)$
8:     if $swap = 1$ then
9:         $\mathcal{N}_{trg} \leftarrow$ random neuron in $\mathcal{G}_{trg}$
10:        Swap $\mathcal{N}_{rand}$ with $\mathcal{N}_{trg}$
11:    else
12:        Move $\mathcal{N}_{rand}$ to $\mathcal{G}_{trg}$
13:        if $\mathcal{G}_{rand}$ becomes empty then
14:            $\mathcal{G}_{vctm} \leftarrow$ random group with more than one neuron
15:            $\mathcal{N}_{vctm} \leftarrow$ random neuron in $\mathcal{G}_{vctm}$
16:            Move $\mathcal{N}_{vctm}$ to $\mathcal{G}_{rand}$
17:        end if
18:    end if
19: else
20:    if $\mathcal{G}_{rand}$ has more neurons then
21:        $\mathcal{L}_{src} \leftarrow$ layer with at least two groups
22:        Move a neuron from a single-neuron group in $\mathcal{L}_{src}$ to another group
23:        Move $\mathcal{N}_{rand}$ to the now empty group in $\mathcal{L}_{src}$
24:    end if
25: end if
26: return $\mathcal{G}_n$
```

Annealing Schedule In this approach, the geometric cooling schedule as outlined previously [61] is adopted, illustrated in Figure 4.4 and mathematically represented in Equation (35). This schedule is characterized by a constant α, typically ranging from 0.8 to 0.99, whose value is experimentally determined. The constant α is a crucial factor, being less than one yet proximate to it, influencing the rate of temperature reduction at each iteration k.

$$T_{k+1} = T_k \times (1-\alpha)^k \tag{35}$$

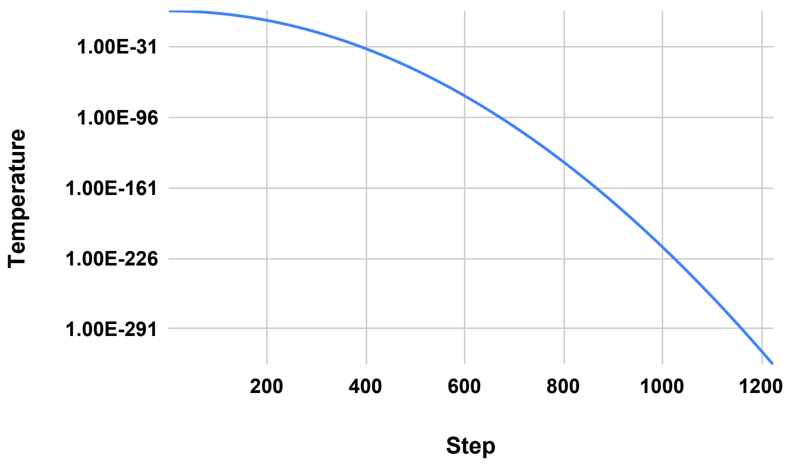

Figure 4.4 Illustration of the cooling schedule with $\alpha = 0.999$ depicted on a logarithmic scale.

The initial temperature setting is aligned with the energy level of the initial solution specific to each problem. This configuration permits the system to initially accept suboptimal solutions with a higher likelihood, thereby facilitating a more exhaustive exploration of the solution space. The annealing process is designed to persist until the system temperature effectively reaches zero. Notably, the iteration count is not predetermined but is instead scaled according to the problem size, guided by the initial energy levels.

The energy calculation of the system leverages Equation (23) from Section 4.1.1. Additionally, for the computation of acceptance probabilities, the Metropolis–Hastings algorithm, introduced previously [62], forms the basis of the method.

4.3. Proposed Method for Node Mapping

In this section, a metaheuristic model based on SA is introduced and tailored for the node mapping process in mesh-based 2D NoC architectures.

4.3.1. SA-based Node Mapping Method

The SA-based node mapping method adapts the principles of Simulated Annealing to efficiently map neuron groups to processing elements (cores) within a 2D mesh NoC architecture. This approach extends the methodologies discussed in Section 4.2.2.

Initial Configuration Determination The initial configuration for the SA algorithm consists of a mapping state, represented as a one-dimensional array, S, where each array element corresponds to a processing element (core) in the NoC topology. The length of this array is equal to the number of processing elements, P. In this array, the value at each index indicates the group that is mapped to the corresponding core. Specifically, if $S[i] = \mathcal{G}j$, it implies that the neuron group $\mathcal{G}j$ is assigned to the core $\mathcal{C}_i$.

To set up the initial conditions for the simulated annealing process, including the initial temperature, final temperature, and iteration count, guidelines from [63] are utilized. These parameters are calibrated to start with a high acceptance probability for new states (98%) and to reduce to no improvement (0%) by the end of the process.

Neighbor State Generation and Annealing Configuration The generation of a neighboring state in the SA algorithm is conducted by randomly selecting and swapping the core assignments of two groups. This swap introduces variability in the mapping configuration, aiding the exploration of potential solutions.

The cooling schedule, crucial to the annealing process, follows the formulation presented in Equation (35) from Section 4.2.2. This schedule effectively manages the gradual reduction of the system's temperature, which is central to the simulated annealing methodology.

The objective function for this node mapping process remains consistent with that of the ILP model, aiming to minimize the total communication cost within the NoC architecture. This cost is calculated using the formula provided in Equation (34). The simulated annealing process iteratively improves the mapping solution by assessing its performance based on this objective function.

In summary, the SA-based node mapping method strategically utilizes simulated annealing principles to optimize the distribution of neuron groups across the cores in a 2D mesh NoC, thereby enhancing overall system performance and efficiency.

5. EXPERIMENTAL RESULTS

This section offers a comprehensive evaluation of both the ILP algorithm and SA-based approach developed for neuron grouping and node mapping. This evaluation compares with an alternative heuristic approach detailed in literature [38]. The heuristic method, illustrated in Figure 3.3, strategically groups neurons within the same layer and optimizes the number of neurons in each group to balance computational demands. This methodical approach is key to understanding the effectiveness and efficiency of different grouping and mapping strategies in DNN acceleration. In the mapping phase, a systematic strategy is applied, particularly designed to align with the unique features of the mesh architecture. The process initiates at the top-left vertex of the mesh, with neuron groups being sequentially assigned following their respective layer sequence. This assignment progresses linearly along the X-axis of the mesh. Upon reaching the end of a row, the mapping seamlessly transitions to the leftmost position of the next row. This technique ensures a coherent and systematic allocation of neuron groups throughout the mesh, facilitating efficient data communication and processing within the network-on-chip (NoC) framework.

For executing the ILP formulations, the Gurobi optimizer framework [64] is utilized. The SA algorithm, on the other hand, is implemented in the C programming language. The experimental evaluations were conducted on a computer system equipped with a Ryzen 5 3600 processor, featuring 6 cores and 12 threads, clocked at 3.60GHz. This system is further complemented by 32GB of RAM and operates under the Windows 10 operating system.

The experimental analysis involved ten distinct DNN models, each characterized by varying numbers of layers and neurons. These models have been systematically enumerated in Table 5.1. The first half of the benchmarks, encompassing five models, were sourced from existing literature. To comprehensively assess the performance of both the ILP and SA algorithms, especially in scenarios with escalated neuron counts, an additional five DNN models were constructed. These models were specifically designed with varying neuron quantities, thereby providing a diverse range of test cases to evaluate the scalability and robustness of the proposed algorithms under different network complexities.

Table 5.1 Summary of the Employed Benchmarks in Experimental Analysis.

Benchmark Name	**ID**	**Neurons in Layers**	**Total Neurons**
Churn Modelling Problem [65]	B1	11-6-6-1	24
Design of Galvanized Steels [66]	B2	3-9-9-3	24
Modelling of Supercapacitors [67]	B3	10-10-10-1	31
Object Recognition Problem [68]	B4	5-6-7-7-6-5	36
LeNet (Fully Connected Part)	B5	120-84-10	214
Custom Generated 1	C1	14-30-10-3	57
Custom Generated 2	C2	12-36-20-1	69
Custom Generated 3	C3	24-62-16	102
Custom Generated 4	C4	36-48-54-6	144
Custom Generated 5	C5	84-54-38-16	192

5.1. Evaluating the Impact of Pruning on Communication Weight

To assess how the pruning of Deep Neural Networks (DNNs) influences the communication weight, a series of experiments were conducted. These experiments involved varying the percentage of pruning applied to the DNNs. The focus was to identify a suitable pruning percentage that would be used in subsequent experiments. Benchmark B2 was selected for this analysis.

The results, depicted in Figure 5.1, illustrate the correlation between increased pruning percentages and the corresponding reduction in overall communication cost. This relationship is crucial for optimizing the network-on-chip (NoC) performance in DNN applications.

In subsequent analyses, the focus will be on examining the impacts of different levels of network pruning, specifically comparing scenarios with 50% pruning and fully connected structures. For this comparison, benchmarks will undergo random pruning, as referenced in the previous studies [69]. Effective pruning strategies aim to systematically eliminate non-essential or redundant connections and neurons, thereby reducing the size of the DNN without significantly affecting its accuracy.

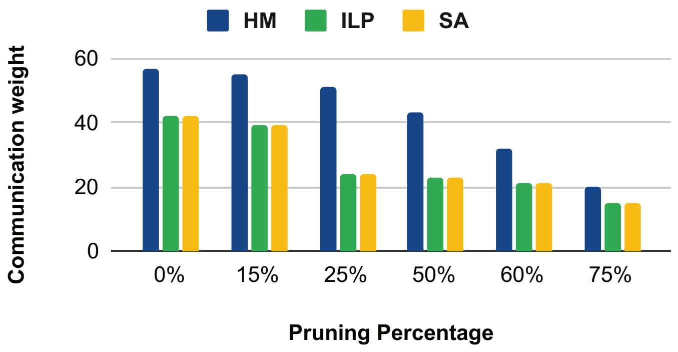

Figure 5.1 Variation in communication weight for benchmark B2 with different pruning percentages ($\delta = 1.0$, mesh size = 3x3).

The referenced study successfully implemented pruning at varying levels, including 70% for Kaldi, 79% for LeNet5, 67% for AlexNet, 48% for ResNet-50, and 51% for the Transformer model, with minimal impact on accuracy. These results informed the decision to adopt a 50% pruning percentage for the current investigation, providing a balance between network simplification and performance retention.

5.2. Neuron Grouping Results

The focus of this experimental analysis is on the optimization of neuron grouping, a critical aspect previously explored in the context of 2D-mesh NoC architectures for DNN accelerators. The primary objective is to minimize communication weight among neuron clusters, ensuring balanced computational loads among processing elements, a topic detailed in earlier sections. For this purpose, a delta value of $\delta = 1.0$ is adopted, aligning with the optimization approaches discussed in the 'Proposed Method' section.

To provide a comprehensive overview, Table 5.2 compiles the communication weight outcomes for fully connected DNNs (with a pruning percentage of 0%). In contrast, Table 5.3 illustrates the results obtained for DNNs pruned by 50%. These tables also feature a comparative analysis, delineated in the final three columns, which depict the percentage change (%Δ) in communication weights. This change is evaluated as a decrease, offering insights into the efficiency of different neuron grouping methods.

Table 5.2 Neuron grouping communication weight results for pruning percentage of 0%.

Benchmark/Mesh	HM	ILP	SA	%Δ ILP/HM	%Δ SA/HM	%Δ ILP/SA
B1 (3x3)	83	51	51	-38.55	-38.55	0.00
B2 (3x3)	60	42	42	-30.00	-30.00	0.00
B3 (3x3)	80	70	70	-12.50	-12.50	0.00
B4 (3x3)	45	38	44	-15.56	-2.22	-13.64
C1 (3x3)	176	112	126	-36.36	-28.41	-11.11
C2 (4x4)	420	236	272	-43.81	-35.24	-13.24
C3 (4x4)	583	368	368	-36.88	-36.88	0.00
			Average %Δ	**-29.46**	**-26.26**	**-5.43**

Table 5.3 Neuron grouping communication weight results for pruning percentage of 50%.

Benchmark/Mesh	HM	ILP	SA	%Δ ILP/HM	%Δ SA/HM	%Δ ILP/SA
B1 (3x3)	40	33	33	-17.50	-17.50	0.00
B2 (3x3)	45	33	33	-26.67	-26.67	0.00
B3 (3x3)	66	42	42	-36.36	-36.36	0.00
B4 (3x3)	39	31	31	-20.51	-20.51	0.00
C1 (3x3)	148	101	104	-31.76	-29.73	-2.88
C2 (4x4)	336	205	212	-38.99	-36.90	-3.30
C3 (4x4)	523	321	308	-38.62	-41.11	4.22
			Average %Δ	**-30.06**	**-29.83**	**-0.28**

These results offer invaluable insights into the impact of different neuron grouping strategies, particularly the ILP and SA-based methods previously formulated and discussed. The findings demonstrate the substantial reduction in communication weight achieved by these methods when compared to heuristic approaches, reinforcing the discussions from the 'Method' section regarding the potential of ILP and SA in optimizing NoC-based DNN accelerators.

Figures 5.2 and 5.3 visually articulate these findings. They present a graphical representation of the neuron grouping communication weights for fully connected and 50% pruned DNNs, respectively. The graphical illustrations are normalized to a scale of $[0, 100]$ to facilitate a clearer and more intuitive understanding of the data.

In summary, these experimental results not only validate the theoretical underpinnings discussed in previous sections but also provide empirical evidence for the effectiveness of

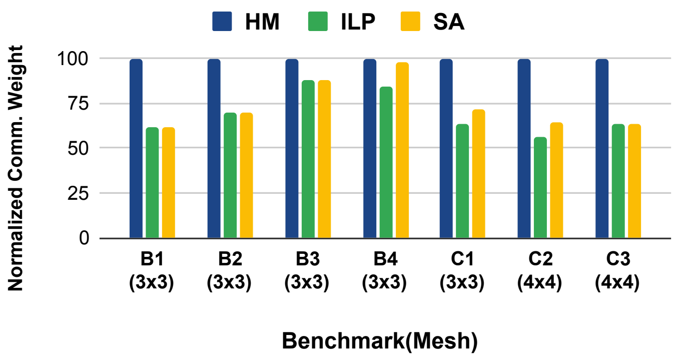

Figure 5.2 Neuron grouping communication weight results for fully connected DNNs (normalized to $[0, 100]$).

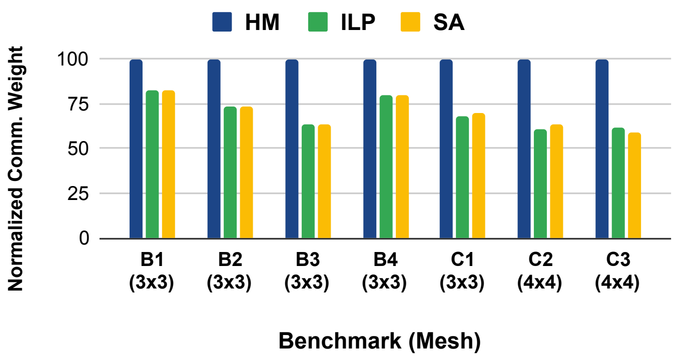

Figure 5.3 Neuron grouping communication weight results for 50% pruned DNNs (normalized to $[0, 100]$).

the proposed neuron grouping methods. The data underscores the significance of strategic neuron grouping in enhancing the overall performance of NoC-based DNN accelerators, especially in the context of varying levels of network pruning.

The exploration of neuron grouping methods for large-scale benchmarks is a significant aspect of this study, particularly in the context of fully connected and 50% pruned DNNs. Table 5.4 offers a detailed analysis of the communication weight results for these benchmarks, focusing on the Heuristic Method (HM) and Simulated Annealing (SA) outcomes. It is noteworthy that the Integer Linear Programming (ILP) model, extensively discussed in the 'Proposed Method' section and 'Existing ILP Approaches to Neuron Grouping and Mapping' subsection, was not feasible for these larger benchmarks due to its high computational requirements and memory constraints. This limitation underscores the complexities associated with scaling ILP solutions for larger neural networks, a challenge that was anticipated in the methodological considerations.

Table 5.4 HM and SA neuron grouping communication weight results for big benchmarks.

Benchmark	**Fully Connected DNN Results**			**50% Pruned DNN Results**		
(Mesh)	**HM**	**SA**	**%Δ SA/HM**	**HM**	**SA**	**%Δ SA/HM**
B5 (8x8)	5376	5292	-1.56	4615	3808	-17.49
C4 (5x5)	1014	726	-28.40	983	573	-41.71
C5 (8x8)	3084	1186	-61.54	2787	2352	-15.61
	Average %Δ		**-30.50**			**-24.93**

The results presented in Table 5.4 are segregated into two categories: Fully Connected DNN Results and 50% Pruned DNN Results. This forking aligns with the earlier discussion in the Proposed Method section regarding the impact of pruning on communication efficiency in NoC architectures. The comparison between HM and SA for these extensive benchmarks reveals intriguing insights. The percentage change (%Δ SA/HM) column quantitatively illustrates the relative performance of SA against HM.

For fully connected networks, the SA method demonstrates a noticeable reduction in communication weight compared to HM. This finding is in line with the hypothesis presented in the Proposed Method section, where SA's capability to navigate complex solution spaces efficiently was highlighted. The reduction in communication weight is more pronounced in the 50% pruned DNN results, complying the effectiveness of pruning in reducing the complexity of neuron grouping tasks and enhancing the performance of metaheuristic methods like SA.

These outcomes are pivotal for understanding the scalability of different neuron grouping approaches in the context of large DNN models. They provide empirical evidence supporting the theoretical discussions around the suitability of metaheuristic methods like SA for large-scale optimization problems in NoC-based DNN accelerators. The results also suggest a direction for future research, particularly in improving heuristic methods or developing new metaheuristic strategies that can handle the growing complexity of modern DNNs more efficiently.

The experiments conducted for neuron grouping aimed at minimizing communication weight among neuron clusters while considering the computational load distribution on processing elements, with a focus on $\delta = 1.0$. The outcomes for both fully connected DNNs (0% pruning) and 50% pruned DNNs are summarized in Tables 5.2 and 5.3. These tables highlight the percentage changes ($\%\Delta$) in communication weight from one method to another, revealing a trend of decrease across different benchmarks.

In fully connected DNNs, both ILP and SA methods consistently demonstrated significant reductions in communication weight compared to HM, with ILP showing a slight advantage over SA. This trend is noticeable across various DNN models with different sizes and complexities. The percentage changes range from slight to substantial, reflecting the complexity and specific characteristics of each benchmark.

For the 50% pruned DNNs, similar patterns are observed. The reduction in communication weight is even more pronounced, indicating the efficacy of both ILP and SA in optimizing neuron grouping in pruned networks. The results underline the potential of pruning as a strategy to enhance the efficiency of NoC-based DNN accelerators.

The normalized results, depicted in Figures 5.2 and 5.3, further illustrate the effectiveness of both ILP and SA in reducing communication weight. These visual representations offer a clear comparison between the methods, with the improvements brought by ILP and SA being evident at a glance.

An interesting observation from the experiments is the performance of ILP and SA in large-scale benchmarks. While ILP is constrained by its computational demand, particularly regarding memory, SA demonstrates a robust performance across all benchmarks. This is evident from the results for large-scale benchmarks in Table 5.4, where SA not only outperforms HM but also presents feasible solutions for all tested models.

These findings highlight the practical implications of using SA for neuron grouping in NoC-based DNN accelerators, especially when dealing with large networks. The scalability and efficiency of SA make it a suitable choice for real-world applications, where computational resources and memory are critical constraints.

In the exploration of neuron grouping methods for DNN accelerators, a critical observation was made in the case of the C3 benchmark, where the Simulated Annealing (SA) method demonstrated superior performance over the Integer Linear Programming (ILP) approach. This outcome can be traced back to the inherent limitations of ILP in dealing with large-scale problems. Specifically, ILP's failure to derive the optimal result within an eight-hour running time constraint underscores its high computational complexity, which becomes increasingly pronounced as the problem size escalates. Consequently, the practicality of employing ILP in large benchmarks is significantly hampered by these computational challenges.

The evaluation of neuron grouping methods in this study adopted a novel approach, focusing on the communication weight of the mesh topology. This perspective diverges from traditional methods that primarily consider the number of neurons. By integrating the communication effects into the grouping strategy, the proposed methods optimize not just the distribution of neurons, but also the processing load across the network. This approach ensures a more balanced and efficient utilization of processing elements, contributing to a reduction in overall system latency and potentially enhancing energy efficiency.

Such a holistic evaluation strategy is particularly relevant in the context of NoC-based DNN accelerators, where the interplay between computational load and communication overhead is crucial. The results from this study, therefore, provide valuable insights into optimizing neuron grouping for DNN accelerators, paving the way for more effective NoC architectures. This is especially pertinent in large-scale DNNs, where traditional methods may fail to address the complex dynamics of neuron interactions and communication pathways.

In summary, the insights gleaned from these experiments offer a comprehensive understanding of neuron grouping dynamics in DNN accelerators. The findings underscore the importance of considering communication weights and load distribution in the grouping process, especially for large-scale applications where computational efficiency and practicality are paramount.

5.3. Node Mapping Results

The experimental section dedicated to node mapping results delves into the implementation and evaluation of the proposed Simulated Annealing (SA) and Integer Linear Programming (ILP)-based methods. These methods, designed for application mapping on Network-on-Chip (NoC) architectures with a 2D-mesh topology, aim to minimize the total communication cost. The logic underpinning the SA and ILP approaches for mapping follows the principles outlined in the earlier sections of this study.

For a comprehensive analysis, all possible combinations of ILP and SA for both the grouping and mapping phases of the problem were examined. These combinations have been precisely tested to determine their effectiveness in reducing communication costs within the NoC framework. The performance of these combinations was then benchmarked against results obtained using a heuristic method (HM-DirX), which serves as a standard comparison in the field.

This accurate testing approach, encompassing various combinations of grouping and mapping strategies, was crucial to providing a holistic understanding of the optimization potential in NoC architectures. The results obtained from these experiments were instrumental in identifying the most efficient methods in terms of communication cost reduction.

By comparing the SA and ILP methods with the HM-DirX approach, the experiments were designed to highlight each method's relative advantages and potential drawbacks. This comparison was especially pivotal in determining the practicality and scalability of the proposed methods for real-world applications.

In the realm of NoC architectures, where the optimization of communication pathways is of predominant importance, these results offer critical insights. They demonstrate the proposed methods' feasibility and provide a benchmark for future improvements and innovations in NoC mapping strategies.

Furthermore, these results contribute significantly to the broader discourse on NoC-based DNN acceleration. By providing experimental evidence on the effectiveness of different mapping strategies, this research aids in the development of more efficient and cost-effective NoC systems. This is particularly relevant as the complexity of DNN models and the demand for more efficient hardware accelerators continue to grow.

In summary, the node mapping results section presents key findings that advance the understanding of application mapping in NoC architectures. It underscores the importance of choosing the right combination of grouping and mapping strategies to optimize communication costs, a critical factor in the performance of NoC-based systems. The comparative analysis with HM-DirX further enriches these insights, offering valuable benchmarks for current and future research in NoC mapping strategies.

In the study, a comprehensive analysis of node mapping strategies for DNNs on mesh-based NoC architectures was undertaken. The approaches encompassed both metaheuristic Simulated Annealing (SA) and Integer Linear Programming (ILP) methodologies. Table 5.5 collates the results, providing insights into the total communication costs incurred under various methodological combinations. These include heuristic mapping (HM-DirX), ILP-based grouping and mapping (ILP-ILP), and combinations of ILP and SA in both grouping and mapping phases (ILP-SA, SA-ILP, SA-SA). The analysis spans different benchmarks, offering a comparative view under two specific scenarios: Fully Connected DNN Mapping Results and 50% Pruned DNN Mapping Results.

This table format is instrumental in evaluating the effectiveness of each method in reducing communication costs, a critical factor in the performance of NoC systems. The data present a clear comparison between heuristic methods and the proposed ILP and SA approaches, underlining the superior performance of the latter in most cases.

The detailed assessment presented in Table 5.5 is critical in understanding the nuances of node mapping in DNN accelerators. It provides a clear demonstration of how strategic methodological choices can significantly influence the efficiency of NoC systems. The results reinforce the concept that a well-optimized node mapping strategy, particularly in

Table 5.5 Communication cost results for mapping of DNNs onto mesh-based NoCs.

	Fully Connected DNN Mapping Results					50% Pruned DNN Mapping Results				
Benchmark	**HM-DirX**	**ILP-ILP**	**ILP-SA**	**SA-ILP**	**SA-SA**	**HM-DirX**	**ILP-ILP**	**ILP-SA**	**SA-ILP**	**SA-SA**
B1 (3x3)	145	79	79	77	77	85	25	25	41	41
B2 (3x3)	125	62	62	63	63	81	44	44	44	44
B3 (3x3)	150	107	107	111	111	120	60	60	60	60
B4 (3x3)	71	41	41	54	54	61	32	32	33	33
C1 (3x3)	344	177	177	206	206	306	152	152	154	154
C2 (4x4)	1224	436	436	600	586	949	373	373	578	561
C3 (4x4)	1406	778	778	776	776	1213	685	681	631	625

large-scale DNN models, is imperative for achieving reduced communication costs and, consequently, enhanced overall system performance.

The effectiveness of these methodologies is visually represented in Figures 5.4 and 5.5, where the results from Table 5.5 are normalized for a more comprehensible comparison. It is evident from these graphical analyses that the combinations of grouping and mapping strategies devised in this research consistently surpass the performance of the HM-DirX benchmark across all evaluated scenarios.

A notable trend is the superior efficacy of configurations where ILP-based grouping is integrated with either SA or ILP mapping methodologies. These combinations consistently demonstrate enhanced performance in comparison to those involving SA-based grouping. This pattern underscores the robustness of ILP in constructing optimal groupings, which subsequently contributes to more efficient mapping outcomes. However, the limitations of ILP, particularly in terms of computational time, become apparent in more extensive benchmarks. This is illustrated in the 50% pruned C3 benchmark, as depicted in Figure 5.5, where ILP fails to converge to an optimal result within the stipulated eight-hour timeframe.

In scenarios involving larger benchmarks, such as the C3 model, SA emerges as a more viable and effective solution. The flexibility and scalability of SA, especially under the constraints of time and computational resources, make it an advantageous approach for large-scale applications. This finding aligns with the earlier observations in the neuron grouping experiments, reinforcing the notion that SA's adaptability to diverse and complex problem spaces makes it a robust choice for NoC-based DNN accelerator optimization.

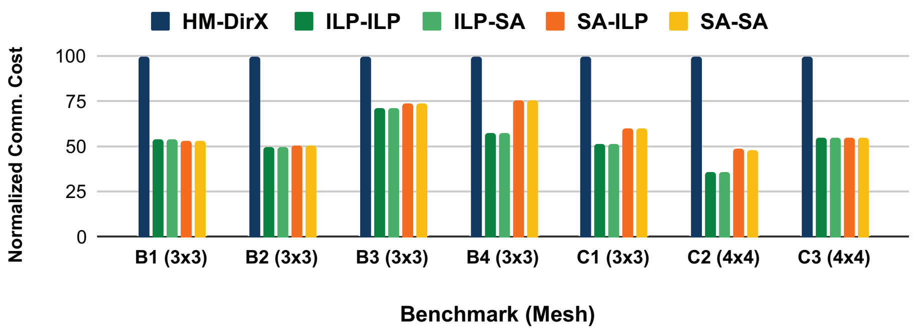

Figure 5.4 Communication cost results for mapping of fully connected DNNs (normalized to $[0, 100]$).

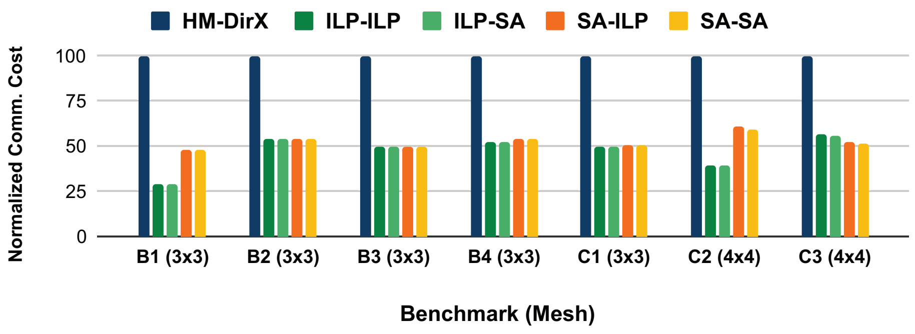

Figure 5.5 Communication cost results for mapping of 50% pruned DNNs (normalized to $[0, 100]$).

In Table 5.6, the relative performance enhancements offered by the SA and ILP methodologies over the heuristic HM-DirX approach are quantified. Notably, when different mapping strategies are applied in conjunction with identical grouping techniques, the observed variance in performance is marginal, approximately 0.2%. This subtle deviation underscores the efficacy of the SA-based mapping approach that has been developed. Conversely, a more pronounced improvement, approximately 6%, is observed when ILP grouping is employed in tandem with a consistent mapping strategy. This finding highlights the superior efficiency of the ILP grouping method in this context.

Table 5.6 Percentage improvements of communication load when SA and ILP methods are used versus heuristic HM-dirX.

Pruning Percentage	Grouping: ILP Mapping: ILP	Grouping: ILP Mapping: SA	Grouping: SA Mapping: ILP	Grouping: SA Mapping: SA
0% Pruned Benchmarks	46.35%	46.35%	40.34%	40.51%
50% Pruned Benchmarks	52.63%	52.68%	47.17%	47.49%

As delineated in Table 5.6, the amalgamation of the proposed methods manifests a substantial performance boost, ranging from 40% to 50%, compared to established heuristic techniques. More specifically, the results for benchmarks with 50% pruning exhibit an approximate 7% enhancement over those of fully connected networks. This finding aligns with the expectations and is attributable to the inherent characteristics of the ILP and SA grouping methodologies. The differential performance between pruned and fully connected benchmarks accentuates the adaptability and effectiveness of the methods in varying network configurations.

5.3.1. Analysis of the Proposed Methods on Scaled and Pruned CNN Architectures: Insights from LeNet and AlexNet Evaluation

Convolutional Neural Networks (CNNs), which are at the leading edge of deep learning advancements, have been instrumental in revolutionizing the field of computer vision. Unlike traditional neural network architectures, CNNs are designed with a specific focus on processing and learning from the complex spatial hierarchies that are inherent in image data. Prominent examples of CNNs, such as LeNet and AlexNet, are tailored to efficiently handle tasks such as image recognition and classification, demonstrating their specialized capabilities in these areas.

LeNet, developed in the 1990s, was a pioneering model primarily focused on recognizing handwritten digits. It set the groundwork for subsequent, more sophisticated models. AlexNet, introduced in 2012, represented a significant evolution in the field with its deeper structure and expanded parameter set, leading to a remarkable level of accuracy on benchmark image datasets. To adapt these architectures to computational limitations and to facilitate a thorough experimental analysis, both were subject to scaling and pruning adjustments. Specifically, LeNet underwent a scale-down by a factor of 16, while AlexNet was scaled down by a factor of 512. In addition, versions of these models with 50% pruning were evaluated with care to assess the balance between network complexity and computational efficiency. Table 5.7 encapsulates a comparative overview of these benchmark models.

The decision to scale down these architectures was not merely a response to computational constraints but also a deliberate strategy to investigate the behavior of CNNs in resource-limited settings. This process was conducted with careful consideration of the distinctive neural connectivity patterns in CNNs, which are notably different from those in traditional Artificial Neural Networks (ANNs). This approach was vital to preserve the structural and functional essence of the original designs.

Table 5.7 CNN benchmarks.

Benchmark Name	Scale-Down Factor	ID	Neurons in Layers	Total Neurons
LeNet	16	B6	294-74-100-25-8-6-10	517
AlexNet	512	B7	301-567-136-364-84-126-126-84-18-8-8-1	1823

The use of Integer Linear Programming (ILP) methods, commonly employed in network optimization, was found to be impractical for the scaled dimensions of the study, primarily due to their inherent computational intensity. As a result, the methodology shifted towards employing Simulated Annealing (SA) for the network mapping process. This strategic decision was influenced by the ability of SA to effectively explore extensive solution spaces, a quality that remains advantageous even in scaled-down scenarios.

In the experimental analysis, as depicted in Table 5.8 and Figure 5.6, a comparative assessment of communication costs using various mapping strategies on CNN benchmarks was conducted. These benchmarks included both fully connected and 50% pruned network configurations. The study considered two distinct grouping-mapping methodologies: HM-SA and SA-SA.

For the fully connected networks, the findings indicate a substantial reduction in communication costs using the SA-SA mapping methodology, especially when compared to the HM-SA mapping. This trend is exemplified in the B6 benchmark, where the SA-SA mapping significantly outperforms the HM-SA approach. A similar pattern is observed in the B7 benchmark, with the SA-SA mapping reducing costs by nearly half relative to

Table 5.8 Communication cost results for mapping of CNNs onto mesh-based NoCs.

	Fully Connected Mapping Results		**50% Pruned Mapping Results**	
Benchmark	**HM-SA**	**SA-SA**	**HM-SA**	**SA-SA**
B6 (6x6)	584	192	422	192
B7 (9x9)	11133	5629	9856	5377

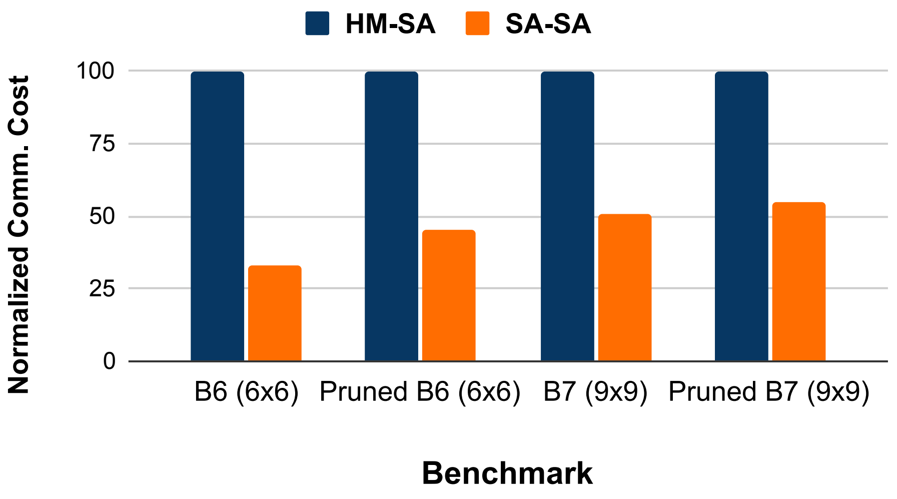

Figure 5.6 Communication cost results for mapping of CNNs (normalized to $[0, 100]$).

HM-SA. In the context of the 50% pruned networks, the communication cost for benchmark B6 remains consistent for the SA-SA mapping but exhibits a notable decrease under the HM-SA approach. Conversely, for the larger B7 benchmark, both mapping methodologies demonstrate a significant reduction in communication costs.

These results underscore the superiority of the SA-SA mapping approach in optimizing communication costs, particularly in more complex network benchmarks. Furthermore, the data highlights the impact of network pruning on communication efficiency, with a marked cost reduction in the HM-SA mapping approach for pruned networks.

6. CONCLUSION

In this study, an enhanced model incorporating Simulated Annealing (SA) for neuron grouping and mapping, along with a heuristic method for neuron grouping, has been introduced. Additionally, an Integer Linear Programming (ILP) approach was employed for comparative analysis. All methodologies have been customized for Deep Neural Network (DNN) accelerators within 2D-mesh-based Network-on-Chip (NoC) architectures, emphasizing the reduction of communication load as a primary objective. This model distinguishes between the communication load among individual neurons, indicative of single-neuron processing, and that among neuron clusters, representing processing within a group of neurons.

Existing neuron grouping heuristics, commonly referenced in the literature, were enhanced to aim for an equitable distribution of workload across the chip. This enhancement was integrated with the well-established Dir-X NoC mapping method to formulate the final model. The proposed SA-based approaches are designed to minimize communication costs while ensuring a balanced processing load like ILP. While the ILP methods yield optimal results, their computational and spatial demands limit their practicality for larger benchmarks and IoT devices. However, they serve as a valuable baseline for evaluating the metaheuristic approaches on smaller-scale benchmarks. As evidenced by the experiments, the SA-based methods deliver near-optimal results swiftly, with minimal performance discrepancies compared to the ILP outcomes, irrespective of the benchmark size. On average, the approaches led to a 40-50% improvement in communication cost efficiency.

The methodologies proposed are adaptable to any mesh-based NoC architecture used in DNN acceleration. Future research will extend to developing grouping and mapping strategies for other NoC topologies, particularly those with irregular, application-specific designs. Additionally, there is a plan to tailor these methods to suit various types of neural networks.

REFERENCES

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, **2016**.

[2] Ajeet Ram Pathak, Manjusha Pandey, and Siddharth Rautaray. Application of deep learning for object detection. *Procedia Computer Science*, 132:1706–1717, **2018**. ISSN 1877-0509. doi:https://doi.org/10.1016/j.procs.2018.05.144. International Conference on Computational Intelligence and Data Science.

[3] Shashi Pal Singh, Ajai Kumar, Hemant Darbari, Lenali Singh, Anshika Rastogi, and Shikha Jain. Machine translation using deep learning: An overview. In *2017 International Conference on Computer, Communications and Electronics (Comptelix)*, pages 162–167. **2017**. doi:10.1109/COMPTELIX.2017.8003957.

[4] Nhan Cach Dang, María N. Moreno-García, and Fernando De la Prieta. Sentiment analysis based on deep learning: A comparative study. *Electronics*, 9(3), **2020**. ISSN 2079-9292. doi:10.3390/electronics9030483.

[5] Geert Litjens, Thijs Kooi, Babak Ehteshami Bejnordi, Arnaud Arindra Adiyoso Setio, and Et al. Ciompi. A survey on deep learning in medical image analysis. *Medical image analysis*, 42:60–88, **2017**.

[6] Thibaut Théate and Damien Ernst. An application of deep reinforcement learning to algorithmic trading. *Expert Systems with Applications*, 173:114632, **2021**. ISSN 0957-4174. doi:https://doi.org/10.1016/j.eswa.2021.114632.

[7] Louis Marceau, Lingling Qiu, Nick Vandewiele, and Eric Charton. A comparison of deep learning performances with other machine learning algorithms on credit scoring unbalanced data. **2020**. doi:1907.12363.

[8] Jiankun Wang, Tianyi Zhang, Nachuan Ma, Zhaoting Li, Han Ma, Fei Meng, and Max Q-H Meng. A survey of learning-based robot motion planning. *IET Cyber-Systems and Robotics*, 3(4):302–314, **2021**.

[9] Ratheesh Ravindran, Michael J Santora, and Mohsin M Jamali. Multi-object detection and tracking, based on dnn, for autonomous vehicles: A review. *IEEE Sensors Journal*, 21(5):5668–5677, **2020**.

[10] Amrita S Tulshan and Sudhir Namdeorao Dhage. Survey on virtual assistant: Google assistant, siri, cortana, alexa. In *International symposium on signal processing and intelligent recognition systems*, pages 190–201. Springer, **2019**.

[11] Ali Bou Nassif, Ismail Shahin, Imtinan Attili, Mohammad Azzeh, and Khaled Shaalan. Speech recognition using deep neural networks: A systematic review. *IEEE access*, 7:19143–19165, **2019**.

[12] Weiting Zhang, Dong Yang, and Hongchao Wang. Data-driven methods for predictive maintenance of industrial equipment: A survey. *IEEE Systems Journal*, 13(3):2213–2227, **2019**. doi:10.1109/JSYST.2019.2905565.

[13] Javier Villalba-Diez, Daniel Schmidt, Roman Gevers, and Et al. Ordieres-Meré. Deep learning for industrial computer vision quality control in the printing industry 4.0. *Sensors*, 19(18), **2019**. ISSN 1424-8220. doi:10.3390/s19183987.

[14] Udit Gupta, Carole-Jean Wu, Xiaodong Wang, Maxim Naumov, Brandon Reagen, and Et al. Brooks. The architectural implications of facebook's dnn-based personalized recommendation. In *2020 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 488–501. **2020**. doi:10.1109/HPCA47549.2020.00047.

[15] Vivienne Sze, Yu-Hsin Chen, Tien-Ju Yang, and Joel S Emer. Efficient processing of deep neural networks: A tutorial and survey. *Proceedings of the IEEE*, 105(12):2295–2329, **2017**. doi:1703.09039.

[16] Jinsu Lee, Sanghoon Kang, Jinmook Lee, Dongjoo Shin, and Et al. Han. The hardware and algorithm co-design for energy-efficient dnn processor on edge/mobile devices. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 67(10):3458–3470, **2020**.

[17] Ananda Samajdar, Jan Moritz Joseph, Yuhao Zhu, and Et al. Whatmough. A systematic methodology for characterizing scalability of dnn accelerators using scale-sim. In *2020 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, pages 58–68. IEEE, **2020**.

[18] William J Dally, Yatish Turakhia, and Song Han. Domain-specific hardware accelerators. *Communications of the ACM*, 63(7):48–57, **2020**. doi:10.1145/3361682.

[19] Jose Duato, Sudhakar Yalamanchili, and Lionel Ni. *Interconnection networks*. Morgan Kaufmann, **2003**.

[20] L. Benini and G. De Micheli. Networks on chips: a new soc paradigm. *Computer*, 35(1):70–78, **2002**. doi:10.1109/2.976921.

[21] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, and Et al. Millberg. A network on chip architecture and design methodology. In *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*, pages 117–124. **2002**. doi:10.1109/ISVLSI.2002.1016885.

[22] Tobias Bjerregaard and Shankar Mahadevan. A survey of research and practices of network-on-chip. 38(1):1–es, **2006**. ISSN 0360-0300. doi:10.1145/1132952.1132953.

[23] Md Farhadur Reza and Paul Ampadu. Energy-efficient and high-performance noc architecture and mapping solution for deep neural networks. In *Proceedings of the 13th IEEE/ACM International Symposium on Networks-on-Chip*, NOCS '19. Association for Computing Machinery, New York, NY, USA, **2019**. ISBN 9781450367004. doi:10.1145/3313231.3352377.

[24] Norman P. Jouppi, Cliff Young, Nishant Patil, David Patterson, Gaurav Agrawal, Raminder Bajwa, and Et al. Bates. In-datacenter performance analysis of a tensor processing unit. In *Proceedings of the 44th Annual International Symposium on Computer Architecture*, ISCA '17, page 1–12. Association for Computing

Machinery, New York, NY, USA, **2017**. ISBN 9781450348928. doi:10.1145/3079856.3080246.

[25] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence*, AAAI'17, page 4278–4284. AAAI Press, **2017**. doi:10.5555/3298023.3298188.

[26] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778. **2016**. doi:10.1109/CVPR.2016.90.

[27] Song Han, Xingyu Liu, Huizi Mao, Jing Pu, Ardavan Pedram, Mark A. Horowitz, and William J. Dally. Eie: Efficient inference engine on compressed deep neural network. *SIGARCH Comput. Archit. News*, 44(3):243–254, **2016**. ISSN 0163-5964. doi:10.1145/3007787.3001163.

[28] Seyedeh Yasaman Hosseini Mirmahaleh and Amir Masoud Rahmani. Dnn pruning and mapping on noc-based communication infrastructure. *Microelectronics Journal*, 94:104655, **2019**. ISSN 0026-2692. doi:https://doi.org/10.1016/j.mejo.2019.104655.

[29] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. *Commun. ACM*, 60(6):84–90, **2017**. ISSN 0001-0782. doi:10.1145/3065386.

[30] Yu-Hsin Chen, Tien-Ju Yang, Joel Emer, and Vivienne Sze. Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 9(2):292–308, **2019**.

[31] Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, **2015**. doi:10.1145/3065386.

[32] Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Quantized neural networks: Training neural networks with low precision weights and activations. *The Journal of Machine Learning Research*, 18(1):6869–6898, **2017**. doi:1609.07061.

[33] Tailin Liang, John Glossner, Lei Wang, Shaobo Shi, and Xiaotong Zhang. Pruning and quantization for deep neural network acceleration: A survey. *Neurocomputing*, 461:370–403, **2021**. ISSN 0925-2312. doi:https://doi.org/10.1016/j.neucom.2021.07.045.

[34] Adam Santoro, Sergey Bartunov, Matthew M. Botvinick, Daan Wierstra, and Timothy P. Lillicrap. One-shot learning with memory-augmented neural networks. *CoRR*, abs/1605.06065, **2016**. doi:1605.06065.

[35] Snaider Carrillo, Jim Harkin, Liam J McDaid, Fearghal Morgan, Sandeep Pande, Seamus Cawley, and Brian McGinley. Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations. *IEEE Transactions on Parallel and Distributed Systems*, 24(12):2451–2461, **2012**.

[36] Ali Yasoubi, Reza Hojabr, Hengameh Takshi, Mehdi Modarressi, and Masoud Daneshtalab. Cupan – high throughput on-chip interconnection for neural networks. In *Neural Information Processing*, pages 559–566. Springer International Publishing, Cham, **2015**. ISBN 978-3-319-26555-1. doi:10.1007/978-3-319-26555-1_63.

[37] Hardik Sharma, Jongse Park, Naveen Suda, Liangzhen Lai, and Et al. Chau. Bit fusion: Bit-level dynamically composable architecture for accelerating deep neural network. In *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, pages 764–775. **2018**. doi:10.1109/ISCA.2018.00069.

[38] Kun-Chih (Jimmy) Chen, Masoumeh Ebrahimi, Ting-Yi Wang, and Yuch-Chi Yang. Noc-based dnn accelerator: A future design paradigm. In *Proceedings of*

the 13th IEEE/ACM International Symposium on Networks-on-Chip. Association for Computing Machinery, New York, NY, USA, **2019**. ISBN 9781450367004. doi:10.1145/3313231.3352376.

[39] A. Jenasamanta and S. Mohapatra. Classification of juvenile delinquency using bayesian network learning: a comparative analysis. *Atlantis Highlights in Social Sciences, Education and Humanities*, **2022**. doi:10.2991/ahsseh.k.220105.013.

[40] X. Duan and P. Hou. Research on teaching quality evaluation model of physical education based on simulated annealing algorithm. *Mobile Information Systems*, 2021:1–8, **2021**. doi:10.1155/2021/4407512.

[41] R. Sivapuram and R. Picelli. Topology optimization of binary structures using integer linear programming. *Finite Elements in Analysis and Design*, 139:49–61, **2018**. ISSN 0168-874X. doi:https://doi.org/10.1016/j.finel.2017.10.006.

[42] Stratos Papadomanolakis and Anastassia Ailamaki. An integer linear programming approach to database design. In *2007 IEEE 23rd International Conference on Data Engineering Workshop*, pages 442–449. **2007**. doi:10.1109/ICDEW.2007.4401027.

[43] Xiaoxiao Liu, Wei Wen, Xuehai Qian, Hai Li, and Yiran Chen. Neu-noc: A high-efficient interconnection network for accelerated neuromorphic systems. In *23rd Asia and South Pacific Design Automation Conference*, pages 141–146. **2018**. doi:10.1109/ASPDAC.2018.8297296.

[44] Kun-Chih Chen and Ting-Yi Wang. Nn-noxim: High-level cycle-accurate noc-based neural networks simulator. In *2018 11th International Workshop on Network on Chip Architectures (NoCArc)*, pages 1–5. **2018**. doi:10.1109/NOCARC.2018.8541173.

[45] Kun-Chih Jimmy Chen, Masoumeh Ebrahimi, Ting-Yi Wang, Yuch-Chi Yang, and Yuan-Hao Liao. A noc-based simulator for design and evaluation of deep neural networks. *Microprocessors and Microsystems*, 77:103145, **2020**.

[46] Yao Xiao, Yuankun Xue, Shahin Nazarian, and Paul Bogdan. A load balancing inspired optimization framework for exascale multicore systems: A complex networks approach. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 217–224. **2017**. doi:10.1109/ICCAD.2017.8203781.

[47] Kun-Chih Jimmy Chen, Chun-Chuan Wang, Cheng-Kang Tsai, and Jing-Wen Liang. Dynamic mapping mechanism to compute dnn models on a resource-limited noc platform. In *2021 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*, pages 1–4. **2021**. doi:10.1109/VLSI-DAT52063.2021.9427320.

[48] Ali Yasoubi, Reza Hojabr, and Mehdi Modarressi. Power-efficient accelerator design for neural networks using computation reuse. *IEEE Computer Architecture Letters*, 16(1):72–75, **2017**. doi:10.1109/LCA.2016.2521654.

[49] Arash Firuzan, Mehdi Modarressi, and Masoud Daneshtalab. Reconfigurable communication fabric for efficient implementation of neural networks. In *2015 10th International Symposium on Reconfigurable Communication-centric Systems-on-Chip (ReCoSoC)*, pages 1–8. **2015**. doi:10.1109/ReCoSoC.2015.7238097.

[50] Jan Moritz Joseph, Murat Sezgin Baloglu, and Et al. Pan. Newromap: Mapping cnns to noc-interconnected self-contained data-flow accelerators for edge-ai. In *Proceedings of the 15th IEEE/ACM International Symposium on Networks-on-Chip*, NOCS '21, page 15–20. Association for Computing Machinery, New York, NY, USA, **2021**. ISBN 9781450390835. doi:10.1145/3479876.3481591.

[51] Seyedeh Yasaman Hosseini Mirmahaleh, Midia Reshadi, and Et al. Shabani. Flow mapping and data distribution on mesh-based deep learning accelerator. In *Proceedings of the 13th IEEE/ACM International Symposium*

on Networks-on-Chip, NOCS '19. Association for Computing Machinery, New York, NY, USA, **2019**. ISBN 9781450367004. doi:10.1145/3313231.3352378.

[52] S. Murali and G. De Micheli. Bandwidth-constrained mapping of cores onto noc architectures. In *Proceedings Design, Automation and Test in Europe Conference and Exhibition*, volume 2, pages 896–901 Vol.2. **2004**. doi:10.1109/DATE.2004.1269002.

[53] Md Farhadur Reza, Dan Zhao, and Hongyi Wu. Task-resource co-allocation for hotspot minimization in heterogeneous many-core nocs. In *2016 International Great Lakes Symposium on VLSI*, pages 137–140. **2016**. doi:10.1145/2902961.2903003.

[54] Salih Bayar and Arda Yurdakul. An efficient mapping algorithm on 2-d mesh network-on-chip with reconfigurable switches. In *2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS)*, pages 1–4. **2016**. doi:10.1109/DTIS.2016.7483808.

[55] Chi-Hong Hwang and Allen C-H Wu. A predictive system shutdown method for energy saving of event-driven computation. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 5(2):226–241, **2000**.

[56] Trevor Pering, Tom Burd, and Robert Brodersen. The simulation and evaluation of dynamic voltage scaling algorithms. In *Proceedings. 1998 International Symposium on Low Power Electronics and Design (IEEE Cat. No. 98TH8379)*, pages 76–81. IEEE, **1998**.

[57] Xin-She Yang. Chapter 1 - introduction to algorithms. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms*, pages 1–21. Elsevier, Oxford, **2014**. ISBN 978-0-12-416743-8. doi:https://doi.org/10.1016/B978-0-12-416743-8.00001-4.

[58] Christian Blum and Andrea Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, **2003**. ISSN 0360-0300. doi:10.1145/937503.937505.

[59] Xin-She Yang. Chapter 3 - random walks and optimization. In Xin-She Yang, editor, *Nature-Inspired Optimization Algorithms*, pages 45–65. Elsevier, Oxford, **2014**. ISBN 978-0-12-416743-8. doi:https://doi.org/10.1016/B978-0-12-416743-8.00003-8.

[60] Dimitris Bertsimas and John Tsitsiklis. Simulated annealing. *Statistical Science*, 8, **1993**. doi:10.1214/ss/1177011077.

[61] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, **1983**. doi:10.1126/science.220.4598.671.

[62] Nicholas Metropolis, Arianna W. Rosenbluth, Marshall N. Rosenbluth, Augusta H. Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The Journal of Chemical Physics*, 21(6):1087–1092, **1953**. doi:10.1063/1.1699114.

[63] Richard J. Wagner Matthew Perry. Python module for simulated annealing. `https://github.com/perrygeo/simanneal`, **2019**.

[64] Gurobi optimizer. `https://www.gurobi.com/`.

[65] Amir Ali. Artificial neural network (ann) with practical implementation. `https://medium.com/machine-learning-researcher/artificial-neural-network-ann-4481fa33d85a`. Accessed: 3-Nov-2022.

[66] Edgar O. Reséndiz-Flores, Gerardo Altamirano-Guerrero, Patricia S. Costa, Antonio E. Salas-Reyes, Armando Salinas-Rodríguez, and Frank Goodwin. Optimal design of hot-dip galvanized dp steels via artificial neural networks and

multi-objective genetic optimization. *Metals*, 11(4), **2021**. ISSN 2075-4701. doi:10.3390/met11040578.

[67] Jiashuai Wang, Zhe Li, Shaocun Yan, Xue Yu, Yanqing Ma, and Lei Ma. Modifying the microstructure of algae-based active carbon and modelling supercapacitors using artificial neural networks. *RSC Adv.*, 9:14797–14808, **2019**. doi:10.1039/C9RA01255A.

[68] Sugiarto, Indar and Pasila, Felix. Understanding a deep learning technique through a neuromorphic system a case study with spinnaker neuromorphic platform. *MATEC Web Conf.*, 164:01015, **2018**. doi:10.1051/matecconf/201816401015.

[69] Marc Riera, José María Arnau, and Antonio González. Dnn pruning with principal component analysis and connection importance estimation. *J. Syst. Archit.*, 122(C), **2022**. ISSN 1383-7621. doi:10.1016/j.sysarc.2021.102336.