Algorithms and Architectures for Spiking Neuromorphic Systems

Vu Huy The

A DISSERTATION
SUBMITTED IN FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF DOCTOR OF PHILOSOPHY
IN COMPUTER SCIENCE AND ENGINEERING

Graduate Department of Computer and Information Systems
The University of Aizu
2019

to Advance Knowledge for Humanity
THE UNIVERSITY OF AIZU
· 1993 ·

The thesis titled

Algorithms and Architectures for Spiking Neuromorphic Systems

by

Vu Huy The

is reviewed and approved by:

Chief referee
Professor, The University of Aizu, Japan
Abderazek Ben Abdallah
Ben Abdallah

Professor, The University of Aizu, Japan
Toshiaki Miyazaki
T. Miyazaki
宮崎

Professor, The University of Aizu, Japan
Tsuneo Tsukahara
T. Tsukahara

Professor, The University of Aizu, Japan
Junji Kitamichi
Junji Kitamichi
北道

Professor, Keio University, Japan
Hideharu Amano
天野 英晴

The University of Aizu

2019

Contents

List of Figures

List of Tables

List of Abbreviations

2D-NoC	Two dimensional Network-on-Chip
3D-IC	Three dimensional Integrated Circuit
3D-NoC	Three dimensional Network-on-Chip
3D-SIC	Three dimensional Stacked Integrated Circuit
ASIC	Application-Specific Integrated Circuit
ANN	Artificial Neural Network
BLoD	Bypass-Link-on-Demand
BC	Broadcast
CPU	Central Processing Unit
CT	Crossbar Traversal stage
DRAM	Dynamic Random Access Memory
ECC	Error Correction Codes
FIFO	First-In-First-Out
FT-KMCR	Fault-Tolerant K-means based MultiCast Routing algorithm
FTSP-KMCR	Fault-Tolerant Shortest-Path KMCR
HDL	Hardware Description Language
HF	Hopfield neural network
IC	Integrated Circuit
KMCR	K-means based MultiCast Routing algorithm
MC	Multicast
NoC	Network-on-Chip
PE	Processing Element
RAB	Random-Access-Buffer
RC	Routing Computation stage
RNDC	Randomly Connected Neural Network
RTL	Register-Transfer Level

SA	Switch Allocation stage
SRAM	Static Random Access Memory
SNN	Spiking Neural Network
SNPC	Spiking Neural Processing Core
SoC	System-on-Chip
SP-KMCR	Shortest- Path K-means based MultiCast Routing algorithm
TSV	Through Silicon Via
UC	Unicast

To my wife, my parent, and my family, with all my love

Acknowledgments

First, I would like to convey my deepest gratitude to my supervisor Prof. Abderazek Ben Abdallah for his guidance, support, and encouragement. Also, I would like to thank Prof. Toshiaki Miyazaki, Prof. Tsuneo Tsukahara, Prof. Junji Kitamichi of The University of Aizu and Prof. Hideharu Amano of Keio University for taking the time to revise my thesis. Moreover, my sincere gratitude to Prof. Yuichi Okuyama for his help and support during the past three years.

Aizu is really memorable for me because of my great friends and colleagues. I am thankful to Dr. Akram Ben Ahmed of Keio University and Dr. Dang Nam Khanh of Vietnam National University for their significant help and discussion. I want to thank all the members of the Adaptive Systems Laboratory and my friends at The University of Aizu. Their supportive words and encouraging messages kept me motivated to work harder and be a better researcher and person. Not to forget to appreciate the staff of The University of Aizu for their assistance.

My endless love goes to my parent and family who always endow me with infinite support and unconditional love. Last but not least, my most heartfelt thanks are due to my dearest stunning wife, *Hien-san*. She always stood by me and endowed me with her endless love and support.

Vu Huy The,
September 2019,
Aizu, Japan

Thesis advisor: Professor Abderazek Ben Abdallah Vu Huy The

Algorithms and Architectures for Spiking Neuromorphic Systems

Abstract

Inventing the powerful machine like the human brain has been a driving force in computing for decades. The von Neumann architecture has been considered to be a clear standard for such the system. However, the significant differences in the organization, power consumption requirements, and the computational power of von Neumann architecture compared to a biological brain leads to creating alternative architectures. Brain-inspired computing or neuromorphic computing is a biologically inspired approach created from highly connected neurons to not only model neuroscience theories but also solve machine learning problems. The term neuromorphic was first introduced by Carver Mead in 1990, where it referred to very large scale integration (VLSI) with analog components to mimic biological neural systems.

In recent years, artificial neural networks (ANNs) with efficient learning methods (e.g., backpropagation) have shown a remarkable improvement in terms of accuracy (even better human-level) for large-scale visual/auditory recognition and classification tasks. Particularly, the convolution neural network (CNN) and recurrent neural network (RNN) have shown promising tools for a wide range of applications such as image, video, and speech. To reach considerable achievement, state-of-the-art neural networks, however, tend to deeply increase their number of layers and size (i.e., deep learning). Consequently, they require hardware platforms with a huge amount of computation as well as power consumption. On the other hand, spiking neural networks (SNNs) was proposed to not only mimic efficiently the behavior of biological neurons but also make neuromorphic systems extremely power-efficient with tens of pJ per connection.

However, implementing a scalable interneuron communication architecture is one of the major challenges for hardware-based SNNs. The architecture is required to maintain a huge amount of traffic created from a massive number of neurons and synapses accommodated on neural computation units. Furthermore, since the arrival time of spikes is used to encode the information, timing violation in such communication architecture affects the overall performance of SNNs. A shared bus as a communication medium is a poor choice for implementing a large-scale complex SNN chip/system because adding neurons decreases the communication capacity of the chip and may affect the neuron's firing rate due

to increasing length of the shared bus. Moreover, the nonlinear increase in neural connectivity is too significant to be directly implemented using a dedicated point-to-point communication scheme. Two-dimensional packet-switched network-on-chip (2D-NoC) has been considered as a potential solution to deal with the interconnection problems found in previously proposed shared communication medium based SNNs. However, such interconnect strategies make it difficult to achieve a high level of parallelism and scalability with low power consumption, especially in large-scale SNN chips.

We also consider three-dimensional network on chips (3D-NoCs) which take advantage of 3D Integrated Circuits (3D-ICs) and mesh-based network on chip (NoCs) opening a promising architecture for SNNs. They offer scalability and parallelism of NoCs that are enhanced in the third dimension thanks to the short wire length and the low power consumption of 3D-ICs interconnects. Consequently, 3D-NoCs are considered to be one of the most advanced and suitable for SNN systems, with capabilities of extremely high bandwidth, efficient scalability, and low power. However, to take the advantages for SNNs, 3D-NoC demands an efficient multicast routing algorithm to deal with a high traffic pattern where a presynaptic neuron sends spikes to a subset of postsynaptic ones. Furthermore, due to the complex nature of 3D-ICs and the continuing shrinkage of the semiconductor components, 3D-NoC based systems are becoming susceptible to a variety of faults. Especially in SNNs, when connections are faulty, the post-synaptic neuron becomes silent or near-silent (i.e, firing rate reduction). This may degrade overall system performance.

Starting from the facts mentioned above, this dissertation proposes algorithms and architectures for spiking neural network systems based on 3D-NoC (3DNoC-SNN). First, a performance assessment for 3DNoC-SNN is presented to analyze the system performance with different spiking neural network topologies, spike routing methods (i.e., unicast, multicast and broadcast), and in both with and without faults occurring in the system. This analytical model aims to early analyze the system architecture before actual implementation. Second, this dissertation proposes novel multicast spike routing algorithms which are a combination of k-means clustering and tree-based routing method. Adopting k-means is as a partition method helping to get overall balanced traffic and then improve system performance. Moreover, a fault-tolerant multicast routing algorithm is also proposed to deal with connection faults in the system, in which primary and backup routing paths are pre-defined. When faults appear in the primary route, routers switch incoming spike packages via the backup path. This reduces recovery overhead, average latency, and enables the system to avoid timing violation of SNNs. Finally, architecture and hardware design and evaluation of the proposed 3DNoC-SNN system are presented to evaluate the proposed works, as well as compare with the analytical model.

スパイキングニューロモルフィックシステムのためのアルゴリズム及びアーキテクチャ

概要

数十年にわたり、人間の脳のような強力な計算機を発明することがコンピュータの分野においての原動力とされてきた。フォン・ノイマン型アーキテクチャは、これらのようなシステムにおいて、明らかな基準とされている。しかし、その構成における重大な違いである、電力消費量、生物の脳に比べたフォン・ノイマン型アーキテクチャの計算能力は、新たなアーキテクチャの創出につながった。脳に想起された、もしくは、脳の構造を模した計算システムという新たな計算手段は、高度に結びついている神経細胞から創出され、人間の脳構造を模した理論を形成するだけでなく、機械学習における問題を解くことにつながっている。”Neuromorphic”という専門用語は１９９０年にCarver Meadにより最初につくられたものであり、それはアナログな部品を付帯した超大規模集積回路(VLSI)による生物的神経細胞システムの模倣を指したことばからきている

近年では、人工神経細胞ネットワーク(ANNs)と、誤差逆伝搬法のような効率的な学習手法が、大規模な視覚・聴覚的認識と分類において、精度の観点から、顕著な功績を示しており、それはときに人間のレベルを凌駕することもあった。特に、畳み込みニューラルネット(CNN)と再帰型ニューラルネットは、画像、動画、音声のような幅広い分野における有望なツールとしての成果を示している。著しい成果、最先端なニューラルネットに達するとき、そこには深層ニューラルネットとよばれるような、深く増加された層や大きさのネットワークが形成される。結果として、それらは大規模な計算量と消費電力を必要とするハードウェアプラットフォームを必要とする。一方で、スパイキングニューラルネット(SNN)は、生物の神経細胞を効率的に模倣するだけでなく、非常に電力効率の良い脳の構造を模したシステムの構成（一つの結びつきにつき、数十 pJ 程度）を可能にする。

しかしながら、拡張可能な神経細胞の通信アーキテクチャを実装することは、ハードウェアを基盤とした SNN の実装における大きな課題となっている。そのアーキテクチャは、膨大なニューロンとその計算に用いられる接続部における、大規模な通信網の制御性を維持することが必要とされしまう。さらに、スパイクの到達時間はデータの加工に用いられ、タイミング違反は SNN の処理全体に影響を与えてしまう。通信手段としての共有バスは、ニューロン数を増やすことは通信容量を減少させることにつながるため、大規模で複雑な SNN 回路/システムの実装において乏しい選択であり、共有バスの長さを増加させることからそれはニューロンの発火率に影響するとされる。さらに、ニューロンの接続における非線形的増加はとても著しく、ポイント・ツー・ポイント型の通信に適用されるような直接の接続は実装できない。二次元パケットスイッチ型ネットワーク・オン・チップ（NoC）は、先で述べられたような共有バスを媒体とした SNN の実装おける相互通信問題に対する潜在的な解決策として考えられている。しかし

ながら、そのような相互通信における戦略は、高い並列性と拡張性および大規模な SNN 回路における低消費電力を獲得する上で大きな困難を要することにつながる。

私たちは、SNN において有望なアーキテクチャとされる３次元階層における集積に利点を持つ３次元ネットワーク・オン・チップについても考えます。それらは、３次元化することにより縮小されたワイヤーのおかげで、ネットワーク・オン・チップにおいての拡張性と並列性を提供する。結果として、非常に高い帯域幅と効率的な拡張性、低消費電力により、３次元型 NoC はもっとも SNN のシステムにおいて最も適しているとされるものの中の一つである。しかしながら、SNN においてこのような利点を受けるために、３次元型 NoC は、高度な交通形態に対処するための、効率的なマルチキャストルーティングアルゴリズムを必要とする。さらに、３次元型 NoC の複雑性と継続的なセミコンダクタ部品の縮小により、３次元型 NoC を元ととしたシステムは様々な欠陥に対して影響を受けやすくなってしまう。特に SNN において、接続に欠陥が生じた場合、ポストシナプティックなニューロンは、発火率の現象に見られるような、静もしくはほとんど静な状態になってしまう。これはシステム全体の性能を低下させてしまう。

以上に述べられた事実をはじめ、この論文では３D-NoC をベースとしたスパイキングニューラルネット(3D-NoC-SNN)のためのアルゴリズムとアーキテクチャを提案する。第一に、３DNoC-SNN の性能評価は、ユニキャスト、マルチキャスト、ブロードキャストのようなスパイクルーティングの方法、そしてシステムに発生する欠陥のあるかないかにおいて、異なるトポロジーの SNN との比較で行われた。第二に、この論文は画期的な、k-平均法とツリーベースのルーティング方法の組み合わせによるマルチキャストスパイクルーティングアルゴリズムについて提案する。分割方法としてk-平均法を採用することは、全体的にバランスの取れた交通を可能にし、システムの性能を向上させることにつながった。さらに、フォールトトレラントなマルチキャストルーティングアルゴリズムは、主要な経路とバックアップ経路を設けることにより、システムにおける接続の欠落に対処することにも役立てられた。主要ルートにおいて欠陥が現れた時、ルーターはバックアップ経路を用いて、入力スパイクのパッケージを切り替える。これは復旧にかかるオーバーヘッド、平均遅延を削減し、システムがタイミング違反を回避することを可能とした。最後に、アーキテクチャとハードウェアの設計と提案された 3D-NoC-SNN システムは、解析的な比較により提案された仕事についての評価を示した。

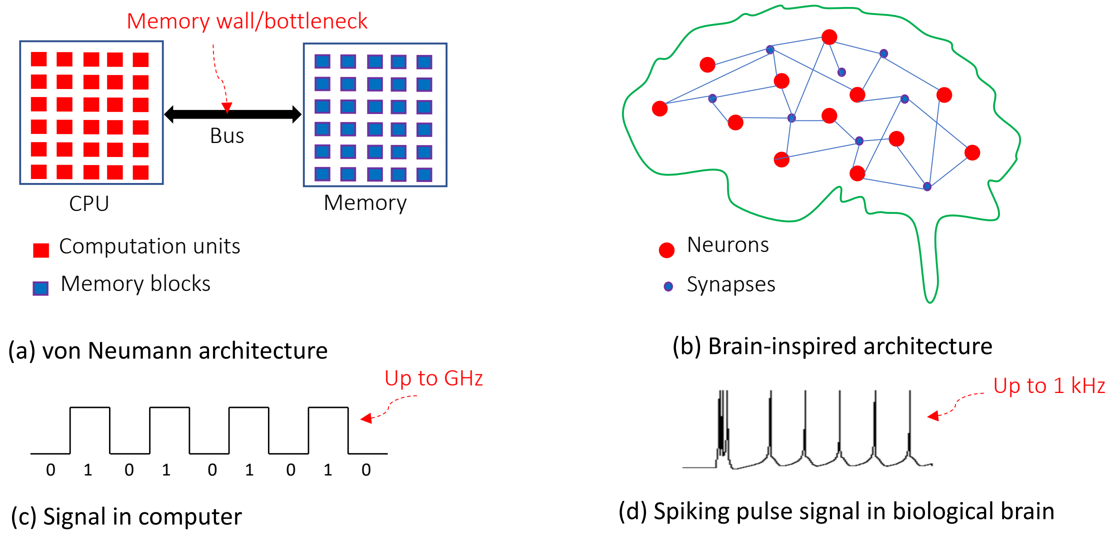
1

Introduction

1.1 Brain-inspired Computing: Towards a New Computation Paradigm

Von Neumann Architecture

The Von Neumann architecture, that was presented 60 years ago, is still a solid base for computer design. The architecture operates in a sequential manner where data is fetched from memory. This concept is very powerful in building supercomputing machines used in a wide range of applications such as quantum mechanics, weather forecasting, climate research, and so on. However, the architecture has faced challenges: (1) in the traditional von Neumann architecture, both data and instructions are stored in the memory, as shown



(a) von Neumann architecture

(b) Brain-inspired architecture

(c) Signal in computer

(d) Spiking pulse signal in biological brain

Figure 1.1: Computation paradigm is shifted from (a) von Neumann (centric computation) to (b) brain-inspired computing (distributed computation)

in Figure 1.1 (a). From this, the CPU can fetch the instructions from the memory and compute arithmetic operations on the data; it however cannot do both at the same time. Consequently, this results in the well-known memory wall problem of the data movement between the CPU and the memory which has become the bottleneck of the entire system [1]. (2) The continued success of the development of the modern von Neumann computing system was secondly enabled by increasing the transistor integration density, followed by the multicore architecture. This was presented in Moore's law predicting that the integration density is doubled every 18 months. The silicon semiconductor industry has shown extraordinary achievements throughout its history. However, since the density of data continuously escalates, extracting valuable information from this huge amount of data becomes computationally expensive, even for supercomputers. Furthermore, when transistors are getting smaller and their power density keeps constant, the questions of domination of dynamic power and the increase in leakage current [2] are raised. Consequently, this slows down the transistor switching rate as well as the overall speed of the system if an efficient cooling mechanism is not employed.

As the end of Moore's law seems closer than ever (see Figure 1.2), computer scientists

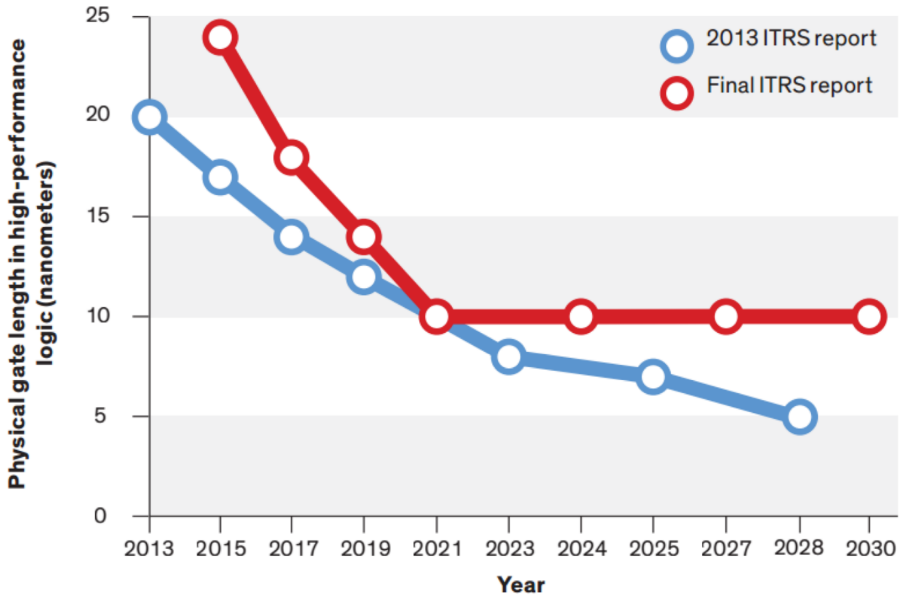


Figure 1.2: End of the road - a shringking challenge of of physical gate length at 10nm [3].

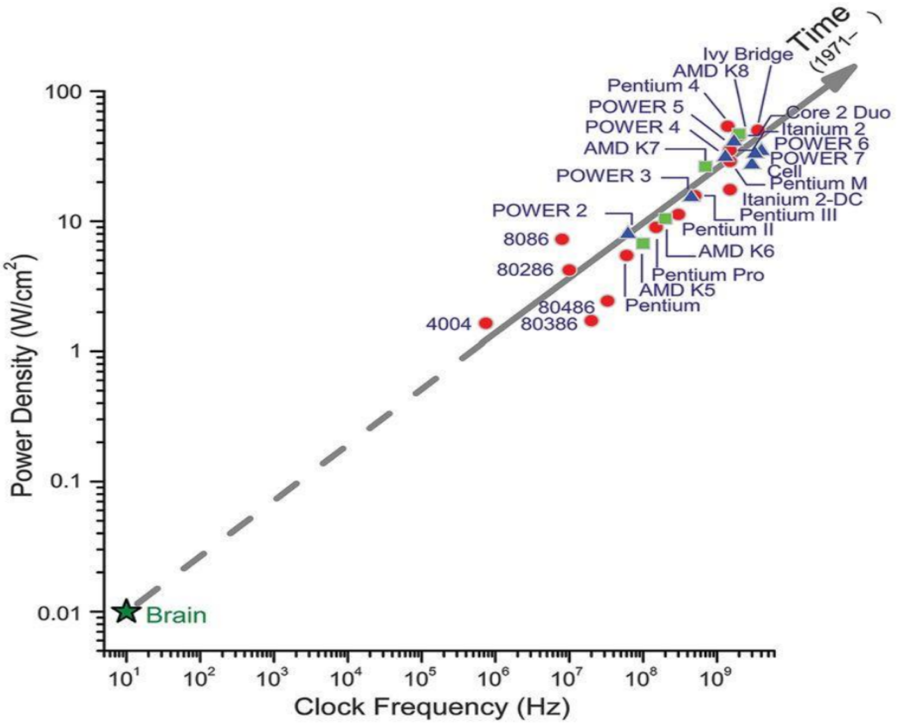


Figure 1.3: Power density and clock frequency challenges of conventional computing architecture [4].

Table 1.1: Comparison of von Neumann and Neuro-inspired computing

von Neumann	**Neuro-inspired computing**
Very high operation frequency (GHz)	Low operation frequency (KHz)
Centric computation	Distributed computation
Low parallelism	High parallelism
Low power efficiency	High power efficiency

have been exploring to build machines as complex and efficient as our brain, dealing with power density and clock frequency challenges of the conventional architecture, as shown in Figure 1.3. Our brain works completely different compared to traditional von Neumann architecture. In fact, there are many secrets behind how the human brain works. What we know is that it distributes computation and memory (see Figure 1.1(b)) among 100 billion biological neurons, and each of them is highly connected with thousands of others via synapses. Neurons communicate with each other through spikes (i. e., short electrical pulses or spikes). The brain is a powerful computation system that helps us survive, adapt, and predict while consuming tens of watts, as summarized in Table 1.1.

Brain-inspired Computing

Brain-inspired computing or neuromorphic computing is a biologically inspired approach created from highly connected neurons to not only model neuroscience theories but also solve machine learning problems. The term neuromorphic was first introduced by Carver Mead in 1990 [5], where it referred to very large scale integration (VLSI) with analog components to mimic biological neural systems. Such systems can be categorized as non-spiking and spiking approaches. First, the non-spiking approach is referred to as the implementation of traditional artificial neural networks (ANNs), in which it aims to improve throughput over power (or acceleration purpose). In recent years, ANNs have shown a remarkable improvement in terms of accuracy (even better human-level [6]) for large-scale visual/auditory recognition and classification tasks. Particularly, the convolution neural network (CNN) [7] and recurrent neural network (RNN) [8] have shown promising tools for a wide range of applications such as image, video, and speech. Nowa-

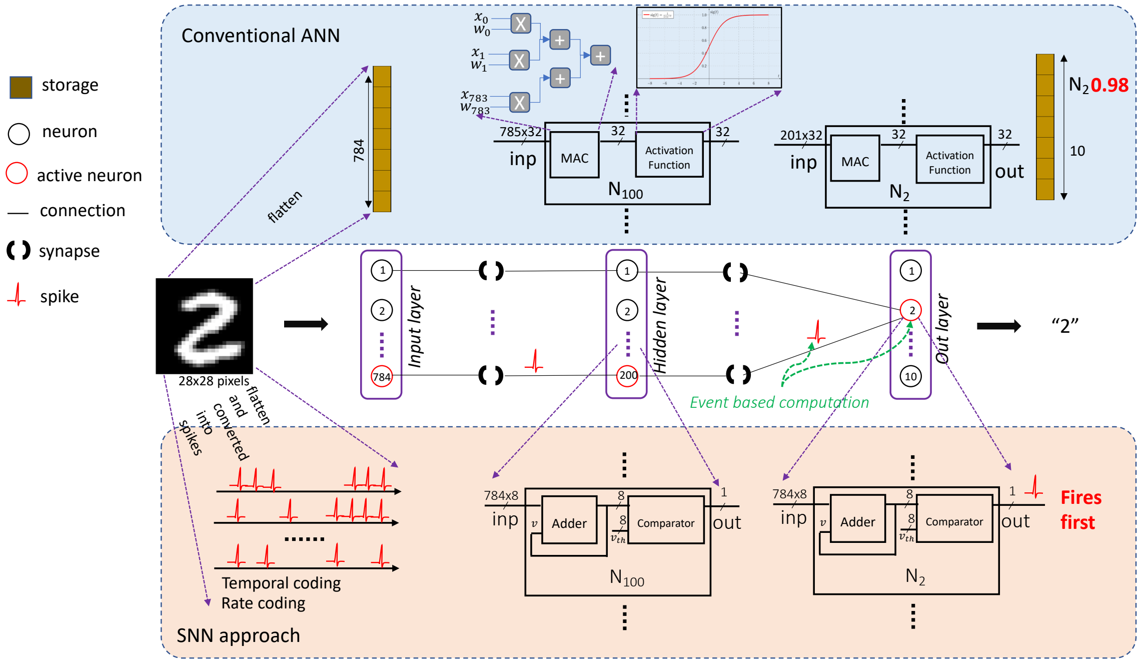


Figure 1.4: An illustration of ANN and SNN hardware implementations for a handwritten digit recognition application.

days, they are typically trained by using graphic processing units (GPUs) or on the cloud side. To reach considerable achievement, state-of-the-art neural networks tend to deeply increase their number of layers and size (i.e., deep learning). For example, Residual Network (ResNet) [9] has 152 layers to achieve 3.57% error on the ImageNet test set (1st place on the ILSVRC 2015 classification challenge [10]). However, this leads to challenges for hardware systems in terms of computation, memory and communication resources. For example, Google's autoencoder [11] was implemented on a cluster of 16,000 processing cores consuming ∼100 kW of power to successfully recognize faces of cats from ten million images captured from YouTube videos.

The second approach based on spiking neural networks (SNNs), witnesses increasing attention both to gain a better understanding of the brain and to explore novel biologically-inspired computations. SNNs have been successfully applied for solving practical problems such as visual recognition and classification tasks [12]. Besides, implementations of neuromorphic hardware have enabled large-scale networks to run in real-time, which is a critical requirement for several applications, including neuro-robotics control, brain-machine interfaces, and robotic decision making. In principle, SNNs attempt to mimic the information processing in the mammalian brain based on parallel arrays of neurons which communicate via spike events. Unlike the typical multi-layer perceptron networks where neurons fire at each propagation cycle, the spiking neurons fire only when a membrane potential reaches a specific value. In SNN, information is encoded using various encoding schemes, such as coincidence coding, rate coding or temporal coding [13]. There have been many spiking neuron models proposed. SNN typically employs integrate-and-fire neurons model [14] in which a neuron generates voltage spikes (roughly 1ms in duration per spike) that can travel down nerve fibers if it receives enough stimuli from other neurons with the presence of external stimuli. These pulses may vary in amplitude, shape, and duration, but they are generally treated as identical events. To better model, the dynamics of the ion channel in a biological neuron, which is nonlinear and stochastic, the Hodgkin-Huxley [15] conductance-based neuron is often used. However, the Hodgkin-Huxley model is too complicated to be used for a large scale simulation or hardware implementation.

Software simulation of SNN is a flexible method for investigating the behavior of neuronal systems. However, simulation of a large (deep) SNN system in software is slow. An alternative approach is a hardware implementation, which provides the possibility to generate independent spikes accurately and simultaneously output spikes in real-time. Hardware implementations also have the advantage of computational speedup over software simulations and can take full advantage of their inherent parallelism. Hardware implementation of ANN and SNN is explained in Figure 1.4. Specialized hardware architectures with multiple neuro-cores could exploit the parallelism inherent within neural networks to provide high processing speeds with high power-efficiency, which make SNNs suitable for embedded neuromorphic devices and control applications. For example, a TrueNorth chip [16] consists of 4,096 neuro-synaptic cores with one million integrate-and-fire neurons and 256 million SRAM synapses. It consumes only 65 mW of power to perform real-time (30 frames/s) object recognition tasks.

1.2 Motivation: Power, Scaling, and Reliability Challenges

Inventing the powerful machine like the human brain has been a driving force in computing for decades. The von Neumann architecture has been considered to be a clear standard for such a system. However, the significant differences in the organization, power consumption requirements, and the computational power of von Neumann architecture compared to a biological brain leads to creating alternative architectures. As inspired by the biological brain, neuromorphic systems require high computation power that can perform operations in parallel. Such systems are more suitable for real-time applications such as: real-time control [17], real-time digital image reconstruction [18], and autonomous robot control [19]. These systems emphasize many simple processing components (i.e., as a form of neurons) combined with dense interconnections between them (i.e., as the form of synapses). This makes that traditional von Neumann architectures are not able to meet this requirement because of the bottleneck [1] coming from the separation of memory and processing unit. Therefore, there is a high demand for neuromorphic systems that can deal with this bottleneck as well as challenges related to end of Moore's law and the end of Dennard scaling. Furthermore, extremely low power operation is an important motivation at the moment and this is inspired by the human brain which performs extremely complex computations with small power, about 20 watts. It, therefore, motivates us to adopt spiking neural networks (SNNs) in our work instead of artificial neural networks (ANNs).

A major challenge of neuromorphic implementation is scalability. It requires a scalable interneuron communication architecture that can maintain a huge amount of traffic created from a massive number of neurons and synapses accommodated on neuron computation units. Since arrival time of spikes is used to encode the information, timing violation in such communication architecture affects the overall performance of SNNs. A shared bus as a communication medium is a poor choice for implementing a large-scale

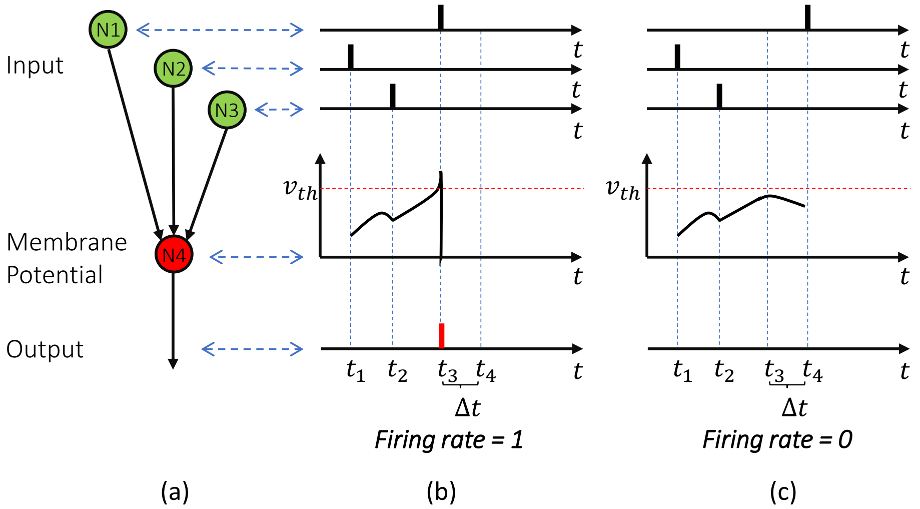


Figure 1.5: Neuron firing rate over different input arrival times: (a) Example of a postsynaptic neuron (N4) receiving incoming spikes from three presynaptic neurons, (b) Firing rate =1, (C) Firing rate =0.

complex SNN chip/system with multicast routing because adding neurons decreases the communication capacity of the chip and may affect the neuron's firing rate due to increasing length of the shared bus. Moreover, the nonlinear increase in neural connectivity is too significant to be directly implemented using a dedicated point-to-point communication scheme. Two-dimensional packet-switched network-on-chip (2D-NoC) [20] has been considered as a potential solution to deal with the interconnection problems found in previously proposed shared communication medium based SNNs [21, 22]. However, such interconnect strategies make it difficult to achieve high scalability with low power consumption, especially in large-scale SNN chips. From another hand, the routing algorithm also plays a vital role in neuron communications because it influences the load balance across the network and the overall latency of system [20]. Since the traffic pattern in a given SNN is in a one-to-many fashion, where a presynaptic neuron sends spikes to a subset of postsynaptic ones, the use of conventional unicast-based routing in large-scale SNNs is inefficient [23].

One of the other main problems of hardware implementations for SNNs is their reliability potential. Although it has been claimed that SNNs have some intrinsic fault-tolerance

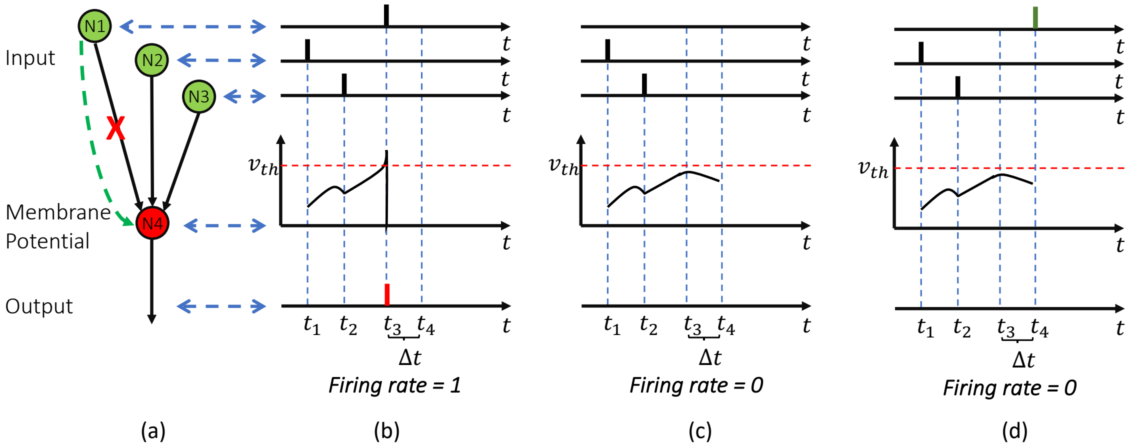


Figure 1.6: Example of the connection-fault effect on the firing rate: (a) a postsynaptic neuron (N4) receiving incoming spikes from three presynaptic neurons, (b) with no connection fault, the firing rate = 1, (c) with the N1-to-N4 connection fault, the firing rate = 0, (d) long latency of a connection with an inefficient routing algorithm resulting in the firing rate = 0.

properties thanks to their massive and parallel structures inspired by the biological neural models, it is not always the case when it comes to practical cases [24]. In fact, with the challenges inherited from the continuing shrinkage of semiconductor components, the implementation of SNNs in hardware exposes them to a variety of faults [24]. The fault risk becomes even more important as we move towards integrating large-scale SNNs for embedded systems when the yield becomes a major problem [25]. When considering the inter-neuron communication reliability, faults may affect the system performance, especially when they occur in critical applications (e.g., aerospace, autonomous car, biomedical, etc.). Such failures can result in undesirable inaccuracies or even irreversible severe consequences. In SNNs, when faults occur in the inter-neuron connections, the postsynaptic neurons become silent or near-silent (low firing activity). As shown in Figure 1.6 (c), at the presence of a broken link in the N1-to-N4 connection, the membrane potential of N4 fails to reach the threshold that would allow it to fire an output spike, as it is the case in Figure 1.6 (b). This leads to a reduction in the firing rate of the postsynaptic neuron. Consequently, it may have an impact on the overall performance of SNN models based on the rate coding method [26]. Neurons with low firing rates become more susceptible to noisy firing rates and temporal jitter of spikes resulting in an increase of the variance [27]. As a result, it demands efficient fault-tolerant techniques. In such mechanisms, the recovery time is one of the important requirements. The long latency of a fault-tolerant routing method may influence the firing rate, as shown in Fig. 1.6 (d). It may impact especially, SNN models using a temporal coding method that is based on the relative timing between spikes. Therefore, the challenge to find efficient fault-tolerant solutions is becoming more important with the integration of large SNNs onto silicon. Routing algorithms are considered as one of the most efficient recovery mechanisms in SNNs as they play a vital role in neuron communication performance. In addition, when considering fault-tolerance requirements, the routing algorithm should be carefully chosen in order to minimize the inter-neuron communication latency; otherwise, the postsynaptic node accuracy can be compromised despite the fact that the failure has been worked around. Figure 1.6 (d) illustrates a clear example of such a case. In this figure, we can see that the long latency due to an inappropriate routing can prevent the postsynaptic neuron to timely fire the output spike.

1.3 Dissertation Goals and Contributions

In this thesis, we present algorithms and architectures for spiking neural network systems based on 3D-NoC, named 3DNoC-SNN. The system exploits the inherent 3D structure of the brain to reduce the communication distances between neurons and allows the seamless implementation of large-scale SNN-based computing systems. The evaluation results show essential characteristics, such as low-latency, high throughput, maintaining the traffic at high fault-rates and low power footprint that make the proposed architecture suitable for large-scale SNN-based embedded AI implementations. The main contributions of this dissertation are summarized as follows:

1. **A performance assessment for 3DNoC-SNN**. The assessment is done by providing an analytic model to analyze the system performance with different spiking neural network topologies, spike routing methods (i.e., unicast, multicast and broadcast),

and in both with and without faults occurring in the system. The goal is to provide an efficient and accurate performance assessment to early understand and evaluate the advantages and drawbacks of potential neural network topologies before the actual hardware development of the SNN system.

2. **Multicast spike routing algorithms for 3DNoC-SNN**. In SNNs, a neuron needs to send their output spikes to thousands of others. In addition, neurons also have different spiking operation modes with different spike rates. As a result, an efficient multicast routing method is highly demanded. This thesis proposes novel routing algorithms which are a combination of k-means clustering and tree-based routing method. Adopting k-means is as a partition method helping to get overall balanced traffic and then improve system performance as well.

3. **A fault-tolerant routing mechanism to deal with link faults in the 3DNoC-SNN system**. In SNNs, when faults occur in inter-neuron connection, the postsynaptic neuron becomes silent because it does not receive enough inputs (spikes) from presynaptic ones. To deal with this issue, this thesis proposes a new fault-tolerant routing algorithm where it pre-defines primary and backup routing paths. When faults appear in the primary route, routers switch incoming spike packages via the backup path. This reduces recovery overhead, average latency, and enables the system to avoid timing violation of SNNs.

1.4 Dissertation Organization

The remaining parts of this thesis is organized as follows:

- In chapter 2, we first overview neural networks, its generations, and topologies. We then present the fundamental of neural networks and their hardware implementation.

- Chapter 3 presents some of the important related works which deal with inter-

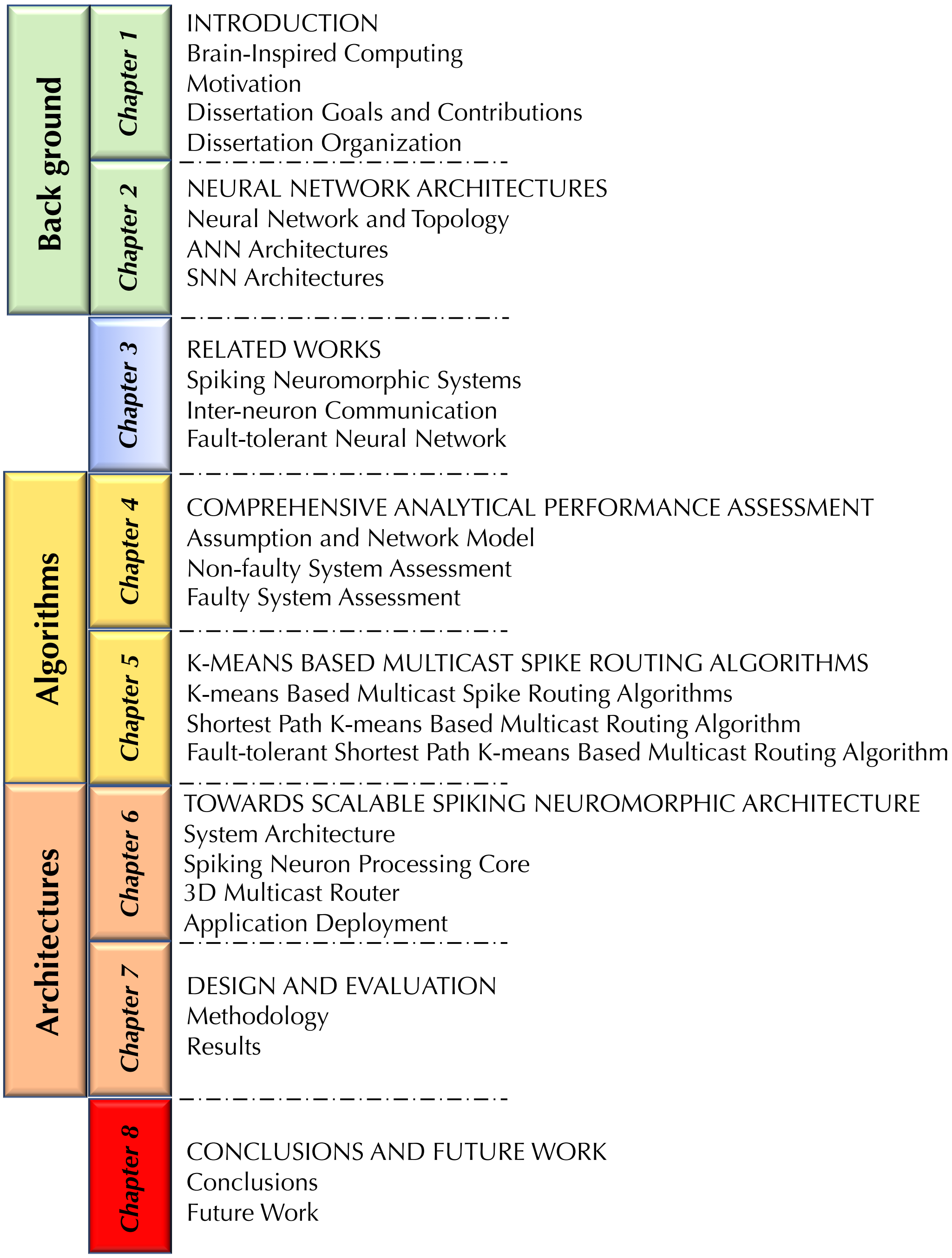

Back ground
Chapter 1
INTRODUCTION
Brain-Inspired Computing
Motivation
Dissertation Goals and Contributions
Dissertation Organization
Chapter 2
NEURAL NETWORK ARCHITECTURES
Neural Network and Topology
ANN Architectures
SNN Architectures
Chapter 3
RELATED WORKS
Spiking Neuromorphic Systems
Inter-neuron Communication
Fault-tolerant Neural Network
Algorithms
Chapter 4
COMPREHENSIVE ANALYTICAL PERFORMANCE ASSESSMENT
Assumption and Network Model
Non-faulty System Assessment
Faulty System Assessment
Chapter 5
K-MEANS BASED MULTICAST SPIKE ROUTING ALGORITHMS
K-means Based Multicast Spike Routing Algorithms
Shortest Path K-means Based Multicast Routing Algorithm
Fault-tolerant Shortest Path K-means Based Multicast Routing Algorithm
Architectures
Chapter 6
TOWARDS SCALABLE SPIKING NEUROMORPHIC ARCHITECTURE
System Architecture
Spiking Neuron Processing Core
3D Multicast Router
Application Deployment
Chapter 7
DESIGN AND EVALUATION
Methodology
Results
Chapter 8
CONCLUSIONS AND FUTURE WORK
Conclusions
Future Work

Figure 1.7: Dissertation organization.

neuron connection challenges in neuromorphic systems. Furthermore, we also provide a survey of fault-tolerance in neural network systems.

- Chapter 4 provides an analytical model to assess the performance of 3DNoC-SNN under different spiking neural network topologies and routing methods. It is conducted under considerations of the fault and no-fault appearance.

- Chapter 5 is dedicated to introducing the proposed multicast routing algorithms. First, it describes a K-means based multicast routing algorithm (KMCR). It then presents an improved algorithm of KMCR, named SP-KMCR. Finally, a fault-tolerant multicast routing algorithm (FTSP-KMCR) that based on SP-KMCR is presented.

- In Chapter 6, we present the system architecture. We then describe its two main components that are a spiking neural processing core (SNPC) as a main computational unit and a 3D router where the proposed spike routing algorithms are implemented. Furthermore, some aspects of deploying the applications onto the system are also presented.

- We dedicate Chapter 7 for presenting implementation, evaluation, and results. We first describe how to implement and evaluate the proposed architecture and algorithms. After that, we provide a comprehensive evaluation and results of the proposed works.

- Finally in Chapter 8, we end this thesis with the conclusion before discussing further future works.

2

Neural Network Architecture: Background

In this chapter, we first present an overview of the artificial neural network, its generations, and topologies. After that, we introduce implementations of the artificial neural network and also spiking neural network, in each of which we consider the neuron model, learning, and existing implementations.

2.1 Neural Network

Overview

A coarse biological neuron, shown in Figure 2.1 (a) is considered to be an information processing system. *Dendrites* play a role as input devices, where input signals are collected. The neuron will process the signals and then produce output signals which are propagated along its *axon*. Finally, the axon transmits the signals via synapses to dendrites of other neurons. It is important to emphasize that this model of a biological neuron is very coarse, and there are many different types of neurons, each of which has different properties.

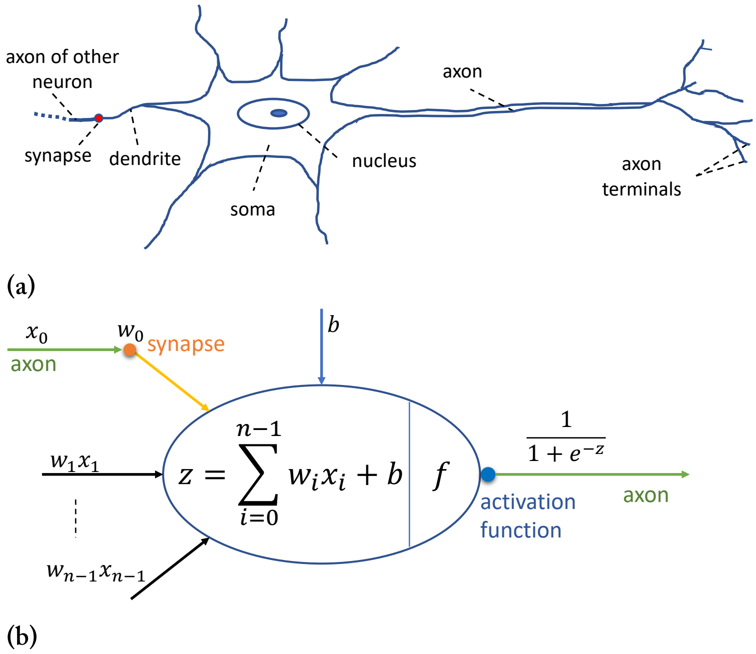


Figure 2.1: (a) a cartoon drawing of a biological neuron (b) a mathematical model of a neuron.

In the computational model of a neuron, shown in Figure 2.1 (b), each output signal (e.g., x_0) from a previous neuron is multiplied with a weight (e.g., w_0). This weight presents the synaptic strength at that synapse. Dendrites carry the signals (e.g., w_0x_0) to the cell body where they all get summed. In the basic model, if the final sum exceeds a certain threshold, the neuron can *fire*, sending a spike along its axon. An activation function is modeled as the *firing rate* of the neuron.

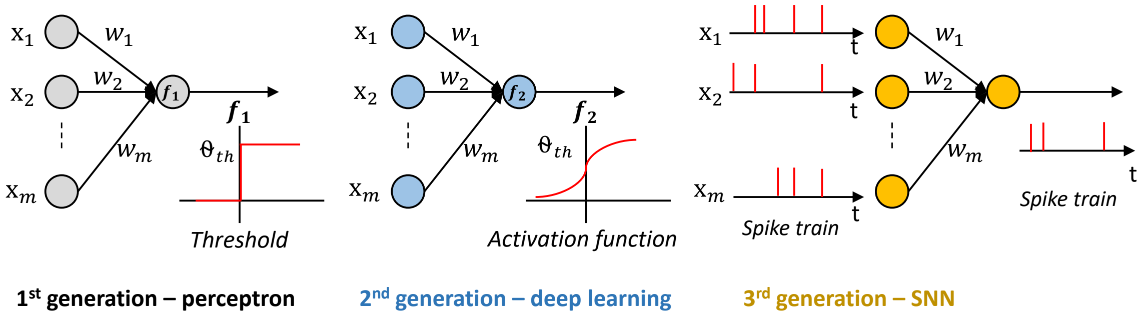


Figure 2.2: Generations of artificial neural network.

Neural networks can be classified into three generations according to their computation units [28, 29], as shown in Figure 2.2:

- **The first generation:** networks have neurons as computational units, and these neurons are referred to as perceptrons or threshold gates. These networks only process with digital inputs and outputs, boolean functions, and a single hidden layer. Multilayer perceptrons, Hopfield networks, and Boltzmann machines are typical examples of this kind of neural network.

- **The second generation:** each neuron in the network applies an activation function with a continuous set of possible output values, such as sigmoid or polynomial or exponential functions. Feedforward, recurrent sigmoidal neural networks, and radial basis function units are considered typical examples of this generation. Moreover, these systems compute not only arbitrary boolean functions but also functions with analog inputs and outputs. Furthermore, neural networks in this generation support learning algorithms based on gradient descent.

- **The third generation:** Spiking neural networks are considered to be a closer approach to modeling biological neurons than previous ANNs. Biological neural systems use the timing of single-action potentials (or spikes) to encode information. Basically, each spiking neuron has a membrane potential which is integrated by incoming pikes. When the membrane potential exceeds a threshold, the neuron fires (i.e., a spike is generated).

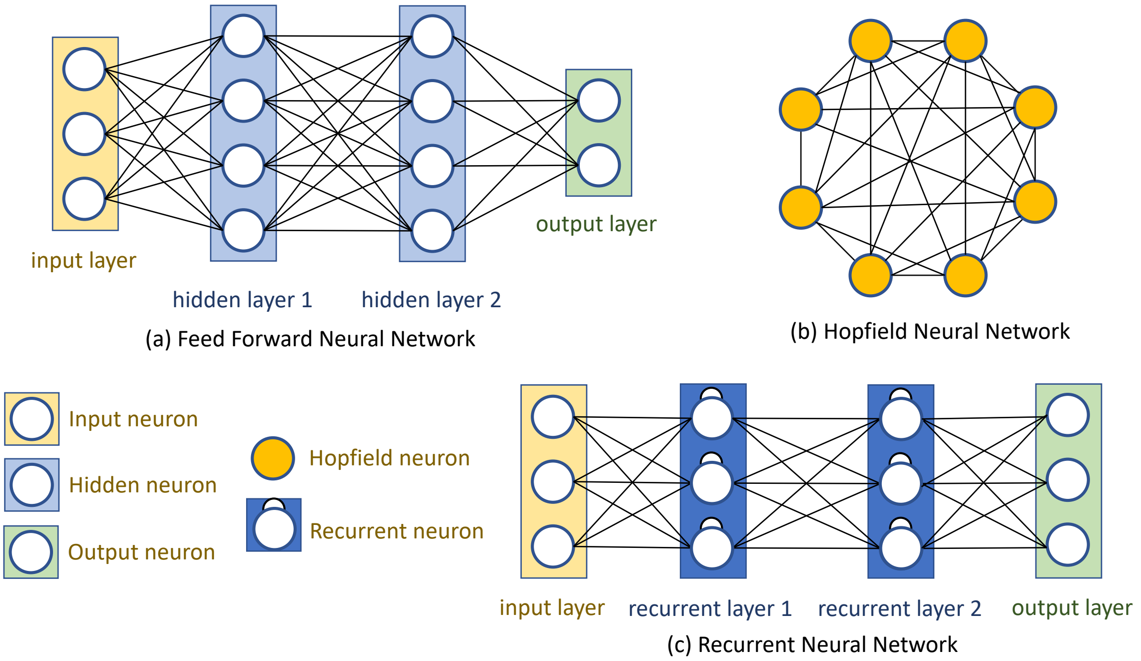


Figure 2.3: Some common neural network topologies.

Neural Network Topologies

Neurons can be connected together in different manners. Typical neural network topologies are summarized, as illustrated in Figure 2.3:

- **Feed forward neural networks:** Figure 2.3 (a) depicts a feed forward neural network (FF or FFNN). This network is organized into separate layers of neurons: input, hidden, and output layers. In this architecture, there are many connections between neurons across layers, but not within a layer. Information is fed from the front to the back. This network usually is used with the back-propagation training method. There are some other neural networks with the same topology as FFNNs. If neurons use a simple binary function, this architecture is called *Perceptron (B)* or *Multilayer perceptron (MLP)*. The simplest network, with two input neurons and one output neuron, can be used to model logic gates. *Radial basis function (RBF)* [30] networks are FFNNs with neurons having radial basis functions. RBFs are suitable for pattern recognition and classification.

- **Hopfield neural network (HFs):** HFs [31] are quite different compared to the systems mentioned above, as shown in Figure 2.3 (b). In this architecture, each neuron is connected to others. Neurons play distinct roles when the network is trained, they are input, hidden, and output, corresponding to before, during, and after training respectively. HFs offer a model for understanding human memory. They are also used as content-addressable memories. *Boltzmann machines (BMs)* [32] are pretty similar to HFs. However, BMs are composed of some input neurons, while the others are hidden neurons. The input neurons will become output neurons after each update of the full network.

- **Recurrent neural networks (RNNs):** As shown in Figure 2.3 (c), RNNs are pretty similar to FFNNs, but the hidden layers are replaced by recurrent neural layers. Unlike FFNNs, neurons in RNNs are not only fed from the previous layer but also from the previous pass of themselves [33]. This results in different outputs when changing the order of information in feeding. RNNs can be used in many areas where the data form can be represented as a sequence such as a string of text. Thus, RNNs are regularly used in autocompletion systems and machine translation. A big drawback of RNNs is the vanishing/exploding problem when using gradient descent technique.

2.2 Artificial Neural Network

2.2.1 Learning Rules

One of the major challenges for neuromorphic designs is how to implement learning algorithms. In general, implementation of learning algorithm can be performed on-chip or off-chip depending on many factors such as neural network models and hardware resources.

Supervised Learning

Backpropagation (BP), a supervised learning method, is the most commonly used algorithm for programming neuromorphic systems. It can be employed in many neural network models such as feed-forward neural networks, recurrent neural networks, and convolution neural networks. The simple way to implement BP in hardware is off-chip [34, 35]. In this case, PB is performed on a traditional host machine. After that, pre-trained parameters are transferred or configured into the target neuromorphic chip. While this method benefits for taking precision of software implementation and requiring lower hardware resource, it is not suitable for systems that have to re-train frequently. However, on-chip BP implementations have been used in many neuromorphic systems [36, 37]. Besides, variations of BP that are optimized or simplified for neuromorphic systems are also implemented [38, 39]. There are other on-chip learning implementations for convolution neural networks [40, 41], Boltzmann machines [42], Restricted Boltzmann machines [43] and deep belief networks [44].

Unsupervised Learning

Compared to supervised learning, implementations of unsupervised learning are less popular. There have been some on-chip ones implemented in neuromorphic systems. Most of them were based on self-organizing maps or self-organizing learning rules [45–47].

2.2.2 Fundamental Implementation

For ANN neurons, the inputs, weights, and outputs are presented as real values. In general, implementations of ANN neurons can be categorized in two ways: analog and digital. Analog implementations offer power efficiency with low area cost and high processing speed, but their downside is a limited accuracy due to being susceptible to noise and difficulty in presenting real values. On the other hand, digital implementations benefit high computation precision, high reliability, and programmability, but they suffer high area cost and high latency compared to analog approaches.

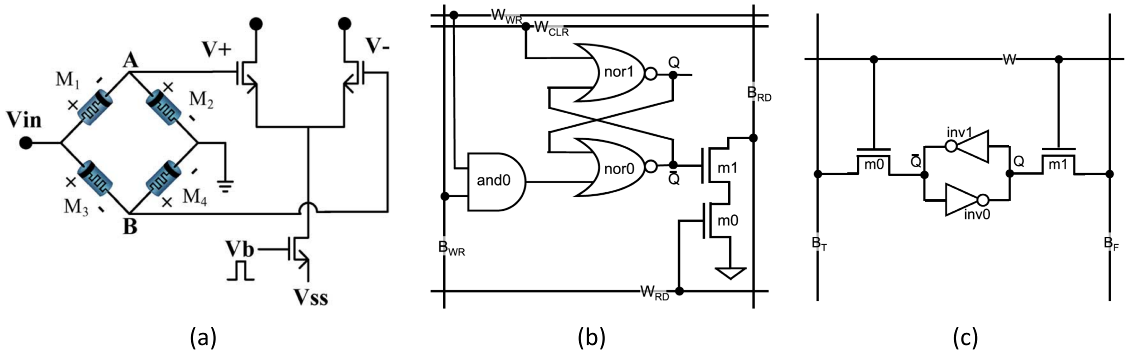


Figure 2.4: Synapse implementation: (a) Analog: memristor bridge synaptic circuit [48]. Digital: (b) 12T scheduler SRAM Cell (simplified) (c) 6T core SRAM cell [16].

Digital neuron implementations are relatively straightforward thanks to supporting powerful design tools. Also, synaptic strengths (weights) are implemented by using registers, latches, or SRAM (see Figure 2.4(b-c)). In addition, all elementary operations such as adders, subtracters, and multipliers can be implemented by available standard circuits [49] while implementation of non-linear activation functions such as sigmoid use specialized hardware, approximated mechanisms (i.e., piecewise-linear function) [35], or look-up tables [50].

Apart from digital CMOS technologies, FPGAs are very attractive due to its programmability and short development time. An FPGA implementation for a general-purpose neuron is presented in [51]. In [52], authors discussed how to implement a single neuron with parallel computation blocks, in addition to bit precision and use of look-up table. Other attractive works were presented in [53, 54]. In these works, authors also discussed implementations and optimization methods of arithmetic operations on FPGAs such as shift add neural arithmetic for fast perceptron and non-linear activation functions.

Analog neuron implementations: In the analog domain, weights are usually implemented by using registers [55], charge-coupled devices [56], capacitors [57, 58], floating gate EEP-ROMS [59], or memristor in recent years (see Figure 2.4(a)). For non-linear activation functions, the characteristic can sometimes be captured directly or using some approximation functions [60]. Although it is difficult to implement a coherent set of all

Table 2.1: Platform comparison for neuromorphic implementation [62].

	Parallel computer	FPNN in FPGA	DSP	FPGA	Analog ASIC	Digital ASIC
Speed	+	+	-	+	+++	++
Area	- -	+	-	+	+++	++
Cost	- -	++	++	++	- -	- -
Design time	+	+++	++	++	- -	- -
Reliability	++	++	++	++	- -	+
- - very unfavorable, - unfavorable + favorable, ++ very favorable, +++ highly favorable						

the basic elements, some operations can be easy to achieve by exploiting simple physical effects [61], such as accumulator can be presented by summing currents (Kirchhoff's current law). Table 2.1 compares different platforms for hardware implementation of neural networks.

2.2.3 Existing Architectures

Digital platform: This is mainly the kind of available neural network platforms, with several categories. First, bit-slice architectures where a processor is composed of modules processing bit-field for ”slice” of an operand. Micro Devices' MD1220 neural bit slice [63] is an example. It composes of eight neurons that have eight 16-bit synapses. Slice platforms generally employ off-chip learning. Another kind of platforms is based on single instruction multiple data (SIMD), in which each PE runs the same instruction at the time on different datasets. For instance, work in [64] proposes a SIMD based processor optimized for image processing. It composes of 16 PEs, each has 24-bit 2K-word local memory and support 24 instructions. Array-based architectures are common designs, especially for neural network acceleration purpose. This architecture composes of multiple PEs in an array where they operate synchronously in the pipeline manner. They are optimized to be very suitable for implementing matrix multiplication [65], a common computation in ANNs. To connect PEs together, these systems use various communication architectures such as common bus [66], ring [67], and network-on-chip [68]. Apart from the platforms mentioned above, other approaches are self-organizing feature map (SOFM) [69] and Digital Signal Processing (DSP) processor [70].

Field Programmable Gate Array (FPGA:) Is also an attractive platform for implementing ANNs thanks to very short development time, low cost, and configurability. However, its downside is resource limitation in implementing large neural network sizes. In [71], a bitstream arithmetic approach is proposed for dealing with resource limitation. Due to the advantages, FPGAs are employed in many applications such as real-time hand detection and tracking [72] and face tracking and identity verification in video sequances [73].

Analog Platforms: Compared to digital approaches, analog platforms are less common due to challenges in implementations. In the early stage of analog implementations, Intel introduced Electrically Trainable Analog Neural Network (ETANN) 80170NX chip [59]. It is a powerful analog chip composing of 64 fully-connected neurons and 10240 synapses, supporting on-chip learning. While neuron states are presented in voltages, weights use floating gates. ETANN can be scaled up to 1024 neurons with 81,920 synaptic weights, using direct-pin/bus interconnection. Authors in [74] proposed a mixed-signal CMOS feed-forward chip using capacitors for weights. It supports on-chip learning implementing Random Weight Change algorithm to be suitable for direct feedback control. Another work [75] implemented a continuous-time recurrent neural network. The interesting thing here is that while available states are expressed by voltages, neural signals are conveyed as currents. This makes the system relatively robust and scalable thanks to neural signals being maintained over long distances.

2.3 Spiking Neural Network

In this section, we first present fundamental of spiking neural network regarding encoding methods, neuron models, and learning rules. We then briefly introduce interconnection architecture and platforms for implementing spike neuromorphic systems.

2.3.1 Neural Coding Methods

As mentioned above, biological neurons use spikes (action potentials) which are short electrical pulses to communicate among them. In a small area of the cortex, there are thousands of spikes that are emitted in every millisecond. This raises the question of how the spikes can be encoded to contain the information? In recent year, there have been many efforts to answer this question, but none of them has proposed a general method. Some coding methods are described as in [13]. This section summarizes two of the most common coding methods which are rate coding and temporal coding. Unlike rate coding method, temporal coding considers the timing of spikes.

Rate Coding

- **Spike Count Rate (Average over time):** This method is determined by the average number of spikes in an interval time, as (2.1):

$$v_{sc} = \frac{n_{spike}}{\Delta t} \tag{2.1}$$

  where n_{spike} is the spike number counted, Δt is the interval (time window). The length of Δt depends on neural models used. This coding method has been successfully used for experiments involving sensory and motor system.

- **Spike Density Rate (Average over several runs):** In this coding method, the same stimulation sequence is repeated K times. From there, the number of spikes n_K is summed over all repetitions. The rate coding method is expressed by (2.2):

$$v_{sd} = \frac{n_K}{K\Delta t} \tag{2.2}$$

  where Δt is the period of a repetition. Although this method is not used by biological neurons, it is a useful method for evaluating neuron activity.

- **Population activity rate (Average over several neurons)**: In the brain, there are a huge number of neurons. Many of them have the same characteristics and interact with the same stimuli. Therefore, this method is proposed to measure the firing rate of a population of neurons, as (2.3):

$$v_{pa} = \frac{n_p}{N\Delta t} \tag{2.3}$$

where n_p is the total number of spike number generated by N neurons, Δt is the time window.

Temporal Coding (Spike coding)

- **Time to First Spike:** This coding method considers the timing of the *first* spike after the stimulus onset. In this coding method, a neuron could signal a strong stimulation if it fires shortly after the reference signal, the latter spike becomes weaker.

- **Phase:** Evidence of the phase coding method was found in the hippocampus of the rat [76]. The hippocampus and some other areas of the brain have oscillations of some global variable which are an internal reference signal. From there, the phase of spikes could be used to encode the information.

- **Correlations and Synchrony**: This coding method comes from the relevance of rank order of spike patterns and synchrony between neurons which could encode information. For example, three neurons that fire with different relative delays might signal a different stimulus.

2.3.2 Neuron Models

A biological neuron is composed of three main components: dendrites, an axon, and a cell body. Information is propagated between neurons through chemical or electrical transmissions (action potentials or spikes). The typical behavior of the neuron can be described as follows: (1) From dendrites, incoming spikes are accumulated at the cell body, and this results in a change in the voltage potential across the cell membrane of the

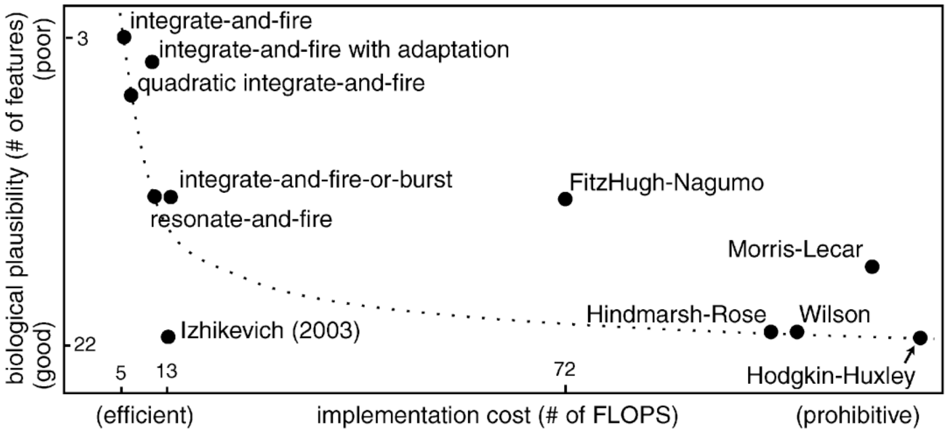


Figure 2.5: A comparison of spiking neuron models in terms of implementation cost and biological plausibility [77].

neuron. (2) When the membrane exceeds a determined threshold, the neuron ”fires” - a spike is generated. This spike then travels along the axon to other neurons.

SNNs can simulate the high level of biological neurons by using individual spikes. Many models of spiking neurons have been proposed. Most of them were implemented in a way to exhibit the same behavior mentioned above. However, they can be different from one model to another. Choosing the appropriate model depends on the user requirements. A pool of spiking neural models was discussed regarding the biological plausibility and computational efficiency in [77].

Hodgkin-Huxley

In the early 1950s, Hodgkin-Huxley neural model was proposed [78]. It presents a mathematical description of the electric current through the membrane potential v giving the details of spike generation, as given in (2.4)

$$\frac{dv}{dt} = (\frac{1}{C})I - g_k n^4(v - E_k) - g_{Na} m^3 h(v - E_{Na}) - g_L(v - E_L) \tag{2.4}$$

where C is the capacitance of the circuit (Figure 2.6 (a)), I is the external current, conductances are potassium g_k, sodium g_{Na}, and leakage g_L. Gating parameters n, m, and h are determined by (2.5), (2.6), and (2.7), respectively

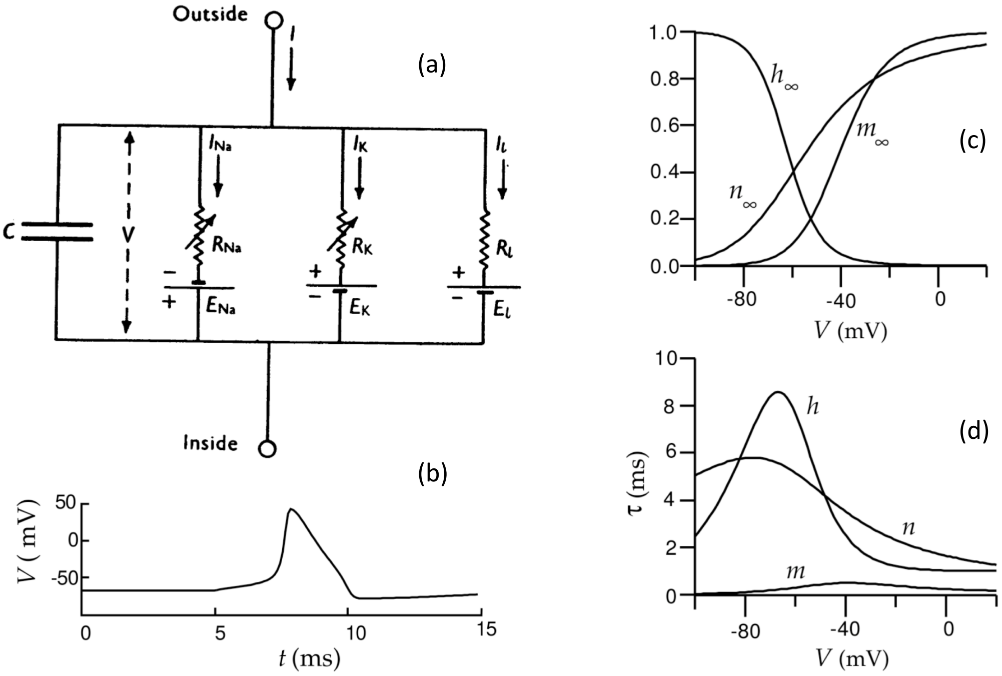


Figure 2.6: The Hodgkin-Huxley model: (a) the schematic diagram presents the membrane potential, in which current injection starts at t = 5 ms as (b), while (c) and (d) show the dependency of the gating variables n, m and h on the membrane potential v [79].

$$\frac{dn}{dt} = (n_\infty(v) - n)/\tau_n(v) \tag{2.5}$$

$$\frac{dm}{dt} = (m_\infty(v) - m)/\tau_m(v) \tag{2.6}$$

$$\frac{dh}{dt} = (h_\infty(v) - h)/\tau_h(v) \tag{2.7}$$

The Hodgkin-Huxley model is the most biological plausible, as shown in Figure 2.5. However, its complexity with many parameters consumes a huge amount of hardware resource. It, therefore, is extremely expensive for large scale implementations.

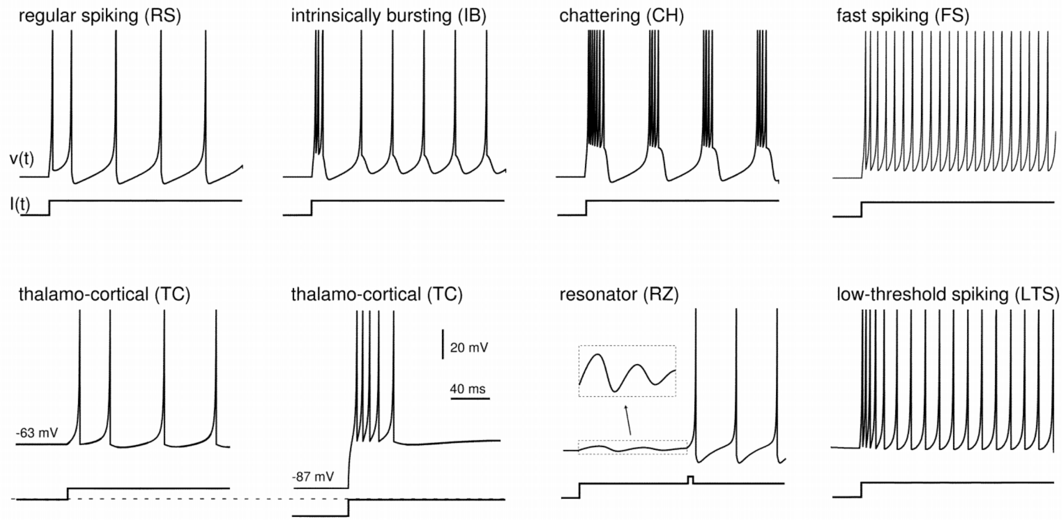


Figure 2.7: Known types of the Izhikevich neuron with different values of the parameters a, b, c, d [80].

Izhikevich

Compared to Hodgkin-Huxley, a less complex model was proposed by Izhikevich [80]. The model is described by the following equations:

$$\frac{dv}{dt} = 0.04v^2 + 5v + 140 - u + I \tag{2.8}$$

$$\frac{du}{dt} = a(bv - u) \tag{2.9}$$

$$\begin{cases} v \leftarrow c \\ u \leftarrow u + d \end{cases} \quad if v \ \geq 30mV \tag{2.10}$$

where v is the membrane potential of the neuron, u is a membrane recovery variable, I is the neuron current, a, b, c, d are parameters of the models, in which the various values of these parameters result in different types of neuron, as shown in Figure 2.7. When membrane potential v exceeds the threshold (30mV), the membrane potential v and recovery variable v are reset as 2.10.

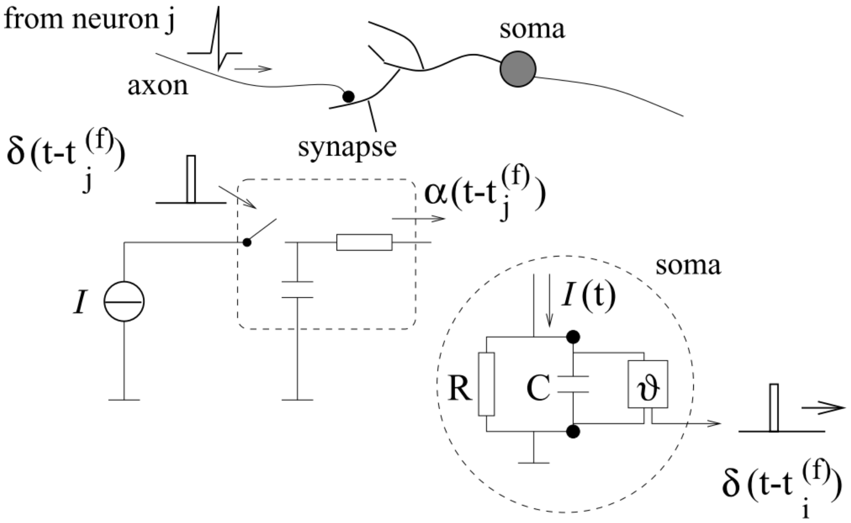


Figure 2.8: Schematic diagram of the LIF model (soma, the circuit is in the dashed circle) [13].

Leaky Integrate and fire

Leaky Integrate and Fire (LIF) model is one of the most common used in SNN. This model is described as the following equations:

$$\frac{dv}{dt} = I + a - bv \tag{2.11}$$

$$v \leftarrow c, \ if\, v \geq v_{th} \tag{2.12}$$

where v is the membrane potential of the neuron, I is the neuron current, a, b, and c are parameters of the model. When the membrane potential v exceeds a threshold v_{th}, it will be reset to c.

The basic circuit presenting LIF model is shown in Figure 2.8. It composes of a capacitor C and a resistor R that are connected in parallel and driven by a current $I(t)$.

In summary, among the existing spiking models, Hodgkin-Huxley [78], Izhikevich [80], and Leaky Integrate-and-Fire (LIF) [81] are often used. The Hodgkin-Huxley type is the best when measurable physiological parameters are highly considered. However, it

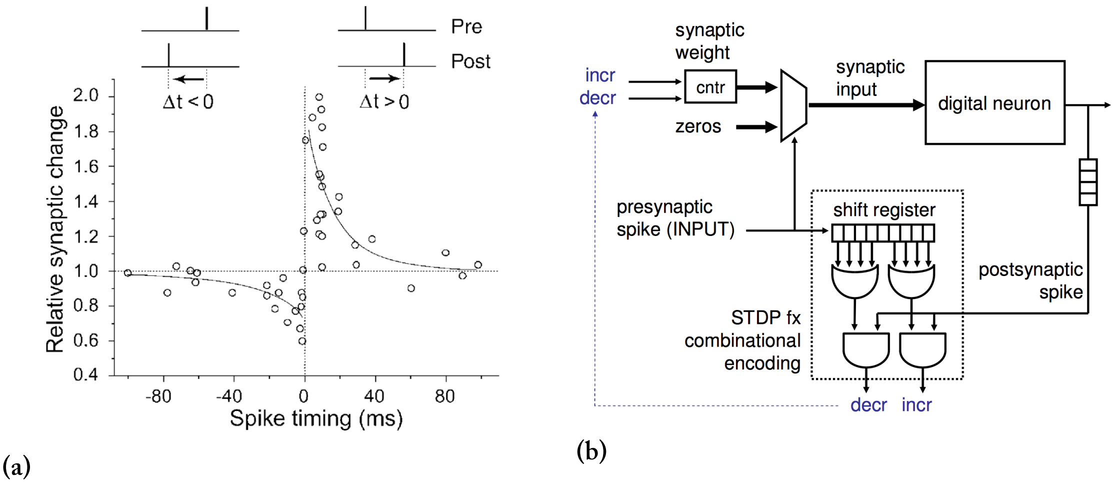


Figure 2.9: STDP: (a) Spike-timing window of STDP characterized in hippocampal cultures [86] (b) a minimum complexity digital implementation of STDP [87].

composes of many coefficients. This leads to challenges when implementing large SNNs because of high cost. In contrast, we can simulate hundreds of thousands of neurons when using LIF neural model; but, it is incapable of producing rich spiking patterns. Finally, the Izhikevich exhibits a good compromise in terms of biophysical similarity and computational cost. It is close to the Hodgkin-Huxley model in biological plausibility while analogous to the LIF in computational complexity.

2.3.3 Spiking Neural Network Learning Rules

Unsupervised Learning

Spike timing dependent plasticity (STDP) is the most popular learning rule implemented in neuromorphic systems [82–84]. It is Hebbian-based coming from observation in biological brain [85]. The operation of STDP basically depends on the arrival time of coming spikes, in which the synaptic weigh will be increased when the spike arrived before post-synaptic neuron "fire" and vice versa, as illustrated in Figure 2.9a. It is an unsupervised learning rule and generally implemented on-chip thanks to its friendly hardware resource, as shown in Figure 2.9b.

Supervised Learning

Apart from the majority of STDP, spiking neuromorphic system also adopted supervised learning rules. In this case, such systems use a ”teacher” signal during the training phase. Besides, another work [88] successfully implemented spike-driven synaptic plasticity (SDSP) learning rule. Unlike STDP, this learning rule induces an update each time a pre-synaptic spike occurs. On the other hand, backpropagation is also adopted for spiking systems [89]. In [90], authors first train an ANN with BP, then convert into SNN by mapping real-value inputs/activations to average firing rates of Poisson spikes. This mechanism can be adopted to implement on spiking hardware as an off-chip learning method.

2.3.4 Communication Network

Communication architectures for spiking neuromorphic systems are responsible for delivering spikes between neuro-cores/tiles. They can be categories as intra-chip and inter-chip. For inter-chip, address event presentation (AER) are commonly employed [91, 92]. In AER, each neuron has a unique address. Whenever a neuron generates a spike, its address is sent to post-synaptic neurons by a high-speed digital bus. AER is suitable for SNN implementations since it only needs to be active whenever neurons fire. To scale up the system, a hierarchical AER as a tree structure was implemented in [93].

On the other hand, network-on-chip (NoC) is commonly implemented for on-chip communication. In the early stage of the implementation, buses are employed in some systems [94]. However, works in [23, 95] evaluated and compared four architectures: bus, tree, point to point, and mesh. The results show that mesh with multicast offers the highest performance for SNN implementations. Furthermore, AER also is used for on-chip communication [96, 97].

2.3.5 Existing Architectures

Digital: Full custom ASIC chips have been common platforms for spiking neuromorphic implementations. Two well-known examples of this kind of implementation are

TrueNorth [16] and SpiNNaker [98]. While TrueNorth only supports the leaky integrate and fire neuron model with no on-chip learning, SpiNNaker offers extreme flexibility in terms of the neuron model, synaptic model, and learning algorithm. However, TrueNorth benefits energy efficiency consuming 25 pJ per connection, while the figure for SpiNNaker is 10 nJ per connection, as reported in [99]. Also, FPGAs are commonly used for implementing spiking neuromorphic systems [100–102]. They can be implemented an apart of the system and also as final implementations. While FPGAs are considered to be a great choice for acceleration over software simulations [103], they are not targeted as platforms for achieving low power.

Analog: There are some characteristics making analog platforms to be suitable for spiking implementations, such as conservation of charge, amplification, thresholding, and integration. Therefore, there are a large number of implementations [104, 105]. Besides, analog platforms were also designed to operate in the subthreshold mode [5, 106], and also a superthreshold mode for the speed-up purpose [107]. On the other hand, field programmable analog arrays (FPAAs) have been used as other analog platforms [108]. They are also customized for neural network implementation such as field programmable neural array (FPNA) [109] and Neuro FPAA [110] where they provide programmable components such as neurons, synapses.

Mixed signal: These platforms are also common for neuromorphic systems [111, 112] to take advantages of both analog and digital platforms. In these works, weights or some other parameters are stored in digital memories to enable the system less noisy and more reliable [113, 114]. Furthermore, inter-chip and intra-chip communication architectures are also implemented in digital [115]. On the other hand, neurons are generally in the form of analog. Two well-known systems for this kind of implementation are Neurogrid [116] and BrainScales [107].

2.4 Conclusion

In this chapter, we overviewed the artificial neural networks including the spiking neural network as the latest generation and how they are implemented. Compared to conventional approaches, SNNs offer not only the capability of simulating biological neural networks but also extreme energy efficiency thanks to event-based operations and fewer operation computations. In the next chapter, we focus on how prior works tried to solve the interconnect challenge in spiking neuromorphic and faults in neural networks.

3

Related Works

In this chapter, we first review some state-of-the-art spiking neuromorphic systems in both software- and hardware-based SNN implementations. We then present interconnect architectures proposed for SNN systems; they are buses, 2D NoCs, and 3D NoCs. Finally, we present works relating to fault tolerance in neural network systems.

3.1 Spiking Neuromorphic Systems

For software-based simulation, the *Blue Brain* project [120] is a popular simulation platform for SNNs, and its approach is similar to several other proposed simulation methods [121]. The *Blue Brain* system can simulate up to 108 simple neurons or up to 104 very

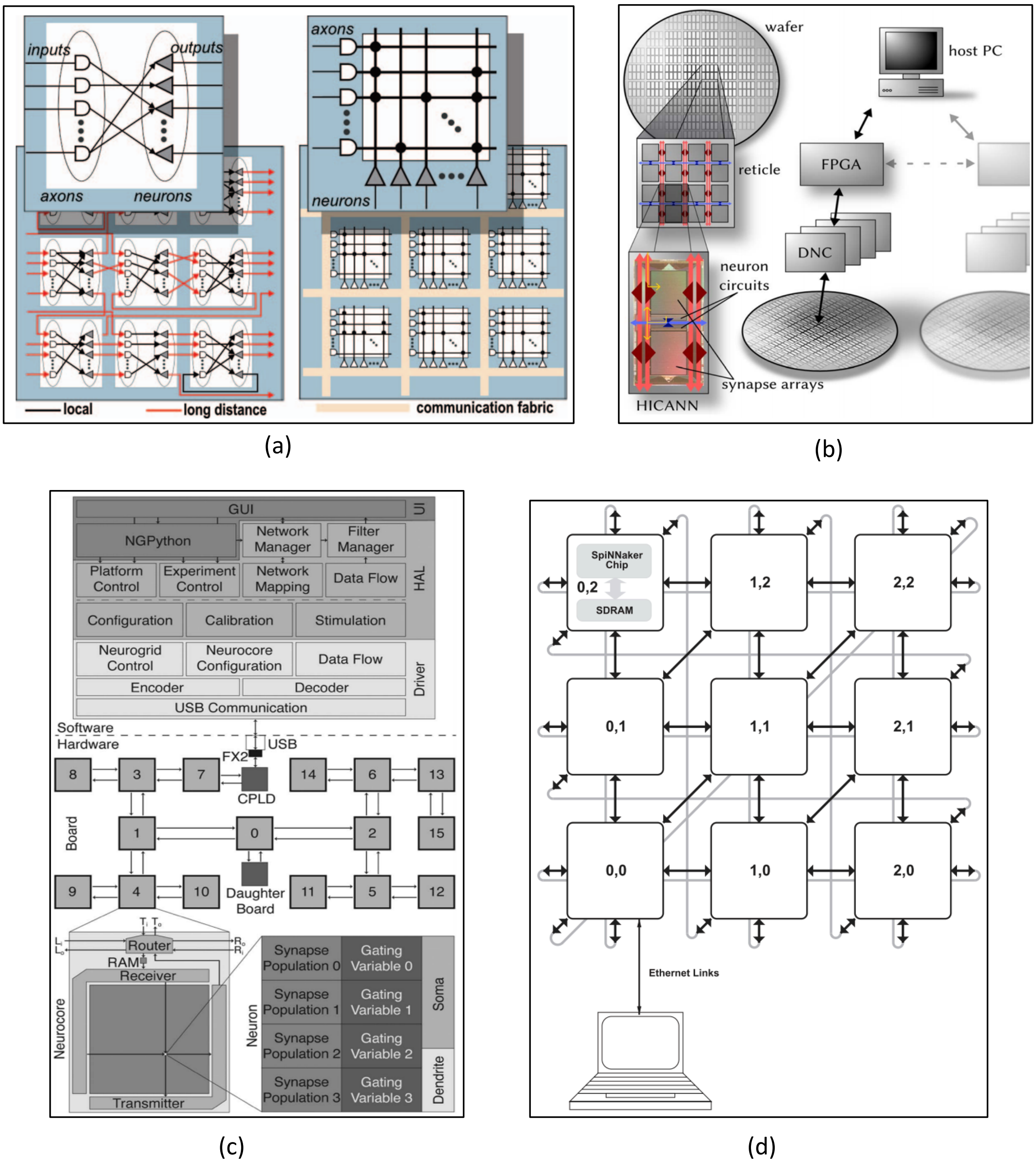
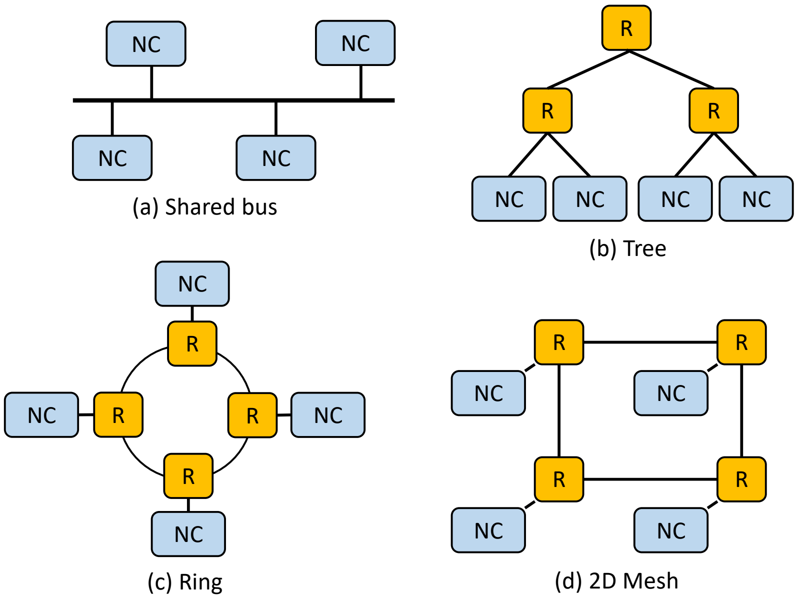


Figure 3.1: (a) TrueNorth: consisting of neurosynaptic cores, tiled in a 2D array: logical representation (left) and physical implementation (right) [117]. (b) The architecture of the BrainScaleS wafer-scale hardware system [118]. (c) Neurogrid architecture: software and hardware [116] (d): SpiNNaker consists of computational units using ARM processors and a 2D triangular mesh interconnect architecture [119].

complex neurons as well as local and global synaptic plasticity rules defined for each neuron. The simulation environment is supported on the IBM *Blue Gene/L*, a system using 8,192 PowerPC CPUs, each running at 700 MHz and arranged in a torus interconnection network [120]. However, this approach is expensive in terms of power consumption which is in the order of hundreds of kilowatts. Besides, this system is considerably slow (low level of parallelism). Thus, it does not achieve biologic real-time execution on large-scale networks.

SpiNNaker project [98] proposes a full custom digital and massively parallel system targeted to implement spiking neural network applications. The project is aimed to simulate up to a billion neurons in real-time and support multiple neural models as well. In order to reach the targets. the architecture composes of 2D-arranged computational nodes connected via a triangular mesh NoC. In each node, there are 18 ARM968 processing cores, in which 16 ones are dedicated for implementing spiking neurons (up to 1,000 neurons), another core is used for monitoring, and the other is kept as a spare for dealing with fault-tolerance problems. In addition, another NoC system is used to connect the processing cores. By using embedded core, SpiNNaker will be extremely flexible in neuron models and learning method. However, it suffers high energy efficiency, at about 10 nJ per connection [99].

TrueNorth [4] is another well-known hardware-based SNN, a full custom ASIC design. Each chip contains 4,096 neural cores composing of 256 integrate-and-fire neurons. As a result, the system can simulate one million neurons. Besides, a NoC system is also used to connect spiking cores. The chip can operate in a partial asynchronous and synchronous manner. Therefore, TrueNorth chip is able to get better energy efficiency compared to SpiNNaker, it consumes 25 pJ per connection. However, it has some shortcomings such as fixed neuron model, limited programmable connectivity, and no on-chip learning.



(a) Shared bus

(b) Tree

(c) Ring

(d) 2D Mesh

Figure 3.2: Interconnect architectures for neuromorphic systems.

3.2 Inter-neuron Communication

Hardware implementations were proposed as alternative solutions to overcome the problems of the software simulation mentioned above. Such systems require a high-parallelism scalable interconnect architecture to convey a huge number of spike generated from neuro-cores. Hierarchical-bus, point-to-point, or NoC interconnect architectures are widely used, as illustrated in Figure 3.2. In this section, We survey various interconnect platforms with spike routing methods for spiking neuromorphic systems, as shown in Figure 3.3.

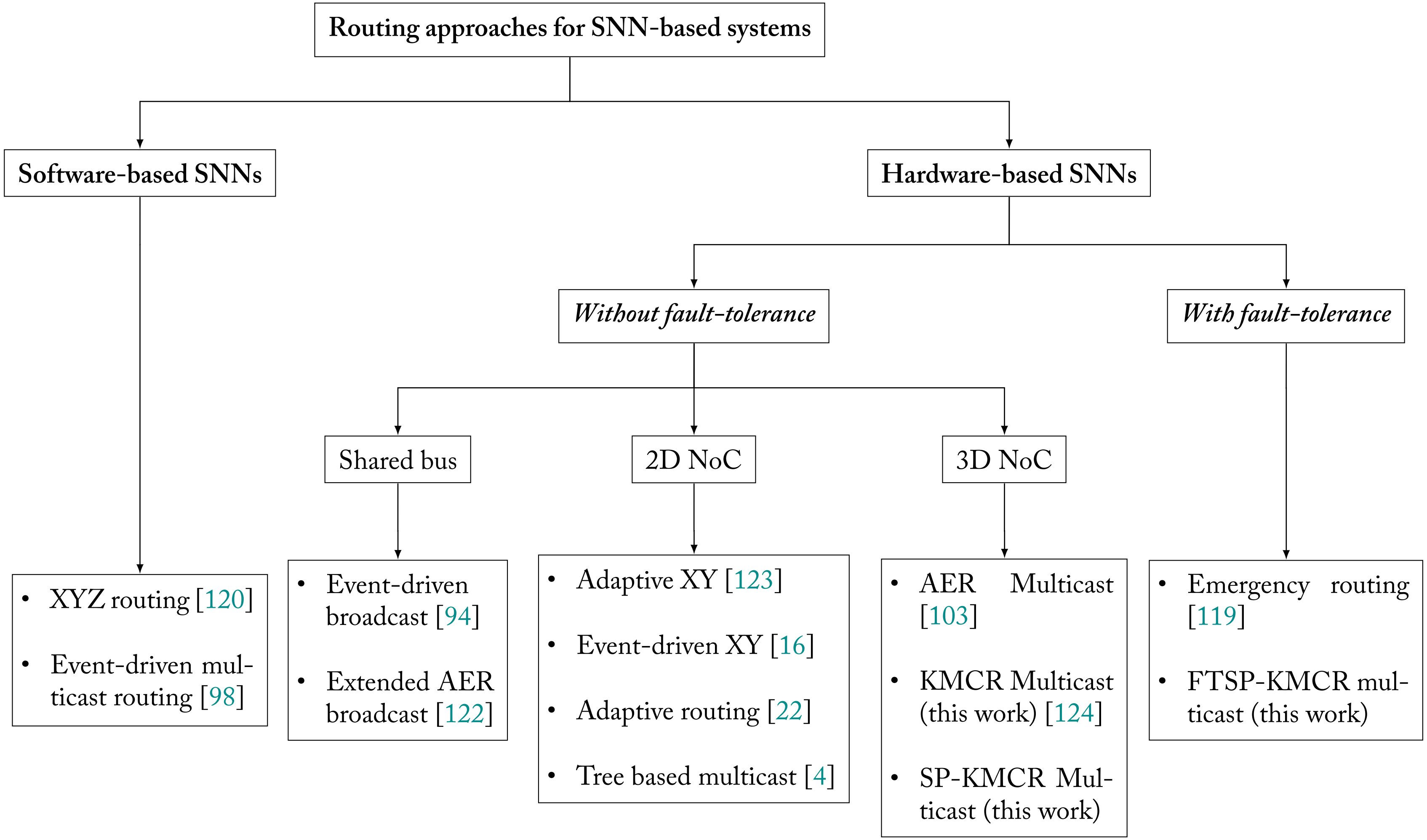

Routing approaches for SNN-based systems
Software-based SNNs
Hardware-based SNNs
Without fault-tolerance
With fault-tolerance
Shared bus
2D NoC
3D NoC
• XYZ routing [120]
• Event-driven multicast routing [98]
• Event-driven broadcast [94]
• Extended AER broadcast [122]
• Adaptive XY [123]
• Event-driven XY [16]
• Adaptive routing [22]
• Tree based multicast [4]
• AER Multicast [103]
• KMCR Multicast (this work) [124]
• SP-KMCR Multicast (this work)
• Emergency routing [119]
• FTSP-KMCR multicast (this work)

Figure 3.3: Summary of SNN routing methods on various interconnect platforms.

3.2.1 Hierarchical Bus-based Spike Routing

Low-cost shared-bus based SNN architectures are proposed in [94, 122]. Although these approaches support multicast and broadcast routing, they suffer from the limitation of scalability when the network size increases. Other works were proposed in [125, 126]. These architectures boosted the throughput; but, they were limited to small-size neural networks.

3.2.2 2D Packet-swiched-based spike routing

There are many ongoing SNN research projects based on 2D-NoC interconnects [4, 21–23, 99]. Hereafter, we only review a few well-known projects. The *Neurogrid* project [116] uses analog computation to emulate ion-channel activity and a digital communication scheme to support synaptic connections. The main building block is the neuro-core, which can accommodate a total of 65,536 quadratic integrate-and-fire neuron models, and it uses an external FPGA and bank of SRAMs for digital communication between neighboring neuro-cores. The Neurogrid has a limitation on the maximum number of neurons per layer (up to 2,175 neurons) that makes it unable to offer biological real time behavior [116].

H-NoC [22] is based on a hierarchical star-mesh topology to connect neurons. The *H-NoC* is organized into three layers: module, tile, and cluster. At the bottom, each module router can connect up to ten neural cells, each of them as a main neural computation element can host one or multiple neurons. In the same fashion, ten module routers are connected to a tile router. An attractive work in [127] proposed a combination of hierarchical architecture and mesh routing strategies. The architecture consists of multiple levels of routers.

In SpiNNaker[98], the interconnection between each node is handled by a NoC using six links, which is wrapped into a triangular lattice; this lattice is then folded onto a surface of a toroid. A node composes of processor cores and two NoC routers, in which one handles the communication between the microprocessors and the peripherals, and the second controls the communications between processors and neighbor nodes. *FACETS*

[128] uses a mixed-signal and high-density hardware neural network architecture based on a combination of analog neurons and a digital multilayer bus communication scheme; all of them placed on an uncut wafer. The *FACETS* hardware model consists of a large number of ASICs containing the analog neuron and synapse circuits. A full wafer can comprise 384 HICANN chips [128], resulting in a total of 196,608 neurons per wafer. To support the neurons interconnection, this architecture uses a combination of hierarchical buses for handling neuron communication inside the wafer, and off-wafer routers implemented on an FPGA based on a 2D-torus topology. *FACETS* can offer hardware acceleration with up to 10 μs inter-spike interval per wafer. However, the architecture consumes a large amount of power estimated to 1kW per wafer [128].

Another work, named *ClosNN*, is presented in [21]. The *ClosNN* system uses a customized NoC architecture based on Clos topology for the neural network. It is designed to overcome with a high diameter of mesh and low bisection bandwidth of a hierarchical tree. The architecture suffers from wire/router physical limitations.

3.2.3 3D Packet-swiched-based Spike Routing

The work in [129] investigated the architecture and design of a 3D stacked neuromorphic accelerator. The architecture targeted processing applications on a CMOS vision sensor next to the first neural network layer. The authors claimed that only modest adaptations would be required to use the system for other applications. The 3D stacking architecture used face-to-face bonding of two 20cm wafers using micro-bumps.

A recent work was presented in [103] about a real-time digital neuromorphic system for the simulation of large-scale conductance-based SNNs. The architecture was implemented in six Altera Stratix III FPGA boards to simulate one million neurons [103]. An *AER* multicast routing mechanism was used for inter-neuron communications. Although the NoC architecture meets the requirements of the system, it is hardly deployed in embedded neuromorphic systems [130].

Apart from the works mentioned above, routing methods for NoC-based SNNs need

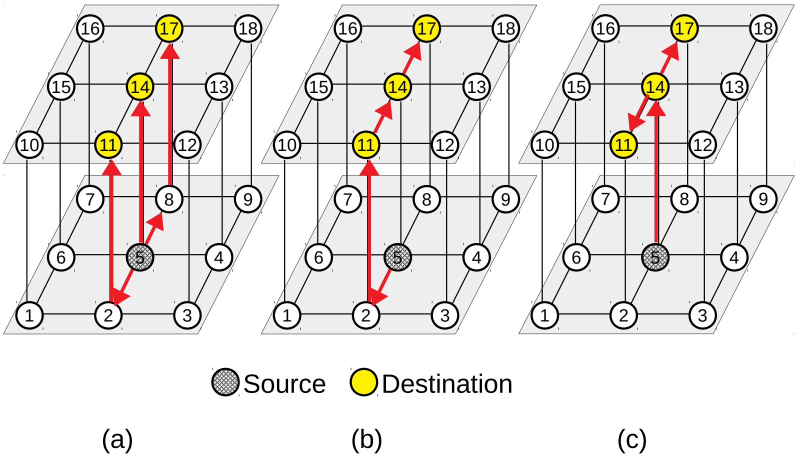


Figure 3.4: Multicast routing mechanisms: (a) Unicast-based (b) Path-based (c) Tree-based.

to be taken into consideration. This is because the spike routing method affects the load balance across the network and also the spike latency. In general, these works can be classified as unicast-based [131], path-based [132], and tree-based [133]. A comparison between these methods is presented in [132]. The basic ideas of these algorithms are shown in Figure 3.4. Compared to the others, unicast-based [131] is an easy way of implementing multicast with no hardware overhead. This is because a multicast package will be replicated at the source node and sent sequentially to destinations. However, a drawback of this method is that it requires a high start-up latency before injecting the packet into the network. Besides, it also leads to a large amount of traffic because of the injection of multiple copies.

In the path-base [132], a routing path is established from source and to each destination. Before sending, every packet header needs to contain a list of all destinations. Whenever the packet reaches a target, the information of that destination will be removed from the header. This helps the packet to be sequentially delivered to all destinations. A disadvantage of this method is that it requires a long time for the packet preparation at the source node. Besides, when increasing the size of destination sets (large size of SNNs), it is not efficient to implement because of a large header size of packets.

Drawbacks of path-based can be overcome by tree-based [133]. In this approach, a

”virtual” tree is constructed with the source node as the root and destinations as leafs. The packets are sent from the source, going along branches, and reaching given destinations. Apart from advantages, a shortcoming of this method is high congestion in wormhole networks [134].

3.3 Fault-tolerant Neural Network

There have been many works proposed to solve the fault occurrence in hardware implementations of neural networks [24]. A taxonomy of fault-tolerant approaches is shown in Figure 3.5

3.3.1 Learning-based approaches

These methods are based on modified conventional learning rules for dealing with faults occurring in neural networks systems. In [135], authors presented a fault-tolerant technique based on temporary injecting faults in hidden neurons during the training process. In this method, one to three neurons are randomly injected for each input example. Another work in [136] presented a modified training rule by adding a regularization term to the cost function. A work based on the backpropagation was proposed in [137], to dealing with faults in classification tasks. In this learning method, weights are constrained under a limited range. In summary, although the modified learning methods do not require any external interactions afterward, they suffer a significant increase in the computation cost and take a long time for the training process.

Apart from the methods mentioned above, retraining methods are also wildly used. In [138], authors proposed a method that performs retraining periodically to improve fault-tolerance in GPGPUs systems. This method does not require either reprogramming or recompilation. Work in [139] proposed a retraining method for dealing with the impacts of timing errors in hardware-based neural networks. In this method, the retraining process is performed when output results are influenced by timing errors. Authors in [140] presented a new learning rule mimicking self-repair capability of the brain, in which the learning rule could reestablish the firing rate of neurons when synaptic faults occur.

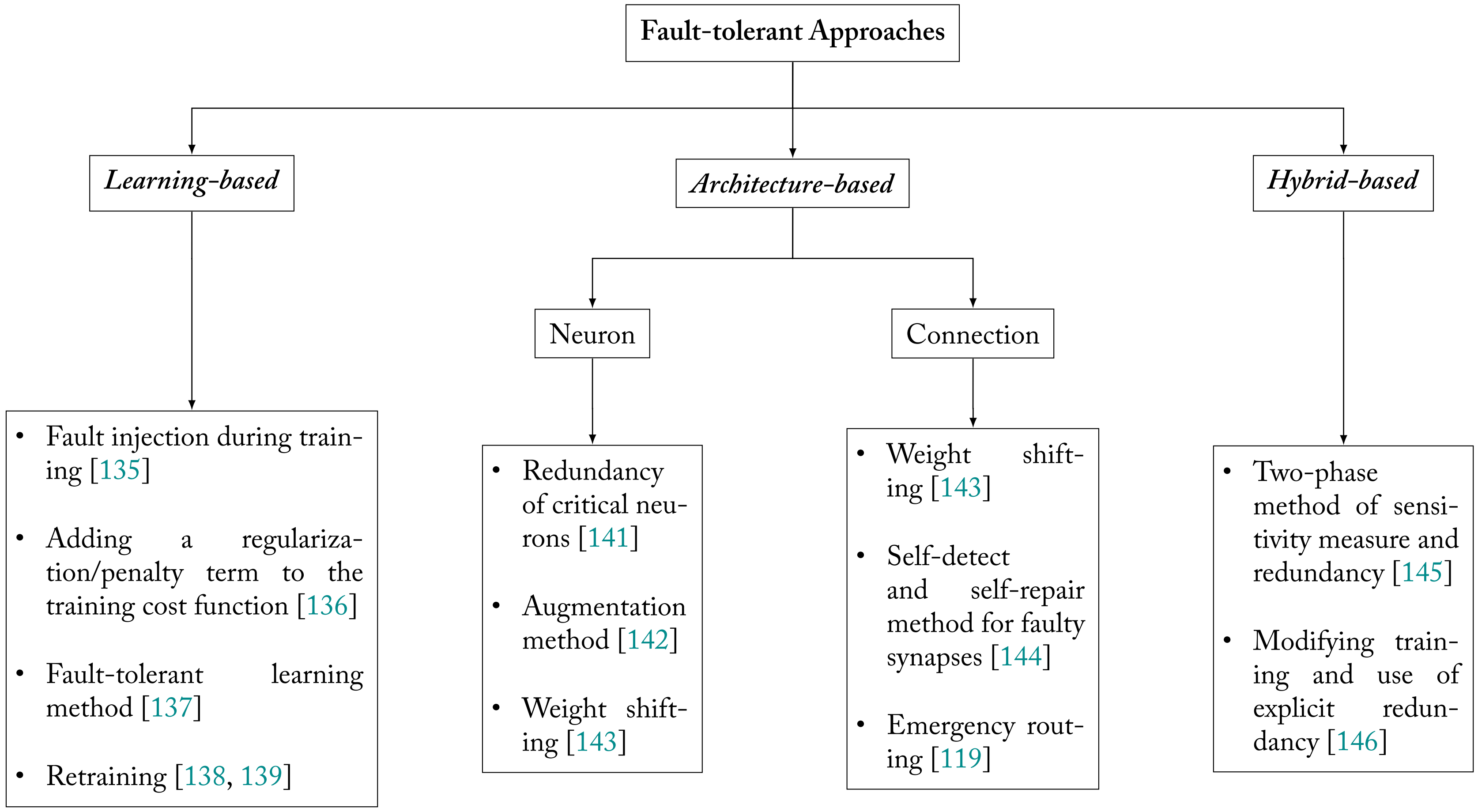

Fault-tolerant Approaches
Learning-based
Architecture-based
Hybrid-based
Neuron
Connection
Fault injection during training [135]
Adding a regularization/penalty term to the training cost function [136]
Fault-tolerant learning method [137]
Retraining [138, 139]
Redundancy of critical neurons [141]
Augmentation method [142]
Weight shifting [143]
Weight shifting [143]
Self-detect and self-repair method for faulty synapses [144]
Emergency routing [119]
Two-phase method of sensitivity measure and redundancy [145]
Modifying training and use of explicit redundancy [146]

Figure 3.5: Taxonomy of fault-tolerant approaches for neural network architectures.

3.3.2 Architecture-based approaches

Regarding architecture-based, fault-tolerant methods are mainly based on the redundancy of the architecture. The redundancy is implemented in pre-trained networks including hidden neurons and their connections. Work in [141] proposed a fault-tolerant architecture with the redundancy of certain critical neurons. This reduces the hardware cost of the system. In this method, multiple sets of weight are stored in a processor, recomputing neural computations with multiple processors enables the system to detect and correct the faults in the processor, from there improving fault tolerance. Another work [142] also presented the redundancy of critical hidden neurons combined with a simple technique, named augmentation. In the proposed method, the weight of the connections between augmented neurons and ones in the output layer is half of its original one.

Apart from faulty neurons, faults in the connection between neurons have also been concerned. In dealing with faults occurring connections and neurons, a method named weight shipping was proposed in [143]. In this method, when faults appear in some connections, their weights are shifted to other fault-free connections of the same neuron. Besides, for a faulty neuron, its output connections are examined to be faulty. A self-repairing hardware architecture was proposed in [144], as shown in Figure 3.6. This architecture features self-detect and self-repair synaptic faults and maintains the system performance with a fault rate of 40%. However, the experiment was taken with only two neurons, and the architecture may suffer a scalability limitation due to its area overhead. In SpiNNaker [119], an emergency routing was proposed to deal with congested or broken links in a 2D-NoC torus topology. The algorithm is based on redundancy in the NoC architecture to automatically redirect a blocked packet through adjacent links to its destination. This enables the system to avoid the timing violations of SNNs when congestion or faults occur.

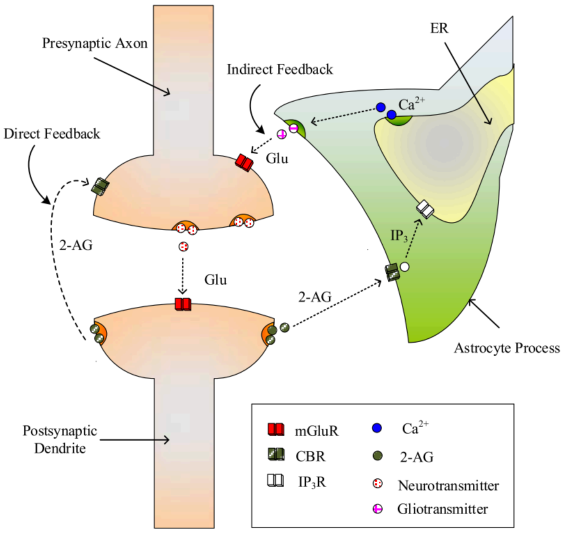


Figure 3.6: A self-detect and self-repair mechanism mimicking capability in the human brain [144]. This mechanism is based on indirect feedback from the astrocyte cell (i.e., the most abundant type of glial cell in the brain), by regulating the synaptic transmission probability of release when faults occur.

3.3.3 Hybrid approaches

Hybrid approaches are based on a combination of the learning-based and architecture-based methods. In [145], a two-phase method was proposed to considerably improve the fault tolerance of the system. At the first phase, by feeding input and measuring the sensitivity, less important hidden neurons are removed. After that, some redundant neurons are added, the network is then retrained. The evaluation results show a fault-tolerant improvement of the system for two multiclass classification problems. This work was then extended in [146]. In this work, the authors proposed three methods: (1) during the backpropagation training, weights are restricted to have low magnitudes to avoid fault-tolerant degradation caused by high magnitude weights. To achieve the desired performance, hidden nodes are automatically added to the network. (2) During the training process, artificial faults are injected to some neurons and connections. (3) unimportant neurons are removed, while new neurons are added to share the role of critical neurons in the network. These methods were evaluated and the results showed better robustness compared to other approaches.

3.4 Conclusion

In summary, we discussed in this chapter some of the important large-scale SNN systems. We also described some exiting works dealing with faults in the neural networks. We spent the main part of this chapter reviewing many proposed works for interneuron architecture. From these, we can find that 3D-NoCs are promising architectures for SNN implementation, offering high parallelism, scalability, and small footprint. Therefore, it motivates us to propose algorithms and architectures for spiking neural network systems based on 3D-NoC in this thesis. In the next chapter, we present a performance assessment model to analyze the performance of the 3DNoC architecture under different SNN topologies and spike routing methods.

4

Comprehensive Analytic Performance Assessment

For hardware implementations of spiking neural networks, an efficient inter-neuron connection architecture is a major challenge. This comes from thousands of neurons need to send their spikes to their post-synaptic ones. Therefore, this chapter presents an assessment method to help designers early understand and evaluate the advantages and drawbacks of their potential neural network topologies and spike routing schemes. The assessment method provides an analytic model to analyze the performance of 3D mesh NoC over variants of two main neural network topologies and three communication methodologies (i.e., unicast, multicast, and broadcast). In addition, the assessment is performed under both with and without connection faults occurring in the system.

The organization of this chapter is as follows: Section 4.1 presents assumptions for the analytical model. While Section 4.2 performs analysis of the architecture without fault injection, Section 4.3 considers injection of faults into the system. Finally, this chapter presents the summary and discussion in the last section.

4.1 Assumption and Network Model

The proposed method was inspired by the work performed in [23, 95], where the authors analyzed different 2D interconnect topologies for neural networks over various spike routing protocols. Our assessment is also performed for Hopfield NN (HF) and Randomly Connected NN (RNDC) topologies over unicast (UC), multicast (MC), and broadcast (BC) routing algorithms. Hopfield and RNDC neural networks represent various connectivity structures of SNN. In Hopfield [147], every single neuron connects to all other ones in the network. On the other hand, connectivity of RNDC [148] simply represents varieties of NN model such as feed-forward NN and deep belief NN, in which the connection probability exponentially decreases with the distance between neurons. Furthermore, the ”O” notation is used to compare the analysis results between the different spike routing methods.

We consider a system comprising of n PEs and n routers arranged in a $\sqrt[3]{n} \times \sqrt[3]{n} \times \sqrt[3]{n}$ 3D mesh [149–153], as shown in Figure 4.1. Assuming in this network model that each PE has one spiking neuron.

The total number of links in a 3D-mesh NoC is given by Equation (4.1)

$$TL_{3DMesh} = 3\sqrt[3]{n^2}(\sqrt[3]{n} - 1). \tag{4.1}$$

As in [154], the mean distance between two nodes in 3D-mesh can be determined by

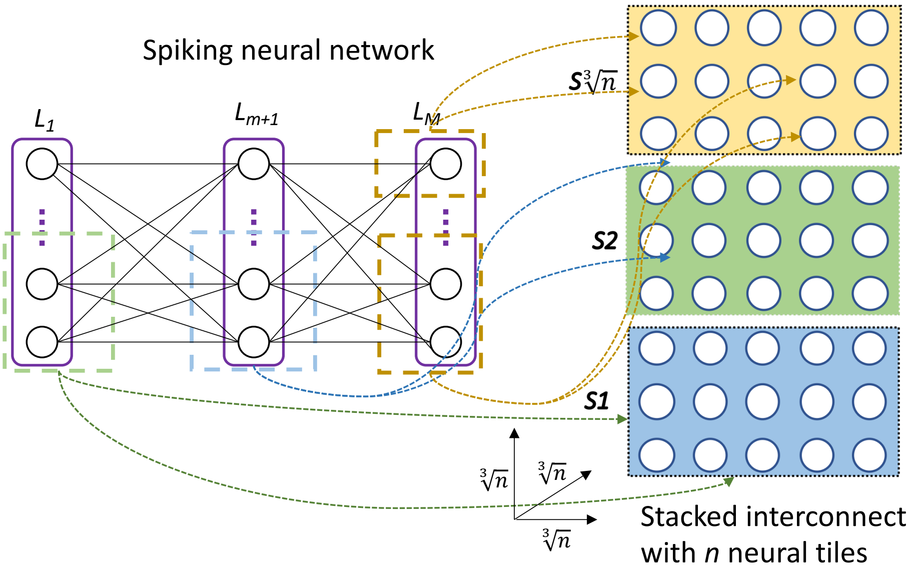


Figure 4.1: 3D mesh NoC architecture with n neural tiles (PEs).

(4.2)

$$\overline{Dist}_{3DMesh} = \frac{\sqrt[3]{n^2} - 1}{\sqrt[3]{n}}. \tag{4.2}$$

In terms of cost and performance metrics, we also use similar ones in [23, 95]. Effective bandwidth (BW_{eff}) is performed to evaluate the architecture performance indicating the parallelism level of the architecture, as (4.3).

$$BW^{nn}_{eff,3DMesh,cast} = \frac{\overline{w}.TL_{3DMesh}}{TotalDist^{nn}_{3DMesh,cast}}.f_{NoC}.U_{NoC}, \tag{4.3}$$

where nn = {HF, RNDC}, $cast$={UC, MC, BC}, $\overline{w}$ is the number of wires per link, f_{NoC} is the link frequency, and U_{NoC} is the link utilization factor for 3D-mesh NoC. In the case of $U_{NoC} = 1$, BW_{eff} reaches the maximum. From this, we can perform the average spike injection rate of each PE (delivered rate) indicating capability of the architecture in delivering spikes, as (4.4):

$$f^{HF}_{p,out,cast} = \frac{BW^{nn,3DMesh,cast}_{eff}}{n}. \tag{4.4}$$

In SNNs, since a neuron cannot emit fire again after the refractory period ($T_{refractory}$), the maximum spike frequency (offered spike rate) of a neuron is $1/T_{refractory}$. From this, K is defined to determine the level of matching of delivered spike rate and offered spike rate, as (4.5).

$$K = \frac{f^{n}_{p,out}}{f^{n}_{spike,max}}. \tag{4.5}$$

From (4.5), $K > 1$ means that the architecture can deliver all spike injected by the neuron in each PE corresponding to multiple being accommodated on a single PE, and vice versa. In terms of hardware complexity, area cost as total wire area and power dissipation as dynamic power consumed on link and gate capacitance are also estimated (described later).

4.2 Non-faulty System Assessment

4.2.1 Performance Analysis of Hopfield NN Based on a 3D-mesh

In this section, we analyze the performance of the Hopfield neural network (NN) on a 3D-mesh NoC over Unicast, Multicast, and Broadcast based spike routing schemes.

Unicast-based Spike Routing

When a neuron sends a packet (spike) to all the other neurons, the node within the 3D-NoC needs to send *n-1* packets to all the others. Therefore, the total number of hops traversed by a spike is given by Equation (4.6)

$$TotalDist^{HF}_{UC,3DMesh} = (n-1).\overline{Dist}_{3DMesh} = \frac{(n-1)(\sqrt[3]{n^2}-1)}{\sqrt[3]{n}}. \tag{4.6}$$

From (4.6), the effective bandwidth of a 3D-mesh NoC system is determined by

$$BW^{HF}_{eff,3DMesh,UC} = \frac{\overline{w}.TL_{3DMesh}}{TotalDist^{HF}_{3DMesh,UC}}.f_{NoC}.U_{NoC} = \frac{3\overline{w}}{(\sqrt[3]{n}+1)(1-\frac{1}{n})}.f_{NoC}.U_{NoC} = O\left(\frac{1}{\sqrt[3]{n}}\right), \tag{4.7}$$

where $\overline{w}$ is the number of wires per link, f_{NoC} is the link frequency, U_{NoC} is the link utilization factor for 3D-mesh NoC. With *n* PEs, the average spiking rate of a single PE is given by Equation (4.8)

$$f^{HF}_{p,out,3DMesh,UC} = \frac{BW^{HF}_{eff,3DMesh,UC}}{n} = \frac{3\overline{w}}{n(\sqrt[3]{n}+1)(1-\frac{1}{n})}.f_{NoC}.U_{NoC}. \tag{4.8}$$

Besides, the maximal firing frequency for a unicast based NoC is expressed as (4.9)

$$f^{HF}_{spike,max,UC} = \frac{1}{T_{refractory}} \cong \frac{1}{10n.T_{cycle}} = \frac{f_{NoC}}{10n}, \tag{4.9}$$

where $T_{refractory}$ is the period after a spike is generated, during that time the neuron cannot fire again, T_{cycle} is the link delay ($T_{cycle} = 1/f_{NoC}$). As mentioned above, in order to send a spike the source node needs to send $n-1$ packets, it thus takes $n.T_{cycle}$ (not including router delay because it is a constant, independent of network size). In this case, we select $T_{refractory} \cong 10n.T_{cycle}$.

From dividing (4.8) by (4.9), we can determine how many neurons may fire at the maximal rate. This is represented by K, as given in (4.10)

$$K = \frac{f^{HF}_{p,out,3DMesh,UC}}{f^{HF}_{spike,max,UC}} = \frac{30.\overline{w}.U_{NoC}}{(\sqrt[3]{n}+1)(1-\frac{1}{n})} = O\left(\frac{1}{\sqrt[3]{n}}\right). \tag{4.10}$$

Multicast and Broadcast Based Routing Schemes

For these routing methods, since each PE only sends a packet for each spike to the others, the number of hops is determined by the following formula:

$$TotalDist^{HF}_{MC/BC} \cong n. \tag{4.11}$$

The efficient bandwidth, the frequencies, and K metric for Multicast and Broadcast are calculated by the following formulas ((4.12), (4.13), (4.14), (4.15)):

$$BW^{HF}_{eff,MC/BC} = \frac{3\overline{w}\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{n}.f_{NoC}.U_{NoC} = 3\overline{w}(1-\frac{1}{\sqrt[3]{n}}).f_{NoC}.U_{NoC} = O(1) \tag{4.12}$$

$$f^{HF}_{p,out,MC/BC} = \frac{BW^{HF}eff, MC/BC}{n} = \frac{3\overline{w}(\sqrt[3]{n}-1)}{n\sqrt[3]{n}}.f_{NoC}.U_{NoC} \tag{4.13}$$

$$f^{HF}_{spike,max,MC/BC} = \frac{1}{T_{refractory}} \cong \frac{1}{T_{cycle}\overline{Dist}_{3DMesh}} = \frac{\sqrt[3]{n}}{\sqrt[3]{n^2}-1}.f_{NoC} = O\left(\frac{1}{\sqrt[3]{n}}\right) \tag{4.14}$$

$$K = \frac{f^{HF}_{p,out,MC/BC}}{f^{HF}_{spike,max,MC/BC}} \cong \frac{3\overline{w}(\sqrt[3]{n^2}-1)}{n\sqrt[3]{n}}.U_{NoC} = O\left(\frac{1}{\sqrt[3]{n^2}}\right). \tag{4.15}$$

With the total link TL_{3DMesh} and the number of wires per link $\overline{w}$, the area cost of 3D-mesh architecture can be expressed by

$$A_{3DMesh} = W_p.\bar{l}.\overline{w}.TL_{3DMesh} = 3.W_p.\bar{l}.\overline{w}.\sqrt[3]{n^2}(\sqrt[3]{n}-1) = O(n), \tag{4.16}$$

where W_p is the wire pitch for a given technology, $\bar{l}$ is the average link length. Dynamic power dissipation on links and gate capacitance are estimated as shown below:

$$P_{3DMesh} = \frac{V^2_{dd}}{R_0.\bar{l}}.\overline{w}.TL_{3DMesh} = \frac{V^2_{dd}}{R_0.\bar{l}}.\overline{w}.U_{NoC}.3\sqrt[3]{n^2}(\sqrt[3]{n}-1) = O(n), \tag{4.17}$$

where V^2_{dd} is the supply voltage, R_0 is the wire resistance. Table 4.1 summarize the evaluated metrics for implementing Hopfield NN on a 3D-mesh NoC system. For 3D-mesh, MC and BC offer better results than UC, at the same power consumption. Regarding the bandwidth, while MC and BC are independent of the network size, UC suffers from the scale-up problem. Furthermore, MC and BC provide higher spiking rate compared to UC, and thus higher throughput as well. In comparison to a 2D counterpart, 3D-mesh NoC shows higher power efficiency.

Table 4.1: 2D mesh NoC [23] vs 3D mesh NoC analyzed for Hopfield NN

Metric	2D Mesh			3D Mesh (This work)		
	UC	MC	BC	UC	MC	BC
BW_{eff}	$O(1/\sqrt{n})$	$O(1)$	$O(1)$	$O(1/\sqrt[3]{n})$	$O(1)$	$O(1)$
Area	$O(n)$					
Power	$O(n)$					
Spiking frequency	$O(1/n)$	$O(1/\sqrt{n})$	$O(1/\sqrt{n})$	$O(1/n)$	$O(1/\sqrt[3]{n})$	$O(1/\sqrt[3]{n})$
K	$O(1/\sqrt{n})$			$O(1/\sqrt[3]{n})$	$O(1/\sqrt[3]{n^2})$	$O(1/\sqrt[3]{n^2})$

4.2.2 Performance Analysis of RNDC NN Based on a 3D-mesh

As in [23], we define the probability ($p(a,b)$) of having a connection between a neuron a and a neuron b in a 3D-mesh NN architecture by the formula (4.18):

$$p(a,b) = \frac{C}{8\pi\lambda^3} e^{-D(a,b)/\lambda}, \tag{4.18}$$

where $C = N_{links} = ||p(.)||$ is the mean number of connections per neurons, λ is a constant presenting spatial connectivity, and $D(a,b)$ is Euclidean distance from a to b. From this probability, the mean distance ($\overline{Dist}^{RNDC}$) between the connected neurons is determined by the following formula:

$$\overline{Dist}^{RNDC} = \frac{1}{8\pi\lambda^3} \iiint\limits_{x,y,z} \sqrt{x^2+y^2+z^2} e^{-\sqrt{x^2+y^2+z^2}/\lambda} \mathrm{d}x\mathrm{d}y\mathrm{d}z = \frac{24\pi\lambda^4}{8\pi\lambda^3} = 3\lambda \tag{4.19}$$

Unicast Based Routing

We first consider the case of the unicast method where a neuron sends C packets to post-synaptic ones. Thus, the average hop count is given by:

$$TotalDist^{RNDC}_{3DMesh,UC} = \overline{Dist}^{RNDC}.N_{links} = 3\lambda C. \tag{4.20}$$

Since the average number of connection C is independent of the NoC dimension, C is kept the same ($C \cong \sqrt{n}$) to compare with 2D-mesh fairly. On the other hand, λ is a measure of locality, a small λ means that neuron is connected more locally and vice versa. For a 2D-mesh NoC, $\lambda \cong \sqrt[3]{n}$ leading to optimal performance. Thus, for a 3D-mesh NoC we can determine λ by the following formula:

$$\lambda \cong \sqrt[4]{n}. \tag{4.21}$$

As a result, the efficient bandwidth is given by Equation (4.22)

$$BW^{RNDC}_{eff,UC} = \frac{\overline{w}3\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{3\sqrt[4]{n}\sqrt{n}}.f_{NoC}.U_{NoC} = \frac{\overline{w}\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{\sqrt[4]{n^3}}.f_{NoC}.U_{NoC} = O(\sqrt[4]{n}). \tag{4.22}$$

Furthermore, frequency of a single PE $f^{UC}_{p,out}$, maximal spiking rate for UC $f^{UC}_{spike,max}$, and K ratio are given as below:

$$f^{UC}_{p,out} = \frac{BW^{RNDC}_{eff,UC}}{n} = \frac{\overline{w}\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{n\sqrt[4]{n^3}}.f_{NoC}.U_{NoC} \tag{4.23}$$

$$f^{UC}_{spike,max} = \frac{1}{10.C.T_{cycle}} = \frac{f_{NoC}}{10\sqrt{n}} = O\left(\frac{1}{\sqrt{n}}\right) \tag{4.24}$$

$$K = \frac{f^{UC}_{p,out}}{f^{UC}_{spike,max}} = \frac{10\overline{w}\sqrt{n}\sqrt[3]{n^2}(\sqrt[3]{n}-1).f_{NoC}}{n\sqrt[4]{n^3}.f_{NoC}}.U_{NoC} = O\left(\frac{1}{\sqrt[4]{n}}\right). \tag{4.25}$$

Multicast Based Routing

For the case of multicast, a packet needs to travel along a 3λ path to reach the first destination, and then plus one hop for each of the others. With the total of C destination nodes, the hop count for each packet is therefore determined by

$$TotalDist^{RNDC}_{3DMesn,MC} = C + \overline{Dist}^{RNDC} = C + 3\lambda \tag{4.26}$$

The bandwidth is given by the following formula:

$$BW^{RNDC}_{eff,3DMesh,MC} \cong \frac{3\overline{w}\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{\sqrt{n}+3\sqrt[4]{n}}.f_{NoC}.U_{NoC} = O(\sqrt{n}). \tag{4.27}$$

As a result, the frequency is given by the following equation:

$$f_{p,out}^{MC} = \frac{BW_{eff,3DMesh,MC}^{RNDC}}{n} = \frac{3\overline{w}\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{n(\sqrt{n}+3\sqrt[4]{n})}.f_{NoC}.U_{NoC}. \tag{4.28}$$

With the link delay T_{cycle}, and average distance between two nodes in the RNDC neural network $\overline{Dist}^{RNDC}$, the maximal spiking frequency is expressed as below:

$$f_{spike,max}^{MC} = \frac{1}{T_{cycle}.\overline{Dist}^{RNDC}} = \frac{f_{NoC}}{3\lambda} = \frac{f_{NoC}}{3\sqrt[4]{n}} = O\left(\frac{1}{\sqrt[4]{n}}\right). \tag{4.29}$$

From (4.28) and (4.29), K ratio is given by following equation:

$$K = \frac{f_{p,out}^{MC}}{f_{spike,max}^{MC}} = \frac{3\overline{w}\sqrt[3]{n^2}(\sqrt[3]{n}-1)3\sqrt[4]{n}}{n(\sqrt{n}+3\sqrt[4]{n})}.U_{NoC} = O\left(\frac{1}{\sqrt[4]{n}}\right), \tag{4.30}$$

where U_{NoC} is the link utilization.

Broadcast Based Routing

Since the RNDC system with broadcast based routing is similar to the case of Hopfield (there is only a difference in network size, C for RNDC, and n for Hopfield), the performance metrics are similar to the Hopfield NN. Furthermore, the area cost and the power consumption are also identical to the Hopfield NN's results. The analyzed results for RNDC neural network are summarized in Table 4.2.

Table 4.2: 2D-mesh NoC [23] vs 3D mesh NoC performance analysis for RNDC NN systems.

Metric	2D Mesh			3D Mesh (This work)		
	UC	MC	BC	UC	MC	BC
BW_{eff}	$O(\sqrt[6]{n})$	$O(\sqrt{n})$	$O(1)$	$O(\sqrt[4]{n})$	$O(\sqrt{n})$	$O(1)$
Area	$O(n)$					
Power	$O(n)$					
Spiking frequency	$O(1/\sqrt{n})$	$O(1/\sqrt[3]{n})$	$O(1/\sqrt{n})$	$O(1/\sqrt{n})$	$O(1/\sqrt[4]{n})$	$O(1/\sqrt[3]{n})$
K	$O(1/\sqrt[3]{n})$	$O(1/\sqrt[6]{n})$	$O(1/\sqrt{n})$	$O(1/\sqrt[4]{n})$	$O(1/\sqrt[4]{n})$	$O(1/\sqrt[3]{n^2})$

In conclusion, the results, shown in Tables 4.1 and 4.2, demonstrate that the 3DNoC-

SNN system achieves higher throughput when compared to the 2D-mesh NoC NN with the same frequency and power dissipation. We also found that, in terms of the routing method, the Multicast scheme outweighs the other routing methods (Broadcast and Unicast).

4.3 Faulty System Assessment

This section mainly analyzes the effects of link faults on the performance of 3D-NoC of spiking neurons architecture. Neural network topologies and communication methods are also emulated as Section 4.2. Considering faults occurring in the system, we assume that the system has a link fault rate α with a uniform distribution. With the link fault rate α, the total number of functional links in a 3D-mesh NoC is given by:

$$TL = 3(1-\alpha)\sqrt[3]{n^2}(\sqrt[3]{n}-1). \tag{4.31}$$

4.3.1 Performance Analysis of Hopfield Neural Network Based on a 3D-mesh

In this section, we analyze the performance of the Hopfield neural network on a 3D-mesh NoC over Unicast, Multicast, and Broadcast based spike routing schemes.

Unicast-based Spike Routing

Considering that a neuron sends a packet (spike) to all the other neurons, the node within a 3D-NoC needs to send *n-1* packets to all the others. Therefore, the total number of hops traversed by a spike is given by:

$$\begin{aligned} TotalDist_{UC}^{HF} &= (n-1).\overline{Dist} \\ &= \frac{(n-1)(\sqrt[3]{n^2}-1)}{\sqrt[3]{n}} \\ &\approx n\sqrt[3]{n}. \end{aligned} \tag{4.32}$$

From equation (4.32), the effective bandwidth of a 3D-mesh NoC system is determined by:

$$
\begin{aligned}
BW^{HF}_{eff,UC} &= \frac{\overline{w}.TL}{TotalDist^{HF}_{UC}}.f_{NoC}.U_{NoC} \\
&= \frac{3\overline{w}(1-\alpha)(\sqrt[3]{n}-1)}{\sqrt[3]{n^2}}.f_{NoC}.U_{NoC} \\
&= O\bigg(\frac{1}{\sqrt[3]{n}}\bigg),
\end{aligned} \tag{4.33}
$$

where $\overline{w}$ is the number of wires per link, f_{NoC} is the link frequency, U_{NoC} is the link utilization factor for a 3D-mesh NoC. With n PEs, the average spiking rate of a single PE is given by:

$$
\begin{aligned}
f^{HF}_{p,out,UC} &= \frac{BW^{HF}_{eff,UC}}{n} \\
&= \frac{3\overline{w}(1-\alpha)(\sqrt[3]{n}-1)}{n\sqrt[3]{n^2}}.f_{NoC}.U_{NoC}
\end{aligned} \tag{4.34}
$$

Besides, the maximal firing frequency for a unicast based NoC is expressed as:

$$
f^{HF}_{spike,max,UC} = \frac{1}{T_{refractory}} \cong \frac{1}{10n.T_{cycle}} = \frac{f_{NoC}}{10n}, \tag{4.35}
$$

where $T_{refractory}$ is the period after which a spike is generated; during that time the neuron cannot fire again. T_{cycle} is the link delay ($T_{cycle} = 1/f_{NoC}$). As mentioned above, in order to send a spike, the source node needs to send $n-1$ packets; thus, it takes $n.T_{cycle}$. Here, we are not including the router delay as it is a constant, independent of the network size. In our analysis, we assume $T_{refractory} \cong 10n.T_{cycle}$, similarly to [95].

By dividing (4.34) by (4.35), we can determine how many neurons may fire at the maximal rate. This is represented by K, as given in (4.36):

$$
K^{HF}_{UC} = \frac{f^{HF}_{p,out,UC}}{f^{HF}_{spike,max,UC}} = O\bigg(\frac{1}{\sqrt[3]{n}}\bigg). \tag{4.36}
$$

Multicast and Broadcast Based Routing Schemes

For these routing methods, since each PE only sends a single packet for each spike which is propagated to the other neurons, the number of hops is determined by:

$$TotalDist^{HF}_{MC/BC} \cong n. \tag{4.37}$$

The efficient bandwidth, the average spiking rate of a single PE, the maximal firing frequency, and the K metric for Multicast and Broadcast are calculated by the equations (4.38), (4.39), (4.40), (4.41), respectively:

$$\begin{aligned} BW^{HF}_{eff,MC/BC} &= \frac{3\overline{w}(1-\alpha)(\sqrt[3]{n}-1)}{\sqrt[3]{n}}.f_{NoC}.U_{NoC} \\ &= O(1) \end{aligned} \tag{4.38}$$

$$\begin{aligned} f^{HF}_{p,out,MC/BC} &= \frac{BW^{HF}eff, MC/BC}{n} \\ &= \frac{3\overline{w}(1-\alpha)(\sqrt[3]{n}-1)}{n\sqrt[3]{n}}.f_{NoC}.U_{NoC} \end{aligned} \tag{4.39}$$

$$\begin{aligned} f^{HF}_{spike,max,MC/BC} &= \frac{1}{T_{refractory}} \cong \frac{1}{T_{cycle}\overline{Dist}} \\ &= \frac{\sqrt[3]{n}}{\sqrt[3]{n^2}-1}.f_{NoC} \end{aligned} \tag{4.40}$$

$$K^{HF}_{MC/BC} = \frac{f^{HF}_{p,out,MC/BC}}{f^{HF}_{spike,max,MC/BC}} = O\left(\frac{1}{\sqrt[3]{n^2}}\right). \tag{4.41}$$

In summary, equations (4.33), (4.36), (4.38), and (4.41) demonstrate the effect of the fault rate on the architecture performance (i.e., in terms of efficient bandwidth and spike rate which a given architecture can maintain) when Hopfield neural network is run on. Compared to UC, MC and BC offer higher bandwidth and the number of neurons can fire at the maximum rate.

4.3.2 Performance Analysis of RNDC Neural Network Based on a 3D-mesh

Unicast based Routing

From (4.18), (4.19), (4.20), and (4.21), the efficient bandwidth can be represented as:

$$\begin{aligned} BW_{eff,UC}^{RNDC} &= \frac{3\overline{w}(1-\alpha)\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{3\sqrt[4]{n}\sqrt{n}}.f_{NoC}.U_{NoC} \\ &= \frac{\overline{w}(1-\alpha)\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{\sqrt[4]{n^3}}.f_{NoC}.U_{NoC} \\ &= O(\sqrt[4]{n}). \end{aligned} \tag{4.42}$$

Furthermore, the average spiking rate of a single PE ($f_{p,out}^{UC}$), the maximal spiking rate ($f_{spike,max}^{UC}$), and the K ratio for UC can be depicted as:

$$\begin{aligned} f_{p,out}^{UC} &= \frac{BW_{eff,UC}^{RNDC}}{n} \\ &= \frac{\overline{w}(1-\alpha)\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{n\sqrt[4]{n^3}}.f_{NoC}.U_{NoC} \end{aligned} \tag{4.43}$$

$$f_{spike,max}^{UC} = \frac{1}{10.C.T_{cycle}} = \frac{f_{NoC}}{10\sqrt{n}} = O\left(\frac{1}{\sqrt{n}}\right) \tag{4.44}$$

$$\begin{aligned} K &= \frac{10\overline{w}(1-\alpha)\sqrt{n}\sqrt[3]{n^2}(\sqrt[3]{n}-1).f_{NoC}}{n\sqrt[4]{n^3}.f_{NoC}}.U_{NoC} \\ &= O\left(\frac{1}{\sqrt[4]{n}}\right). \end{aligned} \tag{4.45}$$

For the case of multicast, a packet needs to travel along a 3λ path to reach the first destination, and then plus one hop for each of the remaining. With a total of C destination nodes, the hop count for each packet is therefore determined by:

$$TotalDist^{RNDC}_{3DMesn,MC} = C + \overline{Dist}^{RNDC} = C + 3\lambda \tag{4.46}$$

Therefore, the efficient bandwidth can be formulated as:

$$\begin{aligned} BW^{RNDC}_{eff,MC} &\cong \frac{3\overline{w}(1-\alpha)\sqrt[3]{n^2}(\sqrt[3]{n}-1)}{\sqrt{n}+3\sqrt[4]{n}}.f_{NoC}.U_{NoC} \\ &= O(\sqrt{n}). \end{aligned} \tag{4.47}$$

Moreover, the average spike rate for each PE is given by:

$$\begin{aligned} f^{MC}_{p,out} &= \frac{BW^{RNDC}_{eff,MC}}{n} \\ &= \frac{3\overline{w}(1-\alpha)(\sqrt[3]{n}-1)}{\sqrt[3]{n}(\sqrt{n}+3\sqrt[4]{n})}.f_{NoC}.U_{NoC}. \end{aligned} \tag{4.48}$$

With a link delay T_{cycle}, and average distance between two nodes in the RNDC neural network $\overline{Dist}^{RNDC}$, the maximal spiking frequency is expressed as:

$$\begin{aligned} f^{MC}_{spike,max} &= \frac{1}{T_{cycle}.\overline{Dist}^{RNDC}} = \frac{f_{NoC}}{3\lambda} \\ &= \frac{f_{NoC}}{3\sqrt[4]{n}} = O\left(\frac{1}{\sqrt[4]{n}}\right). \end{aligned} \tag{4.49}$$

From (4.48) and (4.49), the K ratio is given by the following equation:

$$\begin{aligned} K = \frac{f^{MC}_{p,out}}{f^{MC}_{spike,max}} &= \frac{3\overline{w}(1-\alpha)(\sqrt[3]{n}-1)3\sqrt[4]{n}}{\sqrt[3]{n}(\sqrt{n}+3\sqrt[4]{n})}.U_{NoC} \\ &= O\left(\frac{1}{\sqrt[4]{n}}\right) \end{aligned} \tag{4.50}$$

For the case of broadcast, the RNDC system is similar to the case of Hopfield. The only difference is in the network size: C for RNDC, and n for Hopfield. Consequently, the performance metrics are similar to the Hopfield neural network.

In summary, for the RNDC running on the architecture with a link fault rate α, MC offer higher spiking frequency compared to UC and BC. From the assessment analysis for both Hopfield and RNDC neural network topologies, we can see that the link failure causes performance degradation in the communication architecture. This may lead to timing violations of spikes. Therefore, a low-latency fault-tolerant routing method is imperative to deal with this issue.

4.4 Conclusion and Discussion

This chapter presented a performance analytical model of 3D-mesh based spiking neuromorphic systems. In fact, since the design and implementation of such neuromorphic systems take a long time, this model aims to help designers to earlier evaluate the system before the actual design. The model was performed under different spike routing methods, two kinds of spiking neural network topologies covering different levels of connectivity, and with and without link-fault injection.

From the analyzed results, we can see that multicast spike routing method shows better results compared to unicast-based multicast and broadcast. Besides, the 3D-mesh interconnect architecture performed better 2D-counterpart which was concluded to be the most suitable architecture compared to tree, shared bus, and point-to-point ones [23, 95]. These motivate us to propose efficient multicast spike routings in Chapter 5: two novel multicast routing methods and a new fault-tolerant multicast routing algorithm dealing with inter-neuron connection faults. Furthermore, the implementation and evaluation of the proposed system are presented in Chapters 6 and 7 to validate the analytical model presented in this chapter.

5

K-means Based Multicast Spike Routing Algorithms

In this chapter, spike routing algorithms are presented. We firstly present our first proposed K-means based MultiCast Routing algorithm (KMRC). We demonstrate its key features and advantages and also discuss its weak point. From this, we then present an improvement of KMCR, named Shortest Path K-means based MultiCast Routing (SP-KMCR). This aims to deal with the drawback of KMCR that is high congestion in centroids. In the last subsection, we propose a fault-tolerant multicast routing algorithm based on SP-KMCR, named FTSP-KMCR. This algorithm is proposed to solve interneuron connection faults in NoC based SNN systems.

5.1 K-means Based Multicast Spike Routing Algorithm (KMCR)

In this section, we present our K-means clustering based multicast spike routing for the 3DNoC-SNN system. As mentioned above, the 3D-mesh NoC is suitable for stacking multiple 2D NN layers together in a scalable fashion to create large-scale networks. In SNN, one neuron is typically connected to many others. Thus, there is a significant amount of one-to-many communications between neuron processing cores.

5.1.1 The Proposed Routing Algorithm (KMCR)

The proposed routing algorithm is based on a combination of the K-means clustering method and the tree-based routing [124, 155]. The tree-based [134] is a popular method used in multicast communication. In this routing mechanism, a destination group is partitioned from the source node to form a ”tree” routing path of messages. One major drawback of the tree-based method is high traffic contention due to the high probability of packet blocking at intermediate nodes [134]. To deal with this problem, we adopt the K-means for partitioning a destination set. Employing the K-means comes from the observation that post-synaptic neurons are often neighbors of each other; previous work [156] indicated that SNNs have high inter-neuron communication locality. This enables a neuron group located within the same region to share incoming spikes. Hence, when mapped onto a 3D-NoC system, neurons in a layer are distributed in one core or nearby ones. This enables taking full advantage of K-means to get an effective partition resulting in overall traffic load balance. Besides, K-means also guarantees the smallest number of hops from each destination to its centroids.

The flow chart of the proposed routing algorithm is shown in Figure 5.1. The algorithm firstly partitions destinations into several subgroups. We adopt the K-means clustering mechanism to find a centroid of each subgroup and its labeled targets, in which the centroid is a node with minimal mean distance to all the others in that subgroup. From this, the first part of the routing tree is formed from source nodes to the centroids, and the

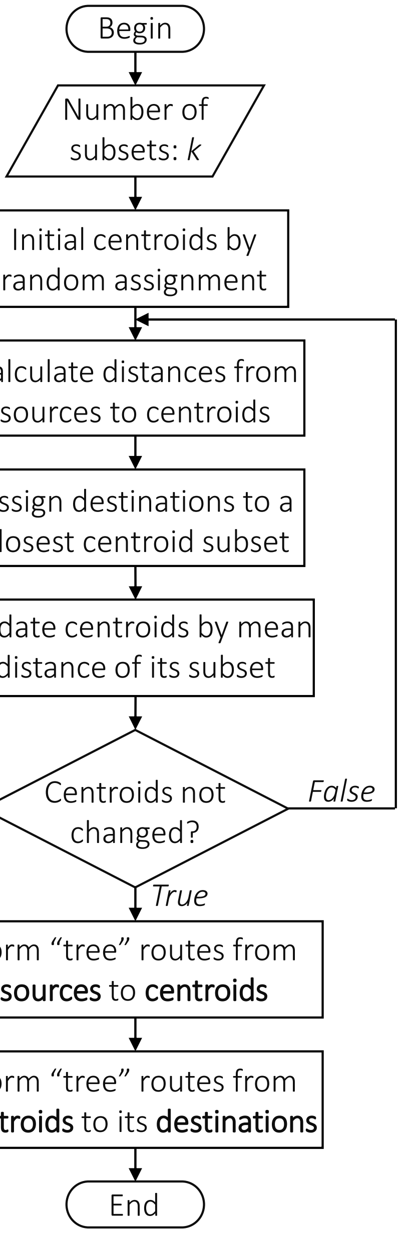

Begin
Number of subsets: *k*
Initial centroids by random assignment
Calculate distances from sources to centroids
Assign destinations to a closest centroid subset
Update centroids by mean distance of its subset
Centroids not changed?
False
True
Form "tree" routes from **sources** to **centroids**
Form "tree" routes from **centroids** to its **destinations**
End

Figure 5.1: Flowchart of the proposed routing algorithm.

other part is a spanning sub-tree from the centroids to their destinations.

To determine the centroids, the algorithm first randomly selects them from the available targets. Then, the algorithm computes the following steps:

- The distances from each destination to the centroids are calculated using the Manhattan distance as shown in line 10 of Algorithm 1.

- Based on these distances, the destinations are, then, assigned to a subgroup which has the nearest centroid.

- Finally, after the subgroups are temporarily formed, the position of the centroids are updated by taking the mean of all its elements. The iteration does not end until centroids are not changed after updating.

The pseudo-code of the implemented algorithm is shown in Algorithm 1.

After determining the centroids, routing paths from source nodes to their targets are formed in two stages. At the first stage, we employ Dimension Order Routing (DOR - a common method for NoCs [20]) to determine routes from each source to the centroids. From here, same routes from a given source to centroids are merged. This leads to a reduction in the numbers of spike packets that need to be transmitted from the source compared to the unicast-based method. Using a particular kind of DOR depends on application mapping method, which will ensure an optimized and balanced traffic (explained more in an example, shown in Figure 5.2). At the end of this stage, a part of ”tree” from the source to centroids is formed. Second, the similar routing calculations in the first stage are computed to establish the other part of ”tree” from centroids to its destinations. After the two stages, the ”tree” route from a given source node to its destination is constructed, and the computed routing information is used to update the routing tables attached to routers.

For an easy explanation and to better understand the algorithm, we show in Figure 5.2 an example of 18×18 fully connected SNN application mapped onto a 6×3×2 3DNoC-SNN system. As shown in the figure, the nodes in *L1* (source nodes) send their outputs

Algorithm 1: KMCR multicast routing pseudo-code

```
/* Input and output                                                          */
Input: // Source node address ($S$), destination node addresses ($T$), and the
        number of centroid nodes ($k$)
        $S = \{s_1(x_1, y_1, z_1),\ s_2(x_2, y_2, z_2), ..., s_n(x_n, y_n, z_n)\}$
        $T = \{t_1(x_1, y_1, z_1),\ t_2(x_2, y_2, z_2), ..., t_m(x_m, y_m, z_m)\}$
        $k$
Output: // Routing paths from S to T
        $P = \{p_1(s_1 \rightarrow T), p_2(s_2 \rightarrow T), \dots p_n(s_n \rightarrow T)\}$

/* Centroid node assignment                                                  */
// Initial centroid nodes by randomly select from $T$
foreach $c_i \in C$ do
    $c_i \leftarrow t_j \in T$
end
// Evaluate centroid nodes
do
    // Calculate the distance between $t_i \in T$ to $c_j \in C$
    foreach $t_i \in T$ do
        $d(t_i, c_j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j|$
    end
    // Assign each destination to its centroid by minimum distance
    foreach $t_i \in T$ do
        $l(t_i) \leftarrow argmind(c_i, t_j)$
    end
    // Update centroid
    foreach $c_i \in C$ do
        $c_i \leftarrow update(mean(t_{ij}))$
    end
while $C$ != $const$;

/* Creating routing tree from each source to centroids                       */
foreach $s_i \in S$ do
    $p(s_i, c_j) \leftarrow DOR\_based\_tree(s_i, c_j)$
end

/* Creating routing tree from each centroid to its destinations              */
foreach $c_i \in C$ do
    $p(c_i, t_j) \leftarrow DOR\_based\_tree(c_i, t_j)$
end
```

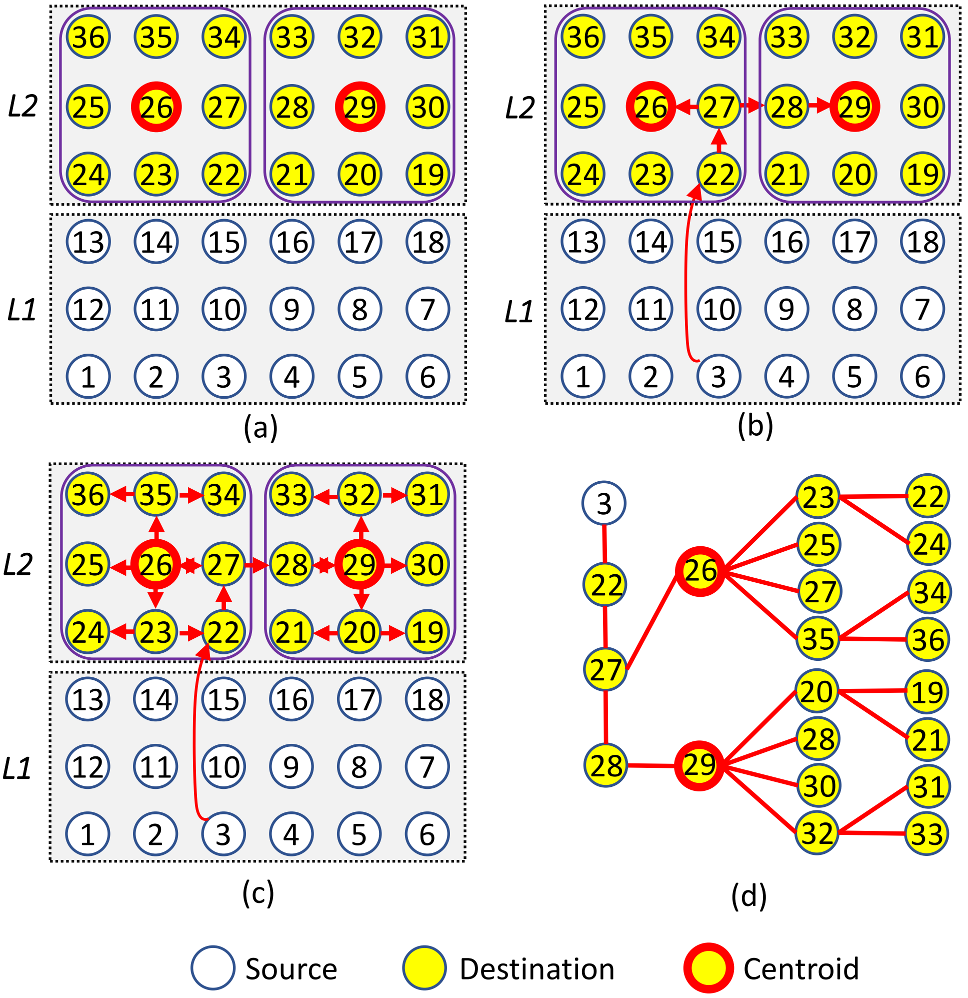


Figure 5.2: Example of the proposed routing algorithm for a 6×3×2 3DNoC-SNN system, where nodes in *L1* send spike packets to all nodes in *L2*: (a) destinations are partitioned by adopting K-means clustering with centroids *26* and *29*, (b) the formation of the first path of the tree from a given source (node *3*) to centroids, (c) the second part of the tree from centroids to its destinations, (d) the routing tree from the given source to destinations.

to all neurons in *L2* (destination nodes). In a particular case, the source node 3 in layer *L1* needs to send spike packets to all nodes in layer *L2*. With the number of clusters *k = 2*, the destination set is partitioned into two subsets with *26* and *29* as centroids (Figure 5.2 (a)). The "tree" route from the source to both centroids is then determined as shown in Figure 5.2 (b). In this mapping method, the ZYX version of the DOR is selected. This enables alleviating traffic contention of the intermediate nodes in the first layer. If either XYZ or YXZ would be used, the source nodes need to send spikes to centroids via *11* and *8* leading to high traffic congestion in these intermediate nodes. After the centroid-to-destination routes are computed, the routing "tree" is formed as shown in Figure 5.2(c, d).

5.1.2 Selection of the Optimal Number of Clusters

As we mentioned, the number of clusters (k) needs to be determined before the proposed routing algorithm (KMCR) is applied. Intuitively, when k is small, the destination set is partitioned into big subsets. This may lead to high congestion in the intermediate nodes like the centroids resulting in high congestion in the network. On the other hand, when k is large, each source node may send multiple copies of a given packet to the centroids. This may also result in high latency. When k equals the number of destinations, our routing algorithm becomes unicast-based multicast. It is important to mention that selection of k mainly depends on the distribution of destination nodes resulted by mapping methods.

Fortunately, there are several good observations that can be taken to select the optimal k. First, as mentioned above, SNNs have a high inter-neuron communication locality. This leads to a situation where neurons in the same group (layer) are mapped onto nearby neural processing cores. This enables k-means clustering algorithm to work efficiently. Second, the number of destination nodes for a typical SNN applications is not large. In fact, the number of neurons in a layer can be hundreds to thousands (efficiency of deep learning based on the multiple-layer model instead of a very large number of neurons in some-layer model), so they can be accommodated in tens of cores (a core contains hundreds of neurons [16], our target for SNPC is 256). Therefore, after mapping the SNN application, the number of clusters can be determined by visualizing the destination distribution. However, in order to select the optimal k for a particular case, it is necessary to evaluate the performance of the system under other different values of k.

Based on the observations mentioned above, the optimal k can be determined by the two following steps:

- Step 1: After mapping SNN application, find the number of clusters by visualizing the destination set.

- Step 2: Evaluate the system varying the values of k (including the number of clusters found in step 1, and some other values) to choose the best case.

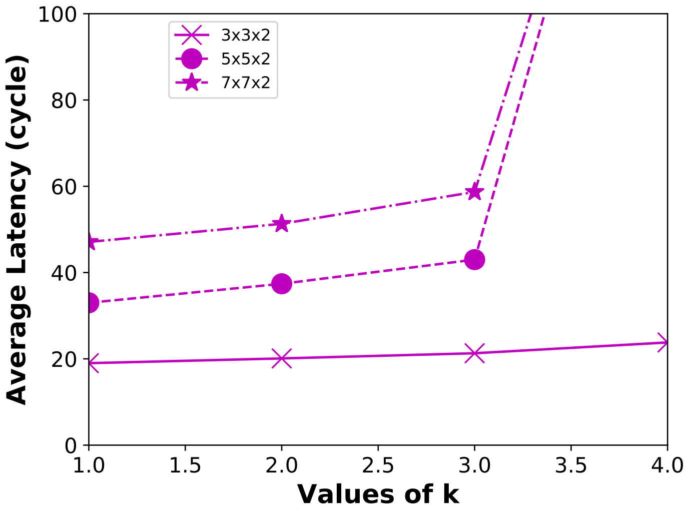


Figure 5.3: Average latency under varying the values of k

As a case study, we evaluated our system performance with different network sizes under variations of k to find the optimal value of k. These experiments are designed to simulate SNNs with two layers, each of which is mapped into separate layers (layer-to-layer mapping). All nodes in the first NoC layer send packets to all the ones in the second layer, as shown in Figure 5.2. This mapping method is a considerable mapping method to take full advantages of the 3DNoC. Furthermore, we can consider that whether an observed cluster should be continuously partitioned or not. Three network sizes are considered: 3×3×2, 5×5×2, and 7×7×2 corresponding to maximal neuron numbers of 2304, 6400, and 12544 in a layer respectively (with 256 neurons/core). For this mapping method, it is easy to find that the number of clusters equals 1 (3×3, 5×5, 7×7 areas).

Figure 5.3 shows the average latency of the systems under different values of k. The evaluation results show that k = 1 is optimal (equal the number of clusters observed). The increase of k leads to high average latency thanks to multiple copies of a given packet (to centroids) injected to networks. This also means that k should be smaller than the number of clusters observed. However, these experiments are special cases, in which destination sets have a nice shape. In other cases, with different SNN architectures and mapping methods, we need to evaluate the system under different k to select the best choice.

5.1.3 Weakpoint

In the KMCR, the source node sends spike packets to centroids which then deliver the spikes to destinations. The use of centroids is to guarantee that the overall distance from them to the destinations is minimum. However, this may cause traffic congestion on the link to the centroids since the traffic from different sources is concentrated there.

5.2 Shortest Path K-means Based Multicast Routing Algorithm (SP-KMCR))

To deal with drawback of KMCR, we propose a new routing method, named Shortest Path K-means based MultiCast Routing algorithm (SP-KMCR) [157, 158]. In the KMCR, the source node sends spike packets to centroids which then deliver the spikes to destinations. The use of centroids is to guarantee that the overall distance from them to the destinations is minimum. On the other hand, in SP-KMCR, after destination subsets are determined by adopting K-means, from a given source, we first calculate the numbers of hops from the source to all the nodes in the subsets. For each subset, we then select a node which has the shortest path to the source (e. g., nodes *22* and *21* in Figure

5.4 (a)). Contrary to the KMCR, the source sends its spike packets to the shortest path node of each subset instead of the centroid node to form the first part of the routing tree, therefore named SP-KMCR. The other part of the routing tree is formed from SP nodes to its destinations, as shown in Figure 5.4 (c). Furthermore, it is worth mentioning that our new method requires more computations for finding the shortest path compared to the KMCR. However, the computations in both KMCR and SP-KMCR are executed off-line. Therefore, the runtime overhead is the same for both algorithms.

5.3 Fault-tolerant Shortest Path K-means Based Multicast Routing Algorithm (FTSP-KMCR)

5.3.1 Proposed Fault-tolerant Routing Algorithm

The shortest path fault-tolerant multicast routing algorithm is based on the SP-KMCR [159, 160]. The basic idea of the FTSP-KMCR is as follows: (1) off-line computations of a primary routing tree from a given source node to its destinations and backup routing branches are performed. (2) After the off-line calculation, the routing tables are configured.

The illustration of the primary and backup routing branches is shown in Figure 5.5. When a faulty primary branch is detected, some pre-planned backup branch(es) is (are) used to bypass the faulty links. The SP-KMCR mechanism is used to calculate the branches (red) in the primary tree. On the other hand, the backup branches are alternative routes of the primary ones. For a considered router (i.e., ”son”), the backup branches (green) are computed for the cases of faults occurring in primary connections. For example, when the father-to-son primary connection is faulty (i.e., pb_1), bb_1 and bb_2 are the backup branches used for maintaining the traffic between the ”father” and ”son”. This is the same for the case where both pb_2 and pb_1 are faulty.

In our proposed algorithm, the computations of primary and backup routes are critical computational tasks. These calculations are performed off-line. This allows to reduce the runtime overhead of the proposed routing algorithm; hence avoiding any possible timing violations in SNNs. As presented in algorithm 3, the source and destination addresses

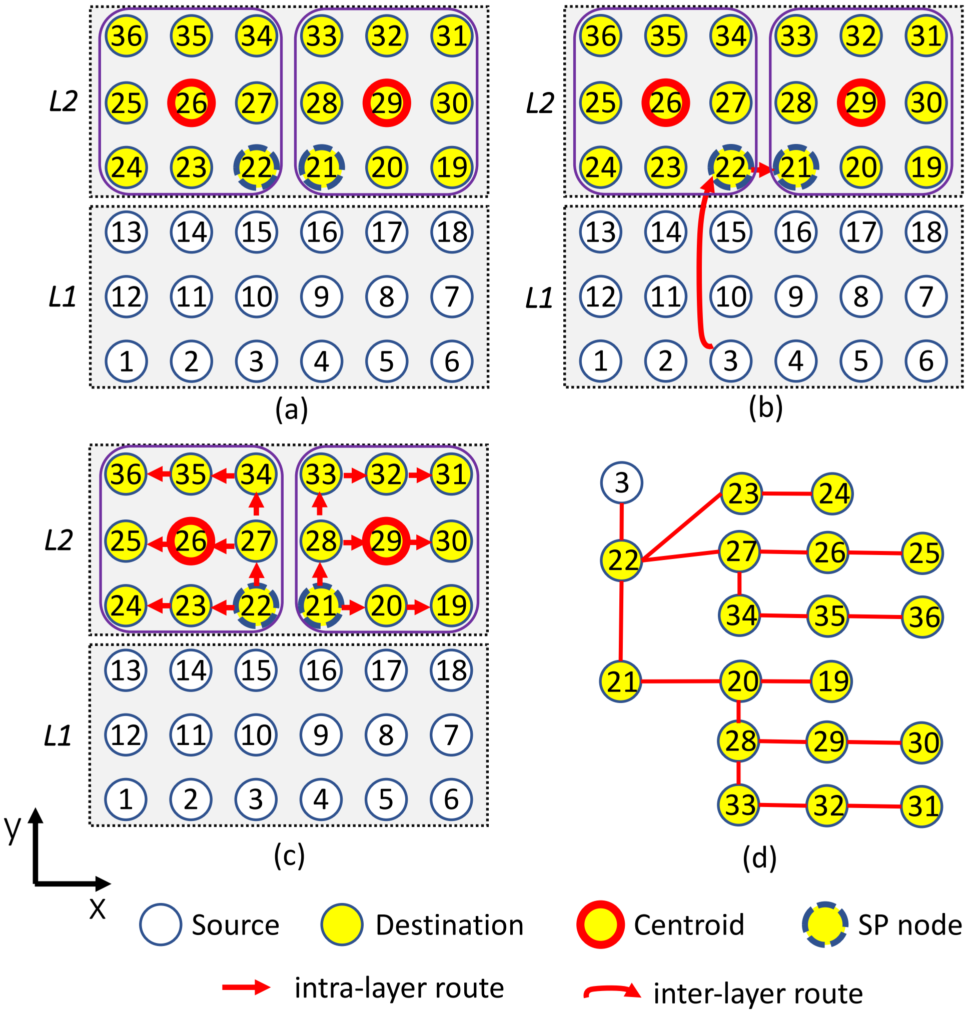


Figure 5.4: Example of SP-KMCR for a 6×3×2 3DNoC-SNN system, where nodes in *L1* send spike packets to all nodes in *L2*: (a) destinations are partitioned by adopting K-means clustering with centroids *26* and *29*, (b) the formation of the first path of the tree from a given source (node *3*) to shortest path node of each subgroup (SP node), (c) the second part of the tree from SP nodes to its destinations, (d) the routing tree from the given source to destinations.

Algorithm 2: SP-KMCR multicast routing algorithm.

```
/* Input and output                                                          */
Input: // Source node address ($S$), destination node addresses ($T$), and the number of subsets ($k$)
        $S$ = $\{s_1(x_1, y_1, z_1),\ s_2(x_2, y_2, z_2),...,s_n(x_n, y_n, z_n)\}$
        $T$ = $\{t_1(x_1, y_1, z_1),\ t_2(x_2, y_2, z_2),...,t_m(x_m, y_m, z_m)\}$
        $k$
Output: // Routing tree from each of source node to the destinations
        $P = \{p_1(s_1 \rightarrow T), p_2(s_2 \rightarrow T), ...p_n(s_n \rightarrow T)\}$
/* Partion the destination set (T) into k subsets                            */
// Initial centroid nodes by randomly select from $T$
foreach $c_i \in C$ do
    $c_i \leftarrow t_j \in T$
end
// Evaluate centroid nodes and their labeled nodes
do
    // Calculate the distance between $t_i \in T$ to $c_j \in C$
    foreach $t_i \in T$ do
        $d(t_i, c_j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j|$
    end
    // Assign each destination to its centroid by minimum distance
    foreach $t_i \in T$ do
        $l(t_i) \leftarrow argmind(c_i, t_j)$
    end
    // Update centroid
    foreach $c_i \in C$ do
        $c_i \leftarrow update(mean(t_{ij}))$
    end
while $C$ != $const$;
/* Finding k shortest-path nodes (SP nodes) for every single source node     */
foreach $s_i \in S$ do
    foreach $t_i \in T^k$ do
        $d(s_i, t_j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j|$
    end
    $sp_i \leftarrow min(d(s_i, t_j))$
end
/* Creating routing path from each source node to its SP nodes               */
foreach $s_i \in S$ do
    $p(s_i, sp_j) \leftarrow DOR\_based\_tree(s_i, sp_j)$
end
/* Creating routing tree from each SP node to its destinations               */
foreach $sp_i \in SP$ do
    $p(sp_i, t_j) \leftarrow DOR\_based\_tree(sp_i, t_j)$
end
```

Algorithm 3: Off-line calculations of the primary and backup branches.

```
/* Input and output */
Input: // Source node address ($S$), destination node addresses ($T$), and the number of subsets ($k$)
        $S = \{s_1(x_1, y_1, z_1),\ s_2(x_2, y_2, z_2),...,s_n(x_n, y_n, z_n)\}$
        $T = \{t_1(x_1, y_1, z_1),\ t_2(x_2, y_2, z_2),...,t_m(x_m, y_m, z_m)\}$
        $k$
Output: //Primary (pr) and backup (bk) branches from S to T
        $P_{pr} = \{p_{pr,1}(s_1 \rightarrow T), p_{pr,2}(s_2 \rightarrow T), ...p_{pr,n}(s_n \rightarrow T)\}$
        $P_{bk} = \{p_{bk,1}(s_1 \rightarrow T), p_{bk,2}(s_2 \rightarrow T), ...p_{bk,n}(s_n \rightarrow T)\}$

/* Centroid node assignment */
// Initial centroid nodes by randomly select from $T$
foreach $c_i \in C$ do
    $c_i \leftarrow t_j \in T$
end
// Evaluate centroid nodes
do
    // Calculate the distance between $t_i \in T$ to $c_j \in C$
    foreach $t_i \in T$ do
        $d(t_i, c_j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j|$
    end
    // Assign each destination to its centroid by minimum distance
    foreach $t_i \in T$ do
        $l(t_i) \leftarrow argmind(c_i, t_j)$
    end
    // Update centroid
    foreach $c_i \in C$ do
        $c_i \leftarrow update(mean(t_{ij}))$
    end
while $C$ != $const$;

/* Finding the shortest paths */
foreach $s_i \in S$ do
    foreach $t_i \in T^k$ do
        $d(s_i, t_j) = |x_i - x_j| + |y_i - y_j| + |z_i - z_j|$
    end
    $sp_i \leftarrow min(d(s_i, t_j))$
end

/* Creating primary and backup branches */
// from each source to SP node
foreach $s_i \in S$ do
    $p_{pr}(s_i, sp_j) \leftarrow DOR^{v.1}\_based\_tree(s_i, sp_j)$
    $p_{bk}(s_i, sp_j) \leftarrow DOR^{v.\neq 1}\_based\_tree(s_i, sp_j)$
end
// from each SP node to its destinations
foreach $sp_i \in SP$ do
    $p_{pr}(sp_i, t_j) \leftarrow DOR^{v.1}\_based\_tree(sp_i, t_j)$
    $p_{bk}(sp_i, t_j) \leftarrow DOR^{v.\neq 1}\_based\_tree(sp_i, t_j)$
end
```

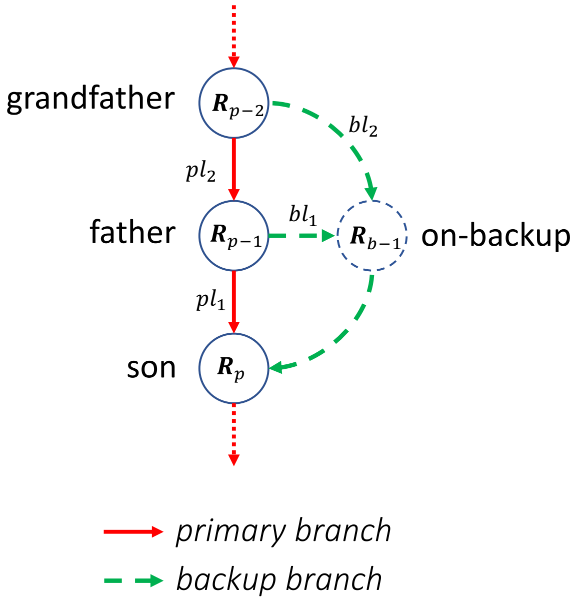


Figure 5.5: Primary and backup branches.

(S, T) and the number of subsets (clusters) (k) are pre-defined as inputs, while output parts are a primary tree (P_{pr}) from each source to destinations and backup branches (P_{bk}). After that, the routing computation is done according to the following steps:

- **Step 1:** from destination addresses, destination subsets are determined by adopting k-means, as shown in lines *6-19*.

- **Step 2:** finding the shortest path from each source to a node (named $sp_i \in SP$) in each subset (with k subsets T^k, a given source node has k SP nodes), as depicted in lines *20-25*.

- **Step 3:** the first part of the primary tree is formed from the source node to SP ones. This is done by adopting dimension order routing (DOR) algorithm [161] from the source to each SP node, then merge with the same route. Alternative variations of the DOR are then adopted to calculate backup branches in order to guarantee that backup branches are separated from the primary routes. For example, if the formation of the primary tree uses DOR of ZYX, the backup branches use other variations of the DOR such as YZX or XZY.

- **Step 4:** following the same computation in step 2, the second part of the primary tree from SP nodes to their destinations in the same group and backup branches are calculated.

After the primary and backup routes are defined, they are used to configure the routing tables in routers. The pre-defined primary and backup routes are suitable for deploying SNN applications since the SNNs are also pre-defined and mapped into the SNN system. Furthermore, this guarantees that the computation overhead of backup branches does not affect the recovery time of the proposed routing algorithm, and also reduces the required hardware cost of the system.

5.3.2 Fault Management Algorithm

After the routing information is configured, the fault-management algorithm is implemented to handle incoming packets, as shown in Figure 5.6. For a given incoming packet, *fault_flag_val* is extracted to indicate whether the packet is in the primary or backup branch. At the same time, the source address is also used to compute its expected primary output port. In the case where *fault_flag_val* = 0 (i.e., the router plays the role of ”father” or ”grandfather”), the calculated *output_port* is then determined to be faulty or not. If it is not faulty, the packet is forwarded to the calculated output port in the primary branch. Otherwise, *output_port* is switched to use a backup_branch, and the *fault_flag_val* is also initiated to inform the next on-backup routers that this packet is on the backup branch. In the case where *fault_flag_val* $\neq 0$ (i.e., the router role is as a on-backup or ”son” router), the *output_port* is routed through the backup route, and *fault_flag_val* is also decreased by one.

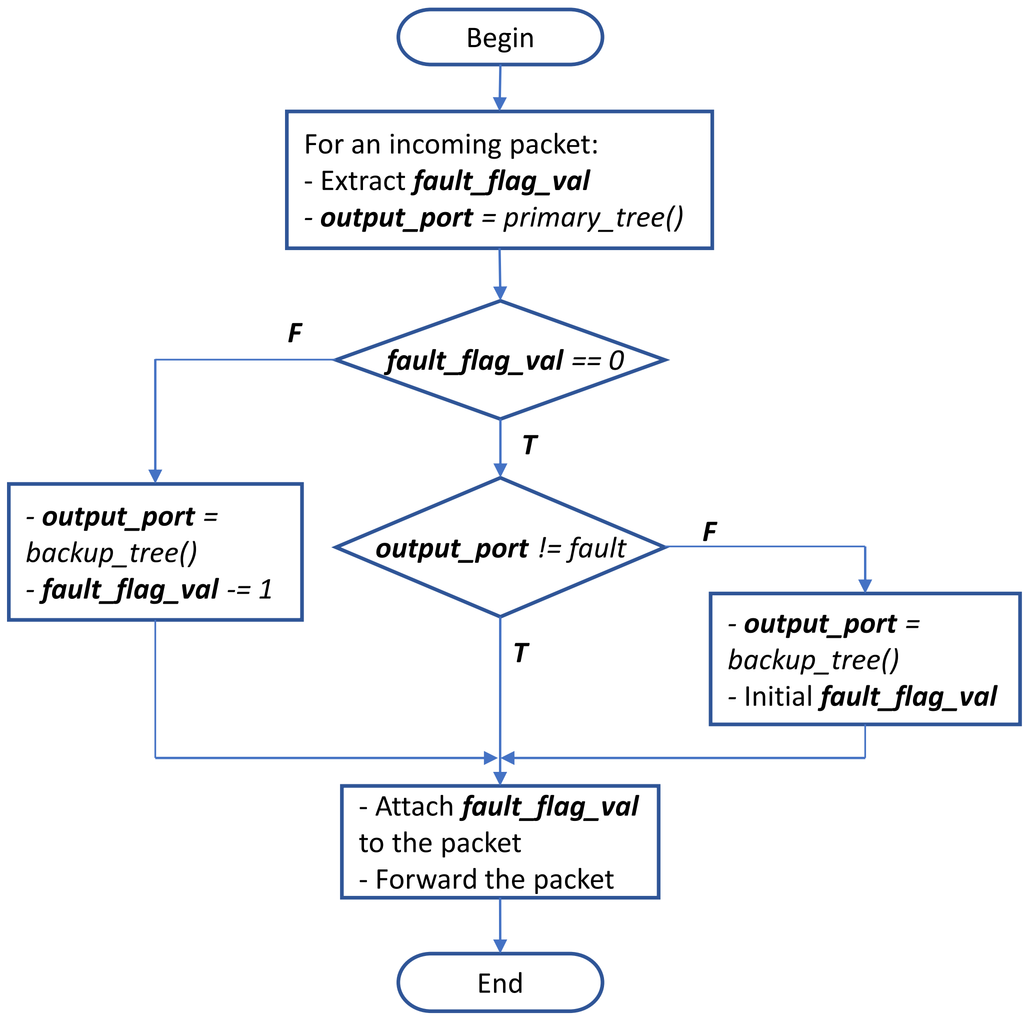


Figure 5.6: Fault-management algorithm applied for ”son”, on-backup, ”father” and ”grandfather” routers.

5.4 Conclusion

In summary, this chapter has presented spike routing algorithms for the 3DNoC-SNN system. All the routing algorithms are based on tree-based multicast routing combined with k-means clustering. They are two multicast routing algorithms, named KMCR and SP-KMCR, and a fault-tolerant multicast routing algorithm, named FTSP-KMCR. In the next chapter, we will present the overall 3DNoC-SNN architecture and its main components, where the proposed routing algorithms are integrated.

6

Towards Scalable Spiking Neuromorphic Architecture

In this chapter, we present the proposed architecture for spiking neural network systems based on 3D-NoC. We first describe the proposed system architecture. After that, two main components of the system are presented; they are a spiking neural processing core (SNPC) and a multicast 3D router (MC-3DR). Finally, considerations of application deployment including application mapping and practical encoding methods are also discussed.

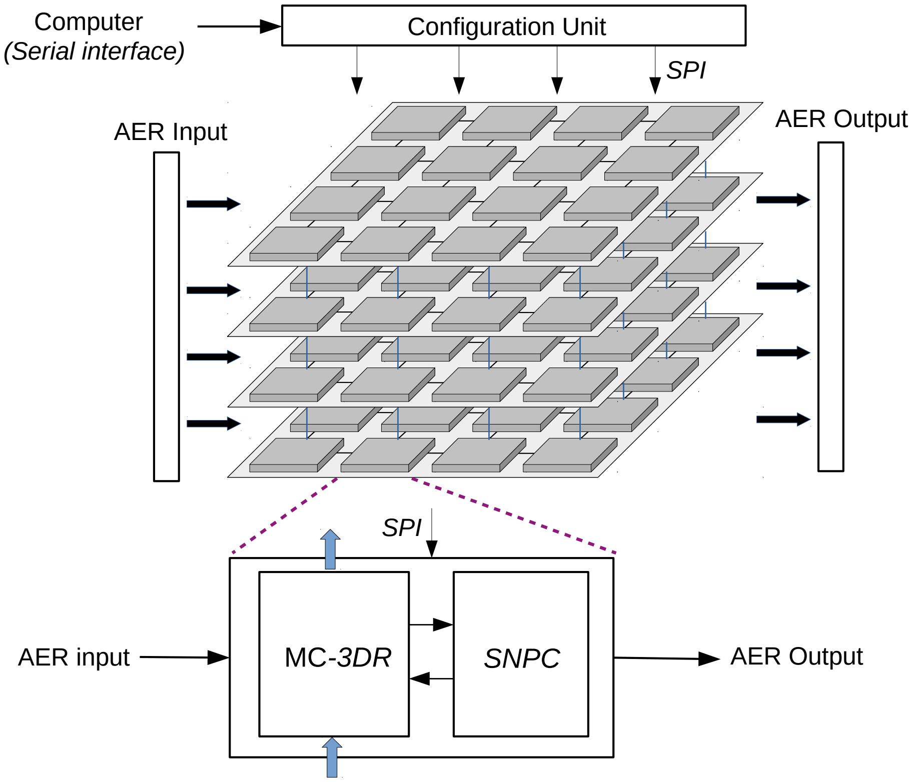


Figure 6.1: Block diagram of system architecture.

6.1 System Architecture

We firstly describe the proposed 3DNoC-SNN architecture which is based on low-latency multicast routing schemes for spike traffic routing. The high-level view of the system architecture is shown in Figure 6.1. As shown in Figure 6.1, The system consists of several stacked 2D layers of spiking neural tiles (4 × 4 2D layers of spiking neural tiles stacked together are shown as an example) and is based on our earlier 3D-NoC architecture [162–166]. A spiking neural tile composes of a spiking neural processing core (SNPC) and a multicast router (MC-3DR). In the context of SNN, a spiking neuron refers to a SNPC, the inter-neuron connectivity is implemented in the form of transmitting spikes (packets) via the scalable 3D-NoC, and the topology refers to the way the neurons are interconnected within the network. Each SNPC within the 3DNoC-SNN is responsible for processing incoming spikes by using an array of spiking neurons (right side of Figure 6.2). In the remaining parts of this chapter, we describe the main components of the 3DNoC-SNN system.

6.1.1 Topology

In our architecture, we employ 3D-Mesh topology because of scalability and high performance. In [23, 95], authors analyzed different 2D interconnect architecture (i.e., mesh NoC, tree, shared bus, and point-to-point) for neural networks over various spike routing protocols. The results showed that 2D mesh NoC with multicast routing is the most suitable for SNNs. As presented in Chapter 4, 3D-mesh architecture outweighs 2D counterpart. This is also verified under experiments in the next chapter.

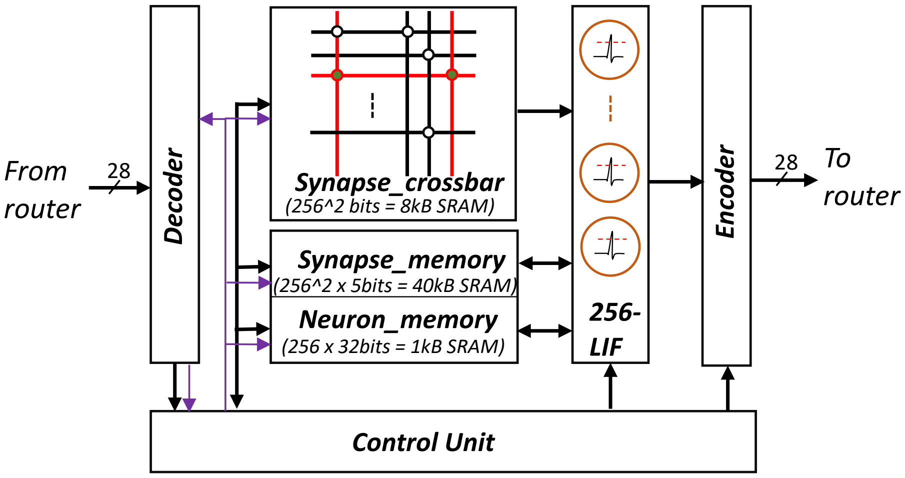
6.1.2 System interface

The proposed system uses two interface blocks to connect outside. First, we adopt address-event presentation (AER) buses to handle input and output events. AER is a popular four-phase handshake communication protocol, considered to be a standard for SNNs [125, 167]. While AER input is used to feed input spikes into the system, AER output is used for monitoring output or connect to other chips in the case of large-scale SNN implements.

Second, the configuration unit is dedicated to system configuration purpose. It receives configuration information from the computer via a serial interface. This information is then sent to neuron tile for configuration via SPI. The information configuration composes of two main parts:

- The first part is the routing path which is for configuring routing tables. This information is the result of the off-line computation of the proposed multicast routing algorithms (see Chapter 5).

- The second part is for the configuration of SNPC. This information contains configuration relating to the synapse and neuron model. The configuration information is mainly: (1) the content of the LUT, used in the decoder. It defines the topology of SNNs. (2) for configuration of memories: bit-lines are configured for the synapse



5bits synapse register format

Input type [0]	Synaptic strength [1:4]

32bits neuron register format

Membrane potential [0:7]	Threshold [8:15]	Leaky value [16:23]	Reset value [24:31]

Figure 6.2: Spiking Neuron Processing Core (SNPC) architecture.

crossbar. Synaptic weights in *syn_mem* for the case of off-chip learning implementations. Neural parameters such as membrane potential of neurons, the threshold, leaky value, and refractory period.

6.2 Spiking Neuron Processing Core (SNPC)

The Spiking Neuron Processing Core (SNPC), depicted in Figure 6.2, is the primary processing unit in the 3DNoC-SNN system. The core composes of several main modules:

- *Decoder* determines post-synaptic neurons for each incoming spike (packet). After arriving the destination neural tile, the incoming spike is forwarded to local SNPC by the local router. Based on ”neuron ID” extracted from the spike packet, the decoder looks up in a LUT to determine the post-synaptic neurons. This information is sent to the Control Unit for neural computation.

- *Control Unit* is designed to control the overall operation of the neural core. It controls both configuration and operation modes of the neural core. It guarantees to update neurons during a single time step.

- *Synapse Crossbar* includes a cross-point array of synapses. Each synapse stores a bit which is able to read, set, or reset, presenting a connection (synapse) between a row (axon) and a column (dendrite). It is read for neural computation and written to after the decode is completed.

- *Syn_mem (Synaptic memory)* stores synaptic information which is used for configuration of the crossbar and synaptic strengths. It is updated in training phase and read in inference operation.

- *Neu_mem (Neural memory)* is used for neural parameters. The parameters are read for neural computations. After the computations, they are updated to store the current status of neurons.

- *LIF Array* is the main computation unit of the neuron core where neural calculations are performed. Data read from the synaptic crossbar, syn_mem, and neu_mem are computed in this unit. Here, multiple leaky-integrate and fire (LIF) neurons are implemented. More precisely, a physical LIF computation unit is implemented while multiple neurons are performed in a sequential manner. This not only takes advantage of the highspeed operation of digital logic but also reduces area cost and power consumption.

- *Encoder* is designed to pack spikes generated from LIF array. After neural computation, if the membrane potential of a neuron exceeds a given threshold, it fires - a spike is generated. This spike is sent to the encoder where the spike is packed into a packet before injecting to the network via the local router.

- *Configuration information* is used for the configuration of the neural core. This information contains configuration relating to the synapse and neuron model. The configuration information is mainly: (1) the content of the LUT, used in the decoder. It defines the topology of SNNs. (2) for configuration of memories: bit-lines

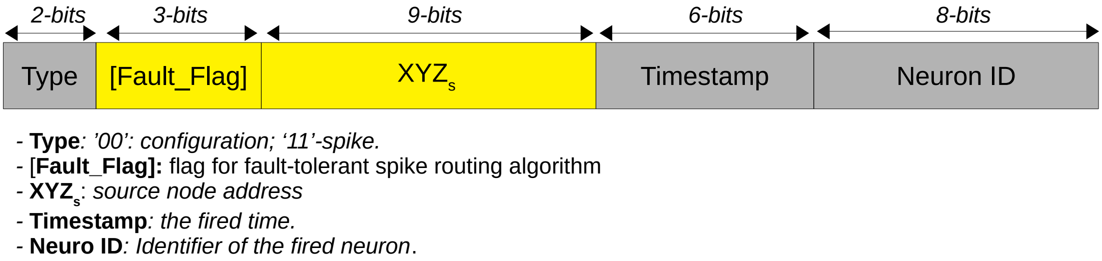

2-bits
3-bits
9-bits
6-bits
8-bits
Type
[Fault_Flag]
XYZ_s
Timestamp
Neuron ID
- Type: '00': configuration; '11'-spike.
- [Fault_Flag]: flag for fault-tolerant spike routing algorithm
- XYZ_s: source node address
- Timestamp: the fired time.
- Neuro ID: Identifier of the fired neuron.

Figure 6.3: Spike packet format.

are configured for the synapse crossbar. Synaptic weights in *syn_mem* for the case of off-chip learning implementations. Neural parameters such as membrane potential of neurons, the threshold, leaky value, and refractory period. The configuration is performed during application mapping before system operates, in which the topology and the parameters of SNNs are determined.

Incoming spikes are first decoded to get the pre-synaptic neural identifiers. Through a crossbar-based synapses, the weight values are accumulated at an array of Leaky Integrate-and-Fire (LIF) neurons [13]. Our choice of this spiking neuron model comes from the trade-off between a design that features most of the biological computational power versus the silicon area and design complexity. The SNPC is based on our recently designed LIF neuron-core prototype [14]. A LIF neuron model is described using five operations namely, synaptic integration, leak integration, threshold, spike firing and reset. From a hardware perspective, its simplicity helps us achieve low area and power consumption.

6.3 Spike packet format

When a neuron fires, a spike packet is transferred to the destination neuron tile through the network. The spike packet format is described as Figure 6.3. It should be mentioned that the packet format is configurable.

- *Type*: It is the header of the packet indicating this packet is either for configuration or spike. If it is '00', this packet is used for system configuration. In this case, the packet body stores configuration data (see Section 6.2). When *Type* = ’11’, it means this is a spike packet. The other values of Type are used for future purpose.

- *Fault_Flag*: This is only used for the fault-tolerant multicast routing algorithm (see Section 5.3). It indicates to the router that the packet arrived via the primary path or backup path. By default, $fault\_flag = 0$. It is initialized at the begin of the backup path (i.e., at node such as ”father” or ”grandfather”. From there, the *fault_flag* value will be decreased by one after every single hop on the backup path and gets to zero at the end (i.e., ”son”).

- XYZ_s: It is the address of the source neuron tile. It is used for spike routing to guarantee that the spike packet is able to reach the destination tile. When a spike packet arrives the input buffer of a router, the address will be extracted for routing computation (i.e., looking up the routing table, see Chapter 5).

- *Timestamp*: In spiking neuron network, the time of the generated spike is used to encode the information (see Section 2.3.1). Whenever the source neuron fires, its spike packet contains Timestamp information of the spike. From there, the information is decoded into the presise time slot of the input spike. This information plays an integral role in Hebbian-based learning rules. For example, in STDP learning rule (Section 2.3.3), the arrival time of the input spike is compared with the spike time of post-synaptic neuron to change the synaptic strength.

- *Neuron ID*: this is the identifier of the pre-synaptic neuron. After a spike packet is delivered to its destination neuron tile, by using a XYZ_s. At this time, Neuron ID is used to determine which are post-synaptic neurons. This process is taken place in the decoder of SNPC. From here, it should be mentioned that the combination of XYZ_s and Neuron ID makes a unique address for each neuron in the whole system.

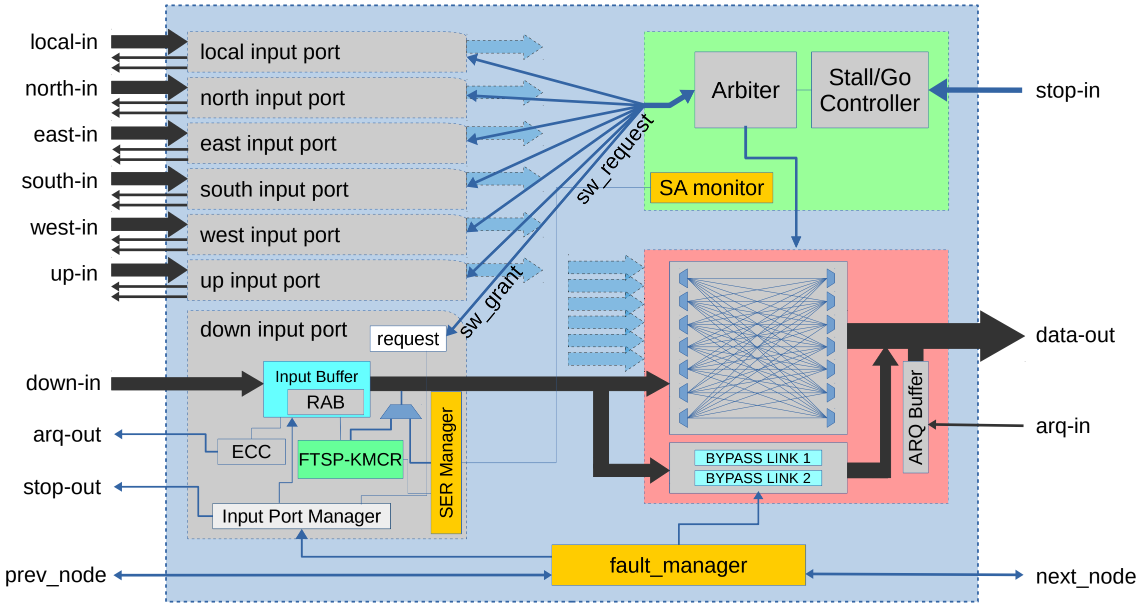


Figure 6.4: Multicast Spike 3D Router architecture (MC-3DR).

6.4 Router Architecture

The multicast 3D router (MC-3DR) architecture is represented in Figure 6.4. Since each neuron can be connected to thousands of other neurons, the MC-3DR supports the proposed multicast routing methods for efficient spike delivery. The MC-3DR is based on our earlier proposed adaptive 3D router architecture (SHER-3DR) [164, 165, 168]. Since the spike times are used to encode information, the MC-3DR should have extremely low latency. Each router in the system has a maximum of 7-input, and 7-output ports, where six input/output ports are dedicated for the neighboring routers and one input/output port is used to connect the switch to the SNPC, the MC-3DR contains seven Input-port modules for each direction in addition to the Switch-Allocator, and the Crossbar module which handles the transfer of spikes to the next SNPC. An Input-port module is composed of two main elements: an input-buffer and a multicast routing module.

The router is designed with four pipeline stages: buffer writing (BW), routing calculation (RC), switch arbitration (SA), and crossbar traversal (CT). At the first stage, an incoming spike (packet) is stored in the *Input Buffer* before being processed. Next, the

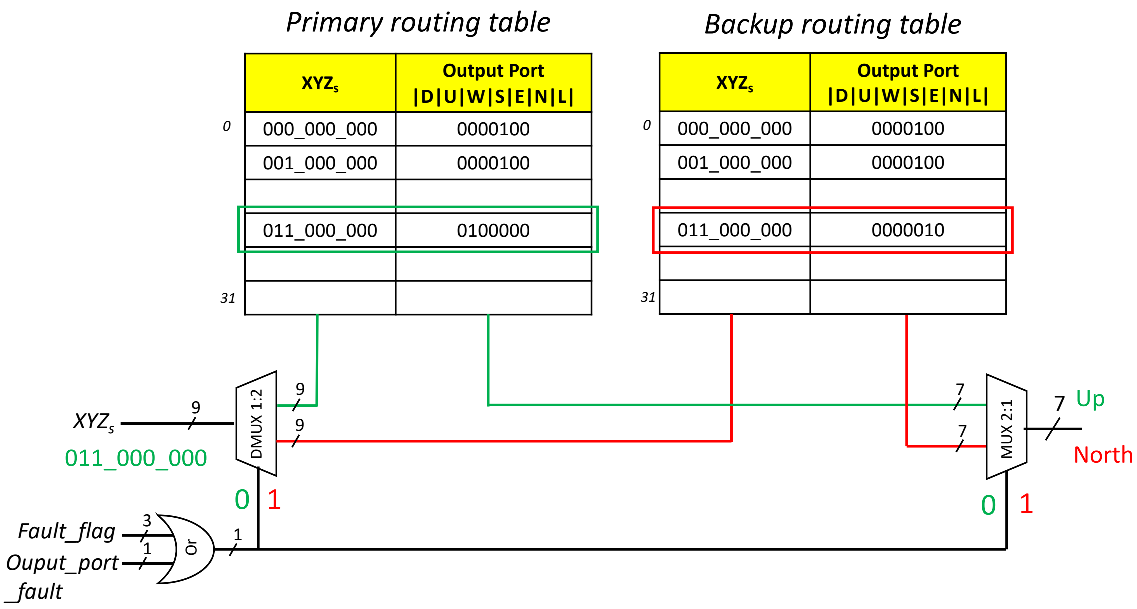


Figure 6.5: Block diagram of routing table architecture. D, U, W, S, E, N, L stand for Down, Up, West, South, East, North, Local respectively.

source address of the packet (X_s, Y_s, Z_s) is extracted and computed to determine which is the output port. After routing computation, a request (*sw_request* signal) is sent to *Switch-Allocator* in order to use the selected output port. The *Switch-Allocator* consists of two main components: *Stall/Go* flow control (the most common use in systems [20]) and *Matrix-arbiter* scheduler. Here, the *Matrix-arbiter* with least recently served priority is employed since it provides fast computation, inexpensive implementation, and strong fairness [20]. Finally, after granted (via *sw_grant* signal), the packet is sent to desired output port passing the crossbar.

6.4.1 Spike routing table

In MC-3DR, one of the main components is the routing computation unit which is also a major concern of this dissertation. As early presented in Chapter 5, routing paths are firstly computed off-line. This aims to not only reduce area cost but also avoid the timing violation caused by extra computation. After the off-line computation, results are used for configuring spike routing tables in routers.

Figure 6.5 described the routing table architecture. After arrived the input buffer, source node address, XYZ_s, is extracted from the input spike packet. The address is then used for determining the desired output port by looking up at routing tables. In the case both *fault_flag* and *output_port_fault* are equal to zero, the desired output port is decided from the primary routing table, and vice versa.

6.4.2 Hard fault tolerance

The proposed router relies on sophisticated recovery techniques based on system reconfiguration with redundant structural resources to handle hard faults in the input-buffers, crossbar, and links [164, 165]These mechanisms aim to alleviate faults occurring in the system.

Fault-tolerant buffer

We inherited a mechanism, named Random Access Buffer (RAB) [164], to solve the deadlock problem in the input buffers. RAB block diagram is described in Figure 6.6. RAB uses a timer for detecting the spike packet being the reason for the deadlock. When the deadlock is detected, the request of the flit will be dropped and RAB then looks for another flit which can be granted. Therefore, RAB is able to recover from transient, intermittent, and permanent faults in the input buffers.

Fault-tolerant crossbar:

In order to deal with faults occurs in the crossbar, we employed our previous work, called Bypass-Link-on-Demand (BLoD) [164]. As shown in Figure 6.7, BLoD is based on providing additional escape channels. When a fault occurs in one or several crossbar links, the links will be disabled, and an appropriate number of the additional links (bypass channels) are enabled.

Fault-tolerant TSV

In our system, we use TSVs [169] as vertical connections between layers of neural tiles. However, because of the high defect rate and clustering distribution, fault-tolerance in TSVs has become a major concern in commercial TSV-based architecture. We, therefore,

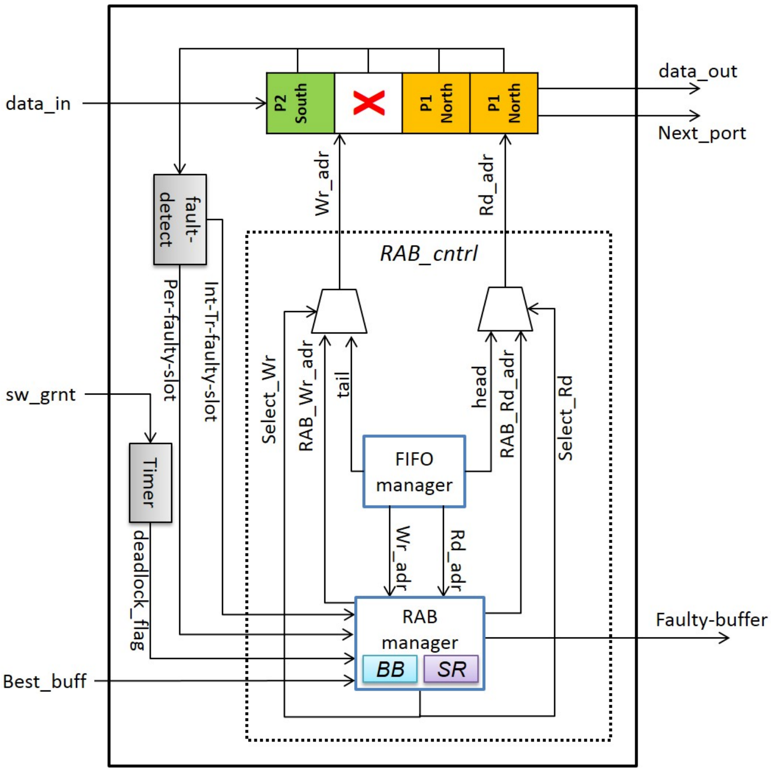


Figure 6.6: Block diagram of Random Access Buffer (RAB) [164].

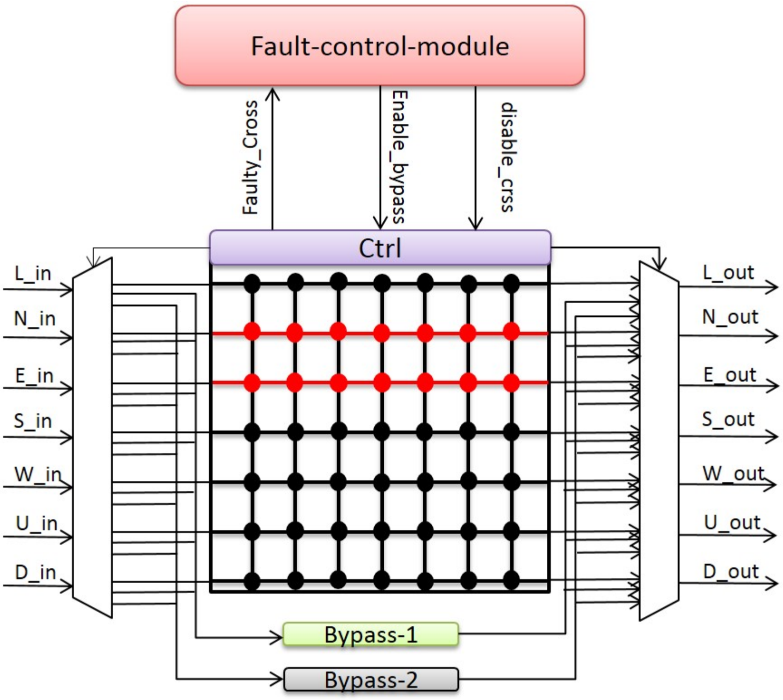


Figure 6.7: Block diagram of Bypass-Link-on-Demand [164].

adopt our previous work which is an architecture sharing TSV clusters [165]. When a TSV cluster defects, the router will borrow a healthy one from its neighbor. This enables our system can handle TSV defect without redundancy of TSVs. As shown in Figure 6.8, each router has TSV clusters and additional supporting modules that perform the sharing algorithm.

6.5 Application deployment

6.5.1 Application mapping methods

It should be mentioned that the mapping strategy of SNNs onto NoC based system plays an important role in deploying SNN applications. It affects not only the overall performance but also the power consumption of the whole system. In [170], the authors proposed two mapping methods: (1) a relatively conventional approach that puts highly communicating tasks together, and (2) an approach based on active degrees of neurons. In this work, we mapped SNNs onto the proposed system in a layer-to-layer fashion to take full advantage of the proposed routing algorithm and the 3D mesh NoC topology, as shown in Figure 7.3. In this mapping method, the neurons in the same network layer are placed in the same system layer, and neurons only send their spike to the ones in the next layer. This approach offers multiple parallel connections between layers (vertical connections), less congestion, and low spike latency when compared to the 2D integration method [165].

6.5.2 Input-data-to-spike conversion methods

Unlike conventional artificial neural network, input data needs to be converted/encoded into spike trains before fed into the spiking neuromorphic systems. In this section, we present widely used conversion methods which can be categorized into two groups: (1) converting from original data sets (2) using converters. Several examples of conversion methods are shown in Figure 6.10.

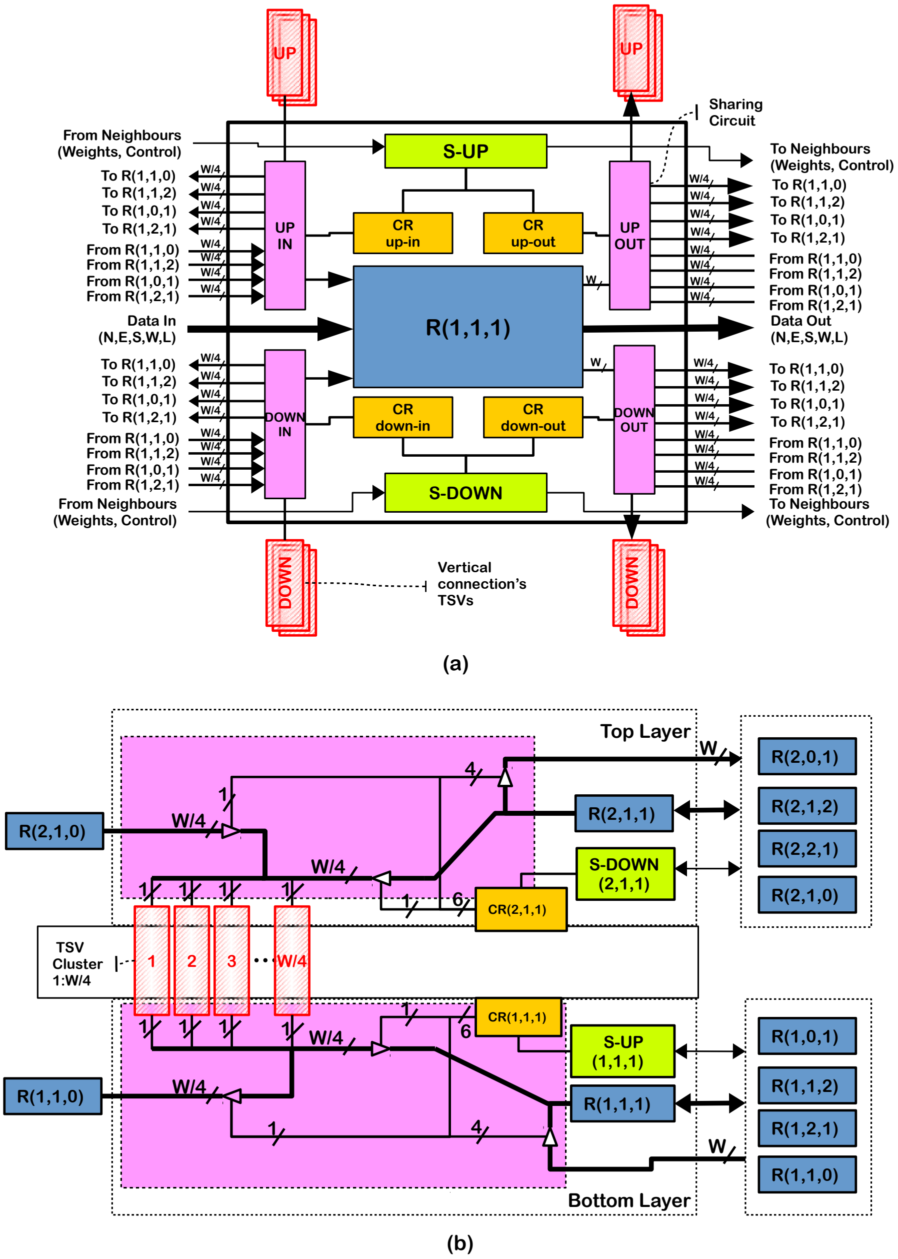


Figure 6.8: Fault-tolerant TSV architecture: (a) router wrapper (b) sharing TSV architecture with TSV cluster (red rectangles) and sharing arbitrators (*S-UP*, *S-DOWN*) [165].

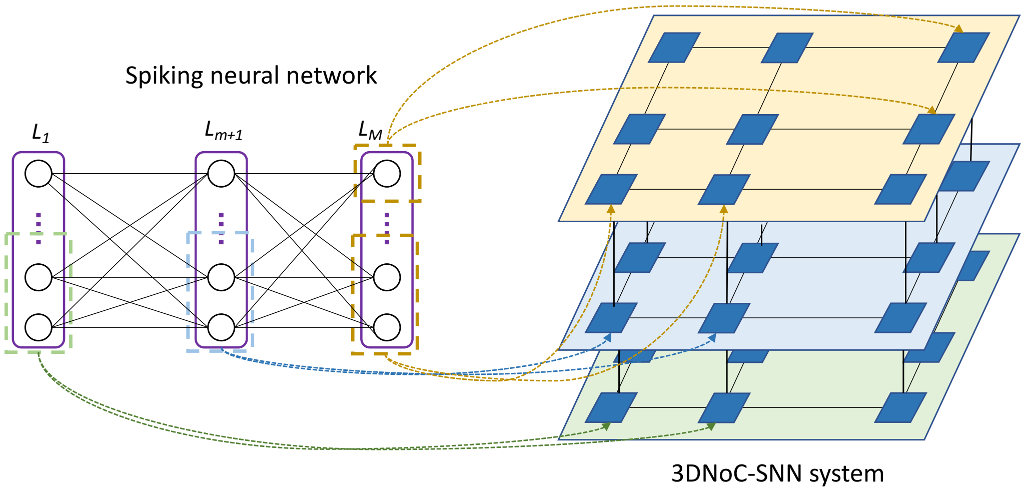


Figure 6.9: Layer-to-layer mapping method.

Converting from original data sets

Since collecting data is very time-consuming, this conversion method benefits from the available data sets.

- Poisson encoding: This is the most common method in converting image data sets into spike trains. In this method, each pixel is converted into a Poisson spike train that its spike rate is proportional to the pixel's density, as shown in Figure 6.10 (A).

- Intensity-to-latency encoding: In this method, pixel intensity is proportional to spike delay. This means that if the intensity of a given pixel is higher, the spike will be generated earlier, as shown in Figure 6.10 (B). Contrary to the Poisson encoding, each pixel is converted into a single spike. This results in a lower spike rate and faster response time.

Using converters

In this method, a converter is employed to convert input data into spikes. In image processing, Dynamic Vision Sensors (DVS) have been used in many works. A setup of using DVS is illustrated in Figure 6.11. When using DVS, it is placed in front of an LCD monitor or a screen. Input images are then displayed slowly during a period (e.g., from

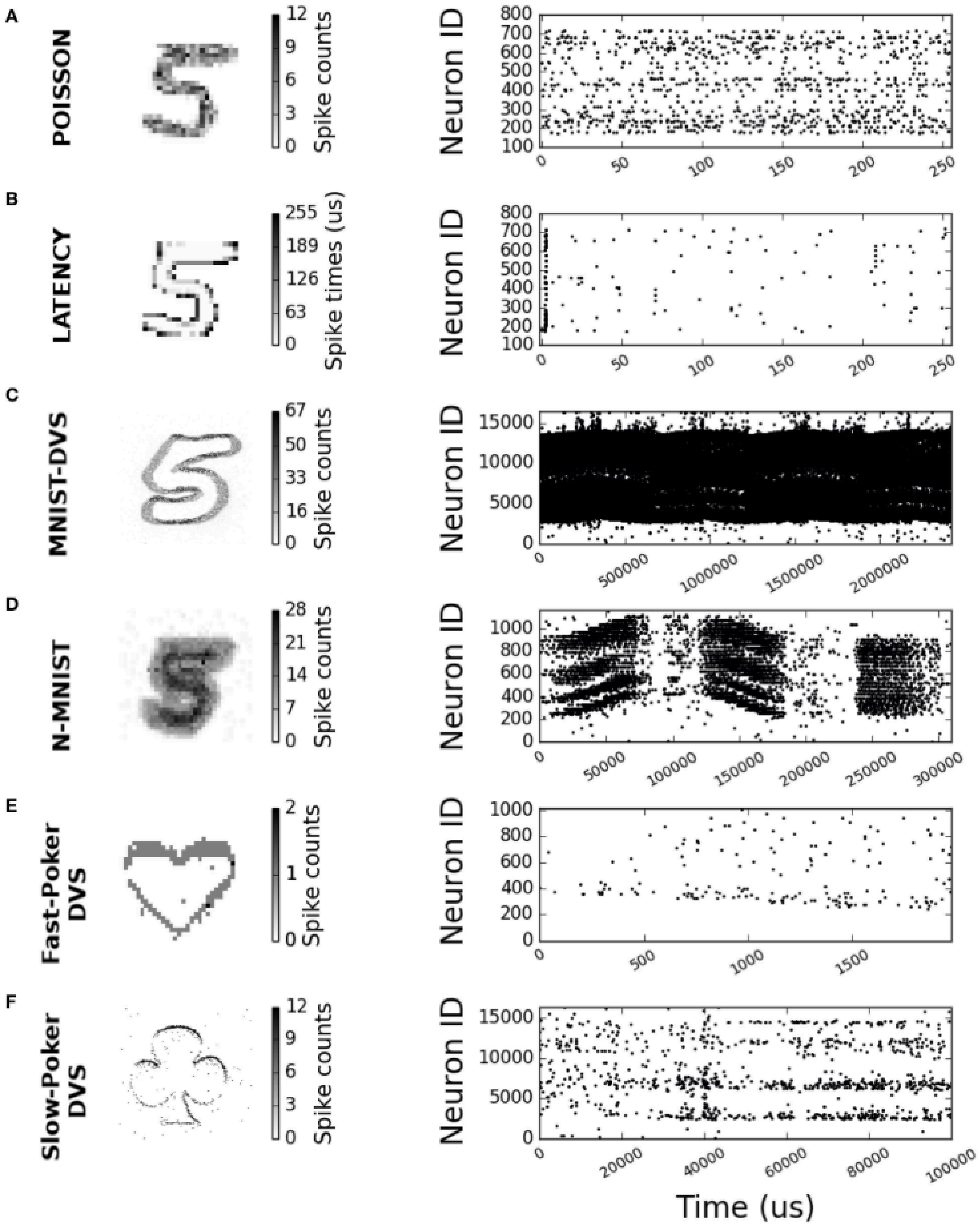


Figure 6.10: 2-D histograms and raster plots for different encoding schemes and neuromorphic data sets. (A) Poisson 28 × 28 input size sample. (B) Latency 28 × 28 input size sample. (C) MNIST-DVS 128 × 128 input size sample. (D) N-MNIST 34 × 34 input size sample. (E) Fast-Poker DVS 32 × 32 input size sample. (F) Slow-Poker DVS 128 × 128 input size sample [171].

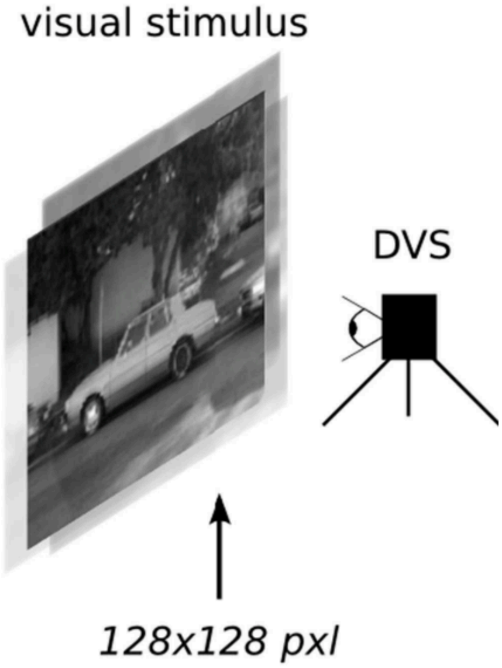


Figure 6.11: Conversion using a DVS camera, captured from [172]. In this setup, two different classes of images (here motorbikes or cars) are displayed on a screen with a small jitter applied at 10Hz. A random subset of the spikes is emitted by the DVS.

hundred milliseconds to tens of seconds). The DVS will record and output spike, as shown in Figure 6.10 (C-F).

7

Design and Evaluation

This chapter is dedicated to implement and evaluate the performance and also hardware complexity of the proposed system. Our proposed system was designed in Verilog-HDL, and synthesized with commercial CAD tool. We first explain how to evaluate the proposed system. After that, we present the evaluation results of each proposed routing method.

7.1 Methodology

The proposed system was implemented in Verilog-HDL . First, we evaluate the maximum spike injection rate supported by a single MC-3DR router. Second, we use both realistic and synthetic benchmarks to study the performance in terms of average latency and throughput of the proposed system. Spike Generators (SGs) and Spike Counters (SCs) were used in the analysis to inject different spike traffic loads into the 3DNoC-SNN system over different network sizes ($2 \times 2 \times 3$, $3 \times 3 \times 3$, and $3 \times 3 \times 4$). These network sizes were carefully selected according to the investigated application requirements. The nodes communication within our 3DNoC-SNN system is in the form of continuous spike streams, where information is coded in the relative timing of spikes. We have to note that different SNN topologies can be created by configuring connections between the 3DNoC-SNN components. Finally, we synthesize the system by using NANDGATE 45nm library [173] to explore the hardware complexity of the proposed system.

The MC-3DR is based on our previous 3D router architectures (SHER-3DR) [164, 165, 168]. The SHER-3DR relies on sophisticated recovery techniques based on system reconfiguration with redundant structural resources to handle hard faults in the input-buffers, crossbar, and links [164, 165], in addition to soft errors in the routing pipeline stages [168]. Similarly to our previous work [165], we also use NCSU FreePDK TSV [174], with TSV size of 4.06μm $\times$ 4.06μm, pitch size of 10μm, and Keep-out Zone of 15μm for prototyping. However, during the evaluation of the 3DNoC-SNN system, all the fault-tolerance techniques which are found in the previous SHER-3DR are disabled.

7.2 Evaluation Results

7.2.1 Spike Injection Rate Analysis

This experiment aims to explore the highest spike injection rate serviced by a given router. As shown in Figure 7.1, the router *R[1,1,1]* is used for forwarding incoming spike packets to their output ports of opposite direction except the spikes injected from *SG7*. For example, the spikes generated from *SG1* after passing through the *R[1,2,1]* arrive to the ”North” input port of the *R[1,1,1]*. After being forwarded to the ”South” output port of the *R[1,1,1]*, they traverse the *R[1,0,1]* before reaching *SC3*. The experiment is conducted under various number of router NoC paths (RNPs) and injection rates (the neuron functionality is not active in this experiment).

Figure 7.2 shows a comparison result between the proposed 3DNoC-SNN architecture

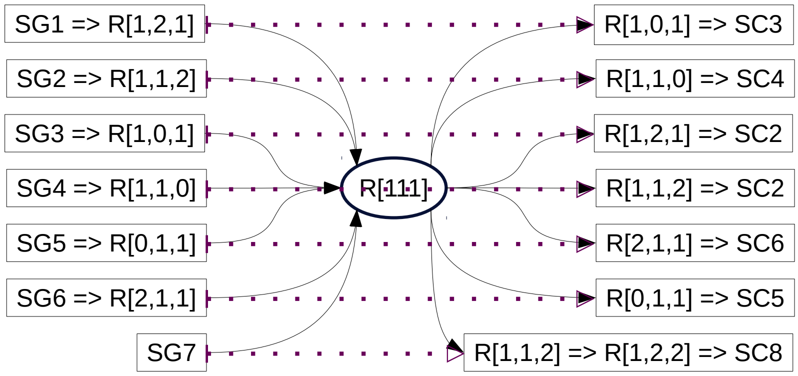


Figure 7.1: Setup for SIR evaluation.

and the work presented in [175] regarding the spike packet loss ratio (SPLR). From this experiment result, we Figure that the proposed method outperforms the previous work. In case of the EMBRACE router, the packet loss begins from the injection rates of 1/32, 1/24, and 1/16 with 4, 3, and 2 RNPs, respectively. Besides, the proposed MC-3DR router can service six RNPs with incoming spikes at every cycle. With seven RNPs, the highest injection rate is 1/2. This result is because EMBRACE router used an eight-state round-robin scheme (five for input ports and three for housekeeping tasks [175]),making the router unable to service high contention traffic (when injection rate and RNP increase). Conversely, our design takes full advantages of a fast and strong-fair Matrix-arbiter, as described in Section 6.

7.2.2 K-means Based Multicast Routing Algorithm Evaluation

Performance Evaluation Under Realistic Benchmarks

To evaluate the performance of the proposed routing method, we selected two well-known applications: (1) Inverted Pendulum and (2) Wisconsin Data-set. These applications are selected because they are suitable for evaluating hardware-based SNNs [176, 177]. The Inverted Pendulum is a standard benchmark for control purpose. The Wisconsin Data-set is an image classifier using Wisconsin Breast Cancer data-set. Our system's

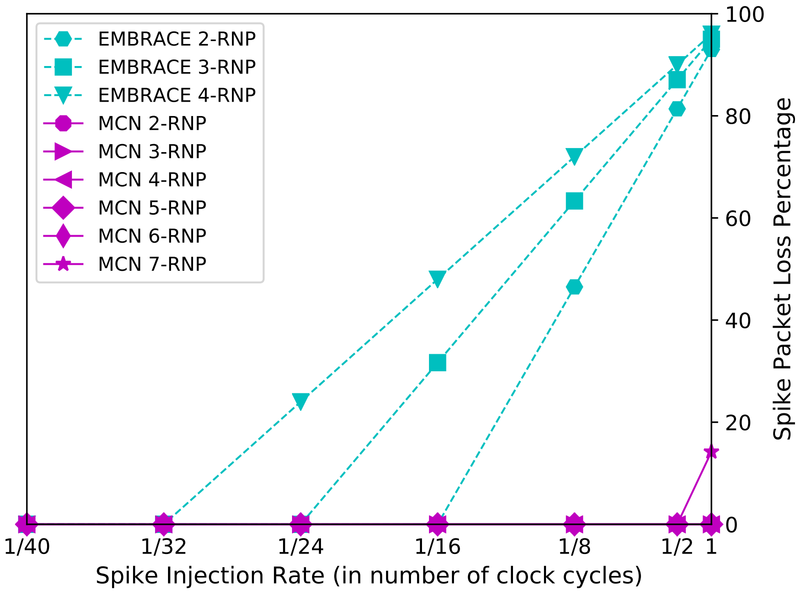


Figure 7.2: Router acceptance rate (RAR) comparison when varying the number of router RNP and SIRs.

configuration parameters for both benchmarks, including the training and testing samples, are inspired by the work in [176]. These applications enable us to evaluate the proposed routing algorithm in real traffic patterns. For the performance metrics, we evaluate the proposed routing regarding latency and throughput. The latency here is defined as the number of clock cycles from when the first bit of a spike packet enters the source terminal until the last bit of the packet arrives at all the destination nodes. The applications were mapped onto the 3DNoC-SNN system in a layer-to-layer manner, as shown in Figure 7.3. In this mapping method, neurons in the same NN layer were placed in the same 3DNoC-SNN layer. This leads to taking full advantages of 3D integration such as reducing the number of hops as well as the overall spike latency when compared to the 2D integration method [165]. Since our main focus in this work is on the network performance, we used Spiking Generator/Counter units (SGC) attached to each multicast router, as shown in Figure 7.3 (a). To make a performance comparison with the proposed algorithm, we also implemented a unicast-based multicast (Section 4.2.1),), named XYZ-UB. XYZ is one of the variations of dimension order routing (DOR). This is a simple algorithm, easy

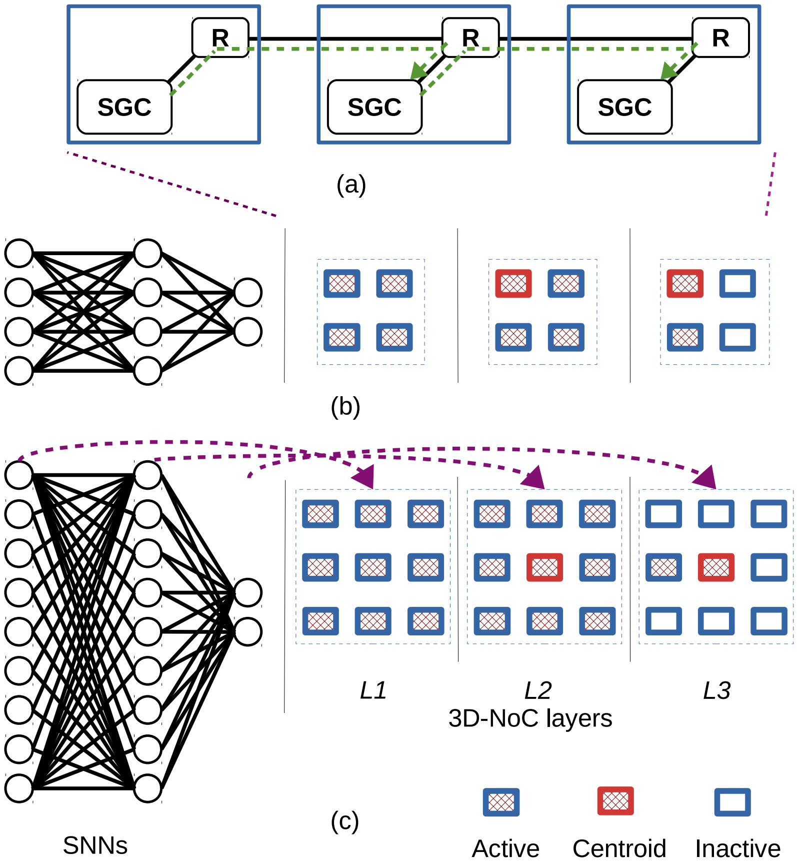


Figure 7.3: Layer-to-layer application mapping: (a) block diagram of a node in each layer (b) Inverted pendulum (c) Wisconsin data-set.

Table 7.1: Realistic simulation configuration.

Parameter/System	XYZ-UB	KMCR (this work)
NoC size	2×2×3 (3D), 4×3(2D)	
	3×3×3 (3D), 9×3 (2D)	
Buffer depth	4	
Switching	Wormhole	
Flow control	Stall-go	
Scheduling	Matrix-Arbiter	
Routing	Unicast-based multicast	Multicast

to implement, and free of deadlock and lifelock [161]. Moreover, both multicast routing mechanisms are implemented in 2D and 3D systems. For a fair comparison, the configuration parameters are kept similar for both routing algorithms, as shown in Table 7.1.

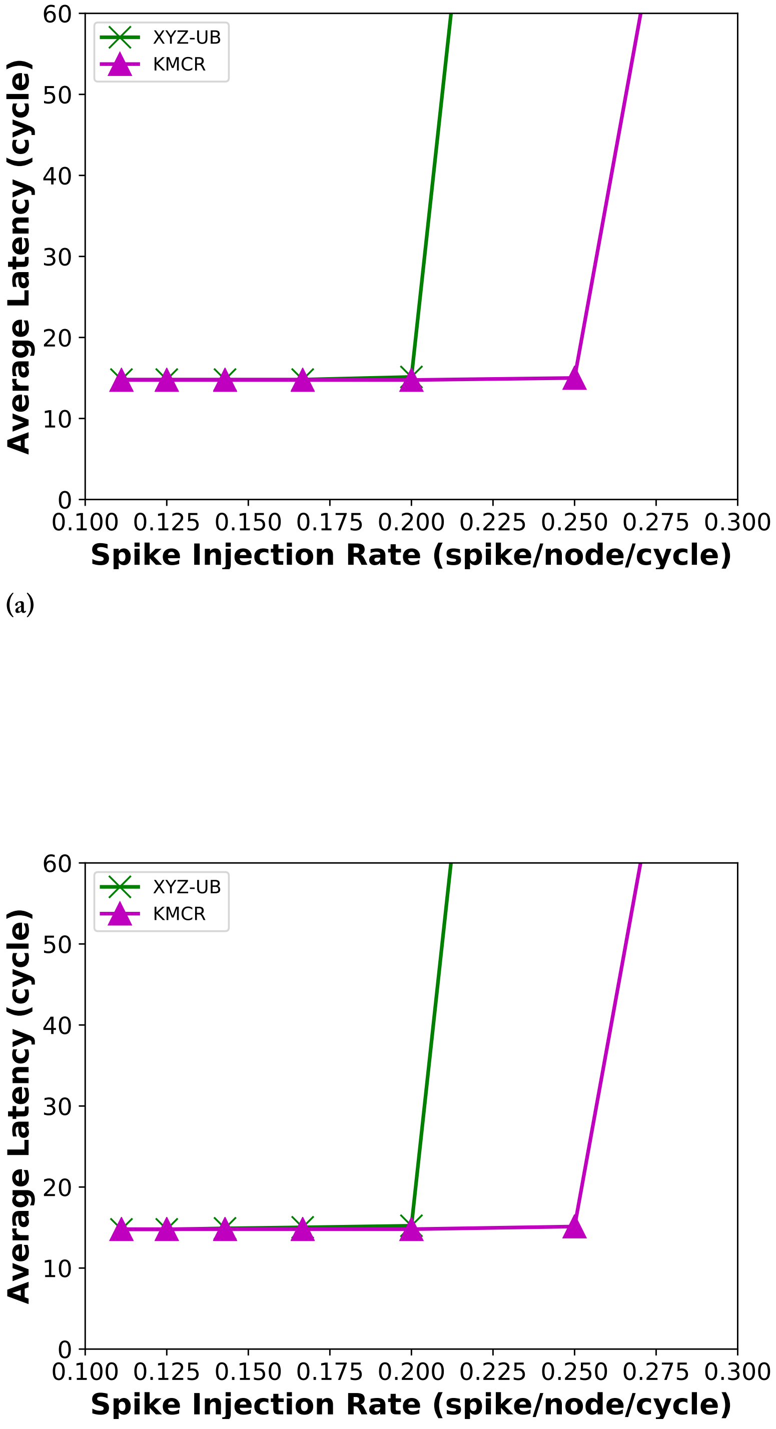
Latency evaluation:

The average communication latency as a function of SIR is shown in Figure 7.4 and 7.5. The evaluation results show that the proposed algorithm demonstrates lower average latency and higher throughput when compared to the XYZ-UB routing method in both 2D and 3D based system configurations. For the Inverted Pendulum application, the 3DNoC-SNN system shows almost a similar average latency in both XYZ-UB and our proposed routing method (see Figure 7.4a). This is because the number of destinations is small (6 destinations/neurons in total), which means low traffic. However, the proposed algorithm enables the system to keep the latency at 25% higher injection rate when compared to XYZ-UB. As shown in Figure 7.4b, the average latency evaluation result for the 2D system configuration is almost similar to the 3D configuration due to the small, realistic NN benchmark.

For Wisconsin Data-set benchmark, the increase of the network size causes a higher latency when compared to the Inverted Pendulum application. The latency also increases with the increase of the injection rate. For 3D systems (Figure 7.5a), the unicast-based system suffers higher latency compared to the other, at about 14.43% at SIR = 1/11. Furthermore, it can only support a lower injection rate compared to the other one which is less than about 22.22% of SIR. For 2D architectures, while system supporting unicast-based routing algorithm cannot maintain SNN traffic, proposed one still works with a little bit higher latency compared to 3D (Figure 7.5b); this shows that the proposed routing mechanism is not only efficient for 3D, but also for 2D systems.

Throughput Evaluation:

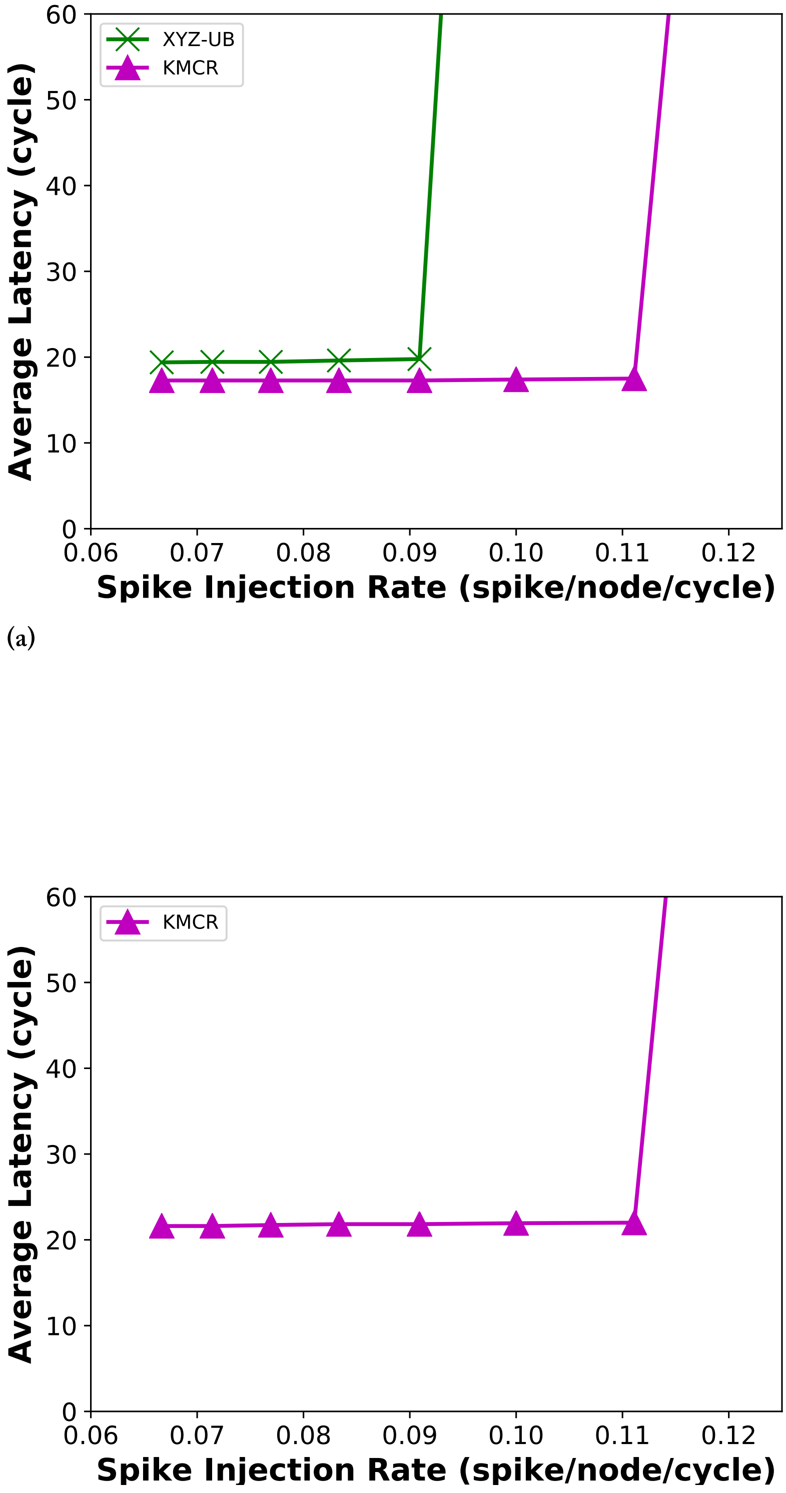
Figure 7.6 and 7.7 shows the evaluation and comparison results regarding the average throughput. The results show that the proposed routing algorithm achieves 24.5% and 22% higher throughput when compared XYZ-UB mechanism over the Inverted Pendu-



(a)

(b)

Figure 7.4: Average latency over various SIRs for Inverted Pendulum in: (a) 3D Domain (b) 2D Domain.



(a)

(b)

Figure 7.5: Average latency over various SIRs for Wisconsin Data-set in: (a) 3D domain (b) 2D Domain.

lum and Wisconsin Data-set benchmarks on 3D domain, respectively.

Performance Evaluation Under Synthetic Benchmark

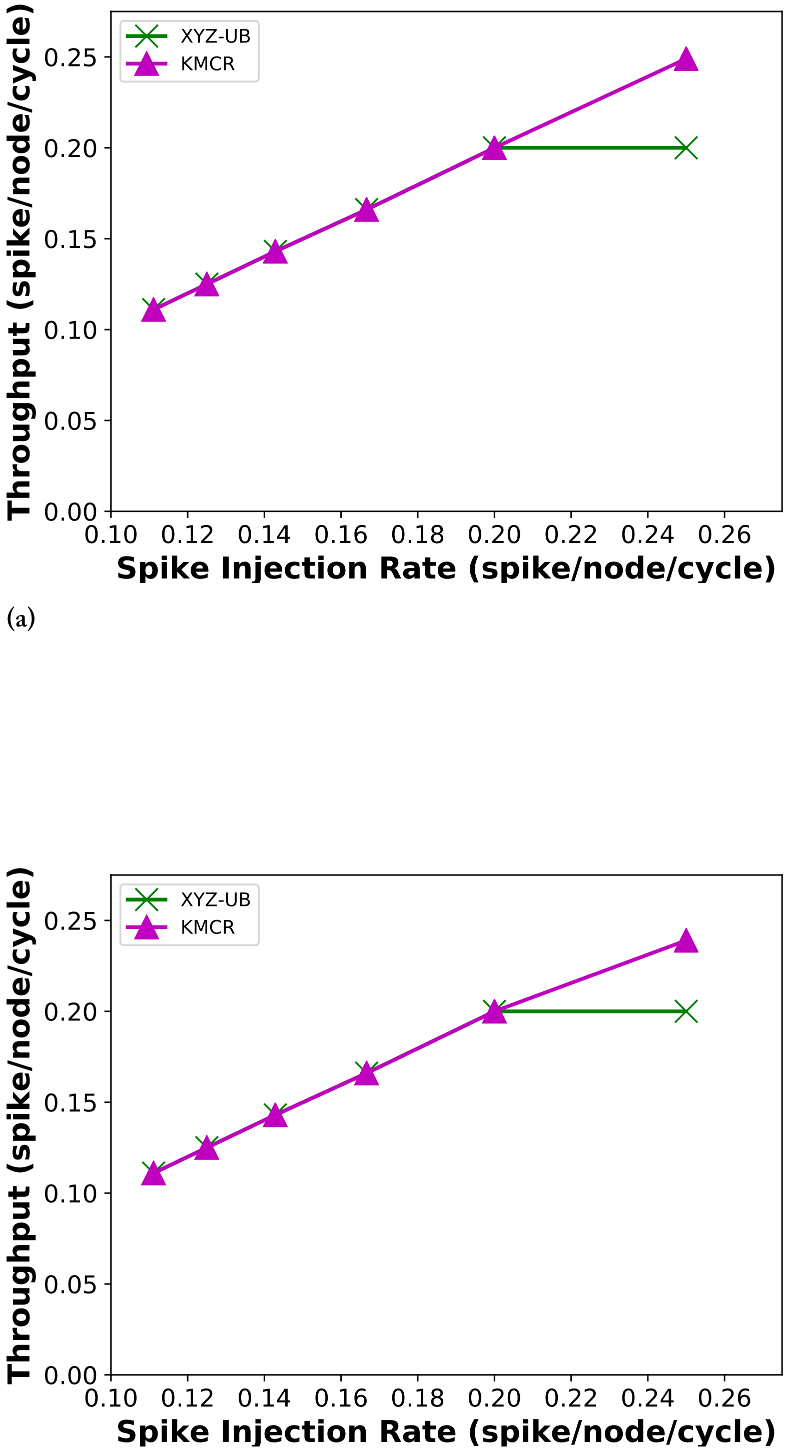
To further explore the performance of our proposed 3DNoC-SNN system, we used larger benchmarks to study again the throughput and latency performance. We compared the performance of the proposed system to other NoC based systems, named Dragonfly [178], H-NoC [22], and Cmesh [179].

In this experiment, we used a NN benchmark with 128 neurons in which each neuron sends its output to all other neurons. The simulation parameters are summarized in Table 7.2. For a fair comparison, we select the same number of neurons per router (N/R ratio = 4) as in Cmesh [179] system. Based on the NN size and the N/R ratio, the network-on-chip size for each system is determined. For our 3DNoC-SNN, the size is $3 \times 3 \times 4$ in which none of the neurons is mapped to the centroid node. This configuration leads to steady traffic between the different network layers.

Table 7.2: Synthetic simulation setup.

Parameter	Dragonfly	H-NoC	Cmesh	3DNoC-SNN (this work)
N/R ratio	8	8	4	4
NoC size	8×2	8×2	8×4	3×3×4
Flit size	32-bits	32-bits	32-bits	31-bits
Buffer deep	8-flits	8-flits	8-flits	8-flits
Routing	Multicast	Multicast	Multicast	Multicast

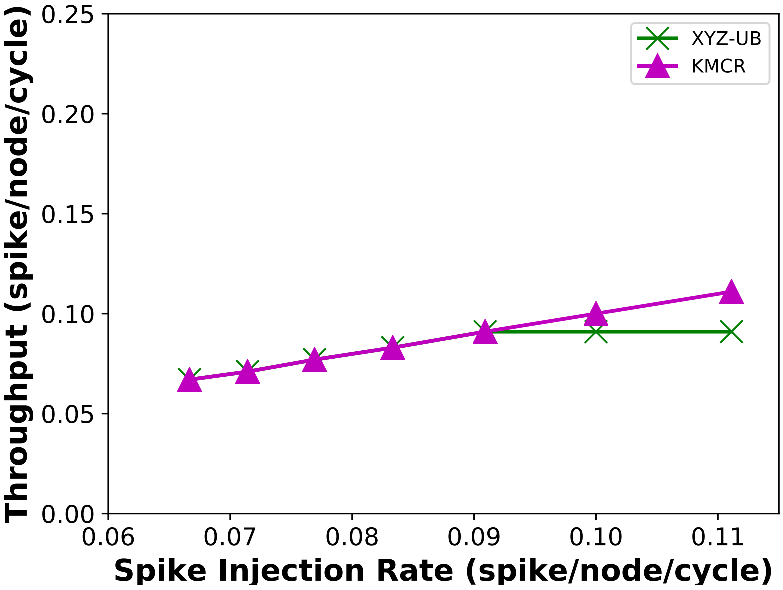
As shown in Figure 7.8, the latency of the proposed 3DNoC-SNN system is almost unchanged for low spike injection rates (SIRs), while the average latency of the other systems (CMesh, H-NoC, and Dragonfly) increases with the increase of the SIRs. This result also proves that our proposed 3DNoC-SNN system can efficiently handle higher spike injection rates. For example, at an injection rate of 0.02875 *spike/node/cycle*, the Dragonfly system experiences almost 2× latency when compared to the proposed 3DNoC-SNN system. Furthermore, the 3DNoC-SNN system can maintain high SIR before saturation point (8.7% compared to the Dragonfly, thanks to our 3D domain topology and the



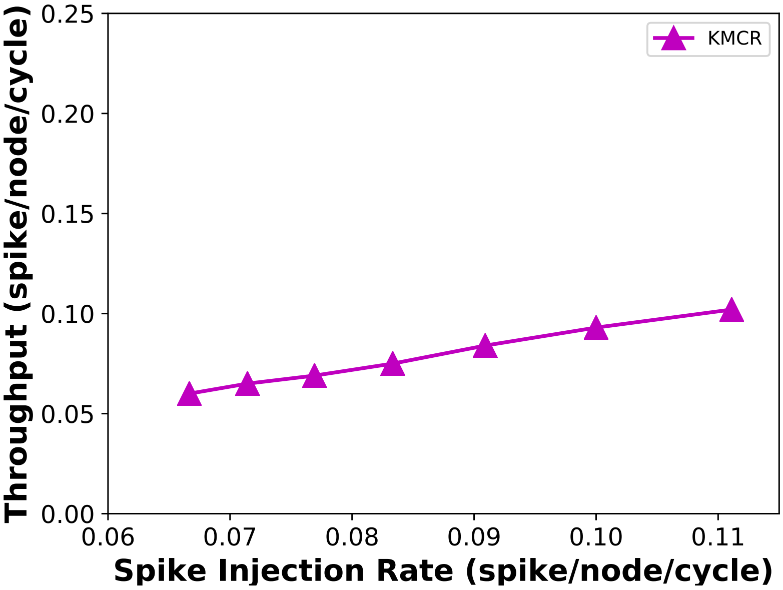
(a)

(b)

Figure 7.6: Average throughput in NoC systems over various SIRs for Inverted Pendulum in: 3D Domain, (b) 2D Domain.



(a)



(b)

Figure 7.7: Average throughput in NoC systems over various SIRs for Wisconsin Data-set in: (a) 3D Domain, (b) 2D Domain.

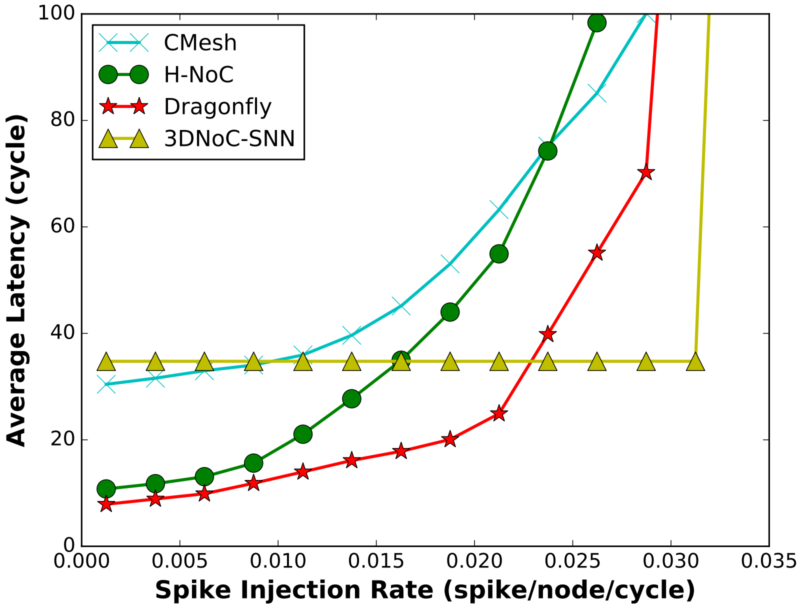


Figure 7.8: Average latency evaluation and comparison over various SIRs.

low-latency multicast routing algorithm.

The throughput of the 3DNoC-SNN system under the synthetic benchmark was also evaluated. Since the average throughput of CMesh, H-NoC, and Dragonfly are not reported in [178], these results are omitted from Figure 7.9. As shown in the figure, there is an increase in the average throughput when increasing injection rate; it reaches 0.0313 spike/node/cycle before the saturation point.

Hardware Complexity Analysis

Table 7.3 compares a single MC-3DR with several other proposed routers for SNN systems regarding area cost and power consumption. The routing table size influences the hardware complexity evaluation result. However, it depends on the selected SNN application. In this experiment, we choose each table having 32 entries (equal to the number of nodes in the synthetic benchmark presented in Section 7.2.2); this is the maximal number of entries which a table may have. In the synthetic evaluation, the entry count is much smaller compared to the maximal number. As shown in table 7.3, the hardware complexity of our design is acceptable. In terms of area cost, the proposed router is about 40.9% higher compared to H-NoC cluster router, while smaller around 2.3× than Clos-

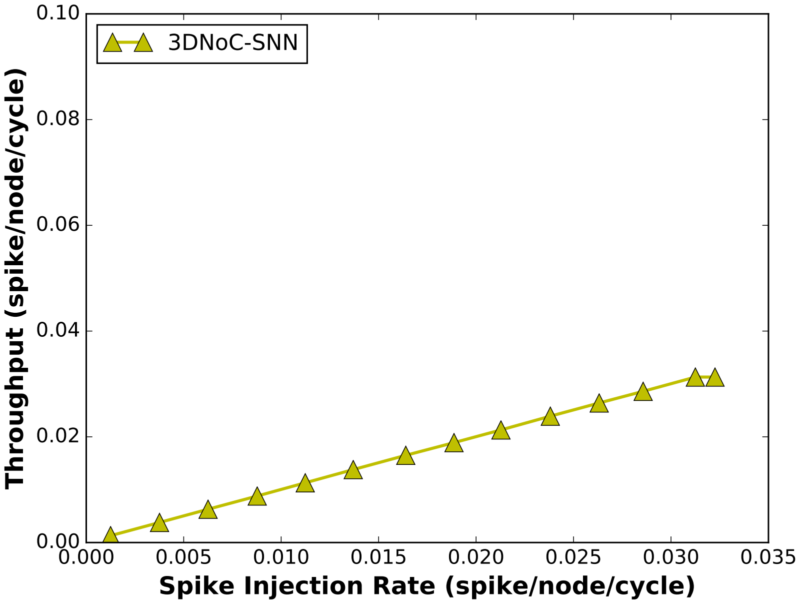


Figure 7.9: Average throughput under the synthetic benchmark over various SIRs.

NoC spine switch. However, it should be mentioned that the hardware complexity of our router does not include the cost of our previous fault-tolerant techniques which are used to handling hard faults in the input buffers, crossbar, and links and also soft errors in the routing pipeline stage. Besides, since the extra computation for finding the shortest-path nodes in each subset is off-line, the SP-KMCR method does not add extra hardware resource compared to the KMCR mechanism.

Table 7.3: MC-3DR Hardware Complexity Evaluation and Comparison.

System	Topology	Area (mm^2)	Power (mW)
EMBRACE router [123], *90nm*	2D Mesh	0.056	1.72
HANA tile router [180], *90nm*	2D Mesh	0.156	28.12
H-NoC cluster router [181], *65nm*	Star-Mesh	0.022	1.19
Clos-NoC spine switch [21], *45nm*	Custom Clos	0.076	-
Clos-NoC leaf switch [21], *45nm*	Custom Clos	0.061	-
MC-3DR router, *45nm* (this work)	3D Mesh	0.031	1.66

Furthermore, we also evaluated the total power consumption of the 3D systems, as shown in Table 7.4. For the Inverted Pendulum, XYZ-UB and KMCR consume 10.41mW and 10.13mW respectively, while the figures for the Wisconsin dataset are 35.26mW and

34.20mW respectively (thanks to the larger network size). This result is due to the proposed routing algorithm which reduces the number of packets injected into the network.

Table 7.4: Power consumption of UCB-XYZ and KMCR under realistic benchmarks.

System	**XYZ-UB**		**KMCR (this work)**	
	Inverted Pendulum	Wisconsin	Inverted Pendulum	Wisconsin
Power (mW)	10.41	35.26	10.13	34.2

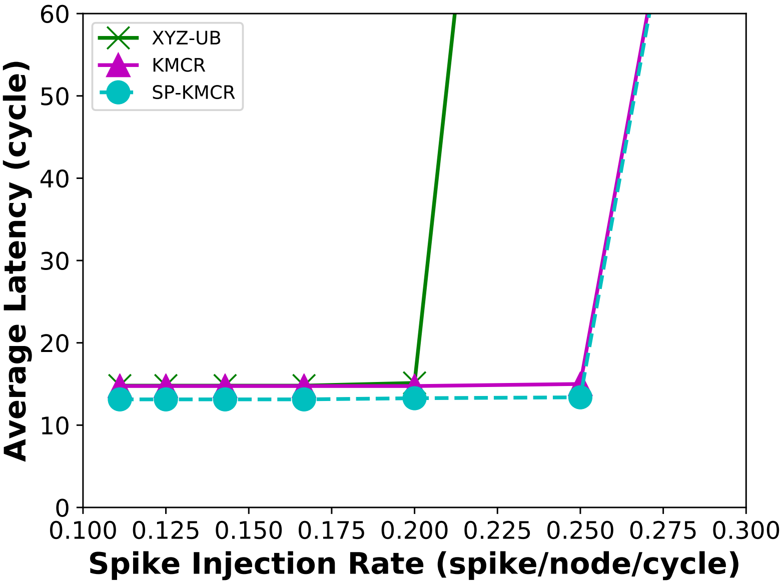
From the performance analytical model, we discovered that the MC routing performs better than UC based routing in both topologies. From the evaluations, the results also prove that the MC based routing performs better than UC in terms of spike injection rate (25%), latency (14.43%), throughput (24.5%), power consumption (3%). This proves that our posed performance analytical model is an accurate and efficient method for earlier performance assessment.

7.2.3 Shortest Path K-means Based Multicast Routing Algorithm Evaluation

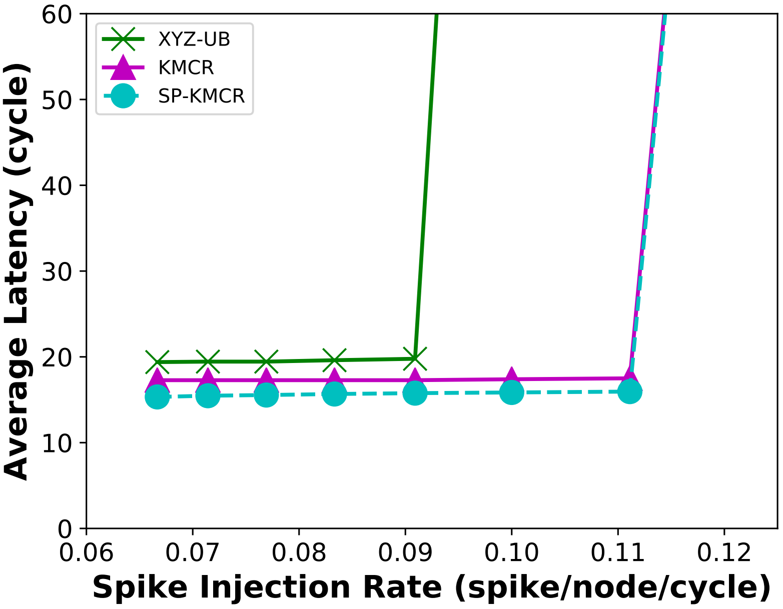
For SP-KMCR, we also use realistic benchmarks to exploire the performance of the proposed system. The experiments are also conducted in both 3D and 2D domains. We compare the SP-KMCR with the previous KMCR and the XYZ-UB in terms of average latency and throughput. The average communication latency as a function of Spike Injection Rate (SIR) is shown in Figure 7.10 and 7.11.

For the Inverted Pendulum application in 3D domain, the KMCR show almost the same average latency as XYZ-UB before reaching its saturation point at SIR = 0.2, as shown in Fig 7.10. However, the SP-KMCR and KMCR sustain almost the same latency while providing 25% higher SIR, when compared to XYZ-UB. This can be explained by the fact that XYZ-UB needs to send multiple copies of a given spike, resulting in high traffic contention. These results mean that the proposed system can maintain high SNN traffic such as fast and bursting operation modes of spiking neurons [13].

On the other hand, SP-KMCR reduces the latency by 12.2% when compared to KMCR before it reaches the saturation point, even with a small network size such as the one used

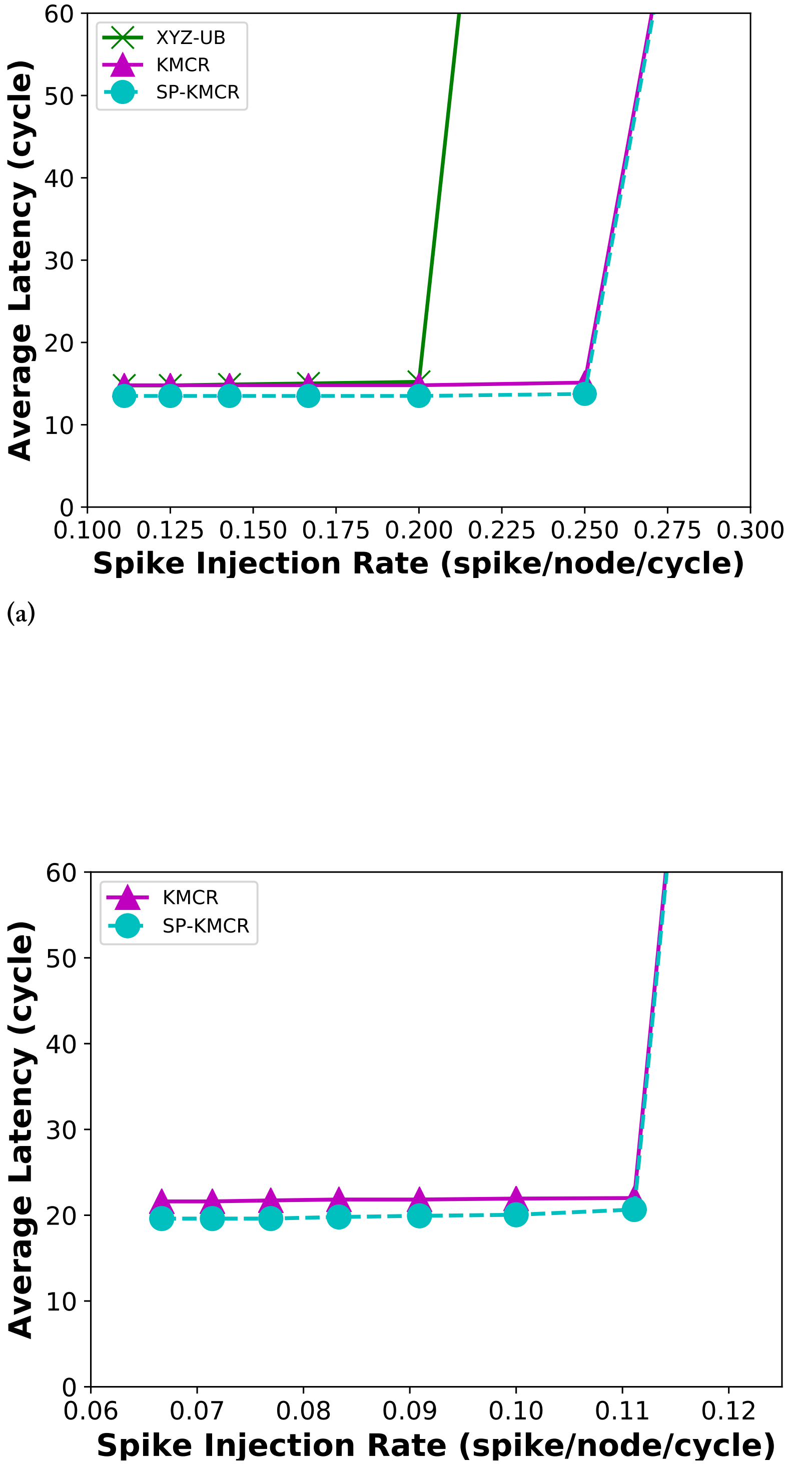


(a)



(b)

Figure 7.10: Average latency over various SIRs in 3D domain: (a) Inverted Pendulum (b) Wisconsin Data-set.



(a)

(b)

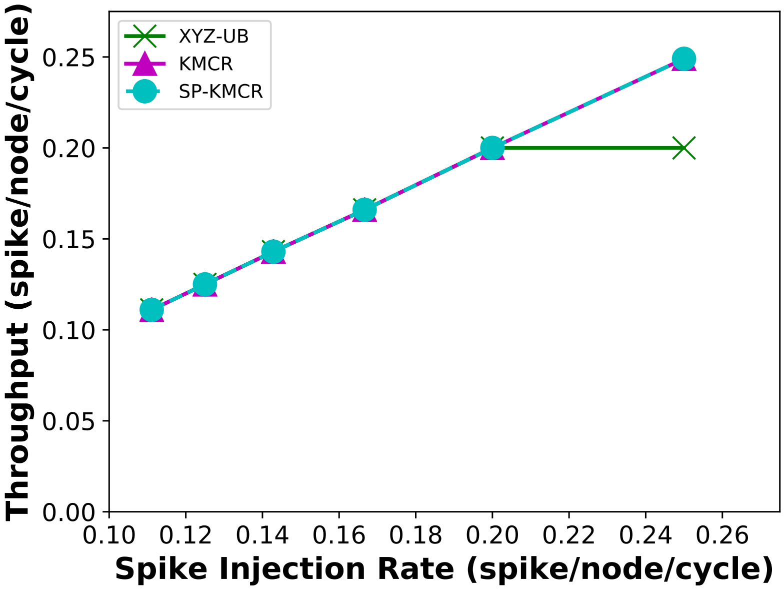
Figure 7.11: Average latency over various SIRs in 2D domain: (a) Inverted Pendulum (b) Wisconsin Data-set.

for the Inverted Pendulum application. This is due to the fact that source nodes in the SP-KMCR send spikes to their shortest path node instead of centroid node in KMCR; resulting in alleviating the congestion in the intermediate node like the centroid. The evaluation results also demonstrate that the SP-KMCR enables systems, running SNN applications with a smaller timestep, to improve their acceleration potential. As shown in Figure 7.11a, the average latency evaluation result for the 2D system configuration is almost similar to the 3D configuration due to the small, realistic NN benchmark.

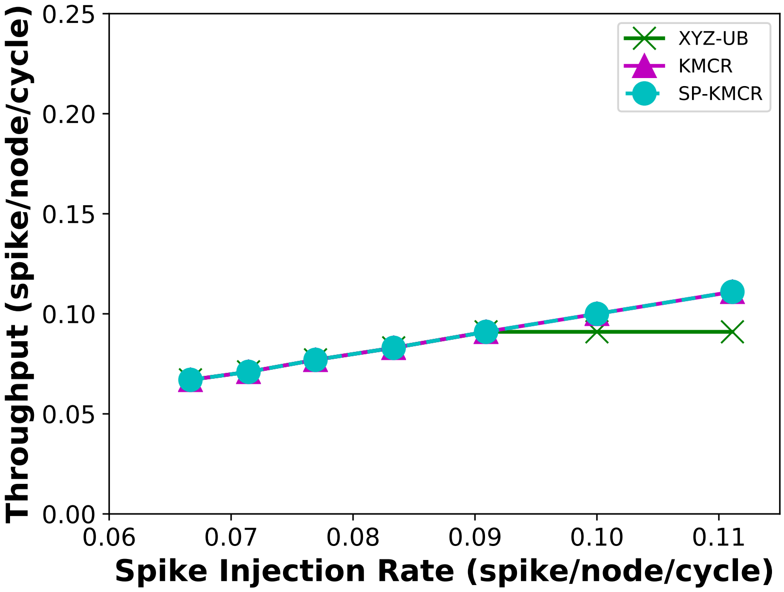
For Wisconsin Data-set benchmark in 3D domain, the increase of the network size causes a higher latency when compared to the Inverted Pendulum application, as shown in Fig 7.10. The latency also increases with the increase of SIR. The unicast-based system suffers higher latency compared to the KMCR by about 14.43%, at SIR = 11%. Furthermore, KMCR can support a higher SIR reaching up to 22.22% when compared to the unicast-based. Compared to the KMCR, the SP-KMCR reduces the average latency by 9.5%, at the highest injection rate. The improvement of SP-KMCR is also kept in 2D domain (see in Fig 7.11b).

Figure 7.12 and 7.13 show the evaluation and comparison results regarding the average throughput. The average throughput of both KMCR and SP-KMCR is similar since they keep the same spike injection rate. The results show that the proposed routing algorithms achieve 24.5% and 22% higher throughput when compared XYZ-UB mechanism over the Inverted Pendulum and Wisconsin Data-set benchmarks on 3D domain, respectively. The evaluation results also show the proposed multicast routing benefits in terms of efficient bandwidth when running SNN applications.

In Section 7.2.2, we evaluated and compared the KMCR with three other existing works: Dragonfly [178], H-NoC [22], and Cmesh [179]. The evaluation results showed that KMCR can maintain a higher SIR before saturation point, by about 8.7% when compared to the best algorithm (i.e., Dragonfly) among the three considered works. This allows us to further believe that the proposed SP-KMCR in this research shows better performance when compared to Dragonfly, H-NoC, and Cmesh.

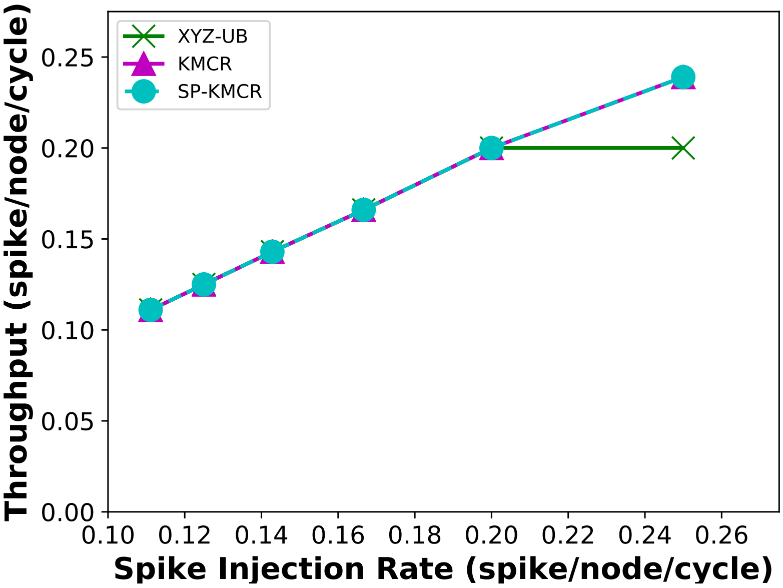


(a)

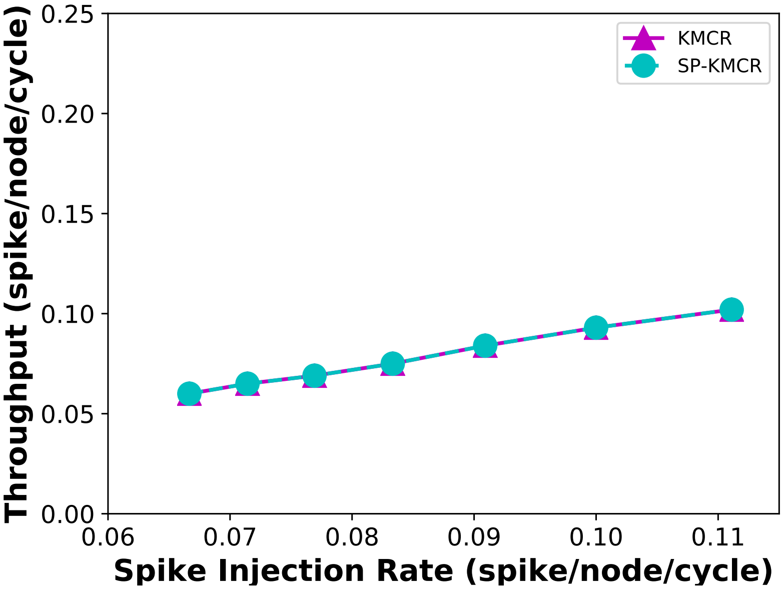


(b)

Figure 7.12: Average throughput over various SIRs in 3D domain: (a) Inverted Pendulum (b) Wisconsin Data-set.

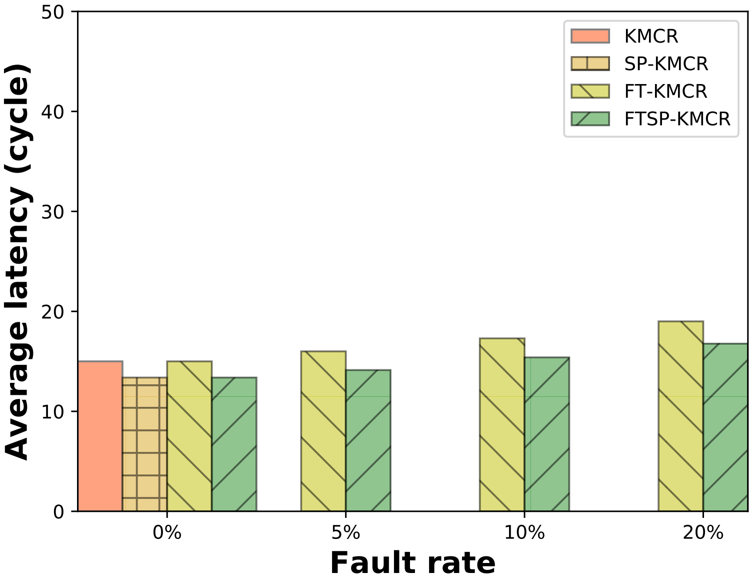


(a)

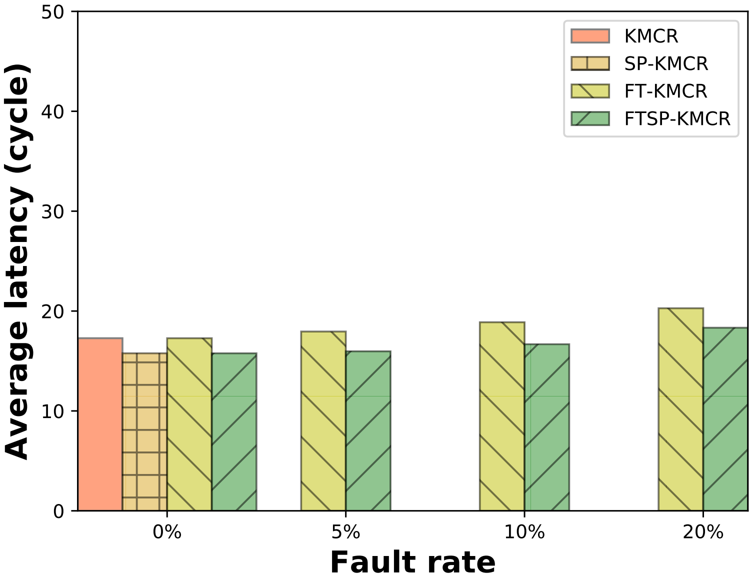


(b)

Figure 7.13: Average throughput over various SIRs in 2D domain: (a) Inverted Pendulum (b) Wisconsin Data-set.



(a)



(b)

Figure 7.14: Average latency over various fault rates: (a) Inverted Pendulum (b) Wisconsin Data-set.

To further explore the improvement of SP-KMCR, we evaluate both KMCR and SP-KMCR with larger network sizes ($3 \times 3 \times 2$, $4 \times 4 \times 2$, and $5 \times 5 \times 2$). Here, all nodes in the first layer send packets to all the other nodes in the second layer. Fig. 7.15 compares the performance of KMCR and SP-KMCR in terms of average latency. The evaluation result shows that the SP-KMCR achieves lower average latency compared to KMCR, at about 10.29%, 16.86%, and 23.57% with 3DNoC sizes of $3 \times 3 \times 2$, $4 \times 4 \times 2$, and $5 \times 5 \times 2$, respectively. It means that the proposed SP-KMCR improves the average latency, especially for large network sizes.

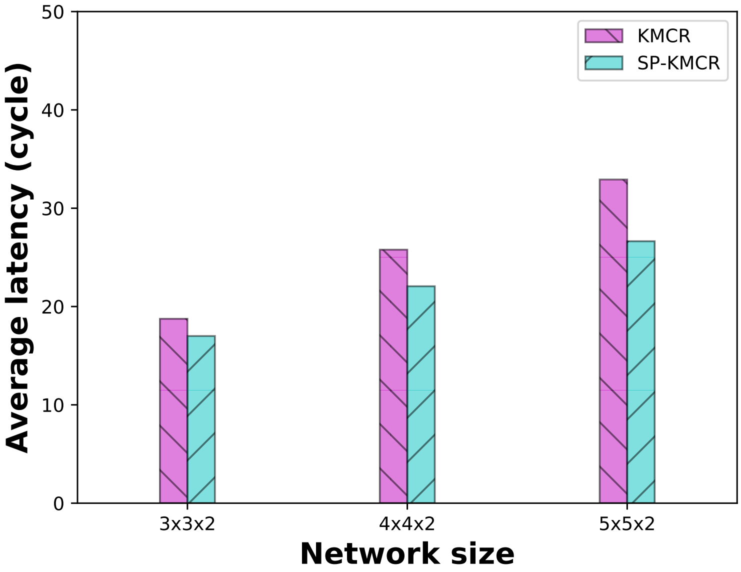


Figure 7.15: Average latency comparison of KMCR and SP-KMCR over different network sizes.

7.2.4 Fault-tolerant K-means Based Multicast Routing Algorithm Evaluation

In this evaluation, we explore the fault-tolerance potential of the proposed algorithm. In this experiment, the fault-tolerant mechanism was added to both KMCR and SP-KMCR baselines, denoted as FT-KMCR and FTSP-KMCR, respectively. We also used realistic benchmarks for the evaluations. The experiments are taken under variation of fault injection rate.

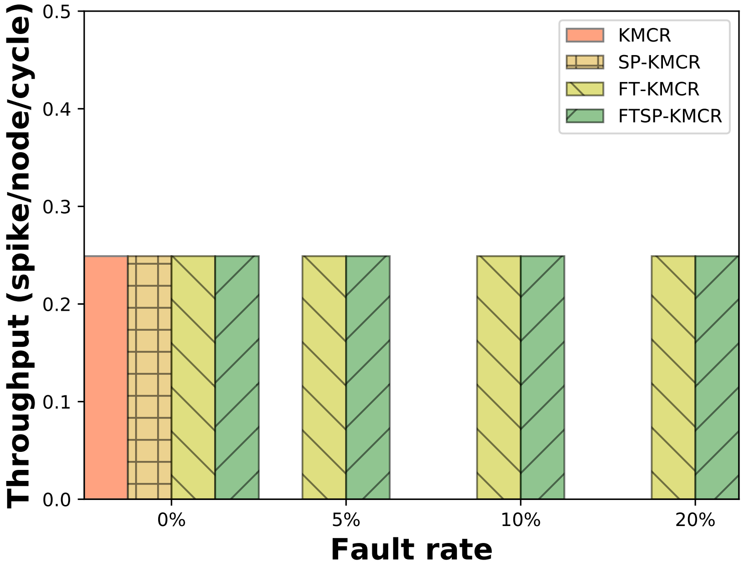
Latency evaluation:

The average latency as a function of fault rate is shown in Figure 7.14. As represented in this figure, the FT-KMCR and FTSP-KMCR keep the same latency compared to their baseline systems when no fault is injected. This is because both fault-tolerant systems and their baseline use the same routing tree. There is also a slight increase in the average latency of the FT-KMCR and FTSP-KMCR when increasing the fault rate. This comes from the fact that the fault-tolerant architectures use backup branches resulting in high traffic at the remaining healthy links. For Inverted Pendulum, the average latency of the FT-KMCR increases by about 6.67%, 15.33% and 26.67% at 5%, 10%, and 20% fault rates, respectively, compared to the KMCR. Here, the maximum spike injection rate is 0.25 spike/node/cycle. Figures for FTSP-KMCR are lower: about 5.61%, 15.10%, and 25.34%. The increase of average latency results in the system running in a longer timestep; however, the system can correctly run SNN applications at a higher fault rate.

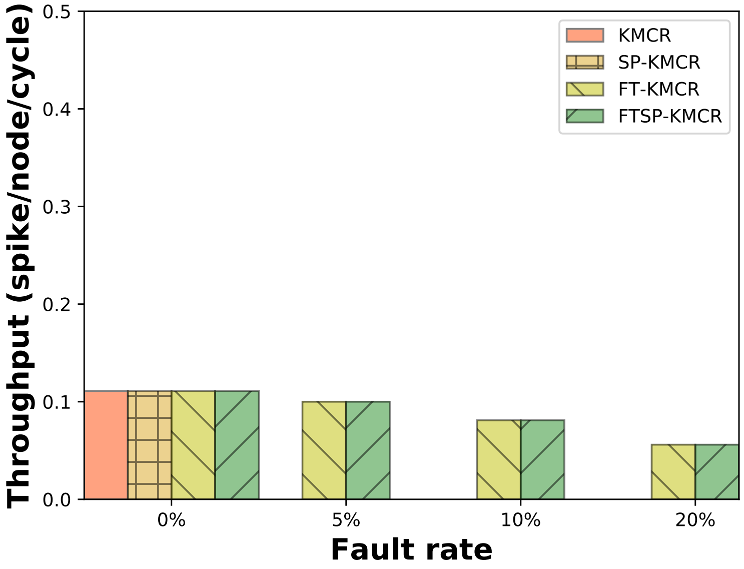
For Wisconsin Data-set, the average latency of FTSP-KMCR increases by 1.27%, 5.77%, and 16.23% when compared to its SP-KMCR baseline system. These evaluation results show that the proposed FTSP-KMCR has lower average latency compared to the FT-KMCR in both applications. The latency reduction is due to two main reasons: first, as explained above, the source nodes in FTSP-KMCR send packets to their shortest path node instead of centroid. Second, since backup routing computations are performed off-line, the runtime overhead of the proposed fault-tolerant technique is the same as its baseline. This helps the system to better reduce the effect of timing violations in SNNs caused by the long latency of recovery mechanisms.

Throughput evaluation:

Figure 7.16 compares the throughput of the proposed and the baseline systems. For the Inverted Pendulum, the architectures show similar average throughput results when increasing the fault rate. This is thanks to the redundancy of the architecture used for



(a)



(b)

Figure 7.16: Average throughput over various fault rates: (a) Inverted Pendulum (b) Wisconsin Data-set.

backup routing paths. For example, as shown in the right side of Figure 5.4 (b), when the ZYX version of the DOR is used for determining the primary tree, all intra-layer links in the first layer ($L1$) are not used. These links can be used as potential backup branches. This allows the proposed fault-tolerant architecture to maintain the communication traffic at the highest spike injection rate (i.e., the rate before the saturation point at 0% fault rate). This is the reason why the throughput is unchanged, while the average latency increases when raising the fault rate due to the larger hop count of the backup branches. On the other hand, there is a decrease in the average throughput of the proposed system under Wisconsin Data-set benchmark when increasing the fault rate. This is caused by the larger number of neurons used in this application resulting in higher contention in primary and backup branches. Therefore, the proposed architecture is not able to keep the same spike injection rate when increasing the fault rate, as it was the case for the Inverted Pendulum application. At a fault rate of 20%, the throughput of FT-KMCR and FTSP-KMCR decreases by 49.5% (i.e., the spike injection rate of 0.056 spike/node/cycle) compared to the system without fault. Nevertheless, the proposed algorithm was capable of correctly delivering all the spikes to their destinations despite this high fault rate. The evaluation results also show that a higher fault rate leads to reduction in the spiking frequency, as analyzed in Chapter 4.

Hardware Complexity Evaluation:

We also evaluated the hardware complexity in terms of area cost and total power consumption of the KMCR and FTSP-KMCR architectures (i.e., the entire network) to observe the extra hardware resources necessary for the proposed fault-tolerant method, as shown in Table 7.5. Regarding the area cost, the FTSP-KMCR uses more area than the KMCR system (about 5.88% and 5.49% for the Inverted Pendulum and the Wisconsin dataset, respectively). This is also consistent with the power consumption results, in which the FTSP-KMCR consumes a higher amount of power (about 5.03% and 4.97% for the Inverted Pendulum and the Wisconsin dataset, respectively). This hardware overhead is

Table 7.5: Power consumption of the KMCR and FTSP-KMCR under the benchmarks.

System	KMCR		FTSP-KMCR	
	Inv. Pen.	Wis.	Inv. Pen.	Wis.
Area (mm^2)	0.102	0.346	0.108	0.365
Power (mW)	10.13	34.20	10.64	35.92

mainly due to the extra hardware needed for the backup branches.

Discussion:

We validated the reliability of the proposed architecture to sustain correct inter-neural communication even at a 20% fault rate, while ensuring small hardware complexity and graceful performance degradation. Nevertheless, a couple of points need to be discussed in order to exploit the full potential of the proposed architecture. Hereafter, we address these challenges and highlight the possible solutions.

The first point to be addressed is the improvement of the proposed fault-tolerant multicast routing algorithm to deal with multiple faults. In particular, when successive multiple faults occur in the primary branches. In the current implementation, the proposed algorithm should forward spike packets through backup branches for every single faulty link. In Figure 5.5, for example, if both pb_1 and pb_2 are faulty, incoming spikes from "grandfather" are firstly forwarded to "father" through backup branches, then from "father" to "son". However, this leads to increased latency due to the additional unnecessary hop travel. This issue can be tackled by using direct backup paths from "grandfather" (or another ancestor) to "son". On the other hand, this method requires that "grandfather" (or another ancestor) has to know the status of the primary branches to "son". To do so, "son" can be continuously monitoring the status of the upstream link. Once detecting a fault there, the "son" should start notifying this information to all the routers on its backup path. In this fashion, whenever and wherever any fault occurs, our algorithm will be able to reconfigure the network to deliver multicast packets as long as the routing table size allows it.

The second challenge to be discussed is the implementation capability of large neural networks. Although the neural network size used in the current experiments is relatively small compared to the popular neural networks, it should be noted that we assumed that each SNPC has one neuron to evaluate the performance of the proposed communication architecture. In the final system, after SNPCs are integrated, each of them will contain 256 neurons. At that time, the system can accommodate almost 7000 neurons with a 3D-NoC network size of 3×3×3, as used in the current evaluation. This is a plausible scenario since, when setting the operation frequency of our 3DFT-SNN to 100MHz, it takes an average of 2.02ns to deliver a spike at a fault rate of 20%. With 256 neurons in an SNPC, it takes 517.12ns. This means that the network architecture can still perform 1,933.8× faster than the real-time requirement of SNNs; that is 1ms (spike rate up to 1KHz) [13]. For further applications require hundred thousands or even millions of neurons, our 3DFT-SNN chips are connected together. This is a common design manner of other SNN systems.

8

Conclusions and Future Work

In this thesis, we presented algorithms and architectures for spiking neural network systems based on 3D-NoC, named 3DNoC-SNN. The proposed system shows essential characteristics, such as low latency, high throughput, high reliability, and low power footprint that make it suitable for large-scale SNN-based embedded artificial intelligence (AI) implementations.

Before designing and implementing the 3DNoC-SNN, we presented a performance assessment method to early explore the performance of the 3D mesh-based inter-connect architecture. The assessment model is taken under different spiking neural network patterns and spike routing method (i. e., unicast (UC), multicast (MC), and broadcast (BC)). The analyzed results show that multicast is the most suitable for implementing SNNs.

Furthermore, we also analyzed the effect of connection faults over the 3DNoC-SNN system. As analyzed in the model, faults occurring connections lead to the degradation of system performance in terms of efficient bandwidth, delivered spike rate and the number of neurons that can be accommodated in each neural tile. As a result, the system demands a fault-tolerant technique to deal with this issue, and from there, avoid timing violation of spiking neuromorphic systems.

To deal with the interconnection challenge of SNN hardware implementations, we proposed a 3D-mesh interconnect architecture and spike multicast routing algorithms. The routing methods are based on the combination of K-means clustering and the tree-based multicast routing method. Adopting k-means as a partition method, helps to get overall balanced traffic and then improve system performance as well. Furthermore, to deal with connection faults, we also proposed a new fault-tolerant multicast routing algorithm that pre-defines primary and backup routing paths. When faults appear in the primary route, routers switch incoming spike packages via the backup path. This reduces recovery overhead, average latency, and enabling the system to avoid timing violation of SNNs.

From the performance evaluation, the results have shown the efficiency of the proposed architecture and algorithms. SNN topologies used in the evaluations are consistent to topologies analyzed in the assessment model, where realistic benchmarks belong to the randomly connected neural network (RNDC), and the synthetic benchmark is presented for Hopfield neural network (HF). While k-means based multicast routing algorithm (KMCR) have shown better result compared to XYZ unicast-based multicast (XYZ-UB) in terms of spike injection rate (25%), latency (14.43%), throughput (24.5%), shortest-path k-means based multicast routing algorithm (SP-KMCR) reduced latency of 12.2% than KMCR. These evaluation results are consistent with the analytical model. Furthermore, both KMCR and SP-KMCR also have good results in the 2D domain and the proposed system achieved better results compared to other 2D architectures (i.e., CMesh, H-NoC, Dragonfly), about 8.7% of higher injection rate and lower almost 2× of latency compared to Dragonfly. This also proves the results in the analytical model. Regarding fault tolerance, the proposed fault-tolerant multicast routing algorithm enables the system to sustain correct traffic communication with a fault rate up to 20%, while only suffering 16.23% longer latency and 5.49% extra area cost when compared to the baseline architecture. These evaluation results mentioned above have shown consistency in the analytical model.

In conclusion, this dissertation has presented algorithms and architectures for spiking neuromorphic systems. It provided a comprehensive set of analytical assessment, spike routing algorithms, and architectures for 3D neuromorphic systems. Furthermore, a fault-tolerant approach also presented to dealing with faults occurring in connections between neural tiles.

Though, there are still other several issues that need to be addressed. The first one is how to map spiking neural networks onto the system because it influences the overall performance of the interconnect architecture. In this thesis, we used the layer-to-layer mapping method, however, a comprehensive mapping method should be investigated. The second one is learning implementation which makes impacts on system performance and hardware complexity as well. Furthermore, the reliability of neuromorphic systems also needs to be investigated. In this work, we focused on connection faults, but other parts such as neurons, memories of the system also need to be concerned.

List of Publications

Refereed Journals

1. **The H. Vu**, Yuichi Okuyama, and Abderazek Ben Abdallah, “Comprehensive Analytic Performance Assessment and K-means based Multicast Routing Algorithm and Architecture for 3D-NoC of Spiking Neurons”, ACM Journal on Emerging Technologies in Computing Systems, 2019, *(in press)*. doi:10.1145/3340963.

2. **The H. Vu**, O. M. Ikechukwu, and Abderazek Ben Abdallah, ”Fault-Tolerant Spike Routing Algorithm and Architecture for Three Dimensional NoC-Based Neuromorphic Systems,” in IEEE Access, vol. 7, pp. 90436-90452, 2019. doi: 10.1109/ACCESS.2019.2925085.

3. **The H. Vu**, Yuichi Okuyama, and Abderazek Ben Abdallah, “Analytical performance assessment and high-throughput low-latency spike routing algorithm for spiking neural network systems”, The Journal of Supercomputing (2019), *(in press)*. doi:10.1007/s11227-019-02792-y.

Refereed International Conferences

1. **The H. Vu** and Abderazek Ben Abdallah, “A Low-latency Tree-based Multicast Spike Routing for Scalable Multicore Neuromorphic Chips”, ACM 5th Interna-

tional Conference of Computing for Engineering and Sciences, Hammamet, Tunisia, 2019.

2. **The H. Vu** and Abderazek Ben Abdallah, “Low-Latency K-Means Based Multicast Routing Algorithm and Architecture for Three Dimensional Spiking Neuromorphic Chips,” 2019 IEEE International Conference on Big Data and Smart Computing (BigComp), Kyoto, Japan, 2019, pp. 1-8. ***(Best Paper Award Runner-Up)***.

3. **The H. Vu**, Yuji Murakami, and Abderazek Ben Abdallah, ””Graceful Fault-Tolerant On-Chip Spike Routing Algorithm for Mesh-Based Spiking Neural Networks,” 2019 2nd International Conference on Intelligent Autonomous Systems (ICoIAS), Singapore, Singapore, 2019, pp. 76-80.

4. **The H. Vu**, R. Murakami, Yuichi Okuyama, and Abderazek Ben Abdallah, ”Efficient Optimization and Hardware Acceleration of CNNs towards the Design of a Scalable Neuro inspired Architecture in Hardware,” 2018 IEEE International Conference on Big Data and Smart Computing (BigComp), Shanghai, 2018, pp. 326-332.

References

[1] S. A. McKee, "Reflections on the memory wall," in *Proceedings of the 1st Conference on Computing Frontiers*, ser. CF '04. New York, NY, USA: ACM, 2004, pp. 162–. [Online]. Available: http://doi.acm.org/10.1145/977091.977115

[2] M. D. Hill, S. V. Adve, L. Ceze, M. J. Irwin, D. R. Kaeli, M. Martonosi, J. Torrellas, T. F. Wenisch, D. A. Wood, and K. A. Yelick, "21st century computer architecture," *CoRR*, vol. abs/1609.06756, 2016. [Online]. Available: http://arxiv.org/abs/1609.06756

[3] R. Courtland, "Transistors could stop shrinking in 2021," *IEEE Spectrum*, vol. 53, pp. 9–11, 2016.

[4] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha, "A million spiking-neuron integrated circuit with a scalable communication network and interface," *Science*, vol. 345, no. 6197, pp. 668–673, 2014.

[5] C. Mead, "Neuromorphic electronic systems," *Proceedings of the IEEE*, vol. 78, no. 10, pp. 1629–1636, Oct 1990.

[6] Y. LeCun, Y. Bengio, and G. E. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015. [Online]. Available: https://doi.org/10.1038/nature14539

[7] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, ser. NIPS'12. USA: Curran Associates Inc., 2012, pp. 1097–1105. [Online]. Available: http://dl.acm.org/citation.cfm?id=2999134.2999257

[8] A. Graves, A. Mohamed, and G. E. Hinton, "Speech recognition with deep recurrent neural networks," *CoRR*, vol. abs/1303.5778, 2013. [Online]. Available: http://arxiv.org/abs/1303.5778

[9] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CoRR*, vol. abs/1512.03385, 2015. [Online]. Available: http://arxiv.org/abs/1512.03385

[10] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.

[11] Q. V. Le, M. Ranzato, R. Monga, M. Devin, K. Chen, G. S. Corrado, J. Dean, and A. Y. Ng, "Building high-level features using large scale unsupervised learning," in *Proceedings of the 29th International Coference on International Conference on Machine Learning*, ser. ICML'12. USA: Omnipress, 2012, pp. 507–514. [Online]. Available: http://dl.acm.org/citation.cfm?id=3042573.3042641

[12] Y. Cao, Y. Chen, and D. Khosla, "Spiking deep convolutional neural networks for energy-efficient object recognition," *Int. J. Comput. Vision*, vol. 113, no. 1, pp. 54–66, May 2015.

[13] W. Gerstner and W. Kistler, *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press, 2002.

[14] K. Suzuki, Y. Okuyama, and A. B. Abdallah, "Hardware design of a leaky integrate and fire neuron core towards the design of a low-power neuro-inspired spike-based multicore soc," in *Information Processing Society Tohoku Branch Conference*, February 2018.

[15] J. H Goldwyn, N. S Imennov, M. Famulare, and E. Shea-Brown, "Stochastic differential equation models for ion channel noise in hodgkin-huxley neurons," in *Phys. Rev. E*, vol. 83, no. 1, 2011, pp. 4190–4208.

[16] F. A. et. al., "Truenorth: Design and tool flow of a 65 mw 1 million neuron programmable neurosynaptic chip," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 10, pp. 1537–1557, Oct 2015.

[17] P. Hasler and L. A. Akers, "Vlsi neural systems and circuits," in *Ninth Annual International Phoenix Conference on Computers and Communications. 1990 Conference Proceedings*, March 1990, pp. 31–37.

[18] J. . Lee and B. J. Sheu, "Parallel digital image restoration using adaptive vlsi neural chips," in *Proceedings., 1990 IEEE International Conference on Computer Design: VLSI in Computers and Processors*, Sep. 1990, pp. 126–129.

[19] L. Tarassenko, M. Brownlow, G. Marshall, J. Tombs, and A. Murray, "Real-time autonomous robot navigation using vlsi neural networks," in *Proceedings of the 3rd International Conference on Neural Information Processing Systems*, ser. NIPS'90. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1990, pp. 422–428. [Online]. Available: http://dl.acm.org/citation.cfm?id=2986766.2986823

[20] A. Ben Abdallah, *Advanced Multicore Systems-On-Chip Architecture, On-Chip Network, Design*. Springer, 2017.

[21] R. Hojabr, M. Modarressi, M. Daneshtalab, A. Yasoubi, and A. Khonsari, "Customizing clos network-on-chip for neural networks," *IEEE Transactions on Computers*, vol. 66, no. 11, pp. 1865–1877, Nov 2017.

[22] S. Carrillo, J. Harkin, L. J. McDaid, F. Morgan, S. Pande, S. Cawley, and B. McGinley, "Scalable hierarchical network-on-chip architecture for spiking neural network hardware implementations," *IEEE Transactions on Parallel and Distributed Systems*, vol. 24, no. 12, pp. 2451–2461, Dec 2013.

[23] D. Vainbrand and R. Ginosar, "Scalable network-on-chip architecture for configurable neural networks," *Microprocess. Microsyst.*, vol. 35, no. 2, pp. 152–166, Mar. 2011. [Online]. Available: http://dx.doi.org/10.1016/j.micpro.2010.08.005

[24] C. Torres-Huitzil and B. Girau, "Fault and error tolerance in neural networks: A review," *IEEE Access*, vol. 5, pp. 17 322–17 341, 2017.

[25] P. M. Furth and A. G. Andreou, "On fault probabilities and yield models for vlsi neural networks," *IEEE Journal of Solid-State Circuits*, vol. 32, no. 8, pp. 1284–1287, Aug 1997.

[26] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[27] M. Pfeiffer and T. Pfeil, "Deep learning with spiking neurons: Opportunities and challenges," *Frontiers in Neuroscience*, vol. 12, p. 774, 2018. [Online]. Available: https://www.frontiersin.org/article/10.3389/fnins.2018.00774

[28] M. Wofgang, “Networks of spiking neurons: The third generation of neural network models,” *Trans. Soc. Comput. Simul. Int.*, vol. 14, no. 4, pp. 1659–1671, Dec. 1997. [Online]. Available: http://dl.acm.org/citation.cfm?id=281543.281637

[29] M. Wolfgang, “On the relevance of time in neural computation and learning,” in *Proceedings of the 8th International Conference on Algorithmic Learning Theory*, ser. ALT ’97. London, UK, UK: Springer-Verlag, 1997, pp. 364–384. [Online]. Available: http://dl.acm.org/citation.cfm?id=647715.735598

[30] Broomhead, D. S., and D. Lowe, “Radial basis functions, multi-variable functional interpolation and adaptive networks,” *Royal Signals and Radar Establishment Malvern (United Kingdom)*, 1988.

[31] H. J. J., “Neurocomputing: Foundations of research,” J. A. Anderson and E. Rosenfeld, Eds. Cambridge, MA, USA: MIT Press, 1988, ch. Neural Networks and Physical Systems with Emergent Collective Computational Abilities, pp. 457–464. [Online]. Available: http://dl.acm.org/citation.cfm?id=65669.104422

[32] H. G. E. and T. J. Sejnowski, “Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1,” D. E. Rumelhart, J. L. McClelland, and C. PDP Research Group, Eds. Cambridge, MA, USA: MIT Press, 1986, ch. Learning and Relearning in Boltzmann Machines, pp. 282–317. [Online]. Available: http://dl.acm.org/citation.cfm?id=104279.104291

[33] J. L. Elman, “Finding structure in time,” *Cognitive Science*, vol. 14, no. 2, pp. 179–211, 1990.

[34] S. Haykin, *Neural Networks: A Comprehensive Foundation*, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 1998.

[35] T. H. Vu, R. Murakami, Y. Okuyama, and A. B. Abdallah, “Efficient optimization and hardware acceleration of cnns towards the design of a scalable neuro inspired architecture in hardware,” in *2018 IEEE International Conference on Big Data and Smart Computing (BigComp)*, Jan 2018, pp. 326–332.

[36] Y. Berg, R. L. Sigvartsen, T. S. Lande, and A. Abusland, “An analog feed-forward neural network with on-chip learning,” *Analog Integrated Circuits and Signal Processing*, vol. 9, no. 1, pp. 65–75, Jan 1996. [Online]. Available: https://doi.org/10.1007/BF00158853

[37] I. Bayraktaroglu, A. S. Ogrenci, G. Dundar, S. Balkir, and E. Alpaydin, “Annsys (an analog neural network synthesis system),” in *Proceedings of International Conference on Neural Networks (ICNN’97)*, vol. 2, June 1997, pp. 910–915 vol.2.

[38] G. Carvajal, M. Figueroa, D. Sbarbaro, and W. Valenzuela, “Analysis and compensation of the effects of analog vlsi arithmetic on the lms algorithm,” *IEEE Transactions on Neural Networks*, vol. 22, no. 7, pp. 1046–1060, July 2011.

[39] M. . Choi and F. M. A. Salam, “Implementation of feedforward artificial neural nets with learning using standard cmos vlsi technology,” in *1991., IEEE International Sympoisum on Circuits and Systems*, June 1991, pp. 1509–1512 vol.3.

[40] J. Fieres, K. Meier, and J. Schemmel, “A convolutional neural network tolerant of synaptic faults for low-power analog hardware,” in *Artificial Neural Networks in Pattern Recognition*, F. Schwenker and S. Marinai, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2006, pp. 122–132.

[41] J. Fieres, J. Schemmel, and K. Meier, “Training convolutional networks of threshold neurons suited for low-power hardware implementation,” in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, July 2006, pp. 21–28.

[42] K. Madani, P. Garda, and F. Devos, “Two analog counters for neural networks implementation,” in *ESSCIRC ’90: Sixteenth European Solid-State Circuits Conference*, vol. 1, Sep. 1990, pp. 233–236.

[43] C. C. Lu, C. Y. Hong, and H. Chen, “A scalable and programmable architecture for the continuous restricted boltzmann machine in vlsi,” in *2007 IEEE International Symposium on Circuits and Systems*, May 2007, pp. 1297–1300.

[44] C. R. Schneider and H. C. Card, “Analog cmos deterministic boltzmann circuits,” *IEEE Journal of Solid-State Circuits*, vol. 28, no. 8, pp. 907–914, Aug 1993.

[45] S. Kumar, K. Forward, and M. Palaniswami, “Performance evaluation of a risc neuro-processor for neural networks,” in *Proceedings of 3rd International Conference on High Performance Computing (HiPC)*, Dec 1996, pp. 351–356.

[46] K. Ben Khalifa, B. Girau, F. Alexandre, and M. H. Bedoui, “Parallel fpga implementation of self-organizing maps,” in *Proceedings. The 16th International Conference on Microelectronics, 2004. ICM 2004.*, Dec 2004, pp. 709–712.

[47] H. Tamukoh and M. Sekine, “A dynamically reconfigurable platform for self-organizing neural network hardware,” in *Neural Information Processing. Models and*

Applications, K. W. Wong, B. S. U. Mendis, and A. Bouzerdoum, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2010, pp. 439–446.

[48] S. P. Adhikari, H. Kim, R. K. Budhathoki, C. Yang, and L. O. Chua, "A circuit-based learning architecture for multilayer neural networks with memristor bridge synapses," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 1, pp. 215–223, Jan 2015.

[49] Y. Dong, S. Bagga, and W. Serdijn, "An inherently linear cmos multiplier," in *Proceedings of the program for research on integrated systems and circuits*, 01 2004, pp. 483–486.

[50] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*, 1st ed. Springer Publishing Company, Incorporated, 2010.

[51] V. Salapura, M. Gschwind, and O. Maischberger, "A fast fpga implementation of a general purpose neuron," in *Proceedings of the 4th International Workshop on Field-Programmable Logic and Applications: Field-Programmable Logic, Architectures, Synthesis and Applications*, ser. FPL '94. Berlin, Heidelberg: Springer-Verlag, 1994, pp. 175–182. [Online]. Available: http://dl.acm.org/citation.cfm?id=647921.740703

[52] A. Muthuramalingam, S. Himavathi, and E. Srinivasan, "Neural network implementation using fpga: Issues and application," *Int. J. Inform. Technol.*, vol. 4, pp. 2–12, 11 2007.

[53] J. Shawe-Taylor, P. Jeavons, and M. Van Daalen, "Probabilistic bit stream neural chip: Theory," *Connection Science*, vol. 3, pp. 317–328, 01 1991.

[54] M. Skrbek and M. Snorek, "Shift-add neural architecture," in *ICECS'99. Proceedings of ICECS '99. 6th IEEE International Conference on Electronics, Circuits and Systems (Cat. No.99EX357)*, vol. 1, Sep. 1999, pp. 411–414 vol.1.

[55] H. P. Graf, L. D. Jackel, R. E. Howard, B. Straughn, J. S. Denker, W. Hubbard, D. M. Tennant, and D. Schwartz, "Vlsi implementation of a neural network memory with several hundreds of neurons," in *AIP Conference Proceedings 151 on Neural Networks for Computing*. Woodbury, NY, USA: American Institute of Physics Inc., 1987, pp. 182–187. [Online]. Available: http://dl.acm.org/citation.cfm?id=24140.24167

[56] A. J. Agranat, C. F. Neugebauer, and A. Yariv, “A ccd based neural network integrated circuit with 64k analog programmable synapses,” in *1990 IJCNN International Joint Conference on Neural Networks*, June 1990, pp. 551–555 vol.2.

[57] T. Morishita, Y. Tamura, and T. Otsuki, “A bicmos analog neural network with dynamically updated weights,” in *1990 37th IEEE International Conference on Solid-State Circuits*, Feb 1990, pp. 142–143.

[58] Eberhard, Duong, and Thakoor, “Design of parallel hardware neural network systems from custom analog vlsi ’building block’ chips,” in *International 1989 Joint Conference on Neural Networks*, 1989, pp. 183–190 vol.2.

[59] Holler, Tam, Castro, and Benson, “An electrically trainable artificial neural network (etann) with 10240 ’floating gate’ synapses,” in *International 1989 Joint Conference on Neural Networks*, 1989, pp. 191–196 vol.2.

[60] B. M. Wilamowski, J. Binfet, and O. Kaynak, “Vlsi implementation of neural networks,” *International journal of neural systems*, vol. 103, pp. 191–197, 2000.

[61] C. Mead, *Analog VLSI and Neural Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 1989.

[62] M. Bohrn, L. Fujcik, and R. Vrba, “Field programmable neural array for feed-forward neural networks,” in *2013 36th International Conference on Telecommunications and Signal Processing (TSP)*, July 2013, pp. 727–731.

[63] M. D. Corp., *Md1220 neural bit slice*. data sheet, lake Mary, 1990.

[64] D. Kim, H. Kim, H. Kim, G. Han, and D. Chung, “A simd neural network processor for image processing,” in *Advances in Neural Networks – ISNN 2005*, J. Wang, X.-F. Liao, and Z. Yi, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 665–672.

[65] N. Bruels, “Ma16 - programmable vlsi array processor for neuronal networks and matrix-based signal processing, user description,” in *Tech. Rep. 1.3*. Siemens AG, Cor- porate Research and Development Division, Munich, Germany, 1993.

[66] M. Glesner and W. Pochmuller, *Neurocomputers: An overview of Neural Networks in VLSI*. Chapman and Hall, London, 1994.

[67] P. Ienne, “Digital hardware architectures for neural networks,” *n SPEEDUP Journal*, vol. Vol. 9, No. 1, pp. 18–25, 1995.

[68] D. Yiping and W. Takahiro, “High performance noc architecture for two hidden layers bp neural network,” in *2008 International SoC Design Conference*, vol. 01, Nov 2008, pp. I–269–I–272.

[69] S. Rueping, K. Goser, and U. Rueckert, “A chip for self-organizing feature maps,” in *Proceedings of the Fourth International Conference on Microelectronics for Neural Networks and Fuzzy Systems*, Sep. 1994, pp. 26–33.

[70] A. A. Dibazar, A. Bangalore, , S. George, W. Yamada, and T. W. Berger, “Hardware implementation of dynamic synapse neural networks for acoustic sound recognition,” in *The 2006 IEEE International Joint Conference on Neural Network Proceedings*, July 2006, pp. 2015–2022.

[71] M. Gschwind, V. Salapura, and O. Maischberger, “Space efficient neural net implementation,” in *Proc. of the Second ACM Workshop on Field-Programmable Gate Arrays*. CONCLUSION, 1994.

[72] M. Krips, T. Lammert, and A. Kummert, “Fpga implementation of a neural network for a real-time hand tracking system,” in *Proceedings First IEEE International Workshop on Electronic Design, Test and Applications ’2002*, Jan 2002, pp. 313–317.

[73] F. Yang and M. Paindavoine, “Implementation of an rbf neural network on embedded systems: real-time face tracking and identity verification,” *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 1162–1175, Sep. 2003.

[74] J. Liu, M. A. Brooke, and K. Hirotsu, “A cmos feedforward neural-network chip with on-chip parallel learning for oscillation cancellation,” *IEEE Transactions on Neural Networks*, vol. 13, no. 5, pp. 1178–1186, Sep. 2002.

[75] B. E. Brown, X. Yu, and S. L. Garverick, “Mixed-mode analog vlsi continuous-time recurrent neural network,” 01 2004, pp. 398–403.

[76] J. O’Keefe, “Hippocampus, theta, and spatial memory,” *Current Opinion in Neurobiology*, vol. 3, no. 6, pp. 917 – 924, 1993. [Online]. Available: http://www.sciencedirect.com/science/article/pii/095943889390163S

[77] E. M. Izhikevich, “Which model to use for cortical spiking neurons?” *IEEE Transactions on Neural Networks*, vol. 15, no. 5, pp. 1063–1070, Sept 2004.

[78] A. Hodgkin and A. Huxley, “A quantitative description of membrane current and its application to conduction and excitation in nerve,” *The Journal of Physiology*, vol. 117, pp. 500–544, 1952.

[79] T. Pfeil, “Exploring the potential of brain-inspired computing,” 2015. [Online]. Available: http://archiv.ub.uni-heidelberg.de/volltextserver/id/eprint/18258

[80] E. M. Izhikevich, “Simple model of spiking neurons,” *IEEE Transactions on Neural Networks*, vol. 14, no. 6, pp. 1569–1572, Nov 2003.

[81] N. Burkitt, “A review of the integrate-and-fire neuron model: I. homogeneous synaptic input,” *Biol. Cybern.*, vol. 95, no. 1, pp. 1–19, Jun 2006.

[82] Y. Dan and M. ming Poo, “Spike timing-dependent plasticity of neural circuits,” *Neuron*, vol. 44, no. 1, pp. 23–30, sep 2004.

[83] and A. Murray, F. Worgotter, K. Cameron, and V. Boonsobhak, “A neuromorphic depth-from-motion vision model with stdp adaptation,” *IEEE Transactions on Neural Networks*, vol. 17, no. 2, pp. 482–495, March 2006.

[84] X. Jin, A. Rast, F. Galluppi, S. Davies, and S. Furber, “Implementing spike-timing-dependent plasticity on spinnaker neuromorphic hardware,” in *The 2010 International Joint Conference on Neural Networks (IJCNN)*, July 2010, pp. 1–8.

[85] S. Song, K. D. Miller, and L. F. Abbott, “Competitive hebbian learning through spike-timing-dependent synapticplasticity,” *Nature Neuroscience*, vol. 3, no. 9, pp. 919–926, sep 2000.

[86] G.-Q. Bi and H.-X. Wang, “Temporal asymmetry in spike timing-dependent synaptic plasticity,” *Physiology & Behavior*, vol. 77, no. 4, pp. 551 – 555, 2002. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0031938402009332

[87] A. Cassidy, A. G. Andreou, and J. Georgiou, “A combinational digital logic approach to stdp,” in *2011 IEEE International Symposium of Circuits and Systems (ISCAS)*, May 2011, pp. 673–676.

[88] C. Frenkel, M. Lefebvre, J. Legat, and D. Bol, “A 0.086-mm^2 12.7-pj/sop 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm cmos,” *IEEE Transactions on Biomedical Circuits and Systems*, vol. 13, no. 1, pp. 145–158, Feb 2019.

[89] S. Yin, S. K. Venkataramanaiah, G. K. Chen, R. Krishnamurthy, Y. Cao, C. Chakrabarti, and J. Seo, “Algorithm and hardware design of discrete-time spiking neural networks based on back propagation with binary activations,” *CoRR*, vol. abs/1709.06206, 2017. [Online]. Available: http://arxiv.org/abs/1709.06206

[90] P. U. Diehl, D. Neil, J. Binas, M. Cook, S. Liu, and M. Pfeiffer, "Fast-classifying, high-accuracy spiking deep networks through weight and threshold balancing," in *2015 International Joint Conference on Neural Networks (IJCNN)*, July 2015, pp. 1–8.

[91] P. Merolla, J. Arthur, R. Alvarez, J. Bussat, and K. Boahen, "A multicast tree router for multichip neuromorphic systems," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 61, no. 3, pp. 820–833, March 2014.

[92] K. A. Boahen, *Communicating Neuronal Ensembles between Neuromorphic Chips*. Boston, MA: Springer US, 1998, pp. 229–259. [Online]. Available: https://doi.org/10.1007/978-0-585-28001-1_11

[93] J. Park, T. Yu, C. Maier, S. Joshi, and G. Cauwenberghs, "Live demonstration: Hierarchical address-event routing architecture for reconfigurable large scale neuromorphic systems," in *2012 IEEE International Symposium on Circuits and Systems*, May 2012, pp. 707–711.

[94] A. Mortara, E. A. Vittoz, and P. Venier, "A communication scheme for analog vlsi perceptive systems," *IEEE Journal of Solid-State Circuits*, vol. 30, no. 6, pp. 660–669, June 1995.

[95] D. Vainbrand and R. Ginosar, "Network-on-chip architectures for neural networks," in *2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, May 2010, pp. 135–144.

[96] S. R. Deiss, R. J. Douglas, and A. M. Whatley, "Pulsed neural networks," W. Maass and C. M. Bishop, Eds. Cambridge, MA, USA: MIT Press, 1999, ch. A Pulse-coded Communications Infrastructure for Neuromorphic Systems, pp. 157–178. [Online]. Available: http://dl.acm.org/citation.cfm?id=296533.296540

[97] F. Sargeni and V. Bonaiuto, "An interconnection architecture for integrate and fire neuromorphic multi-chip networks," in *2009 52nd IEEE International Midwest Symposium on Circuits and Systems*, Aug 2009, pp. 877–880.

[98] S. B. Furber, F. Galluppi, S. Temple, and L. A. Plana, "The spinnaker project," *Proceedings of the IEEE*, vol. 102, no. 5, pp. 652–665, May 2014.

[99] S. Furber, "Large-scale neuromorphic computing systems," *Journal of Neural Engineering*, vol. 13, no. 5, p. 051001, 2016.

[100] A. Jiménez-Fernández, E. Cerezuela-Escudero, L. Miró-Amarante, M. J. Domínguez-Morales, F. de Asís Gómez-Rodríguez, A. Linares-Barranco, and

G. Jiménez-Moreno, “A binaural neuromorphic auditory sensor for fpga: A spike signal processing approach,” *IEEE Transactions on Neural Networks and Learning Systems*, vol. 28, no. 4, pp. 804–818, April 2017.

[101] J. A. Bailey, P. R. Wilson, A. D. Brown, and J. Chad, “Behavioural simulation and synthesis of biological neuron systems using vhdl,” in *2008 IEEE International Behavioral Modeling and Simulation Workshop*, Sep. 2008, pp. 7–12.

[102] T. S. T. Mak, G. Rachmuth, K. Lam, and C. Poon, “A component-based fpga design framework for neuronal ion channel dynamics simulations,” *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, vol. 14, no. 4, pp. 410–418, Dec 2006.

[103] S. Yang, J. Wang, B. Deng, C. Liu, H. Li, C. Fietkiewicz, and K. A. Loparo, “Real-time neuromorphic system for large-scale conductance-based spiking neural networks,” *IEEE Transactions on Cybernetics*, pp. 1–14, 2018.

[104] T. Kaulmann, M. Ferber, U. Witkowski, and U. Rückert, “Analog vlsi implementation of adaptive synapses in pulsed neural networks,” in *Proceedings of the 8th International Conference on Artificial Neural Networks: Computational Intelligence and Bioinspired Systems*, ser. IWANN’05. Berlin, Heidelberg: Springer-Verlag, 2005, pp. 455–462. [Online]. Available: http://dx.doi.org/10.1007/11494669_56

[105] R. A. Nawrocki, S. E. Shaheen, and R. M. Voyles, “A neuromorphic architecture from single transistor neurons with organic bistable devices for weights,” in *The 2011 International Joint Conference on Neural Networks*, July 2011, pp. 450–456.

[106] T. Yu and G. Cauwenberghs, “Analog vlsi neuromorphic network with programmable membrane channel kinetics,” in *2009 IEEE International Symposium on Circuits and Systems*, May 2009, pp. 349–352.

[107] J. Schemmel, A. Grübl, S. Hartmann, A. Kononov, C. Mayr, K. Meier, S. Millner, J. Partzsch, S. Schiefer, S. Scholze, R. Schüffny, and M. Schwartz, “Live demonstration: A scaled-down version of the brainscales wafer-scale neuromorphic system,” in *2012 IEEE International Symposium on Circuits and Systems*, May 2012, pp. 702–702.

[108] S. Nease, S. George, P. Hasler, S. Koziol, and S. Brink, “Modeling and implementation of voltage-mode cmos dendrites on a reconfigurable analog platform,”

IEEE Transactions on Biomedical Circuits and Systems, vol. 6, no. 1, pp. 76–84, Feb 2012.

[109] E. Farquhar, C. Gordon, and P. Hasler, "A field programmable neural array," in *2006 IEEE International Symposium on Circuits and Systems*, May 2006, pp. 4 pp.–4117.

[110] M. Liu, H. Yu, and W. Wang, "Fpaa based on integration of cmos and nanojunction devices for neuromorphic applications," in *Nano-Net*, M. Cheng, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2009, pp. 44–48.

[111] R. H. R. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, jun 2000.

[112] S. Han, "Biologically plausible vlsi neural network implementation with asynchronous neuron and spike-based synapse," in *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 5, July 2005, pp. 3244–3248 vol. 5.

[113] D. Del Corso and L. M. Reyneri, "Mixing analog and digital techniques for silicon neural networks," in *IEEE International Symposium on Circuits and Systems*, May 1990, pp. 2446–2449 vol.3.

[114] S. Satyanarayana, Y. P. Tsividis, and H. P. Graf, "A reconfigurable vlsi neural network," *IEEE Journal of Solid-State Circuits*, vol. 27, no. 1, pp. 67–81, Jan 1992.

[115] G. Charles, C. Gordon, and W. E. Alexander, "An implementation of a biological neural model using analog-digital integrated circuits," in *2008 IEEE International Behavioral Modeling and Simulation Workshop*, Sep. 2008, pp. 78–83.

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

[117] J. S. et. al., "Truenorth ecosystem for brain-inspired computing: Scalable systems, software, and applications," in *SC '16: Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, Nov 2016, pp. 130–141.

[118] M. A. Petrovici, B. Vogginger, P. Müller, O. Breitwieser, M. Lundqvist, L. Muller, M. Ehrlich, A. Destexhe, A. Lansner, R. Schüffny, J. Schemmel, and K. Meier,

"Characterization and compensation of network-level anomalies in mixed-signal neuromorphic modeling platforms," *PLOS ONE*, vol. 9, no. 10, pp. 1–30, 10 2014. [Online]. Available: https://doi.org/10.1371/journal.pone.0108590

[119] J. Wu and S. Furber, "A multicast routing scheme for a universal spiking neural network architecture," *The Computer Journal*, vol. 53, no. 3, pp. 280–288, Apr 2009.

[120] M. Henry, "The blue brain project," *Nature Reviews Neuroscience*, vol. 2, no. 7, pp. 153–159, 2006.

[121] F. Naveros, N. R. Luque, J. A. Garrido, R. R. Carrillo, M. Anguita, and E. Ros, "A spiking neural simulator integrating event-driven and time-driven computation schemes using parallel cpu-gpu co-processing: A case study," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 7, pp. 1567–1574, July 2015.

[122] J. Lazzaro and J. Wawrzynek, "A multi-sender asynchronous extension to the aer protocol," in *Proceedings Sixteenth Conference on Advanced Research in VLSI*, Mar 1995, pp. 158–169.

[123] S. Carrillo, J. Harkin, L. McDaid, S. Pande, S. Cawley, B. McGinley, and F. Morgan, "Advancing interconnect density for spiking neural network hardware implementations using traffic-aware adaptive network-on-chip routers," *Neural Networks*, vol. 33, pp. 42 – 57, 2012.

[124] T. H. Vu, Y. Okuyama, and A. B. Abdallah, "An efficient k-means based multicast routing algorithm and architecture for spiking neural network systems," in *2019 IEEE International Conference on Big Data and Smart Computing (BigComp)*, vol. in press, Fab 2019.

[125] K. A. Boahen, "Point-to-point connectivity between neuromorphic chips using address events," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 47, no. 5, pp. 416–434, May 2000.

[126] S. Liu, J. Kramer, G. Indiveri, T. Delbrück, T. Burg, and R. J. Douglas, "Orientation-selective avlsi spiking neurons," *Neural Networks*, vol. 14, no. 6-7, pp. 629–643, 2001.

[127] S. Moradi, N. Qiao, F. Stefanini, and G. Indiveri, "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (dynaps)," *IEEE Transactions on Biomedical Circuits and Systems*, vol. 12, no. 1, pp. 106–122, Feb 2018.

[128] J. Schemmel, J. Fieres, and K. Meier, "Wafer-scale integration of analog neural networks," in *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, June 2008, pp. 431–438.

[129] B. Belhadj, A. Valentian, P. Vivet, M. Duranton, L. He, and O. Temam, "The improbable but highly appropriate marriage of 3d stacking and neuromorphic accelerators," in *2014 International Conference on Compilers, Architecture and Synthesis for Embedded Systems (CASES)*, Oct 2014, pp. 1–9.

[130] M. A. Ehsan, Z. Zhou, and Y. Yi, "Modeling and analysis of neuronal membrane electrical activities in 3d neuromorphic computing system," in *2017 IEEE International Symposium on Electromagnetic Compatibility Signal/Power Integrity (EMCSI)*, Aug 2017, pp. 745–750.

[131] D. Xiang and K. Shen, "A new unicast-based multicast scheme for network-on-chip router and interconnect testing," *ACM Trans. Des. Autom. Electron. Syst.*, vol. 21, no. 2, pp. 24:1–24:23, Jan. 2016.

[132] M. Ebrahimi, M. Daneshtalab, P. Liljeberg, J. Plosila, J. Flich, and H. Tenhunen, "Path-based partitioning methods for 3d networks-on-chip with minimal adaptive routing," *IEEE Transactions on Computers*, vol. 63, no. 3, pp. 718–733, March 2014.

[133] F. A. Samman, T. Hollstein, and M. Glesner, "Adaptive and deadlock-free tree-based multicast routing for networks-on-chip," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 18, no. 7, pp. 1067–1080, July 2010.

[134] X. Lin and L. M. Ni, "Multicast communication in multicomputer networks," *IEEE Transactions on Parallel and Distributed Systems*, vol. 4, no. 10, pp. 1105–1117, Oct 1993.

[135] C. H. Sequin and R. D. Clay, "Fault tolerance in artificial neural networks," in *1990 IJCNN International Joint Conference on Neural Networks*, June 1990, pp. 703–708 vol.1.

[136] T. Poggio and F. Girosi, "Networks for approximation and learning," *Proceedings of the IEEE*, vol. 78, no. 9, pp. 1481–1497, Sep. 1990.

[137] Naihong Wei, Shiyuan Yang, and Shibai Tong, "A modified learning algorithm for improving the fault tolerance of bp networks," in *Proceedings of International Conference on Neural Networks (ICNN'96)*, vol. 1, June 1996, pp. 247–252 vol.1.

[138] A. Hashmi, H. Berry, O. Temam, and M. Lipasti, "Automatic abstraction and fault tolerance in cortical microachitectures," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, June 2011, pp. 1–10.

[139] J. Deng, Y. Rang, Z. Du, Y. Wang, H. Li, O. Temam, P. Ienne, D. Novo, X. Li, Y. Chen, and C. Wu, "Retraining-based timing error mitigation for hardware neural networks," in *2015 Design, Automation Test in Europe Conference Exhibition (DATE)*, March 2015, pp. 593–596.

[140] M. Naeem, L. J. McDaid, J. Harkin, J. J. Wade, and J. Marsland, "On the role of astroglial syncytia in self-repairing spiking neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 26, no. 10, pp. 2370–2380, Oct 2015.

[141] L. . Chu and B. W. Wah, "Fault tolerant neural networks with hybrid redundancy," in *1990 IJCNN International Joint Conference on Neural Networks*, June 1990, pp. 639–649 vol.2.

[142] M. D. Emmerson and R. I. Damper, "Determining and improving the fault tolerance of multilayer perceptrons in a pattern-recognition application," *IEEE Transactions on Neural Networks*, vol. 4, no. 5, pp. 788–793, Sep. 1993.

[143] C. Khunasaraphan, K. Vanapipat, and C. Lursinsap, "Weight shifting techniques for self-recovery neural networks," *IEEE Transactions on Neural Networks*, vol. 5, no. 4, pp. 651–658, July 1994.

[144] J. Liu, J. Harkin, L. P. Maguire, L. J. McDaid, and J. J. Wade, "Spanner: A self-repairing spiking neural network hardware architecture," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 29, no. 4, pp. 1287–1300, April 2018.

[145] S. Piche, "Robustness of feedforward neural networks," in *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks*, vol. 2, June 1992, pp. 346–351 vol.2.

[146] Ching-Tai Chin, K. Mehrotra, C. K. Mohan, and S. Rankat, "Training techniques to obtain fault-tolerant neural networks," in *Proceedings of IEEE 24th International Symposium on Fault- Tolerant Computing*, June 1994, pp. 360–369.

[147] R. Rojas, *Neural Networks: A Systematic Introduction*. Berlin, Heidelberg: Springer-Verlag, 1996.

[148] R. Legenstein and W. Maass, "Edge of chaos and prediction of computational performance for neural circuit models," *Neural Netw.*, vol. 20, no. 3, pp. 323–334, Apr. 2007. [Online]. Available: http://dx.doi.org/10.1016/j.neunet.2007.04.017

[149] D. K. Nam, M. Michael, O. Yuichi, and A. A. Ben, "A low-overhead soft–hard fault-tolerant architecture, design and management scheme for reliable high-performance many-core 3d-noc systems," *The Journal of Supercomputing*, vol. 73, no. 6, pp. 2705–2729, Jun 2017. [Online]. Available: https://doi.org/10.1007/s11227-016-1951-0

[150] A. B. Ahmed and A. B. Abdallah, "Graceful deadlock-free fault-tolerant routing algorithm for 3d network-on-chip architectures," *Journal of Parallel and Distributed Computing*, vol. 74, no. 4, pp. 2229 – 2240, 2014. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0743731514000045

[151] A. Ben Ahmed and A. Ben Abdallah, "Architecture and design of high-throughput, low-latency, and fault-tolerant routing algorithm for 3d-network-on-chip (3d-noc)," *The Journal of Supercomputing*, vol. 66, no. 3, pp. 1507–1532, Dec 2013. [Online]. Available: https://doi.org/10.1007/s11227-013-0940-9

[152] A. B. Ahmed and A. B. Abdallah, "La-xyz: Low latency, high throughput look-ahead routing algorithm for 3d network-on-chip (3d-noc) architecture," in *2012 IEEE 6th International Symposium on Embedded Multicore SoCs*, Sep. 2012, pp. 167–174.

[153] ——, "Low-overhead routing algorithm for 3d network-on-chip," in *2012 Third International Conference on Networking and Computing*, Dec 2012, pp. 23–32.

[154] B. Parhami, "Exact formulas for the average internode distance in mesh and binary tree networks," *Computer Science and Information Technology*, vol. 1, pp. 165–168, 2013.

[155] T. H. Vu, Y. Okuyama, and A. B. Abdallah, "Analytical performance assessment and high-throughput low-latency spike routing algorithm for spiking neural network systems," *The Journal of Supercomputing*, 2019, (in press).

[156] S. H. Strogatz, "Exploring complex networks," *Nature*, vol. 410, no. 6825, pp. 268–276, mar 2001. [Online]. Available: https://doi.org/10.1038%2F35065725

[157] T. H. Vu, Y. Okuyama, and A. B. Abdallah, "Comprehensive analytic performance assessment and k-means based multicast routing algorithm and architecture for

3d-noc of spiking neurons," *ACM Journal on Emerging Technologies in Computing Systems*, 2019, (in press).

[158] T. H. Vu, Y. Murakami, and A. B. Abdallah, "A low-latency tree-based multicast spike routing for scalable multicore neuromorphic chips," in *ACM 5th International Conference of Computing for Engineering and Sciences, Hammamet, Tunisia*, July 2019.

[159] T. H. Vu, O. M. Ikechukwu, and A. B. Abdallah, "Fault-tolerant spike routing algorithm and architecture for three dimensional noc-based neuromorphic systems," *IEEE Access*, vol. 7, pp. 90 436–90 452, 2019.

[160] T. H. Vu, Y. Murakami, and A. B. Abdallah, "Graceful fault-tolerant on-chip spike routing algorithm for mesh-based spiking neural networks," in *2019 2nd International Conference on Intelligent Autonomous Systems (ICoIAS), Singapore*, February 2019.

[161] C.-H. Chao, K.-Y. Jheng, H.-Y. Wang, J.-C. Wu, and A.-Y. Wu, "Traffic- and thermal-aware run-time thermal management scheme for 3d noc systems," in *Proceedings of the 2010 Fourth ACM/IEEE International Symposium on Networks-on-Chip*, ser. NOCS '10. Washington, DC, USA: IEEE Computer Society, 2010, pp. 223–230. [Online]. Available: http://dx.doi.org/10.1109/NOCS.2010.32

[162] A. B. Ahmed, A. B. Ahmed, and A. B. Abdallah, "Deadlock-recovery support for fault-tolerant routing algorithms in 3d-noc architectures," in *2013 IEEE 7th International Symposium on Embedded Multicore Socs*, Sep. 2013, pp. 67–72.

[163] A. B. Ahmed, T. Ochi, S. Miura, and A. B. Abdallah, "Run-time monitoring mechanism for efficient design of application-specific noc architectures in multi/-manycore era," in *2013 Seventh International Conference on Complex, Intelligent, and Software Intensive Systems*, July 2013, pp. 440–445.

[164] B. A. Akram and B. A. Abderazek, "Adaptive fault-tolerant architecture and routing algorithm for reliable many-core 3d-noc systems," *J. Parallel Distrib. Comput.*, vol. 93, no. C, pp. 30–43, Jul. 2016.

[165] K. N. Dang, A. B. Ahmed, Y. Okuyama, and B. A. Abderazek, "Scalable design methodology and online algorithm for tsv-cluster defects recovery in highly reliable 3d-noc systems," *IEEE Transactions on Emerging Topics in Computing*, pp. 1–1, 2017.

[166] K. N. Dang, A. Ben Ahmed, X. Tran, Y. Okuyama, and A. Ben Abdallah, “A comprehensive reliability assessment of fault-resilient network-on-chip using analytical model,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 25, no. 11, pp. 3099–3112, Nov 2017.

[167] A. Mortara and E. A. Vittoz, “A communication architecture tailored for analog vlsi artificial neural networks: intrinsic performance and limitations,” *IEEE Transactions on Neural Networks*, vol. 5, no. 3, pp. 459–466, May 1994.

[168] K. N. Dang, M. Meyer, Y. Okuyama, and A. B. Abdallah, “Reliability assessment and quantitative evaluation of soft-error resilient 3d network-on-chip systems,” in *2016 IEEE 25th Asian Test Symposium (ATS)*, Nov 2016, pp. 161–166.

[169] D. N. Khanh, A. A. Ben, A. A. Ben, and T. X. T, “Tsv-ias: Analytic analysis and low-cost non-preemptive on-line detection and correction method for tsv defects,” in *2019 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, 2019.

[170] Y. Ji, Y. Zhang, H. Liu, and W. Zheng, “Optimized mapping spiking neural networks onto network-on-chip,” *Algorithms and Architectures for Parallel Processing*, pp. 38–52, 2016.

[171] E. Stromatias, M. Soto, T. Serrano-Gotarredona, and B. Linares-Barranco, “An event-driven classifier for spiking neural networks fed with synthetic or dynamic vision sensor data,” *Frontiers in Neuroscience*, vol. 11, p. 350, 2017.

[172] N. Qiao, H. Mostafa, F. Corradi, M. Osswald, F. Stefanini, D. Sumislawska, and G. Indiveri, “A reconfigurable on-line learning spiking neuromorphic processor comprising 256 neurons and 128k synapses,” *Frontiers in Neuroscience*, vol. 9, p. 141, 2015.

[173] NanGate Inc., “Nangate Open Cell Library 45 nm,” http://www.nangate.com/, (accessed 2019.02.25).

[174] NCSU Electronic Design Automation, “FreePDK3D45 3D-IC process design kit,” http://www.eda.ncsu.edu/wiki/FreePDK3D45:Contents, (accessed 2019.02.25).

[175] S. Pande, F. Morgan, S. Cawley, B. McGinley, S. Carrillo, J. Harkin, and L. McDaid, “Embrace-sysc for analysis of noc-based spiking neural network architectures,” in *2010 International Symposium on System on Chip*, Sept 2010, pp. 139–145.

[176] S. Cawley, F. Morgan, B. Mcginley, S. Pande, L. Mcdaid, S. Carrillo, and J. Harkin, “Hardware spiking neural network prototyping and application,” *Genetic Programming and Evolvable Machines*, vol. 12, no. 3, pp. 257–280, Sep. 2011.

[177] E. Pasero and M. Perri, “Hw-sw codesign of a flexible neural controller through a fpga-based neural network programmed in vhdl,” in *2004 IEEE International Joint Conference on Neural Networks (IEEE Cat. No.04CH37541)*, vol. 4, July 2004, pp. 3161–3165 vol.4.

[178] N. Akbari and M. Modarressi, “A high-performance network-on-chip topology for neuromorphic architectures,” in *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, vol. 2, July 2017, pp. 9–16.

[179] Y. Dong, C. Li, Z. Lin, and T. Watanabe, “Multiple network-on-chip model for high perfor- mance neural network,” *Journal of Semiconductor Technology and Science*, vol. 10, 2010.

[180] J. Liu, J. Harkin, L. P. Maguire, L. J. McDaid, J. J. Wade, and G. Martin, “Scalable networks-on-chip interconnected architecture for astrocyte-neuron networks,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 63, no. 12, pp. 2290–2303, Dec 2016.

[181] S. Carrillo, J. Harkin, L. J. McDaid, S. Pande, S. Cawley, B. McGinley, and F. Morgan, “Hierarchical network-on-chip and traffic compression for spiking neural network implementations,” in *2012 IEEE/ACM Sixth International Symposium on Networks-on-Chip*, May 2012, pp. 83–90.