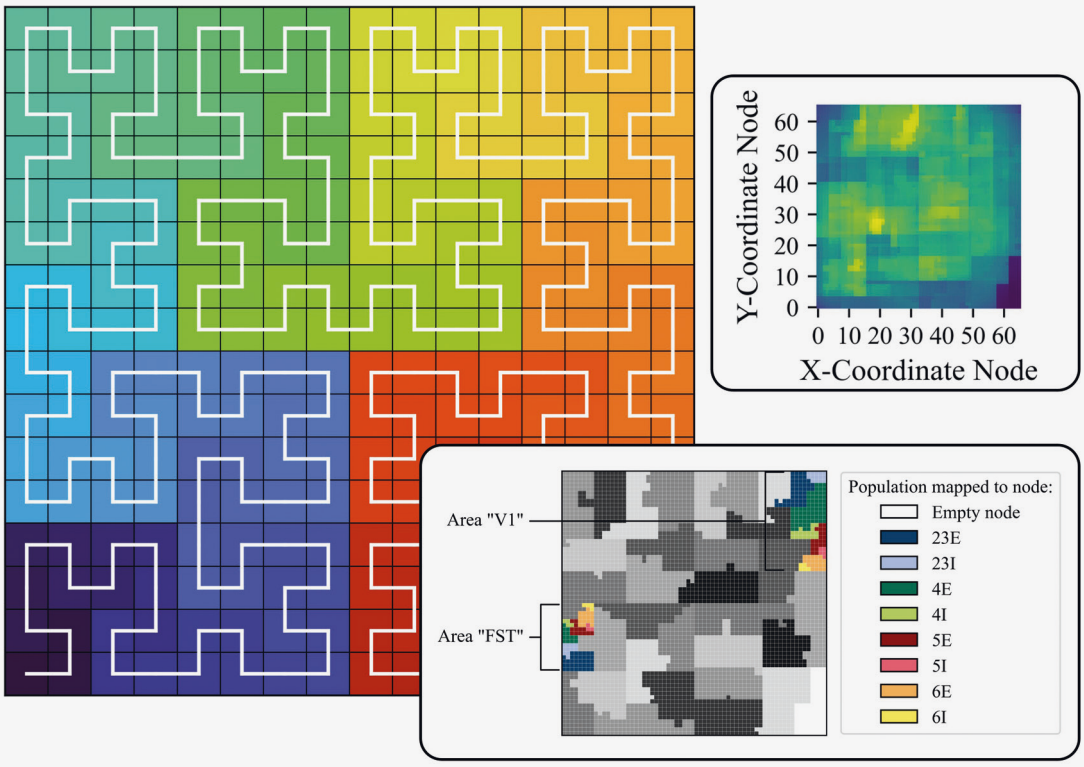


NeuCoNS and Stacked-Net: Facilitating the Communication for Accelerated Neuroscientific Simulations

Robert Kleijnen

Information
Band / Volume 106
ISBN 978-3-95806-788-2

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum

Schriften des Forschungszentrums Jülich
Reihe Information / Information Band / Volume 106

Forschungszentrum Jülich GmbH
Zentralinstitut für Engineering, Elektronik und Analytik (ZEA)
Systeme der Elektronik (ZEA-2)

NeuCoNS and Stacked-Net: Facilitating the Communication for Accelerated Neuroscientific Simulations

Robert Kleijnen

Schriften des Forschungszentrums Jülich
Reihe Information / Information Band / Volume 106

ISSN 1866-1777 ISBN 978-3-95806-788-2

Bibliografische Information der Deutschen Nationalbibliothek.
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der
Deutschen Nationalbibliografie; detaillierte Bibliografische Daten
sind im Internet über http://dnb.d-nb.de abrufbar.

Herausgeber und Vertrieb:	Forschungszentrum Jülich GmbH Zentralbibliothek, Verlag 52425 Jülich Tel.: +49 2461 61-5368 Fax: +49 2461 61-6103 zb-publikation@fz-juelich.de www.fz-juelich.de/zb
Umschlaggestaltung:	Grafische Medien, Forschungszentrum Jülich GmbH
Druck:	Grafische Medien, Forschungszentrum Jülich GmbH
Copyright:	Forschungszentrum Jülich 2024

Schriften des Forschungszentrums Jülich
Reihe Information / Information, Band / Volume 106

D 464 (Diss. Duisburg-Essen, Univ., 2024)

ISSN 1866-1777
ISBN 978-3-95806-788-2

Vollständig frei verfügbar über das Publikationsportal des Forschungszentrums Jülich (JuSER)
unter www.fz-juelich.de/zb/openaccess.

I have no special talent...
I am only passionately curious.

Albert Einstein

Acknowledgement

First and foremost, I would like to start by thanking Prof. Dr.-Ing. Stefan van Waasen for giving me the opportunity to work on this project.

I would like to express my deepest gratitude to my supervisor Dr.-Ing Markus Robens. Thank you for your advice and support through the years, especially during the stressful periods when a deadline was approaching. Your dedication and meticulousness while proofreading my publications as well as this thesis allowed me to achieve the quality of work that I strive for. I could always count on your help and this thesis would not have been possible without your supervision.

Special thanks goes to my former team leader Dr. Michael Schiek. You were always available on short notice to offer your help and insights on my scientific contributions. Even though your career path moved you to a different institute during the final phase of my thesis, your help and guidance over the years are invaluable to me.

I would also like to thank all my colleagues at the Central Institute of Engineering, Electronics and Analytics - Electronic Systems (ZEA-2), whom I have had the pleasure of working with, for their kindness and company. A special mention goes to my fellow PhD students, both those who have already finished and those still pursuing their PhD, who were along with me on this challenging but exciting journey. I cherish the memories we made together.

Thanks should also go to Fredrik Pettersson for his contribution to my research into the use of VLSI-based place and route algorithms to map neurons to a Neuromorphic Computing system.

Finally, I want to express my personal thanks to my parents, family, and friends for their love and support. Your faith in me has pushed me through all the challenges that I have faced over the last couple of years.

Abstract

Investigating the inner workings of the brain runs into multiple challenges. A couple of those challenges are the level of detail which can be obtained by in vivo experiments and the time required to investigate long-term processes. A potential solution to these challenges is the use of simulators which run at an accelerated speed compared to biological real time and allow to probe physiological features that are inaccessible otherwise. Computer systems can simulate parts of the brain, but traditional computer architectures achieve neither the speed-up factor nor the scale desired. The main challenges here are the massively parallel operation of the brain, the decentralisation of memory and the high level of connectivity between neurons. The field of Neuromorphic Computing attempts to solve these challenges by developing computer systems with a fundamentally different architecture. By using the knowledge obtained by neuroscience, the architecture of the system can be based on the structures found in the biological brain to better mimic these characteristics. One of the challenges of such a system is the high throughput, low latency communication of spike events through the system.

This work focuses on these challenges and investigates the communication traffic generated on a neuromorphic system when running biologically representative large-scale spiking neural networks in order to come up with a suitable solution. The investigation of the communication traffic is done using a Python based network simulator, which is presented in this work as well. This simulator analyses the communication traffic with respect to both, the amount of communication data as well as the latency. To prove the correct functionality of the simulator, simulation data are compared against results obtained with already existing models as well as experimental data. This comparison not only proves the correct functionality of the simulator, but also shows off some of the advantages of this tool. The simulator offers a higher level of detail than existing models while also being able to handle more complex heterogeneous connectivity models. This last feature is unique for this tool and is especially important in this work as the heterogeneity is a key characteristic in biological neural networks. Simultaneously, this also allows the evaluation of neuron mapping algorithms by the simulator. To better understand the impact of different network designs, the tool is used to evaluate the performances resulting from a variation of different design aspects such as the topology, the routing algorithm, casting protocol and node size.

The goal of this study is to develop a novel communication network concept that can facilitate the communication in a large-scale neurmorphic system next to providing the tooling for its examination. To achieve this goal, the knowledge obtained during the simulation study is used to conceptualize a new stacked network topology. This network topology shows a reduction of the network load over a factor of 10 and a reduction of the latency up to a factor of 3, while hardly increasing the hardware cost of the network.

Zusammenfassung

Die Untersuchung der inneren Funktionsweise des Gehirns stößt auf zahlreiche Herausforderungen. Einige dieser Herausforderungen sind der Detailgrad, der durch In-vivo-Experimente erreicht werden kann, und der Zeitaufwand, welcher für die Untersuchung langfristiger Prozesse benötigt wird. Eine mögliche Lösung für diese Herausforderungen ist die Verwendung von Simulatoren, die im Vergleich zur biologischen Echtzeit mit einer beschleunigten Geschwindigkeit laufen und es ermöglichen, physiologische Merkmalen zu erfassen, die sonst nicht zugänglich wären. Computersysteme können Teile des Gehirns simulieren, aber herkömmliche Computerarchitekturen erreichen weder die gewünschte Geschwindigkeit noch die gewünschte Größenordnung. Die Hauptprobleme hierbei sind die überwiegend parallele Funktionsweise des Gehirns, der dezentralisierte Speicher und die hohe Konnektivität zwischen Neuronen. Der Bereich des Neuromorphic Computing versucht diese Herausforderungen durch die Entwicklung von Computersystemen mit einer grundlegend anderen Architektur zu lösen. Durch die Nutzung von Ergebnissen aus den Neurowissenschaften kann die Architektur des Systems auf den Strukturen des biologischen Gehirns aufbauen, um so diese Eigenschaften besser nachahmen zu können. Eine der Herausforderungen eines solchen Systems ist die Übermittlung von Spike-Events mit hohem Durchsatz und geringer Latenz.

Diese Arbeit konzentriert sich auf diese Herausforderungen und untersucht den Kommunikationsverkehr, welcher in einem System erzeugt wird, wenn biologisch repräsentative, impulsbasierte neuronale Netzwerke im großen Maßstab ausgeführt werden, um eine passende Lösung zu finden. Die Untersuchung des Kommunikationsverkehrs erfolgt mithilfe eines Python-basierten Netzwerksimulators, der ebenfalls als Teil dieser Arbeit vorgestellt wird. Dieser Simulator analysiert den Kommunikationsverkehr sowohl hinsichtlich der Menge an Daten als auch hinsichtlich der auftretenden Latenz. Um die korrekte Funktionsweise des Simulators nachzuweisen, werden Simulationsdaten mit Ergebnissen bereits bestehender Modelle sowie mit experimentellen Daten verglichen. Dieser Vergleich beweist nicht nur die korrekte Funktionsweise des Simulators, sondern zeigt auch einige Vorteile dieses Tools. Der Simulator bietet einen höheren Detailgrad als bestehende Modelle und ist gleichzeitig in der Lage, komplexere heterogene Verbindungsmodelle zu bewältigen. Letztgenanntes ist ein Alleinstellungsmerkmal für dieses Tool und in dieser Arbeit besonders wichtig, da die Heterogenität ein besonderes Kennzeichen biologischer neuronaler Netze ist. Gleichzeitig wird hierdurch auch die Evaluierung von Algorithmen zur geeigneten Verteilung von Neuronen durch den Simulator ermöglicht. Um die Auswirkungen verschiedener Netzwerkdesigns besser zu verstehen, wird das Tool zur Bewertung der Leistungsfähigkeiten verwendet, die

sich durch Variation verschiedener Designaspekte wie der Topologie, dem Routing-Algorithmus, dem Casting-Protokoll und der Knotengröße ergeben.

Das Ziel dieser Studie ist die Entwicklung eines neuen Konzepts für ein Kommunikationsnetzwerk, das die Kommunikation in einem großen neuromorphen System begünstigt und gleichzeitig die Werkzeuge für dessen Untersuchung bereitstellt. Um dieses Ziel zu erreichen, werden die während der Simulationsstudie gewonnenen Erkenntnisse genutzt, um eine neue geschichtete Netzwerktopologie zu konzipieren. Diese Netzwerktopologie zeigt eine Reduzierung der Netzwerklast um einen Faktor größer als 10 und eine Reduzierung der Latenz bis zu einem Faktor von 3, während die Hardwarekosten des Netzwerks kaum steigen.

Contents

List of Figures

List of Tables

Abbreviations

ACA-project	Advanced Computing Architectures project
AdExp	adaptive exponential integrate and fire
AI	Artificial Intelligence
ANN	Artificial Neural Network
BC	Broadcast
BNN	Biological Neural Network
CAM	Content-Addressable Memory
CC	Clustercast
CM	Connectivity Matrix
CPU	Central Processing Unit
.csv-file	"Comma Separated Values"-file
DE	Delay Extension
DOR	Dimension Order Routing
ESPR	Enhanced Shortest Path Routing
FR	Firing Rate
HiCANN	High-Count Analogue Neural Network
IF	Integrate & Fire
.json-file	"JavaScript Object Notation"-file
LDFR	Longest Dimension First Routing
LIF	leaky integrate-and-fire
LMC	Local Multicast
LUT	Lookup Table
MC	Multicast
NC	Neuromorphic Computing
NER	Neighbour Exploring Routing
NeuCoNS	Neuromorphic Communication Network Simulator
NN	Neural Network

NpN	Neurons per Node
RNDC NN	Randomly Connected Neural Network
SNN	Spiking Neural Network
SpN	Synapses per Neuron
TSpN	Target Synapses per Node
SPpN	Spike Packets per Node
SRC	Spike Source
UC	Unicast
VLSI	Very Large Scale Integration

Chapter 1

Introduction

1.1 Motivation

The mammalian brain is a marvellous structure able to solve complex problems, store vast amounts of data, learn, adapt, tolerate noise and offer a high level of redundancy. All this is realised in a relatively small volume while only requiring a small amount of energy. Even in ancient societies, this complex organ already fascinated scientists, with the first references to the brain dating back to ancient Egypt [1]. Ever since, through human history, mankind has tried to grasp its incredible computational power, but despite countless studies in neuroscience, much of the brains inner workings are still shrouded in a cloud of mystery. One—of many—problems that neuroscientists face when investigating the brain is the impracticality of analysing the brain while operational. While modern neuroimaging techniques are able to measure and visualize the structures and activity in the human brain in a non invasive way, the level of detail obtainable, while being impressive, is still superficial compared to the complexity of the brain structure. Attempting to investigate long-term processes in the brain also faces practicality problems. Asking a test subject to spend more than a couple of hours in a MRI-machine already becomes problematic. Attempting to investigate changes in the brain over the course of weeks or years becomes almost impossible. The only data obtainable would be a collection of snapshots of the brain made at intervals over a longer span of time. A final problem is that the study can only be performed in biological real time and will have to last for the same amount of time as the long-term process under investigation takes. An alternative to the in vivo experiments that has the potential to solve or at least support the solution of these problems is the use of simulation platforms. These platforms can simulate the behaviour of (parts of) the brain and offer a much more detailed view of the potential interaction of individual components of the brain by making use of models. These systems also offer the possibility to perform long-term experiments on a continuous basis. Ideally, these simulations can be run at an accelerated speed compared to biological real time, allowing the analysis of changes happening in the brain over the span of years in a matter of days or hours. While simulations, likely, will never completely replace in vivo experiments, they are certainly a helpful tool for neuroscientists in their

quest to unravel the complex mechanisms implemented in our brains. Unfortunately, current computer systems are not able to simulate any large sections of the brain at speeds close to real time. This is because of the significantly different structure of the brain compared to the basic structure of a traditional computer. This is where the field of Neuromorphic Computing (NC) fits in. NC is an interdisciplinary field which combines the fields of computer science and electrical engineering with neuroscience and biology. Traditionally, computer systems are based on the von-Neumann architecture, which uses a high speed, sequentially operating, central processing unit in combination with a separate centralised memory bank to store data. The brain, on the other hand, relies on a large number of slow but highly interconnected and parallel operating neurons. At the same time, the connections between neurons, called synapses, act as both computational elements and memory elements, achieving in memory computing. Guided by these principles found in the brain, novel NC systems are built to mimic their structure and behaviour better in an attempt to build computer platforms which are more efficient at simulating the brain. One of the challenges faced when designing a NC system, and the main focus of this work, is facilitating the communication of an extensive amount of neural spike events between individual computational nodes. This communication traffic is substantially different from the communication happening in most traditional computer networks. Instead of large streams of data, the spike events are communicated across the network in the form of a large amount of small information packets. Besides the sheer number of information packets passing through the system, the challenge here also lies in the latency of these packets since information is coded in their temporal relations. Technological advancements in communication components alone will not be sufficient to overcome these obstacles and the network concepts themselves will have to be reconsidered. While a couple of NC systems already exist, none of these meets the requirements set out for this work. These requirements are defined by the Advanced Computing Architectures project (ACA-project), which will be discussed in the next section.

1.2 The ACA-Project

This work is part of the Advanced Computing Architecture project [2]. This project is a collaboration between the Jülich Research Centre, the RWTH Aachen University, Heidelberg University, and the University of Manchester. This group covers a broad range of different disciplines, including, but not limited to, neuroscience, physics, and circuit design, but also includes partners that have already designed existing NC systems. Within the project, the different groups are each focused on individual challenges and tasks faced when designing such a NC system. Instead of starting with the design of the smallest computational element, the project assumes a more network-centric view by analysing the requirements needed to simulate networks with sizes of relevant brain

regions. The aim of this project is to make advances in the field of NC and develop a simulation platform capable of simulating large regions of the brain 100 times faster than biological real time with a time resolution of 0.1 ms. Combining these last two requirements translates in a wall-clock simulation step size of only 1 µs. Within this time, the simulator has to execute both the computational task, i.e. process the synaptic inputs and neuron updates, as well as the communication task, i.e. send all generated spike packets to their destinations to be processed in the next time step. At this stage of the project, the time is allocated evenly over these two aspects. This defines the maximum latency of the system and means that the communication has to be done in 500 ns.

1.3 Structure of this Thesis

The focus of this work is the inter-node communication of spike packets in a NC system. However, before the challenges and ideas regarding this task can be explained further, the basics of NC need to be understood first. In order to do so, chapter 2 will cover some basic knowledge required to understand the key parts of this work. Following this, section 3.1 will introduce a static network simulator that was implemented as part of this work to be used as an evaluation tool. A key feature of this simulator is that it can handle heterogeneous connectivity models. However, this also introduces the additional task of determining a neuron map as the specific locations of individual neurons become relevant for the communication traffic. An overview of the mapping algorithms used in this work is given in section 3.2. Continuing in chapter 4, simulation results obtained using the tool introduced before in section 3.1 are presented. To show that the tool functions as it should, this chapter starts with the verification of the network simulator in section 4.1. Noteworthy here is section 4.1.2, which is about the the verification of the simulator by comparing it to experimental traffic data measured on the SpiNNaker system. This section is essentially a segment of an article published previously as part of this work [3] and has high level of similarity with the original paper. After it has been demonstrated that the simulator works correctly, the tool is used to evaluate the different mapping algorithms as well as the different communication network architectures and protocols in sections 4.2 and 4.3, respectively. Subsequently, a novel network concept will be introduced and evaluated in chapter 5. This chapter will start by going through the thought process leading to the new network concept and ends with the evaluation of its performance. Once again, the simulator introduced in section 3.1 will be used as the main tool to perform this evaluation. Finally, in chapter 6 the work is summarised and a discussion and outlook regarding the work is given.

Chapter 2

Fundamentals

2.1 The Mammal Brain

The brain forms the centre of the nervous system in all mammals and is primarily composed of two types of cells, glia and neuron cells. While the glial cells provide structural support, supply nutrients as well as oxygen to the neurons and act as insulation between neurons, the highly interconnected neurons form the processing units of the brain. However, this processing is also greatly affected by the connections the neurons have to each other via so called synapses.

2.1.1 The Neuron

Figure 2.1 shows a schematic representation of the anatomy of a neuron. The neuron is a chemically/electrically excitable cell that can send signals to other cells over long distances. These signals are sent in the form of action potentials, also known as spikes. The process of a neuron firing a spike starts with the neuron receiving incoming signals from synapses at the dendrites. These incoming signals translate into a postsynaptic current, which is either excitatory, making the receiving neuron more likely to fire an action potential, or inhibitory, making it less likely to generate such a spike. The flow of ions that forms these postsynaptic currents accumulates in the soma of the neuron and temporarily depolarises or hyperpolarises the membrane potential. This results in either an excitatory or inhibitory postsynaptic potential over the cells membrane. Generally, when a single excitatory postsynaptic potential occurs, the depolarization will not be sufficient to pass the cells threshold and will not trigger a spike event. Instead, the cell will repolarise its membrane potential over time and return to its resting potential. When, however, multiple excitatory postsynaptic currents occur within short succession of each other, their combined effect is the sum of the individual postsynaptic currents. If the membrane potential resulting from this combined postsynaptic current surpasses the cells threshold potential, the neuron causes a quick rise and fall of the membrane

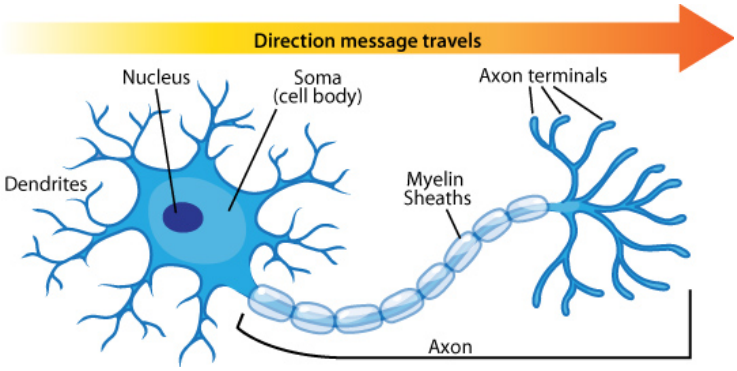


Figure 2.1: A schematic illustration of the neuron. Image by Arizona Board of Regents / ASU Ask A Biologist [4] used under CC BY-SA 3.0

potential, forming a spike. This spike is then sent along the axon to the axon terminals, which form the connections to other neurons over synapses.

2.1.2 Synapses

The synapse is a structure in the brain that forms a connection between individual neurons as described above. When a presynaptic neuron transmits an action potential, this action potential is sent along the axon to the axon terminals. At these axon terminals, the action potential will trigger the release of neurotransmitters into the synaptic cleft, which then bind to receptors on the postsynaptic cell. This, in turn, triggers a postsynaptic potential on the postsynaptic cell and the cycle is complete. The magnitude of the response triggered by a presynaptic spike can be different for each individual synapse. However, the ability of the synapse is far more complex than simply relaying incoming spikes to the postsynaptic neuron. Depending on previous spikes, the strength of the synaptic connection can change, either short term or long term, which is called plasticity. An example of short term plasticity is when multiple presynaptic spikes arrive at the presynaptic terminal in quick succession. As a result the response to the later spikes can either be enhanced or depressed relative to the initial response. When due to the quick succession of incoming spikes the available pool of neurotransmitters is temporarily depleted, the response to the later events will be depressed. If on the other hand the quick succession of incoming spikes results in a surplus of neurotransmitters in the synaptic cleft, the response will be enhanced. An example for long term plasticity is practising for a certain task. By repeating the task multiple times, the same group of neurons responsible for that task are repeatedly firing in sequence. This will trigger the synapses between these neurons to increase in strength, resulting in an enhanced path

for signals to travel over. In future scenarios it becomes easier for signals to follow this path, and thus easier to perform the task without having to think about it. These long term changes in the synaptic strength are what forms the basis for learning processes and memory in the brain.

2.2 Neural Networks

The efficient operation of the biological brain is achieved by networks of highly interconnected neurons. These so called Neural Networks (NNs) allow for all the tasks performed by the brain and have proven to be very efficient at certain tasks. Prime examples of such tasks are speech and image recognition. Recognizing written and spoken letters and words, voices and faces of people, and identifying objects are only a handful of the tasks performed by our brain continuously, on a daily basis, while hardly thinking about it. These Biological Neural Networks (BNNs) are an inspiration for Artificial Neural Networks (ANNs), which attempt to solve the same problems but use artificial neurons and synapses to do so. Because of the complexity of the biological neurons and synapses, it is difficult to fully capture their behaviour in their artificial counterparts. Instead, simplified models are used in place of them. The earliest ANNs relied on the perceptron to mimic the neuron behaviour. This perceptron uses a threshold function $f(\vec{x})$ to determine its output given an input vector $\vec{x}$. The output function can be defined as

$$f(\vec{x}) = \begin{cases} 1, & \text{if } \vec{w} \cdot \vec{x} + b > 0, \\ 0, & \text{otherwise,} \end{cases} \tag{2.1}$$

with $\vec{w}$ a real-valued weight vector and b the (negative) threshold value required to be surpassed in order to "fire". Each element in the input vector $\vec{x}$ represent an input coming from another perceptron (or external input) and is multiplied with a weight value given in the weight vector $\vec{w}$. If the accumulated value of the vector-dot product is larger than the given threshold value $-b$, the perceptron "fires" and its output is set to 1. The first generation of ANNs using this simple model already showed great potential and are able to perform tasks which are deemed difficult for traditional computers. However, in the brain, information is not encoded in the magnitude of a spike, but rather in the presence of a spike. To be more precise, information is encoded in the frequency of spikes sent by neurons as well as by the exact spike timing and the spatio-temporal relation of spikes from different neurons. To mimic this behaviour, more complex neuron and synapse models such as the leaky integrate-and-fire (LIF)-model [5], the Hodgkin–Huxley model [6], or the Izhikevich model [7] are required which also take these aspects into consideration and rely on spike inputs rather than linear inputs. Instead of a combination

of a (non-linear) threshold function with a single linear equation, these models consist of one or more differential equations preceding a non-linear threshold function to model the neuron and synapse behaviours and create a special kind of NN called a Spiking Neural Network (SNN). Capturing the complex behaviour of the neurons and synapses is important for the proper representation of BNNs, but also leads to higher computational loads. Choosing a more complex neuron and/or synapse model will inevitably result in a lower number of neurons and synapses that can be simulated on a single node. While the term NN in general refers to the overall collection of different types of NNs, it will in the rest of this work refer to either biological or artificial SNNs.

2.3 Neuromorphic Computing

Traditional computers have proven very valuable in modern society and enabled the fast growth of technological progress. However, traditional computers have shown to be unable to match performances of the biological brain for specific applications. Where traditional computers rely on the sequential processing of calculations at MHz to GHz frequencies, the brain operates fully parallel at a relative slow frequency. The high level of parallelism is mainly possible because of the very high level of connectivity between neurons, both in regard to fan-out and fan-in. A second fundamental concept in the brain is the decentralisation of memory or even combination of memory and computational units in the form of synapses, i.e. in memory-computing. In contrast to the latter, traditional computers use a centralised memory block from which data has to be read by the Central Processing Unit (CPU), resulting in a larger distance between memory and computation. In regards to the former, traditional computers have a fan-out of approx. 2-6 and a maximum of 20 in most cases, whereas the connectivity in the human brain ranges from 10^3 to 10^5 with an average of 10^4. These fundamental characteristics of the structures in the brain make it inefficient to simulate NNs on a traditional computer, as the underlying hardware is so different. Neuromorphic Computing tries to overcome this problem by implementing either digital or analogue electronics in a way which represents the structure of the biological brain better. By matching the hardware structure to the structure of the algorithms, i.e. the NNs, these systems show a massive improvement in performance for these applications. While this is the general idea of NC, the field can be further divided into two subcategories: cognitive computing and computational neuroscience. The field of cognitive computing focuses on the pattern recognition tasks described previously and Artificial Intelligence (AI), and tries to imitate the biological brain to complete these tasks. However, this imitation does not need to be an exact copy and is mainly inspired by its biological counterpart. The actual ANN used can be significantly different from the BNN, as long as the ANN gives the desired output. A suitable comparison is the design of airplane wings which are inspired by birds. The target here was to create an airplane which can fly and inspiration was taken from nature

in order to do so. However, the intention was never to achieve this target by exactly copying the behaviour of birds and use flapping wings. The sub-field of computational neuroscience, on the other hand, aims at better understanding of the biological brain by performing simulations of BNNs. These simulations offer a higher level of detail under the premises of the model than can be obtained with in vivo experiments and offer far greater control of the experimental circumstances desired. However, in order to do so the simulations should mimic the behaviour of the BNN, including individual neuron behaviour, as closely as possible. The different objectives of these two fields result in different system requirements and priorities for their respective applications. Many ANN used for classification problems and in AI have a relative low connectivity compared to BNN, simplifying the communication task. However, the applications for cognitive computing are often targeted at (portable) consumer electronics whereas a simulation platform for computational neuroscience will generally be located in research facilities. This leads to more stringent requirements in regards to the size and energy consumption of the cognitive computing targeted NC system. Another difference is the requirement for faster than real time simulation of SNNs desired in computational neuroscience. This is often not necessary in cognitive computing as the input data comes in at real time speed.

Despite the differences in requirements between computational neuroscience and cognitive computing, the basic design concept of NC systems for the different applications is still the same. Figure 2.2 shows a generalised block diagram of a NC system. Depending on their respective objectives, each system prioritizes its own requirements, generally at the cost of other specifications, but almost all of them share the following design concept; the system consists off a (large) number of nodes working fully in parallel with each other. Each of these nodes contains computational unit(s), local memory, some form of communication infrastructure and in the case of digital systems a local controller. Every

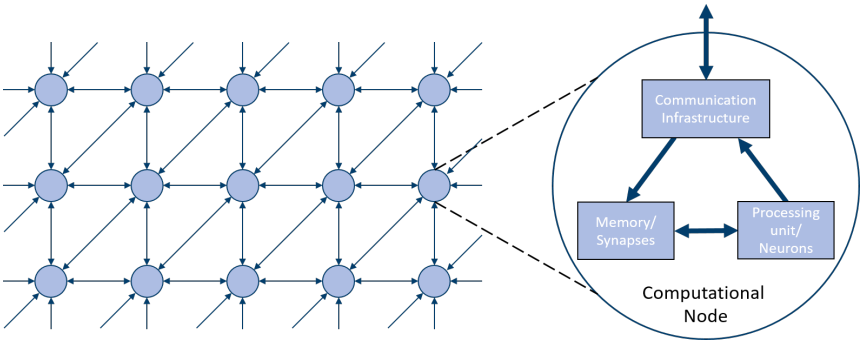


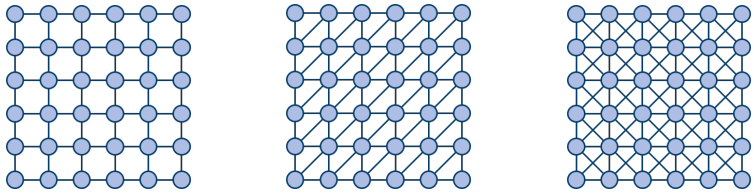
Figure 2.2: An abstract representation of a NC system. (©2021 IEEE)

node simulates a small group of neurons on its own using its own computational unit and local memory to do so. However, in order to simulate larger SNNs, the individual nodes have to communicate with each other. This communication is done by using so called spike packets, which are generated once a neuron fires. As the information encoding in SNNs is done purely in the time domain, the spike packets do not need to carry actual data and only contain the information required to send them to their destinations. However, this does require the system to send spikes to their destinations without adding to much latency, i.e. in the same biological time step in a digital system, as a deviation from doing so would change the information carried by the spike packet. These spike packets are sent to other nodes in the system via a communication network. Each node is connected to this network via the communication infrastructure, generally in the form of a router located on every node. The resulting NC system is a computer architecture that uses decentralised memory units which are located close to the related computational units. This is a much better representation of the biological brain than traditional computer architectures. The traffic load generated on such a system is a large amount of very small information packets of which the latency is critical, compared to fewer large data streams as can be found in traditional computer networks. This communication network and its respective challenges are the main focus of this work.

Different aspects have a significant effect on the performance of the communication network. In this work, mainly the aspects of the network topology, the casting protocol and the routing algorithm are investigated. However, another aspect which affects the communication network is how the routing is performed, either statically, using routing information stored locally in Lookup Tables (LUTs), carried by the spike packet, or is handled by centralised relay nodes [8], or dynamically using traffic information obtained from neighbouring routers. Meanwhile, some networks do not use any type of routing, but rather direct spike packets to their destinations using circuit switches, which are configured during the system start up. However, in this work, the analysis is limited to packet-switched-networks.

2.3.1 Communication Network Topologies

Naturally, the way how the individual nodes are interconnected has a significant impact on the performance of the network. In general, the overall communication traffic in a network reduces when the communication packets have to travel over fewer nodes. Simultaneously, this also has a positive effect on the latency of the system as each component adds to the latency. In an ideal scenario, every node is connected to every other node. In such a network each spike packet only has to pass through the source router and the target router. Simultaneously every link would only need to handle the packets that stem from one of the two connected nodes and targets the other. While being an ideal solution with respect to performance, the cost of such a network rises



(a) (b) (c)

Figure 2.3: Three traditional mesh networks, **(a)** square mesh, **(b)** triangular mesh, **(c)** king mesh.

very quickly as a network with N nodes needs $\frac{1}{2}N \cdot (N-1)$ bidirectional links and $N-1$ I/O-ports (+ one port to connect to the local core) per node. Therefore, each node in the communication network is only directly connected to a couple of other nodes. If a node needs to communicate with another node to which it is not connected via a direct link, the communication is realised over the other nodes. While any arbitrary interconnect, which allows to communicate to all nodes, could theoretically be used in a NC system, it is commonly implemented as a tree network or as a mesh with a regular pattern to facilitate easier routing. Some examples of such mesh topologies are the square mesh, the triangular mesh, and the king mesh as shown in figure 2.3. Alternatively these meshes are referred to according to the degree of each node, i.e. mesh4, mesh6 and mesh8, respectively. These meshes are created by laying out the nodes in a grid pattern and connecting them to their direct horizontal and vertical neighbours. In case of the triangular mesh, a connection is also created to a nodes direct neighbours in one of the diagonal directions, while in the king mesh nodes connect to the direct neighbours in both diagonal directions. Generally, by increasing the degree of the network, shorter routes are created between nodes. However, this is not the only way this can be achieved. In these "flat" networks, the largest distances in the network exists between nodes on the boundaries of the network. To reduce this distance, the opposing edges of the network can be connected to each other as shown in figure 2.4 to form a shortcut from one side of the network to the other. These types of mesh networks are called toroidal or torus networks. Many more standardised topologies do exist that are employed in, for example, high performance computing. However, in this work the analysis will be limited to the regular mesh networks introduced earlier as these are most common in the state-of-the-art NC systems as well as some altered variants of these networks.

2.3.2 Casting Protocols

A second major aspect of the communication network is the casting protocol. The casting protocol defines the method used to distribute a spike packet through the network and

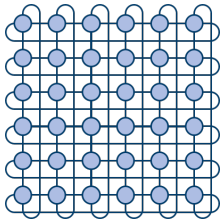


Figure 2.4: A toroidal (square) mesh network.

significantly affects the amount of communication traffic introduced onto the network. Commonly used casting protocols in NC communication networks are Broadcast (BC), Multicast (MC) and Unicast (UC).

Starting with BC, this one-to-all communication art means that any spike packet generated in the network is sent to all nodes in the network. This is done regardless of whether the spike packet is actually needed by the node, i.e. connects to any neuron on the node or not. Another key aspect of this type of casting is that this is not done by sending individual packages to each node, but rather by sending out a single spike packet which branches out. The spike packet starts from the source node and is copied by each router it passes. The router then sends the spike packet onwards into multiple directions while also sending one instance to the local core. This way the spike packet is distributed over the entire network.

MC is another form of a one-to-many communication style and is based on the same idea of branching out spikes. However, this protocol only sends the spikes to its actual target nodes instead of to all nodes. This does make this casting protocol more complex as each node needs to check whether it requires the spike packet as well as where to send the packet to next. However, it also reduces the impact a spike packet has on the communication network as it does not send spikes through the network unnecessarily.

UC, in contrary to the previous two casting protocols, does not branch out spike packets, but sends an individual spike packet to each individual target, resulting in a one-to-one style of communication. Unfortunately, the term UC is defined in two slightly different ways in literature. To differentiate between the two variants, a fourth definition is introduced in this work named Local Multicast (LMC). In this work, UC is defined as sending an individual spike packet to each individual target neuron. Meanwhile, the LMC protocol only sends out a separate packet to each individual target node. The UC protocol will generate the largest amount of communication traffic into a network, as a single spike event injects multiple spike packets in the network, followed by LMC, then BC and finally MC. However, the UC and LMC protocols enable the use of destination based routing of spike packets, which is generally simpler to implement.

2.3.3 Routing Algorithms

The final aspect of a NC communication network considered in this work is the routing algorithm used. The routing algorithm determines which route the spike packets take to reach their destination(s) and can both influence the network load as well as the network latency. While numerous different routing algorithms exist, this work limits its scope to the four routing algorithms introduced in [9].

The first of these algorithms is Dimension Order Routing (DOR). This algorithm prioritizes the movement in one direction over the other(s) and keeps moving in this direction until it cannot get any closer to the destination by doing so. At that point, the movement changes direction and the process repeats. On a regular grid network, such as the square mesh, when prioritizing horizontal movement, this algorithm translates into moving in the horizontal direction until the correct column is reached before starting moving up or down to reach the correct row. An advantage of this algorithm is that the movement in the latter direction does not have to wait on packets moving in the former direction. As a result, this routing algorithm is guaranteed to be deadlock free as no cyclic dependencies can arise. By considering the diagonal connections as their own directions, this algorithm can also be applied to the triangular mesh and the king mesh in a similar fashion.

The Longest Dimension First Routing (LDFR) algorithm is similar to the DOR algorithm and also moves through the mesh in one direction at a time. This time, though, the algorithm does not always prioritizes the same direction when determining the order of directions in which to move. Instead, the algorithm prioritizes movement in the direction the spike packet has to travel the furthest. The goal of this approach is to allow for the sharing of a longer part of the path in case MC is used. It does, however, also introduce the potential for deadlock in the system as different packets will prioritize movement in different directions and cyclic dependencies can arise.

The same goal is also pursued by the Enhanced Shortest Path Routing (ESPR), which is mainly targeted to be used in combination with MC. The algorithm searches for the shortest path between the source and destination, but does so considering the previously calculated paths stemming from the same source. In a grid network, multiple different shortest paths exist between a source and a destination. This algorithm searches for an intermediate node located on one of these possible routes which is closest to the destination. This intermediate node can either be another target node, or a node which is passed while travelling to another target node. If such a node is found, the algorithm determines the shortest route from this intermediate node to the destination using the LDFR algorithm and adds this route to the already existing route to the intermediate node. Because the algorithm only considers nodes that already lie on one of the possible shortest paths as intermediate nodes, the algorithm still returns one of the shortest possible paths.

All three algorithm DOR, LDFR and ESPR always return the shortest possible path and as a result will also yield the same latency for the system when used. This is not the case for the final algorithm called Neighbour Exploring Routing (NER). Similar to ESPR, NER searches for an intermediate node which is closest to the destination and determines the route to this intermediate node. However, this time, the intermediate node does not necessarily need to lie on one of the shortest possible paths, but can also be located further away from the source than the destination. This potentially results in an increase of the network latency when using this algorithm, but offers the potential of decreasing the communication traffic by optimizing the use of MC. A more detailed analysis of the performance of these routing algorithms will be presented later in section 4.3.3.

2.4 Existing NC Systems

The previous section already explained the abstract concept of NC systems. However, as mentioned before, the actual implementation depends greatly on the aimed application and the requirements of the specific system. In the recent decade, a number of NC systems have been developed. General overviews of these systems can be found in [10–12]. Some of the most well known examples of these systems are TrueNorth [13–15], SpiNNaker [16, 17], and BrainScaleS [18, 19]. Each of these systems has its own strengths, but neither of them meets the requirements set out by the ACA-project introduced in section 1.2. To give a better impression of how some of these NC systems work, a brief description of the three named systems is given in the following subsections. As the focus of this work lies on the communication network, this will also be emphasised in these sections.

2.4.1 TrueNorth - IBM

The TrueNorth system is a fully digital NC system developed by IBM to process large amounts of noisy multisensory data in real time, i.e. cognitive computing. The main design requirement for this system is real time processing with 1 ms time steps in combination with low power consumption (65 mW/10^6 neurons). The low power consumption is achieved by operating custom hardware in an event driven fashion. Based on the sparsity of information received per neuron, this minimizes active power as computations are only performed when new information arrives. To ensure global synchronization, the system uses a 1-kHz global synchronization signal. Generally, such clock trees create a significant contribution to the power consumption, however, at the relatively low 1-kHz frequency, this contribution remains minimal and other problems related to large clock trees, such as clock skew, can be neglected. This fully digital implementation and global system synchronization lead to fully deterministic behaviour of the system and

a one-to-one equivalence between design software and hardware. The computational units of the TrueNorth system are the neurosynaptic cores. Each of these cores simulates 256 neurons using an augmented LIF-neuron model [20]. The input to these neurons is given via a 256x256 synaptic array, which allows all-to-all connectivity between the 256 input axons of the core and all local neurons. The output of these local neurons is in turn connected to 256 output axons which are connected to either one of the local input axons or to an input axon of another neurosynaptic core. By tiling 4.096 of these neurosynaptic cores in a 64x64 grid, a TrueNorth chip is created containing one million neurons and 256 million synapses. The system can be scaled up further by connecting 16 chips in a 4x4 grid to create a so called NS16e system. This system uses the TrueNorth native inter-chip interface and acts as a single large scale system. An alternative approach, called scaling out, is the clustering of x NS1e systems–single TrueNorth chip systems with supporting hardware–together and form a NS1e-x system. This type of system is mainly suitable for problems which can be decomposed into smaller problems and solved in parallel. Currently, the biggest TrueNorth system built by IBM is a NS16e-4 system combining scaling up and scaling out to form a 64 million neuron, 16 billion synapse NC system. While the TrueNorth system is capable of implementing an impressive number of neurons with very low energy consumption, the system is not capable of implementing the number of synapses per neuron required by the ACA-project.

2.4.1.1 TrueNorth Communication Network

The communication network in TrueNorth is realised in the manner of a square grid in which the router in each node, i.e. the neurosynaptic core, is connected to its direct horizontal and vertical neighbours, also referred to as a square mesh topology. Once the internal logic of the neurosynaptic core determines that the membrane potential of a neuron surpasses a set threshold, the neuron fires. At this point, a spike packet in the format shown in table 2.1 is generated for the corresponding neuron and sent to the local router. When a router receives a spike packet, either from the local core or from an neighbouring core, it checks its Δx value. Depending on whether this Δx is positive or negative, the spike packet is sent to the right or left neighbouring router and Δx is decreased or increased by one. Once $\Delta x = 0$, the first nine bit are stripped from the spike packet and the same process is repeated for the Δy value to move the spike packet up or down until it reaches its designated target node. This vertical path is implemented completely independent of the horizontal path and ensures deadlock free communication as there is no possibility for cyclic dependencies. As the routing prioritizes movement

Table 2.1: Format of spike packets in the TrueNorth system.

Δx	Δy	Delivery tick	Destination Axon Index	Debug Bits
9 bits	9 bits	4 bits	8 bits	2 bits

in one direction over the other, this routing resembles DOR, which is implemented in a destination based approach by using the relative address to the destination for the routing. The advantage of the combination between this routing protocol and algorithm in this system is that the routers only need to perform a minimal amount of basic arithmetic operations on each incoming spike packet in order to determine which way to send a spike packet next. After a spike packet reaches its destination node, a notion of the spike event is injected in a 16 column by 256 row static random access memory array called the scheduler. Each row in the scheduler corresponds to an input axon of the local core, while each column represents a time step delay for the spike to be processed. According to the delivery tick and the destination axon index stored in the spike packed, a logical one is stored in the memory cell at the intersection of the corresponding row and column. Every time step, the scheduler reads out the next column and initiates the processing of an incoming spike event on the input axons that store a logical one in that column. Because each spike packet only targets a single axon in a single node, but each axon can target up to 256 different neurons, this type of communication can be classified as LMC. This also forms one of the limitations of this system as each neuron can only target up to 256 synapses, and this requires the 256 synapses to be located on the same node. This limitation can be circumvented to some degree by using repeater neurons as branch points for spike packets or by implementing multiple instances of the same neuron on the same node but targeting different nodes. However, this comes at the cost of allocating neuron hardware to increase the connectivity.

2.4.2 SpiNNaker

The Spiking Neural Network Architecture system, better known as SpiNNaker, is developed by the Advanced Processor Technologies Research Group at the University of Manchester and is part of the Human Brain Project. Inspired by the connectivity characteristics of the mammal brain, the project aims to deliver a large scale neuromorphic system suited to model large-scale SNNs in biological real time. The main goal is to better understand information representation and processing in the brain, i.e. computational neuroscience. The system is fully digital and uses a large number of processors to achieve the level of parallelism desired. The basic building block of the SpiNNaker system is the SpiNnaker chip, which is a multi-processor chip that incorporates 18 ARM cores on a single die. Each processor operates as a conventional CPU and simulates a small NN or population of neurons as part of a larger network. Everything ranging from calculating the neuron cell membrane potentials to connectivity, plasticity and axon delays is done in software. This software level approach offers great flexibility; all neuron and synapse model equations as well as parameters can be changed by changing the software, a major advantage of the SpiNNaker system. In principle, it allows a very high degree of complexity of the models. However, taking into regard other requirements, such as simulation time steps, acceleration factor vs. biology, and the number of neurons/synapses simulated

per core, the complexity will be limited. This makes it hard to give exact numbers on the cores performance as this will greatly depend on the implemented neuron model and other simulation parameters. A higher time resolution will come at the cost of fewer neurons or synapses, a simpler neuron or synapse model, a slower simulation speed, or a combination of the five. This trade-off results in a system which can not fulfil all the the requirements set out by the ACA-project simultaneously. The current design can, in theory, be scaled up to facilitate simulations of neural networks of up to a billion neurons (10^9) and a trillion synapses (10^{12}), corresponding to approximately 1% of the size of the human brain with respects to the number of neurons. The power consumption of the system greatly depends on the NN simulated and network activity [21], but peaks at 1 W per chip. However, the full scale system requires additional energy in order to facilitate communication between the higher levels. In SpiNNaker publications, the energy consumption for the full scale system is estimated at 75 kW [16] and 90 kW [17].

2.4.2.1 SpiNNaker Communication Network

The SpiNNaker system is scaled up by connecting individual chips together in a triangular mesh. The communication between the individual chips is facilitated by the router located on every chip. Information is transmitted over the network using four different types of packets: Multicast, Point-to-Point, Nearest Neighbour, and Fixed Route. Each of these packet types has its own purpose and way it is routed. What is relevant in this work is the Multicast packet type. This packet type is used for the communication of neural spike events and is formatted as shown in table 2.2. The first eight bits of the packet form the header–starting with two bits that indicate the packet type, 00 in case of MC–which tell the routers how to handle the spike packet. Besides the regular routing of packets, the SpiNNaker communication network also allows for packets to take an emergency route to a neighbouring node in case the intended link is blocked or defective. After trying for a certain time, measured using the time stamp bits, the emergency routing protocol is started and the corresponding bits set accordingly. Eventually, if this also fails, the spike packet is dropped to prevent deadlocks forming in the system. However, as spikes are dropped, the output of the NN might not be correct any more. The last two header bits are used to indicate the presence of an optional payload in the form of 32 additional bits at the end of the spike packet, which is generally not used for the MC packet type, and a parity bit for error detection. The other 32 bits are used as the routing keys for the routers. As the SpiNNaker system uses source-based routing, this routing key explicitly contains a source ID. To route the packets, the routers check a Content-Addressable

Table 2.2: Format of the multicast spike packet in the SpiNNaker system.

Packet Type	Emergency Routing	Time Stamp	Payload	Parity	Routing Key
2 bits	2 bits	2 bits	1 bit	1 bit	32 bit

Memory (CAM), also known as LUT, for entries matching the routing key of incoming packets. If the routing key is found in the CAM, the corresponding memory row is returned which consists of 24 bits. Each of these 24 bits represent an output of the router, 6 external outputs and 18 to the local cores, and indicate to which of these outputs the spike packet needs to be forwarded. When multiple of these bits are set simultaneously, the packet is sent out of all the corresponding outputs, resulting in the desired MC behaviour. Otherwise, when no matches are found in the LUT, a default routing is applied and the spike packet is forwarded to the output port opposite of the input it came from. This default routing, together with the use of masked "do not care" bits in the LUT, reduces the number of entries required in the LUT and reduces the size of the required CAM. While the MC protocol allows any arbitrary number of outgoing connections per individual neuron, the system is limited by the number of input synapses per neuron. This is due to the compute budget generally being dominated by the number of incoming signals. In principle, one processor core can support up to 10 million connections/s, but this number is highly dependent on the synaptic model used. Especially the use of plastic synapses reduces this number significantly.

2.4.3 BrainScaleS

Another existing NC system, which is part of the Human Brain Project, is the mixed-analogue-digital BrainScaleS system. This system has been developed by a collaboration of research groups lead by the Heidelberg University over a series of projects including the FACETS project and the BranScaleS project. One of the key design concepts of this system is the use of integrated analogue circuitry to model adaptive exponential integrate and fire (AdExp) [18] neurons with conductance based synapses [22]. These analogue circuits operate at a time scale up to 10.000 times faster than real-time. This high speed-up factor makes the system very well-suited for long-term learning experiments, allowing the simulation of year long processes in a matter of hours. Other applications include, but are not limited to, large-scale parameter searches. Besides the high speed-up factor, the analogue implementation also has the benefit that only a few transistors are required, significantly reducing the energy consumption per synaptic event by six orders of magnitude compared to conventional computers. The analogue neurons, and their corresponding synapses, are implemented in a so called High-Count Analogue Neural Network (HiCANN) die, each implementing up to 512 AdExp neurons and 131.072 synapses. These synapses are laid out in two 256x256 synapse arrays with the neurons located in-between. Each of the neurons in this neuron block alternatively connects to one column in either the upper or lower synaptic array, resulting in 256 input synapses per neuron. However, by short-circuiting the membrane potential circuits of adjacent neurons, the maximum number of inputs per neuron can be increased at the cost of a reduced number of effective neurons. The second key design concept employed by BrainScaleS is the wafer-scale integration of the HiCANN dies. This wafer-scale integration

allows an efficient interconnect that accommodates the high speed-up factor, but does introduce some additional challenges. The production of microchips on silicon wafers happens in a step-and-repeat fashion in which parts of the wafer are processed in turns to produce multiple copies of the same chip on a wafer. Using this common manufacturing process, these individual copies, called reticles, cannot be connected to each other and operate as isolated units. To realize the wafer-scale integration, the BrainScaleS wafer undergoes a post-processing during which an additional metal layer is added to create global connectivity across the wafer. With 48 reticles per wafer, and each reticle holding eight HiCANN dies, one full BrainScaleS wafer can simulate up to 196.608 neurons and $5 \cdot 10^7$ synapses. The largest reported BrainScaleS system is a 20-wafer platform built within the Human Brain Project, incorporating up to a maximum of $3.9 \cdot 10^6$ neurons and 10^9 synapses. Unfortunately, even at this scale, the system would only be able to facilitate roughly one twentieth of the number of neurons in the multi-area model, which is used in the ACA-project and will be introduced in section 2.5.2.

2.4.3.1 BrainScaleS Communication Network

The global interconnect formed during the post-processing of the wafer connects all HiCANN dies on the wafer in a square grid. However, this interconnect does not act as a regular mesh network that routes spike packets. Instead, the BrainScaleS system uses switches to create fixed connections between HiCANN dies over a so called "Layer-1" communication system. This "Layer-1" consists of 64 horizontal buses running through the middle of each HiCANN die, while 256 vertical buses are split and run along both sides of the die. Once a neuron fires, an asynchronous serial 6-bit packet is injected on one of the horizontal buses. This 6-bit packet encodes the address of the source neuron on that given bus lane, while arbitration by a priority encoder enables spike packets of up to 64 neurons to be carried on each bus lane. Located at the cross-points of these horizontal and vertical buses are cross bar switches. During the systems start up, these switches are configured and define which buses are connected. So called synaptic switches, located on the sides of each synapse array on the vertical buses are then used to configure connections between the vertical lanes and the synaptic drivers, driving two adjacent rows of synapses. Finally, an address decoder located in each synapse then filters out the spike packet of the corresponding presynaptic neuron to be injected in the synapse. In order to reduce the required area, the cross bars are sparse switch matrices and not every 64x256 junction is switchable. While this does reduce the area of the cross bar, it also limits the system when assigning neurons and synapses and increases the routing complexity. Another challenge is the implementation of the "layer-2" communication system. In order to extend the system beyond one wafer, several wafers can be interconnected via a "Layer-2" communication network. While this enables the simulation of larger SNNs, it significantly reduces the speed-up factor of the system.

2.5 Test cases

While the ACA-project defines the requirements of the overall system, it does not translate these requirements in hardware specifications for the communication network. These specifications not only depend on the design of the communication network, but is also highly dependent on the NN running on the NC system. As of such, the network load on the communication network can only be evaluated correctly by taking a representative NN into account. In the field of cognitive computing, a common benchmark for NNs is the MNIST database and in turn, these NNs are often used as test case for NC systems designed for this application. However, the target of the ACA-project is to develop a NC system for the field of computational neuroscience. This field focuses on biologically representative NNs, which are generally recurrent NNs and have a higher in- and out-degree than those found in cognitive computing. A basic example of such a network is a Randomly Connected Neural Network (RNDC NN). The connectivity in this type of network is defined by a single probability ϵ which indicates the probability that a connection exists between two arbitrary neurons. As this ϵ is equal for all neurons in the network, the connectivity of each also statistically equal and the network is referred to as being homogeneous. More complex homogeneous NNs do exist, however, as long as the connectivity is set the same for all neurons, the NN remains homogeneous. Another key aspect of BNNs, equally important, is that they show a high degree of heterogeneity. The connectivity in these NNs differs greatly between different groups of neurons, both in regards to the number of connections formed as well as in regards to potential targets. Within the ACA-project a couple of test cases are identified. Two of these test cases with such characteristics, which were of special concern within this work, are the cortical microcircuit model and the multi-area model. A basic description of these test cases is given in the following subsections.

2.5.1 Cortical Microcircuit Model

The large scale cortical microcircuit model by Potjans and Diesmann [23] describes the connectivity of approximately 78.000 neurons found under one square mm of the cortical surface. The model consists of the four layers 2/3, 4, 5, and 6, with two populations each, an excitatory and an inhibitory population. Each of the populations has its own specific size, i.e. number of neurons, and uses the LIF neuron model in combination with exponential-shaped postsynaptic currents to model the neuron behaviour. Along with the population size, the model also defines connection probabilities that determine how likely a connection between two independently chosen pre- and postsynaptic neurons is. These connection probabilities depend on the population of both the pre- and postsynaptic neuron, resulting in a heterogeneous network structure. If a connection exists, the synaptic strengths, also known as synaptic weight, and the synaptic delay of

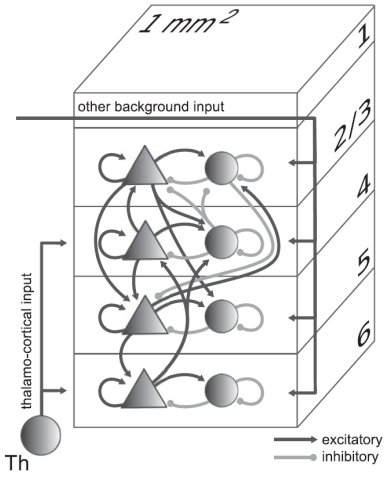


Figure 2.5: Model definition of the cortical microcircuit by Potjans and Diesmann. Only excitatory (black) and inhibitory (grey) connections with connection probabilities > 0.04 are shown. Image by T. Potjans and M. Diesmann [23] - Used under the CC BY-NC 3.0.

the connection are drawn from a Gaussian distribution. Besides the two populations for each of the four layers listed before, the model also contains a "thalamic" population, which models one of a total of three types of inputs to the local cortical network. The other two types of inputs are the "grey matter" and "white matter" external inputs. This model shows asynchronous and irregular activity in combination with layer-specific firing rates, which are similar to the activity observed in biology. The overall structure of the cortical microcircuit is shown in figure 2.5. Layers 2/3, 4, 5, and 6 are each represented by an excitatory (triangles) and an inhibitory (circles) population of neurons.

2.5.2 Multi-Area Model

While the cortical microcircuit is considered to be a large scale test case in some studies within the field of theoretical neuroscience, it is still very small in relation to the size of mammal brains. To put the scale in perspective, a mouse brain is estimated to consist of roughly 70 million neurons [24], while the human cerebral cortex alone already has between 14 and 16 billion neurons [25]. Combining the extensively characterised network structure at the local circuits level and knowledge about long-range connectivity leads to the multi-scale framework of the structure of one hemisphere of the macaque vision-related cortex [26], henceforth referred to as the multi-area model. This model

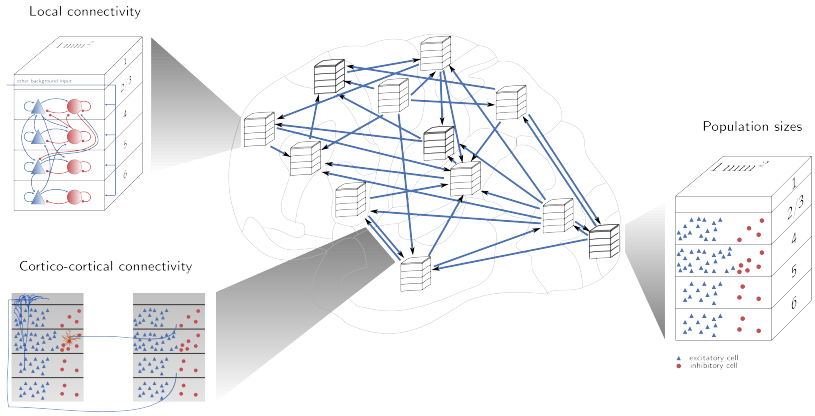


Figure 2.6: Overview of the multi-area model. Image by M. Schmidt et al. [26] - Used under the CC BY-4.0.

consists of 32 areas each representing the volume under one square mm of the cortical surface of that area. The description of each area is based on the cortical microcircuit model and has the same layered structure of excitatory and inhibitory populations[1] but with area- and layer-specific population sizes and connection probabilities. An abstract overview of this model is shown in figure 2.6. The synapses in the model can be divided in four different types: type *I* connections originating from other neurons within the microcircuit under the same 1 mm^2 of cortical surface, type *II* connections originating from the same area but outside of the 1 mm^2 region, type *III* connections originating from other areas included in the model, and type *IV* connections originating from areas not included in the model. The intra-area connectivity (types *I* and *II*) forms roughly 79% of all synapses considered in the model with the other 21% being formed by the type *III* and *IV* connections. However, only the neurons creating the type *I* and *III* connections are actually included in the model. The other connections are treated as external inputs. In total, the model includes 4.1 million neurons with 4.000 to 14.000 synapses per neuron, depending on the area. As the type *II* and type *IV* connections are treated as external inputs, these will not be considered in this work.

2.5.3 External Inputs

Both of the previously described test cases also include external inputs in the model, which stem from neurons outside the range of the model description. Generally, these inputs are generated according to a statistical distribution. Instead of generating these

[1] One exception is the "TH" area, which does not include the layer 4 populations.

inputs off-system and sending them to the cores, a more efficient approach is to generate these input spike trains on the system itself. Unless stated otherwise, in this work it is assumed that these input spike trains are generated on the local core, in which they are needed, and do not need to travel over the communication network.

2.6 Comparable Work

Naturally, as the communication network forms a critical part of any NC system, this problem has been analysed by others as well. Two of such studies within this field that show a large overlap with the work presented here, are [27] conducted by Technion - Israel Institute of Technology and [28, 29] conducted by IDS - RWTH Aachen University. The latter also participated in the ACA-project. The former group evaluates communication network topologies and protocols using an analytical model in their work. The latter group follows the same aim, but they apply a numerical approach for this purpose. Both research groups also propose a novel (hierarchical) network topology.

2.6.1 The Analytical Model by Vainbrand et al.

The study presented in [27] analytically evaluates and compares different network configurations for different types of neural networks. They define the bandwidth of a neuromorphic communication network as

$$\mathrm{BW} = \frac{\overline{w} \cdot \mathrm{TL}_{\mathrm{arch}}}{\mathrm{TD}} \cdot f_{\mathrm{arch}}, \tag{2.2}$$

where $\overline{w}$ is the width of the links, $\mathrm{TL}_{\mathrm{arch}}$ is the total number of links in the architecture, D is the average number of links traversed by each neural spike going to all its destinations, and f_{arch} is the switching frequency of the hardware components in the architecture. If $\overline{w}$ is equal to the number of bits per spike packet, i.e. each link can handle one spike packet per cycle, the equation becomes easier to comprehend. By dividing the number of links in the network by the average number of links occupied by one spike packet, the number of spike packets that can travel through the network simultaneously is calculated. By then multiplying this number with the frequency at which each link operates, the accumulated number of spike packets per seconds that can be handled by the system can be determined resulting in the maximum bandwidth of the architecture. In the original work, this maximum bandwidth is scaled with an empirically determined, architecture-specific utilization factor to account for a reduced f_{arch} because of router delays. In the present work, this scale factor will be neglected in the following. Instead of using the overall networks bandwidth as the performance metric, in this work, the networks are

evaluated by determining the required throughput per hardware element. The required throughput per link–and with it the throughput per I/O-port of the connected node–can be defined as $\overline{w} \times f$. In this case, equation (2.2) can be rewritten as

$$\text{Throughput} = \frac{\text{BW}_{\text{req}} \cdot \text{D}}{\text{TL}_{\text{arch}}}. \tag{2.3}$$

Assuming that the spikes are injected evenly throughout the network, the required bandwidth of the network can also be defined as

$$\text{BW}_{\text{req}} = f_{N,\text{out}} \cdot N, \tag{2.4}$$

with N as the number of nodes in the network and $f_{N,\text{out}}$ as the rate at which each node injects spike packets in the network. As the number of spikes injected per node can also be expressed by the number of Neurons per Node (NpN) multiplied with the average firing rate of each neuron FR_{avg}, this can be converted in

$$\text{BW}_{\text{req}} = \text{NpN} \cdot \text{FR}_{\text{avg}} \cdot N = n \cdot \text{FR}_{\text{avg}}, \tag{2.5}$$

with the total number of neurons in the network

$$n = \text{NpN} \cdot N. \tag{2.6}$$

Finally, the number of links occupied by each neural spike can also be expressed as

$$\text{D} = N_{\text{targets}} \cdot d_{\text{avg}}, \tag{2.7}$$

with N_{targets} as the average number of targets per neuron and d_{avg} as the average distance (in links) to each individual target. Using these expressions in equation (2.3), the throughput required per link can be expressed as

$$\text{Throughput} = \frac{n \cdot N_{\text{targets}} \cdot d_{\text{avg}}}{\text{TL}_{\text{arch}}} \cdot \text{FR}_{\text{avg}}. \tag{2.8}$$

Because these equations assume a uniform distribution of spikes throughout the network, it should be noted that the final equation only returns the average throughput required, as will be discussed in more detail in section 4.1.1. While this approach gives a good initial approximation of the network load, it is highly generalised. Even though the parameters and variables can be set freely, it can be difficult to determine the exact values for each situation. In specific scenarios, the values can be calculated or approximated

relatively well, but these scenarios are all limited to regular communication networks with homogeneous neural connectivity models.

2.6.2 Numerical Model by Kauth et al.

The research performed by the group of IDS - RWTH Aachen takes a similar approach as presented in this work. They describe two tools, a static simulator and a dynamic simulator that can be used to evaluate system architectures, each with its their own strengths. They then use these tools to evaluate communication networks and develop a new concept.

2.6.2.1 Static Simulator

Starting with the static simulator, this tool is mainly intended for the fast exploration of the design space and relies on quick execution of large network evaluations. It operates on the assumption of homogeneous connectivity in the NN and uses one arbitrary node as the starting point for the network load calculation. From this node, it determines the routes the spike packets take to randomly picked target and determines the average bandwidth and latency requirements of the system. Target neurons, thereby, may be determined according to a uniform, a distance based, or a conditionally defined connection probability. While being more versatile and allowing more complex scenarios than the analytical model presented before, this tool is still abstract as it does not consider any queuing of packets, an unbalanced distribution of spike generation or other dynamic behaviours and is limited to homogeneous connectivity models.

2.6.2.2 Dynamic Simulator

The dynamic simulator, on the other hand, does not solely focus on the communication but also considers other bottlenecks of the system such as memory and computational requirements as well. This is done by emulating hardware components and modelling their dynamic behaviour. Actual computations of the neuron behaviour are still omitted and instead the computation latency is determined as function of the neuron model complexity. The individual nodes only generate and absorb incoming spikes according to predefined statistical distributions.

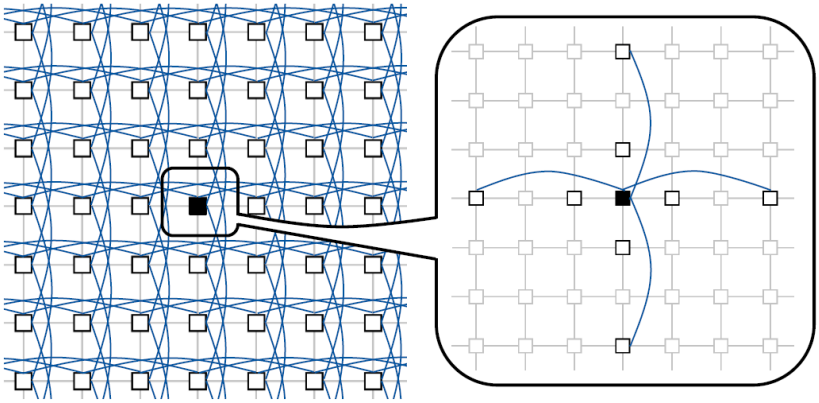


Figure 2.7: Multi-mesh 4-[1, 3] topology as a special instance of the class of multi-mesh topologies. Image by K. Kauth et al. [28].

.

2.6.3 Multi-Mesh Topology Proposed by Kauth et al

In one of their publications [28], this group also introduces a novel communication network topology accompanied by its own casting protocol. The main goal of the proposed casting scheme is to reduce offline route calculations and to eliminate the need for large routing tables. However, it increases the network load and may have an adverse effect on the latency—in regards to the distance the spike packets have to travel. As the reduction of the network load is one of the main focal points in this work, this casting scheme will not be elaborated further. What will be described in more detail is the proposed "multi-mesh" interconnect topology. This network topology starts with a traditional toroidal mesh network and superimposes a mesh with long-distance connections on top of the original mesh. This principle is not limited to just one superimposed mesh and can be realised in the numerous forms of differently sized mesh networks with different degrees per node. To distinguish between the different variations, these multi-mesh topologies are identified as "multi-mesh 4-$[i_0, ..., i_x]$ + 6-$[j_0, ..., j_y]$ + 8-$[k_0, ..., k_z]$". In this form, each list entry i, j and k represents a mesh of the respective length with the given node degree. An example of such a multi-mesh superimposing a length-3 square mesh on top of a traditional toroidal square mesh resulting in a multi-mesh 4-[1, 3] is shown in figure 2.7. The upper level creates shortcuts to nodes further away allowing for low latency communication over longer distances at the cost of additional outgoing links per node. The performance of this network topology will also briefly be evaluated in section 4.3.2.

Chapter 3

Simulation Tool and Neuron Mapping

The development of a communication network that fulfils all requirements outlined for a large-scale NC system by the ACA-project is a complex task. The topology of the communication network, the routing and casting protocols, as well as the number of neurons simulated by a node, all affect the resulting communication load on the network and its latency. In order to find the best solution in the multi-parameter design space, the performance of different communication network topologies and communication protocols has to be analysed. As major support to achieve this goal, the Python based network simulator named Neuromorphic Communication Network Simulator (NeuCoNS) [3, 30] has been developed, which is presented in section 3.1. The choice for this programming language was made because of the versatility and ease of use as well as compatibility with partner institutes within the ACA-project. The unique feature of this tool is its ability to analyse heterogeneous connectivity models in which the connectivity per neuron varies in both the number of connections as well as the destinations. This is a key feature, as heterogeneous NNs are a better representation of the connectivity found in biology. As the connectivity of individual neurons in such a scenario is different, the allocation of neurons to nodes in the system also becomes a determining factor for the network load and latency of the system. This allocation is performed using neuron mapping algorithms. A couple of such mapping algorithms are implemented in NeuCoNS as well and will be further elaborated in section 3.2. Because NeuCoNS is able to process this neuron mapping, it can also be used to evaluate different mapping algorithms. Later in chapter 4, the results obtained using NeuCoNS for different analyses will be presented. The results in section 4.1 will also show a third advantage of NeuCoNS over existing network simulators in terms of an increased level of detail of the estimated network load obtainable.

3.1 Python based Neuromorphic Communication Network Simulator

NeuCoNS is designed to analyse the performance of a communication infrastructure of a NC system at a high level of abstraction and works on a static basis, rather than to simulate the dynamic behaviour of the NN or individual neurons. This performance is assessed based on two aspects, the load on the network, e.g. the number of spike packets travelling through different parts of the network, and the network latency. In a NC system, spike packets are generated by neurons firing and have to be transmitted to all target neurons. As mentioned before, NeuCoNS is not designed for the analysis of actual neuron behaviour and as a consequence, it cannot determine which neuron is firing and sending out a spike. To significantly simplify the simulation, the communication traffic generated by a spike event is calculated for each individual neuron. The overall network load is then determined by accumulating these individual traffic profiles together, potentially with a weight factor to consider differences in activity rates of neurons or neuron types. This design choice was made as it would require an actual NC system to run a simulation considering the spike timings at a large scale. The final output of NeuCoNS is an estimation of the number of spike packets passing through each individual router and link as well as an indication of the latency. Again, because of the high level of abstraction of NeuCoNS, the actual spike timings are not considered and effects such as buffering and queuing in the routers cannot be considered for the determination of the systems latency. Instead, the latency is calculated as the number of routers and (weighted) links a spike packet has to transverse to reach its destination.

During a simulation run NeuCoNS performs the following steps:

- Setting all simulation parameters
- Generation of a NN to act as test case
- Generation of the NC communication network represented by a mathematical graph
- Mapping of the neurons to the graph
- Simulation of a spike event for each neuron
- Weighted accumulation of the communication traffic loads created by the individual neurons.

A more detailed description of how NeuCoNS works and how the network load and latency is calculated, as well as how the output data should be interpreted, will be given in the subsequent subsections.

3.1.1 Simulator Top Level

The execution of the previously listed steps is controlled by NeuCoNSs top level functions. Depending on how the simulation is configured, some of these steps are repeated with different settings to perform multiple individual runs sequentially. NeuCoNS can be started either as a stand alone Python script, interactively via the python console, or within other pieces of Python code. In the first case, the settings have to be set in a .config file, which is read out by NeuCoNS and sets it up accordingly. In the other scenarios, the settings have to be given to the functions directly. This offers a higher degree of flexibility but can be prone to errors if parameters are not set (accordingly). During this step, the topology of the communication network is set, the casting method, routing algorithm and mapping algorithm are chosen, and all other respective parameters are defined. Appendix A shows an example of a .config file used to perform simulations using NeuCoNS; also included in this appendix are lists of parameter settings implemented in the current version of NeuCoNS. The source code along with a more detailed explanation on how to set up and use NeuCoNS can be found in [31].

3.1.2 Connectivity Definition of the Neural Network Test Case

After a generic configuration of NeuCoNS, the following step is to define a NN to act as test case. This test case defines the connectivity of the neurons in the NN and determines which neurons need to communicate with each other. To accurately evaluate the communication traffic on the hardware network, this test case should represent the future use cases accordingly. Because neuron behaviour is not considered by NeuCoNS, the strength of these connections have no direct influence on NeuCoNS and only the presence of the connections has to be defined. NeuCoNS can handle the definition of connectivity in two different ways. A third approach, which was based on connection probabilities depending on the location of the neurons in the network, was implemented as well in an earlier version. However, as this approach did not represent any real test case and prevented the use of certain optimisations, it was dropped from the later versions. A description of this third approach can still be found in [30]. The two remaining connectivity definitions are explained next.

3.1.2.1 Connectivity Netlist

The first approach uses a netlist to exactly define the Firing Rate (FR) of each individual neuron as well as a list of target neurons which need to receive a spike packet in case the source neuron fires. This netlist has to be provided to NeuCoNS in the form of a "JavaScript Object Notation"-file (.json-file). An example of how such an netlist file looks like can be seen in figure 3.1. The ACA-project also sustains its own collection of

"neuron$_0$": {"FR": 1.1, "connected_to": [neuron$_1$, neuron$_4$, neuron$_{12}$, ...]},
"neuron$_1$": {"FR": 1.8, "connected_to": [neuron$_3$, neuron$_5$, neuron$_8$, ...]},
"neuron$_2$": {"FR": 0.3, "connected_to": [neuron$_1$, neuron$_2$, neuron$_{24}$, ...]},
...
...
"neuron$_n$": {"FR": fr_n, "connected_to": [...]}

Figure 3.1: An arbitrary example of a json type netlist file for the use in NeuCoNS.

test cases as described earlier in section 2.5. Consequently, functions to automatically generate netlist files for most of these test cases are implemented in NeuCoNS as well. Ideally, experimentally obtained FR distributions should be used here to create a truly heterogeneous NN, but for the sake of simplicity and to focus the analysis on the influence of the hardware choices rather than the particular characteristics of a given network model, the choice was made to set the FRs to "1" by default. What this means for the interpretation of the output data will be discussed later in section 3.1.6. Unfortunately, because of the large number of neurons and high level of connectivity typically found in NNs, the size of the netlist file can grow proportional to n^2 and quickly becomes impracticable, e.g. approximately 3.7Gb for the cortical microcircuit model with $7.8 \cdot 10^4$ neurons and $2.92 \cdot 10^5$ synapses. A way around this limitation is presented next.

3.1.2.2 Connectivity Matrix

To investigate the communication network for larger scale NNs, an alternative approach was implemented in the later versions of NeuCoNS which relies on a connectivity matrix. The Connectivity Matrix (CM) is stored in a "Comma Separated Values"-file (.csv-file) which is read out at the start of the simulation. In this case, the exact connections between individual neurons are not stored and are not even determined in some cases. Instead, whether a spike packet needs to be sent to a neuron or node is determined statistically during the simulation itself. Each row in the connectivity matrix file represents a population in the NN where the first three columns contain the population name X, the population size N_X in number of neurons, and the populations average firing rate FR_X, set to 1 here by default. All subsequent columns form the actual connectivity matrix and define the probabilities $C_{X,Y}$ that a neuron from population X connects to an individual neuron from another population Y. As an example, the transposed connectivity matrix of the cortical microcircuit is shown in table 3.1. A major advantage of this approach is the reduction of the memory requirement, e.g. the matrix file for the multi-area model is only 247KB with $4.1 \cdot 10^6$ neurons and $4 \cdot 10^9$ synapses.

Table 3.1: The transposed CM for the cortical microcircuit model with firing rates set to 1.

X	L2/3E	L2/3I	L4E	L4I	L5E	L5I	L6E	L6I	TC
N_X	20683	5834	21915	5479	4850	1065	14395	2948	902
FR_X	1	1	1	1	1	1	1	1	1
$C_{X,L2/3E}$	0.1009	0.1689	0.0437	0.0818	0.0323	0	0.0076	0	0
$C_{X,L2/3I}$	0.1346	0.1371	0.0316	0.0515	0.0755	0	0.0042	0	0
$C_{X,L4E}$	0.0077	0.0059	0.0497	0.135	0.0067	0.0003	0.0453	0	0.0983
$C_{X,L4I}$	0.0691	0.0029	0.0794	0.1597	0.0033	0	0.1057	0	0.0619
$C_{X,L5E}$	0.1004	0.0622	0.0505	0.0057	0.0831	0.3726	0.0204	0	0
$C_{X,L5I}$	0.0548	0.0269	0.0257	0.0022	0.06	0.3158	0.0086	0	0
$C_{X,L6E}$	0.0156	0.0066	0.0211	0.0166	0.0572	0.0197	0.0396	0.2252	0.0512
$C_{X,L6I}$	0.0364	0.0011	0.0034	0.0005	0.0277	0.008	0.0658	0.1443	0.0196
$C_{X,TC}$	0	0	0	0	0	0	0	0	0

Furthermore, there are speed advantages to this approach as well. In the netlist based simulation, the target list has to be read out individually for each neuron. This list of target neurons then needs to be converted to a list of nodes on which these neurons are located. Considering the length of the target list and that this has to be done for each individual neuron, this requires a large number of memory accesses, which are relatively slow. With the connectivity matrix based approach this is not necessary. Instead, NeuCoNS loops through the entire network and determines whether a neuron connects to another neuron based on the probabilities. When doing this for each individual neuron pair, there will not be a significant speed gain, if any at all. However, individual neuron pairs only have to be considered when simulating full UC. Other casting methods in which the target nodes forward a single incoming spike packet to all the local target neurons do not require the actual connectivity. For these casting types it only needs to be determined whether any neuron in the node needs to receive the spike packet, i.e. Target Synapses per Node (TSpN) $\geqslant 1$. The probability to connect to at least one neuron in the target node N_i can be calculated as:

$$p(\text{TSpN} \geqslant 1) = 1 - p(\text{TSpN} = 0) = 1 - \prod_{\forall\, n \in N_i} (1 - C_{X_{\text{src}},Y_n}) \tag{3.1}$$

with Y_n the population type of neuron n. Because C_{X,Y_n} is the same for all potential target neurons of the same population, equation (3.1) can be rewritten to:

$$p(\text{TSpN} \geqslant 1) = 1 - \prod_{\forall\, Y_i \in N_i} (1 - C_{X_{\text{src}},Y_i})^{n_{Y_i}} \tag{3.2}$$

with n_{Y_i} the number of neurons of population Y_i located on node N_i. However, most nodes will only contain one type of neurons. Why this would be most likely the case will be discussed more elaborate in sections 3.1.4 and 3.2. In the SpiNNaker system, it is even a design limitation that each core can only simulate one type of neuron, i.e. neurons from the same population. As a result, the probability that a node requires a certain spike packet can be further simplified and calculated as:

$$p(\mathrm{TSpN} \geqslant 1) = 1 - (1 - C_{X_{\mathrm{src}}, Y_i})^{n_{Y_i}} \,. \tag{3.3}$$

Because the calculation only depends on the source neurons type and not the actual neuron itself, this calculation only has to be performed once for every node, for every population and can be reused throughout. The final connectivity of a neuron is then determined by random sampling the nodes with the calculated probabilities. This optimization of combining the target calculation greatly reduces the simulation time.

A draw-back of this approach is that the exact connectivity is not stored anywhere. Within the same simulation platform, the exact same connectivity can often still be recreated by setting a specific seed, but it cannot be recreated by other analysis tools. However, the definition of a NN in the form of population sizes and probabilities of connections between those populations—also known as the projections of one population to another—is a commonly used approach in the field of computational neuroscience and can be found in other NC tools such as NEST as well as the PyNN Python packet. Generally, the connectivity between neurons in biology cannot be determined exactly either for larger NNs due to practical reasons. Instead the connection probabilities between different groups of neurons are measured, making this a valid approach for NeuCoNS as well.

3.1.3 Communication Network Hardware Graph

The next step in the simulation is the creation of a directed graph, which represents the physical communication network. Each vertex in the graph, from here on referred to as node, represents a computational node in the communication network. The directed edges in turn represent the communication links between two nodes and have a weight assigned to them. This weight can be used for the latency estimation to weigh delay contributions of certain longer or slower connections accordingly. However, by default the weight is set to 1. To implement bidirectional links in NeuCoNS, two parallel unidirectional links in opposite directions can be used. The choice to use unidirectional links was made as it is more representative of the actual hardware implementation of bidirectional links where input and output ports are separated as well. By modelling it this way, the possibility that a high traffic load in one direction is cancelled out by low traffic in the other, is eliminated and a higher level of insight can be achieved. If however, for any reason, the

communication link is indeed realised as a shared channel for both directions, the overall number of packets passing over the link can simply be calculated as the accumulation of the two unidirectional links. Because NeuCoNS is not dynamic and does not consider the influence one packet has on the other, the results of this approach will be identical as implementing a bi-directional links.

The implementation of the hardware representing graphs is done in class objects. A top level "Network" class forms the (grand)parent class for all other classes. This (grand)parent class covers all the basic methods the communication network needs to fulfil, such as adding and removing nodes, or edges to or from the graph, reading out the traffic after a simulation run, and resetting the network. The (grand)parent class also contains the more complex methods that are independent of the topologies, such as the UC, LMC and MC methods as well as the Dijkstra routing algorithm, and the random and sequential mapping algorithms. The different topologies, such as the square or triangular meshes found in TrueNorth and SpiNNaker, are then implemented as child classes. These child classes contain an automatic generation method to create the network according to given parameters but also contain the topology specific casting, routing, and mapping methods.

3.1.4 Mapping of the Neurons

As mentioned before, one of the unique features of NeuCoNS is that it can analyse heterogeneous NNs. Because individual neurons or populations in a heterogeneous NN can have different connectivity characteristics, they also have different impacts on the network load. This network load depends on both their own, and their targets location in the network. Consequently, this means that the location of the neuron in the network needs to be considered by NeuCoNS and therefore the neurons need to be assigned to nodes in the network. While this mapping can be done in any arbitrary way, it is beneficial to take the connectivity between neurons into consideration. A significant part of this work was spend on the implementation of an efficient, i.e. effective but fast, mapping approach. The different approaches considered in this work are discussed and evaluated in detail in sections 3.2 and 4.2, respectively. For the netlist based connectivity definition the locations of the individual neurons are important and consequently each neuron has to be assigned individually to a node. For the connectivity probability matrix based approach, this is less important as the exact connectivity is only determined during the simulation itself. In this case, only the type of neuron and the number of neurons of this type mapped to a node has to be stored. In NeuCoNS this is done in the way of a tuple (n, X) in which X is the neuron type and n the number of neurons of this type mapped to the node. Depending on whether multiple neuron types can be mapped to a single node or not, multiple tuples can be stored in a list to map multiple different populations

to a node. This implementation also simplifies the implementation of the probability calculation to connect to a node according to equations (3.2) and (3.3), respectively.

3.1.5 Simulation of the Spike Events

After the neurons have been mapped to the graph, the traffic load created by the individual neurons can be calculated. NeuCoNS iterates through all nodes in the network and subsequently through the neurons on these nodes. Depending on which connectivity declaration is used, the targets of the respective neuron are either read out or determined randomly. Once all targets are determined, the routes from the source node to all target nodes can be calculated. Depending on the routing algorithm, these routes either depend only on the source and target node or depend on the combination of target nodes, as is the case for ESPR and NER. In the former case, the route calculation only has to be performed once for every source-target node pair, allowing further optimization. In the latter case, the route has to be determined individually for each individual source neuron, leading to longer simulation times. With the individual routes determined, these routes can be combined—depending on the casting type—to determine the impact a spike packet has. Finally, with the impact of the individual spike packets calculated, the collective traffic load can be calculated by accumulating the individual contributions, weighted with the firing rate of the corresponding neuron. Simultaneously, the latencies of the spike packets are also determined. Because NeuCoNS is a static simulator and does not consider actual neuron behaviour and spike timing, it cannot take the influence one spike has on the processing of other spikes, such as buffering, into account. Instead, the latency is estimated as the number of routers a spike has to transverse as well as the weighted link length passed. The estimated latency of a neuron is then the maximum distance a spike packet generated by that neuron has to travel to reach all its destinations. The maximum latency of the overall system is in turn the maximum latency for all neurons.

3.1.6 Interpretation of Simulation Output

As output, NeuCoNS gives an estimation of the number of spike packets passing through each router and link as well as an indication of the latency, given as the time a packet requires to reach its destination. The value for the latency is calculated as

$$\text{latency} = \#\text{routers} \cdot t_{\text{router}} + \#\text{links} \cdot t_{\text{link}}, \tag{3.4}$$

in which the number sign (#) means "*number of*" and t_{router} and t_{link} are the time required to pass the respective elements—with t_{link} as the time required to pass a link with the normalised length equal to 1. In a later design phase, where these values can be estimated,

the actual values can be passed to NeuCoNS and used. However, by default the values $t_{\text{router}} = 1$ and $t_{\text{link}} = 0$ are used, which results in the latency being given in the number of routers passed, including the starting router, henceforth referred to as number of "hops". As long as the communication network is not saturated and incoming packets can be processed by the routers upon arrival, the latency will be proportional to the number of hops. However, when the network gets saturated and the router needs to queue incoming spikes and buffer outgoing spikes, the latency will rise independent of the number of hops. To get a proper latency estimation in this case, a dynamic simulation has to be performed which takes the actual spike timing and other dynamic aspects into account.

The eventual traffic load is given as number of packets going through the elements of the network. However, this value greatly depends on how often a neuron spikes and sends out a packet. Because NeuCoNS does not simulate the neural behaviour and thus cannot determine which neurons are actually spiking when and how often, the communication traffic load is estimated according to the predefined FRs. The resulting "number of packets" returned does not include a unit of time. To translate the resulting traffic load into a general units of bits per second, the time frame corresponding to the defined FRs and the required number of bits per packet have to be considered. The defined FRs are given as spikes per arbitrary time frame T_{FR}, i.e. in Hz in case $T_{\text{FR}} = 1$s. Then, the resulting throughput can be calculated as

$$\text{Throughput} = \frac{\#\text{spike packets}}{T_{\text{FR}}} \cdot \#\text{bits per packet} \cdot \alpha\,, \tag{3.5}$$

taking the acceleration compared to biological real time, as mentioned in section 2.3 and represented here by α, into account as well. To achieve an accurate quantitative estimation, the defined FRs of the different neurons or populations should be representative to the future use cases. As these FRs can vary significantly for different scenarios, the choice to set the FRs to 1 for this work unless stated otherwise, can be justified in order to determine the average throughput requirement. To do so, the assumption T_{FR} =0.1s can be made based on the average activity rate of 10Hz found in biology. Additionally, the results from NeuCoNS can also be used for a qualitative comparison between different structures and protocols as this assumption remains the same for all scenarios.

3.2 Neuron Mapping Algorithms

The mapping of the neurons is a decisive step in the determination of the traffic load on the communication network. To reduce both the traffic load on the network as well as the latency in the system, it is advantageous to place neurons which are connected

to each other closely together. Placing pairs of connected neurons close to each other means that spike packets travelling between the two have to cross fewer routers and thus will most likely have a lower latency. At the same time, as the spike packets will pass through fewer resources in the network to reach their destination, they have a lower impact on the network load. The different approaches investigated to determine the neuron mapping are described in the following subsections.

3.2.1 VLSI Place and Route Algorithms

The optimization problem to place connected neurons close to each other can be compared to the place and route stage in Very Large Scale Integration (VLSI) design in which logic gates are moved around in an attempt to place connected gates close to each other and reduce their parasitic resistance and capacitance. Because of the similarity between the two, it makes sense to investigate the use of VLSI place and route algorithms for this purpose as well. Therefore, a couple of common VLSI placement algorithms, such as the Kernighan-Lin algorithm [32] and the quadratic wirelength placement algorithm [33], were adapted to work for NNs and were implemented in early versions of NeuCoNS, which is described in [34]. Although being effective to some degree, they proved to be too inefficient in regards to calculation time, mainly caused by the significantly higher level of connectivity found in NNs. Given the scaling of the computation time and the already long computation time at the small scale they were tested at, this approach had to be abandoned.

3.2.2 Population Based Mapping

Based on the premise that the test case NNs are generally defined as populations and projections, a less complex solution is to look at the mapping per population instead of individual neurons. Considering the ACA-project test cases where the connectivity within a population (and area) is higher compared to the connectivity between populations (and areas), it can be reasoned that an efficient mapping algorithm places neurons from the same population close together as a group. A straightforward approach to achieve this is to map the neurons to the network sequentially. Starting from one point in the network, neurons of a population are mapped to a node until the node is full and the algorithm moves on to the next node. Only after all neurons from a population are placed, the algorithm will move on to map neurons from a different population. This ensures that most neurons are placed on a node together with other neurons from the same population. The next challenge is to ensure that neurons from the same population that are located on different nodes are still relatively close to each other, which translates to the order in which the algorithm iterates through the nodes. This order can be visualised as a path going through the network which the algorithm will follow and will from this

point be referred to as a fill sequence in this work. The fill sequences implemented in the current version of NeuCoNS and how they are generated will be explained in the following.

3.2.2.1 Sequential Mapping

Potentially the simplest solution is to fill the network row-by-row. For illustration, the resulting neuron map for the cortical microcircuit model mapped on a mesh network with 100 NpN is shown in figure 3.2. Each pixel in the illustration represents a node in the network. The different colours represent the populations the neurons on the node belong to. The resulting neuron map shows that all populations are mapped as blocks on the network, but some of the blocks are stretched out over the entire width of the network, resulting in large distances between neurons at either end. At a larger scale, such as the multi-area model, this effect is even worse as shown in figure 3.3. A simple improvement is to alternate the direction through the rows, preventing the partitioning of a population, but this does not prevent the stretching of the blocks. The effect this stretching has on the network load is shown later in 4.2.

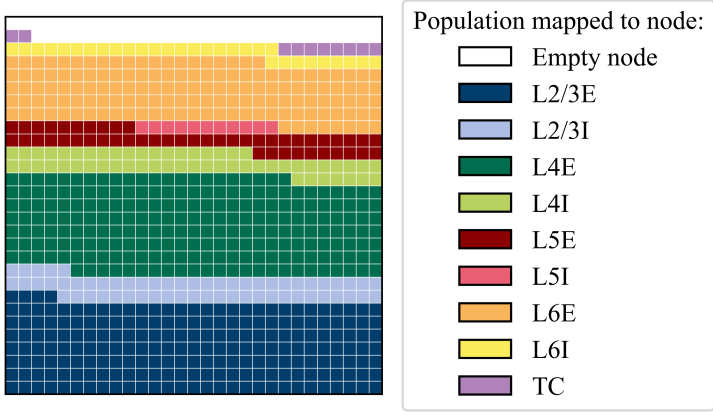


Figure 3.2: Sequential mapping of the cortical microcircuit model to a 29x29 mesh with 100 NpN.

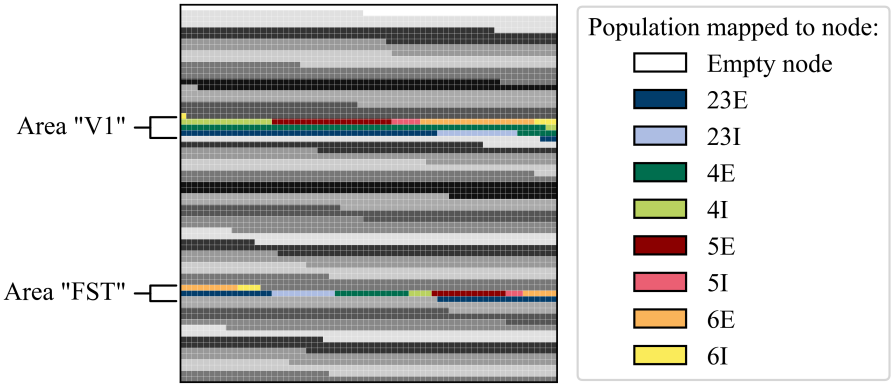


Figure 3.3: Sequential mapping of the multi-area model to a 66x66 mesh with 1000 NpN. For comprehensibility, only the populations of the largest area "V1" and one of the smallest areas "FST" are highlighted by colour keys. The other areas are visualised as different shades of grey to show the mapping of the areas.

3.2.2.2 Population- and Area-Grouping

The second fill sequence aims to create square blocks for each population. The ideal shape to use for the blocks is not a square but depends on the specific topology of the network. Yet, the choice for square blocks was made here, as the maximum distance between two neurons in a block remains close to the minimum for most topologies. This approach can be visualised as filling a large square, i.e. the network, with squares of different sizes, i.e. the populations. There are algorithms to pack a group of squares together in the smallest possible area [35] or pack items in the fewest amount of bins [36]. Before the network graph is generated, the minimum number of nodes N required to map all neurons is calculated and the size of the network is then set to $\lceil\sqrt{N}\rceil$. As a result, the network will have relatively few "empty" nodes which are not used. The mentioned algorithms on the other hand, generally leave empty spaces in their packing solution and thus require a larger network size to be used, making these algorithm unsuitable here. In order to stay within the given network size, an alternative algorithm is proposed, called population-grouping. To fit all populations on the network, the squares have to be deformed into rectangles while preventing them from being stretched to much. To accomplish this, the algorithm takes the maximum number of square blocks that fit next to each other within the width of the network and adds their areas together. Then the height required to achieve the same area over the entire width of the network is

determined. Finally, the blocks are reshaped into rectangles with the calculated height while retaining the same area, and these rectangles are placed next to each other. This process is visualised in figure 3.4.

Figure 3.5 shows the neuron map when this algorithm is used to map the cortical microcircuit to a mesh with 100 NpN. Compared to the sequential mapping, the populations are less stretched out—with exception of the two smallest populations "L5I" and "TC"—as was the intended purpose of the algorithm. However, some flaws of the algorithm can also be recognised for the populations "L4E" and "L5E. After the previous populations have been placed, neurons from these populations are used to fill the row before continuing on the next row. As a result, parts of these populations leak into vacant areas.

The population grouping approach also works for the multi-area model. However, at this scale, another flaw becomes visible and the stretching of the areas starts to become visible again as shown in figure 3.6. This can be prevented by using the areas as the blocks that are mapped instead of the individual populations, grouping neurons from the same area together. Then, after the locations of the areas are determined, the individual populations can be assigned to the region of their corresponding area. Ideally this would be done in a similar fashion, however, due to the way the algorithm is implemented, this hierarchical approach is not possible. Instead the populations are mapped within the areas in the sequential approach discussed before. The effects this choice has on the resulting traffic will be discussed in section 4.2. The resulting neuron map for this scenario is shown in figure 3.7. In general, the algorithm attempts to keep all blocks as square as possible to preserve the locality. This is achieved quite well for the circumstances set here. However, depending on the size difference between blocks on the same row, some of the blocks might be stretched or squeezed heavily diminishing its performance.

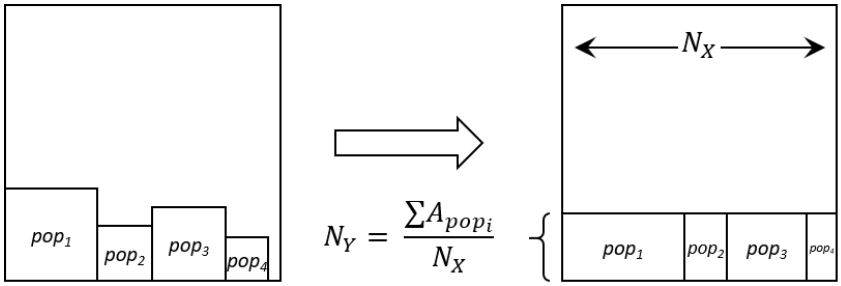


Figure 3.4: Reshaping the square blocks of a population to rectangles.

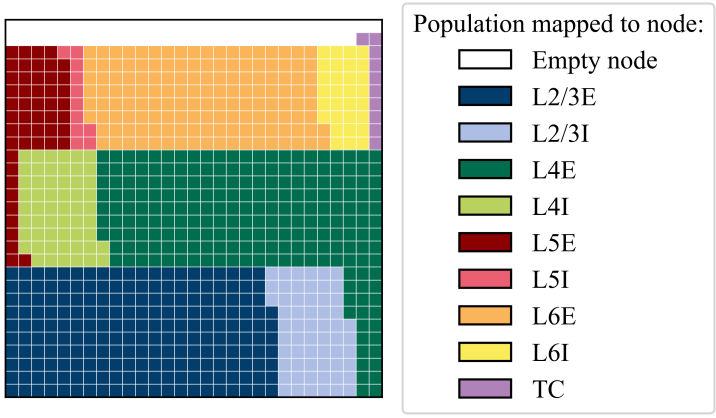


Figure 3.5: Mapping when population grouping is applied to the cortical microcircuit model on a 29x29 mesh with 100 NpN.

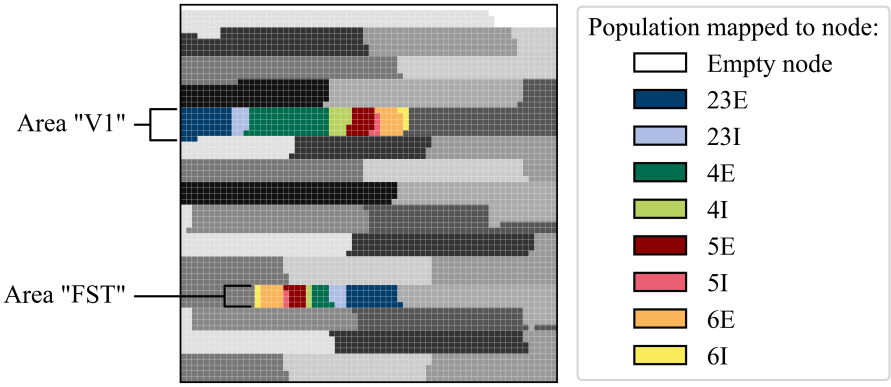


Figure 3.6: Mapping when population grouping is applied to the multi-area model on a 66x66 mesh with 1000 NpN. For comprehensibility, only the populations of the largest area "V1" and one of the smallest areas "FST" are highlighted by colour keys. The other areas are visualised as different shades of grey to show the mapping of the areas.

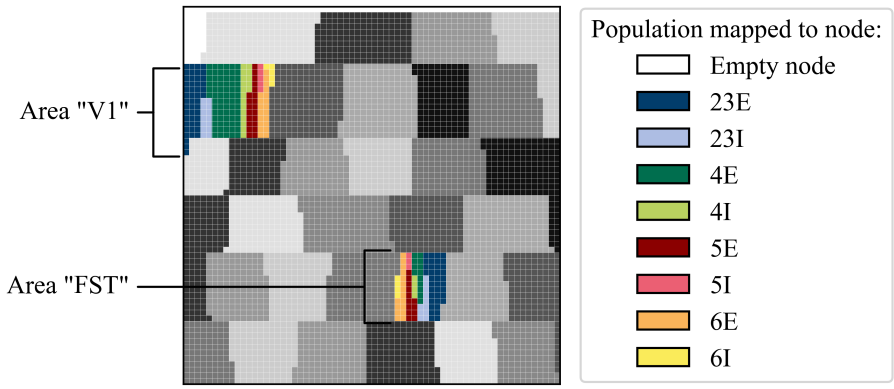


Figure 3.7: Mapping when area grouping mapping is applied to the multi-area model on a 66x66 mesh with 1000 NpN. For comprehensibility, only the populations of the largest area "V1" and one of the smallest areas "FST" are highlighted by colour keys. The other areas are visualised as different shades of grey to show the mapping of the areas.

3.2.2.3 Space Filling Curve

The last fill sequence implemented is based on the Hilbert space-filling curve [37]. The Hilbert curve has the property of preserving locality when converting from a 1-dimensional space to a multi-dimensional space. This can be visualised by mapping a sequence of gradient colours to a grid. When locality is preserved, similar colour tones are clustered together. An example using a 16 by 16 grid is shown in figure 3.8. The white line in this figure shows the actual path of the Hilbert curve.

While locality is preserved when mapping from linear space to a higher dimension, the converse cannot always be true as can be seen in 3.8. While similar shades of colours have coordinates close to each other, some neighbouring cells have completely different colours. These borders are mainly visible at coordinates spaced 2^n cells apart. Nonetheless, often neighbouring cells do have similar shades of colours, showing the degree of clustering the Hilbert curve does provide when used to map from a 2-dimensional space back to a linear space. Because of this property, the Hilbert curve is widely used within the field of computer science in applications such as data space mapping in databases and image compression [38].

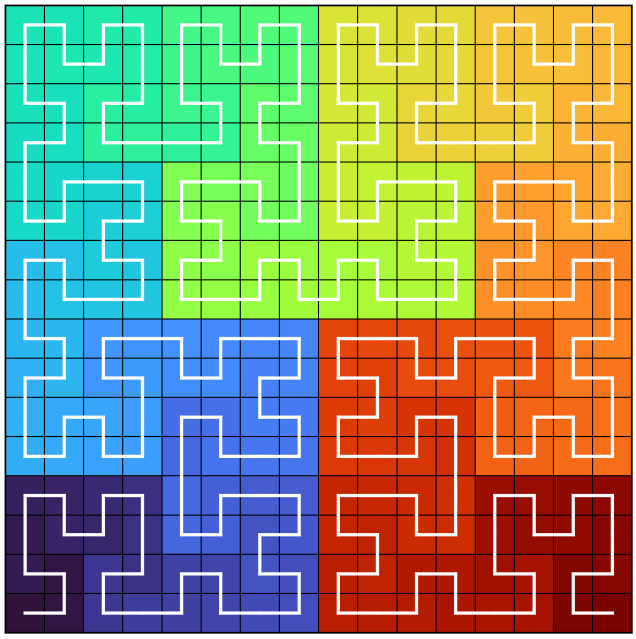
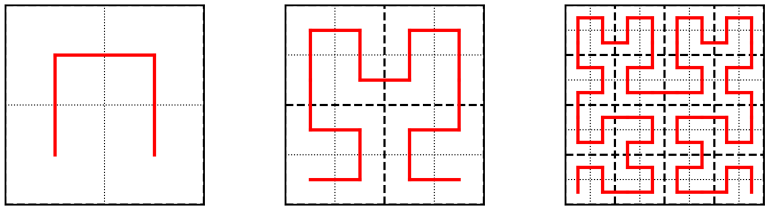


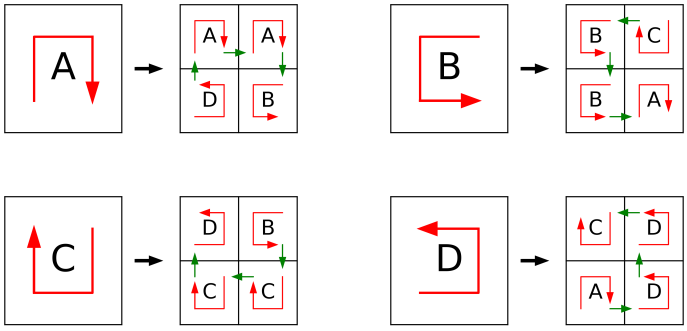
Figure 3.8: Visualization of the Hilbert curve using a colour gradient on a 16 by 16 grid.

The Hilbert curve itself is based on a U-shaped base element, representing the curve over a four celled grid. To generate a Hilbert curve on a 2-dimensional grid, the grid is divided into four sections and arranged according to the U-shape as shown in figure 3.9a. The sections are then iteratively divided into four subsections and arranged in a sequence following (a rotated variant of) the U-shape as shown in figure 3.9b. Repeating this process a third time results in the Hilbert curve shown in figure 3.9c. The production rules of how to replace each of the subsections are shown as well in figure 3.9d. The process can be repeated until each subsection has the size of two by two cells and can be replaced with the sequence corresponding to the U-shape.

A problem that arises here is that after i iterations the resulting Hilbert curve covers a $2^i \times 2^i$ grid. NeuCoNS, on the other hand, uses network sizes of $N \times N$ nodes with any arbitrary number N. To accommodate for this, the algorithm has been altered, mainly to work with odd sized, non-square, (sub)sections. Based on the U-shaped element used in the original Hilbert curve, the goal is to find a path from the lower left corner of the grid to the lower right corner through all cells, without crossing over itself. For B-, C-, and D-type elements, this applies to corresponding corners at the east, north, or west



(a) First iteration. (b) Second iteration. (c) Third iteration.



(d) Transformation rules of the Hilbert curve.

Figure 3.9: The algorithm to generate a Hilbert curve, **(a)** first iteration curve, **(b)** second iteration curve, **(c)** third iteration curve, **(d)** transformation rules of the Hilbert-curve.

border, respectively. If this requirement is fulfilled for a section, the section can replace a quadrant of the U-shaped curve in the original Hilbert curve and the algorithm can continue. However, this is only possible if either the dimension of the corresponding side is even ①, or if both dimensions are odd ②. Thus, in the case that one dimension is odd and the other is even, the respective corners can not lie on the odd side. This means that sections with an even horizontal and odd vertical size either have to be from type A or C, and sections with an odd horizontal and even vertical size have to be from type B or D. Due to the arbitrary network dimension N, the final base elements are not necessarily of the size two by two either but a number of different sized blocks, which are shown in figure 3.10 with the corresponding curves, assuming they are from type A. For types B, C, and D they have to be rotated in the same manner as in the original algorithm.

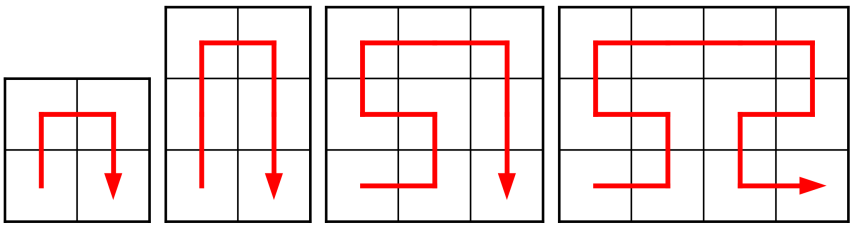


Figure 3.10: The elemental blocks of the altered Hilbert curve for type A.

With this in mind, the new, altered algorithm works as follows. During each step, the x and y dimension sizes of the current section are checked to be either odd or even, and are larger than three. If one dimension is smaller or equal to three, the section can be assigned one of the elemental building blocks shown in figure 3.10, or the rotated version of them. Otherwise, the even dimensions are split into two equal parts and the odd into two parts differing one in size. Which of the two parts needs to be bigger is not arbitrary and is explained later. The algorithms starts of just as the original Hilbert curve, the grid is divided in four smaller subsections and a U-shape is drawn over the four sections. For the approach to work iteratively, each section and its subsections resulting after the division need to fulfil at least one of the following requirements:

① The begin and end corner of the the U-shaped element are located on an even side of the (sub)section. Thus, in the case that one dimension is odd and the other is even, the respective corners cannot lie on the odd side.

② Both dimension of the (sub)section are odd.

Because the starting point is a square shaped network and each section is iteratively divided in four equal—or as close to equal as possible—parts, the difference between the vertical and horizontal dimension can never become larger then one. This also means that if both dimensions are even or both are odd, they need to be equal and thus square. Starting with a square grid, the division of both dimensions always results in to one of the four following scenarios:

1. Both x and y are even. The section fulfils ① and is square. Both dimensions can be divided equally and the resulting subsections will all be square as well. Thus, either ① or ② is true for all subsections and the requirement is fulfilled.

2. Both x and y are odd. The section fulfils ② and is square. The division of such a section with the arbitrary size $(2k+1)\times(2k+1)$ results in the subsections sized $k\times k$ and $(k+1)\times(k+1)$, $k\times(k+1)$ and $(k+1)\times k$ in the four potential configurations shown in figure 3.11. Either k or $k+1$ is odd and the other is even, thus one of the

square subsections fulfils ①, while the other complies with ②. However, the other two subsections only meet the requirement if they are of the correct block types. Assuming that the original section is of type A, the two top subsections will be of type A as well. One of these subsections has the horizontal dimension k while the other has the dimension $k + 1$. The even value of these two complies with ①, independent of its height. The odd value can only do so if its vertical dimension is also odd. Thus, depending on whether k is even or odd, the height of the top row needs to be set to $k + 1$ or k, respectively, to meet ②. As a result, the bottom row, which consists of two blocks of the types B and D, has a height which is even and fulfils ① for these block types. Because the bottom row is independent on horizontal division and the two subsections in the top row are interchangeable, two possible configurations remain and can be chosen arbitrarily for either an even or odd value of k. The same reasoning can be applied to determine the valid configurations for the other types of blocks.

3. x is even and y is odd. This scenario would only fulfil ① if the section is of the correct type. However, because the full size network is a square, this situation can only occur as a result of the division of an odd sized square grid at some earlier iteration. During the previous iteration, the dimensions of this section have been chosen to be as described because this section is either of the A or C type. Because $|x - y| = 1$, the height of the two cells on one of the rows will be equal to their width when dividing the section into subsections. As these two cells are square, they fulfil either ① or ②. If $x/2$ is even, their height is even as well and the height of the other two cells needs to be odd for y to be odd. The other two cells thus have an even horizontal dimension and odd vertical dimension. As this combination would not be a valid configuration for the type B and D blocks, the vertical division needs to be set so that the subsections with these types end up in the two square cells. In the other scenario, where $x/2$ is odd, the height of the square cells is also odd. This means that for y to be odd, the other two cells need an height which is even. Because these cells have an odd width, the only valid block type for these subsections would be either B or D to comply with ①. Thus, in this scenario, the vertical division needs to be set so that the subsections with the type A or C, respectively, are located in the square cells.

4. The reverse scenario where y is even and x is odd can be proven in an identical manner as 3).

Because each of the individual scenarios fulfils the requirements and can only result in one of the listed scenarios, it is proven that the algorithm can be used iteratively for any arbitrary square sized network.

As the population mapping, which attempts to map successive neurons—e.g. neurons from the same population—in close proximity to each other, is based on the same principle

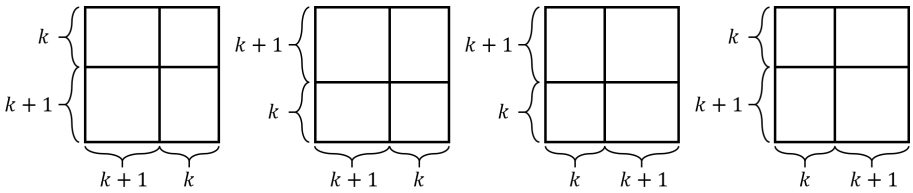


Figure 3.11: The four different configurations when splitting an odd sized square block.

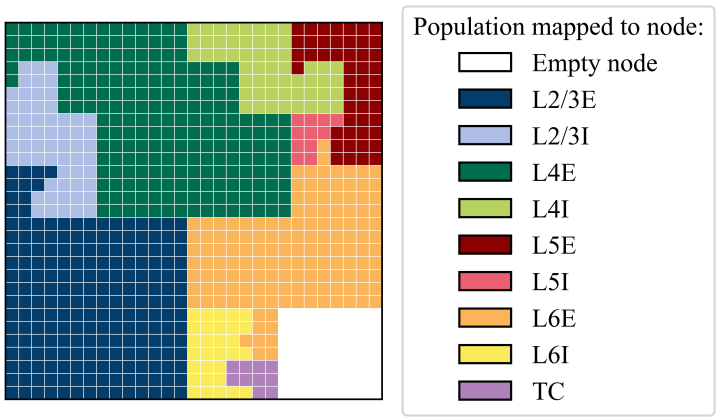


Figure 3.12: Mapping when the space-filling curve is applied to the cortical microcircuit model on a 29x29 mesh with 100 NpN.

as the locality preservation in the Hilbert curve, the space-filling curve should be effective in this scenario as well. Because the original algorithm has been altered, it is no longer referred to as the Hilbert space-filling curve but rather a space-filling curve algorithm in a general sense. Figure 3.12 shows the resulting mapping for the cortical microcircuit when the space-filling curve is used. An additional benefit here is that the space-filling curve does not limit this proximity principle to just neurons from the same population. If applied to the multi-area model, it ensures relative close proximity of neurons from the same area as well, assuming populations from the same area are listed after each other. The inter-area connectivity in the multi-area model, on the other hand, does not follow the same rule and successively declared areas are not necessarily highly connected or even connected at all. A further improvement could be made by rearranging the order in which the areas are declared to better match this approach. Nonetheless, the space-filling curve is an effective approach, which is very flexible, and does not require a

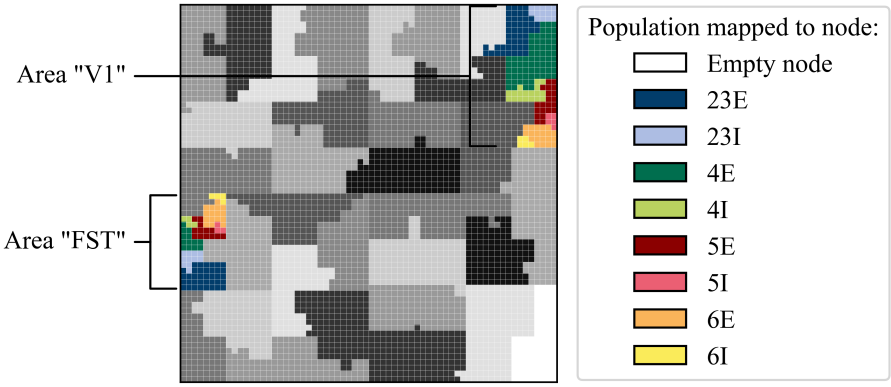


Figure 3.13: Mapping when the space-filling curve is applied to the multi-area model on a 66x66 mesh with 1000 NpN. For comprehensibility, only the populations of the largest area "V1" and one of the smallest areas "FST" are highlighted by colour keys. The other areas are visualised as different shades of grey to show the mapping of the areas.

lot of computations to be perform. The resulting mapping for the multi-area model is shown in figure 3.13. More on the performance of this approach and how it compares to the other approaches introduced before, is discussed in section 4.2.

3.3 Summary and Discussion

In this chapter the network simulator tool NeuCoNS was presented. The unique feature of this simulator is its ability to simulate communication traffic generated by heterogeneous NNs. Because of this, the tool also needs to consider neuron mapping for its communication traffic analysis. Hence, a number of different neuron mapping algorithms, which are implemented in NeuCoNS, have been introduced in this chapter as well.

The current version of NeuCoNS is a useful tool for the evaluation of heterogeneous NC communication networks. However, during the implementation some design choices were made that in retrospect create some limitations. One of those choices is the way the communication network size is determined. In the current version, the network size is always the minimum required square size, except for the topologies based on the existing NC systems, which have fixed sizes. This simulates a network size optimised for the test

case, but this is not a realistic scenario. A real NC system will have a certain size that is generally fixed, or at least requires to be rebuild to change size. If the system is to be used for the simulation of a NN that does not require the full size, parts of the network will remain empty. As a result, the alternative route over the torus connections of the network will be a longer route as the spike packets have to cross over the empty nodes and might not be as useful. Simultaneously, the presence of redundant empty nodes does offer additional flexibility to the mapping algorithm that can potentially lead to a better neuron mapping.

Chapter 4

Simulation Study

In the previous chapter, the network simulator NeuCoNS has been introduced and explained. In addition, a couple of different neural mapping approaches have been discussed. Now, in this chapter NeuCoNS is used to run network simulations and present the results. These simulations are performed with different objectives in mind. First, NeuCoNS is used to verify the correctness of the simulation results with the help of another model, as well as experimental data. Then, the effect, a heterogeneous connectivity model has on the communication traffic, is shown in conjunction with the different implemented mapping algorithms as introduced in the previous chapter. Finally, a collection of traditional and previously introduced network topologies, and communication protocols are analysed to set a benchmark for the novel communication network concept introduced later in chapter 5.

4.1 Verification

Simulators are powerful tools that are used in many fields of study. They are able to predict the behaviour of physical systems and help to better understand them. From a designers point of view, they offer the option to evaluate the performance of a design without the actual need to implement it in hardware. As it is easier to make a change to a model and its parameters describing the design than to change the actual hardware implementation after it has been realised, this approach offers a lot more flexibility at a significantly reduced cost. However, a prerequisite of this approach is that the simulator depicts the physical system accurately. In the initial stages, the operations of NeuCoNS were verified to be correct by manually calculating communication traffic on a very small scale, e.g. 4-16 nodes and less than 25 neurons in total. However, this quickly becomes impracticable and prone to errors when scaling up. To verify that NeuCoNS works as intended on a larger scale as well, a couple of different approaches can be used. A useful approach is to compare the results of NeuCoNS to experimental data measured in an actual NC system. However, when experimental data are unavailable or unsuitable, an alternative approach is to cross-verify NeuCoNS against another simulator, assuming

that the reference works correctly. During the development of NeuCoNS, both these approaches have been used to verify the correctness of the simulator, which will be discussed in more detail in this section. Cross-verification can also be used to verify the correctness of a new version of a simulator against the already verified previous version, a step performed for NeuCoNS with the introduction of each new version.

4.1.1 Cross-verification

The first real verification of NeuCoNS at a descent scale has been done by comparing NeuCoNS against the existing analytical model described in section 2.6.1. The major benefit of this analytical model is that it can accurately describe the average throughput on a system with a relatively simple equation such as (2.8). Because the parameters and variables in the equation can be set as desired, it is simple to match them for both NeuCoNS and the analytical model, and compare the results. However, due to the difficulties to accurately determine the input variables of the equation for complex scenarios, this cross-verification has been limited to homogeneous NNs on a square mesh network. The homogeneous NN that has been used for the verification is a RNDC NN with a connection probability $\epsilon = 0.048$. This value has been chosen as it matches the average connection probability of the cortical microcircuit model. In a RNDC NN, the targets of a neuron are spread out uniformly throughout the network, making the determination of the input variables of the analytical model comprehensible. Referring back to equation (2.8), the parameters and variables that need to be determined for the cross-verification are n, N_{targets}, d_{avg}, TL_{arch} and FR_{avg}. The parameters n and FR_{avg} can be set freely for both NeuCoNS and the analytical model. As of such, FR_{avg} has been set to 1 as the parameter is only used as multiplier in both situations while n has been investigated over a range from 10.000 to approximately 100.000 neurons for the verification. The topology used for the verification is the square mesh in which each node connects to its direct horizontal and vertical neighbours with a node size of 100 NpN. The size of the communication network N has been increased accordingly to n but has been kept at a height to width ratio of 1. For NeuCoNS, n has been incremented in steps of 10.000 neurons, while accordingly scaling N to fit all neurons. However, because the analytical model assumes a fully occupied network, n cannot be increased in fixed increments with a constant NpN and a square network size. Instead, N has been incremented—which in turn increased n—and data points have been created for

$$n = \text{NpN} \cdot N \quad \text{with} \quad N \in [10^2, 12^2, 14^2, 16^2, 18^2, 20^2, 22^2, 24^2, 26^2, 28^2, 30^2, 32^2]$$

resulting in an upper limit of $n = 102.400$. Next, TL_{arch} can be set to $4 \cdot (N - \sqrt{N})$ for the flat network or $4 \cdot N$ for the toroidal mesh as each node in a square mesh has 4 outgoing links. The incoming links do not need to be considered as they are formed by outgoing

links of neighbouring nodes. The remaining two variables depend on a combination of different circumstances. N_{targets} depends on the connectivity of the NN as well as the casting methods used. Here, the commonly used casting methods UC, LMC and MC have been considered for the verification, leading to three different definitions of N_{targets}. In case of UC the average number of targets is equal to the average Synapses per Neuron (SpN) and can be calculated as

$$N_{\text{targets,UC}} = \text{SpN} = n \cdot \epsilon. \tag{4.1}$$

For LMC and MC, on the other hand, N_{targets} is equal to the average number of nodes containing at least one target neuron. Because the RNDC NN has a uniform connectivity, the probability that the number of TSpN is larger than one can be calculated according to equation (3.3) which can be rewritten for the RNDC NN case as

$$P(\text{TSpN} \geqslant 1) = 1 - P(\text{TSpN} = 0) = 1 - (1 - \epsilon)^{\text{NpN}}. \tag{4.2}$$

Thus, in a network with N nodes, this results in

$$N_{\text{targets,LMC}} = N_{\text{targets,MC}} = N \cdot (1 - (1 - \epsilon)^{\text{NpN}}) \tag{4.3}$$

for the LMC and MC casting methods. The final variable that needs to be determined is d_{avg}. This variable depends on the network topology, the casting method, the connectivity scheme, and in case of MC the routing algorithm used. Because of the uniformity of the chosen connectivity model this value can be calculated for UC and LMC as the average distance between nodes in the network. For a regular square mesh topology this is given by the equation

$$d_{\text{avg,square mesh}} = \frac{2}{3}\sqrt{N}, \tag{4.4}$$

or if the opposing sides are connected to each other to create a toroidal network as

$$d_{\text{avg,torus square mesh}} = \begin{cases} \frac{1}{2}\sqrt{N}, & \text{if } \sqrt{N} \text{ is odd,} \\ \frac{N}{2\cdot(N-1)}\sqrt{N}, & \text{if } \sqrt{N} \text{ is even.} \end{cases} \tag{4.5}$$

However, d_{avg} cannot be calculated with a closed form equation in case of MC. In this scenario, the spike sent to multiple target nodes is combined in a single packet at first. Thus, the distance a packet eventually travels cannot be considered as the distance required to reach an individual target node. The distance the packet travels has to be normalised for each individual target node and is the distance from the target node to

either the source, another target node, or a node which lies on the route to another target node. This makes it difficult, if not impossible, to express it as a formula in even this simple scenario with a regular topology and a homogeneous connectivity model. Nonetheless, with the given values for ϵ and NpN, the expected number of target nodes is almost equal to the total number of nodes in the system, $N_{\text{targets}} = 0.9927 \cdot N$ to be precise. This means that almost every node in the network has to be reached by each spike packet, turning MC in BC. Because almost every node is a target, the distance to the nearest branch node, e.g. another target node or an intermediate node, approaches one. This approximation of $d_{\text{avg,MC}} \gtrapprox 1$ can be used in the formula as a lower limit but is only valid for this set of circumstances. Using these values, the average number of packets per link can be calculated for the different scenarios. Figure 4.1 shows the calculated averages for the regular square mesh for the three different types of casting discussed before. The results obtained using NeuCoNS to simulate the same scenarios are also shown as box plots in the same graphs. The same visualizations of the simulation results and calculated averages for the toroidal square mesh are shown in figure 4.2. An explanation how these box plots should be interpreted is given in appendix B.

When comparing the simulation results to the average network load predicted with the analytical model, a couple of observations can be made. First and foremost, the simulated mean values almost perfectly coincide with the calculated values, proving that NeuCoNS works correctly for these scenarios. A second observation is the difference in the level of detail provided by the analytical model and NeuCoNS. The analytical model only calculates the average number of spike packets per link, whereas NeuCoNS determines the number of spike packets on each individual link. This offers more insight in the distribution of the communication traffic. Due to boundary effects in the network, certain regions of the network are less heavily loaded, while other regions are more heavily loaded. This is a result of the nodes in the centre of the network having to process spike packets travelling through the node to reach the other side of the network, while the nodes on the boundary mainly have to process spike packets targeting the node itself. These variations are not observable by just measuring the average values but have a significant impact on the performance of a network as the heavily loaded areas are most likely to get congested and become a bottleneck in the system. When considering the toroidal network, this additional level of detail does not show the same degree of variation. This can be explained by the uniformity of the NN test case used here. Because the toroidal network has no boundaries and the RNDC NN has a uniform connection probability to all neurons in the network, the communication traffic generated by each neuron is statistically identical and the overall communication traffic becomes uniform. The comparison to this analytical model proves the validity of NeuCoNS for the relatively simple scenarios discussed. However, as explained before, the analytical model is unsuitable to be used for more complex scenarios. Another network model, which is able to consider more complex topologies and communication protocols, is the one discussed in section 2.6.2. A further cross verification of NeuCoNS has been performed

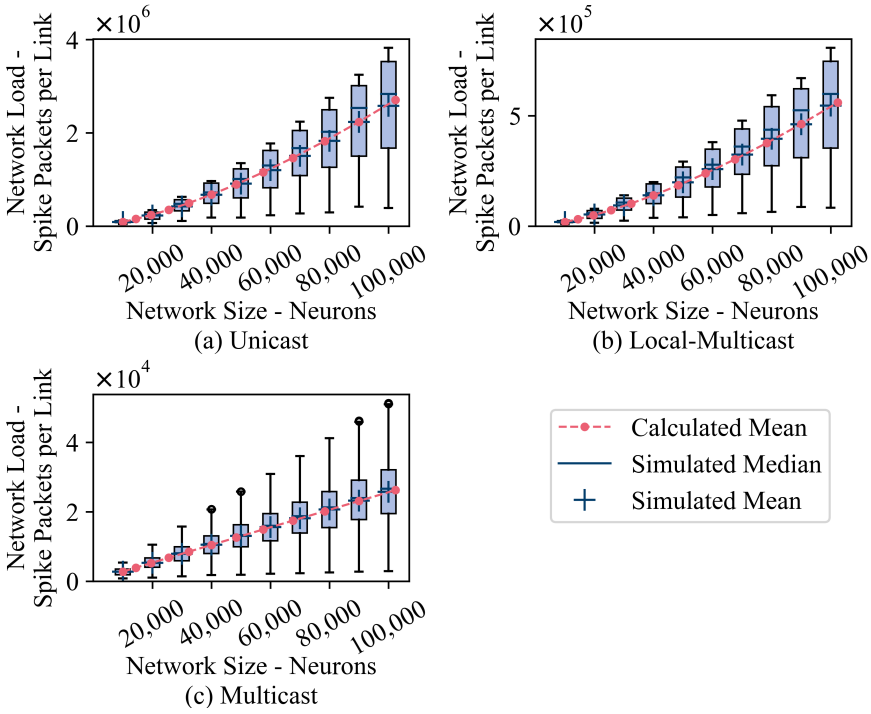


Figure 4.1: Estimation of the required throughput of a RNDC NN on a square mesh. (©2021 IEEE)

with this model as well. This comparison fostered confidence in NeuCoNS, however, due to the confidential nature of involved communications, simulation data were not retained and cannot be shown here.

4.1.2 Verification against Experimental Data

While NeuCoNS can be verified against the other models, these network models only work for homogeneous NN models and as of such are limited in their use. The unique feature of NeuCoNS is its ability to analyse—biologically representative—heterogeneous NNs and should be verified with respect to this feature as well. The best way to validate the correctness of NeuCoNS is to run a network load analysis for a test case and compare the results to empirical data measured running the same test case on an actual NC platform. Unfortunately, no access to such a system was available during this work, but results obtained from a hardware system are conveniently reported in [39, 40]. As

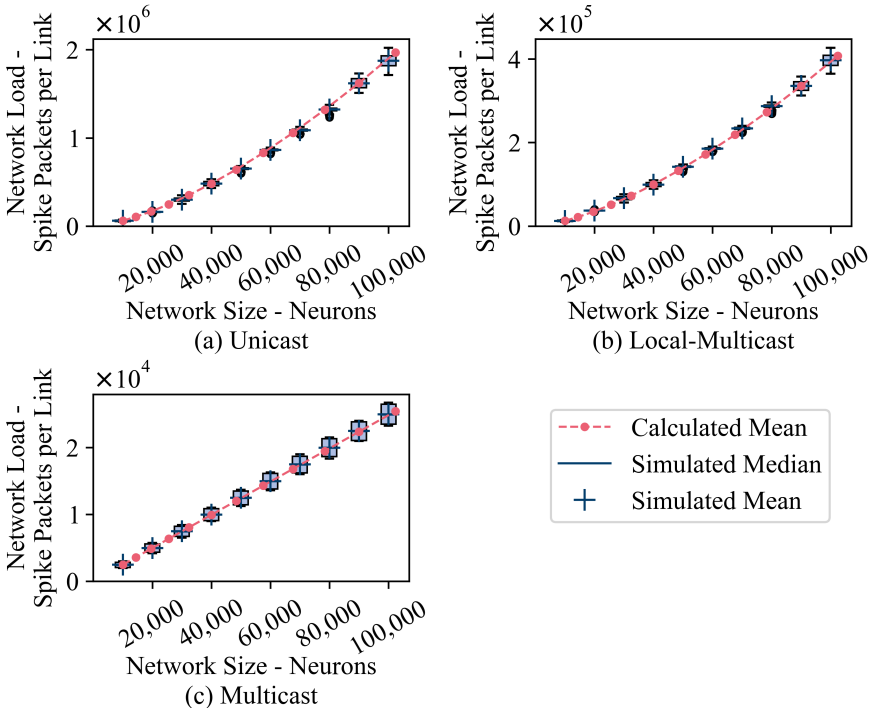


Figure 4.2: Estimation of the required throughput of a RNDC NN on a square toroidal mesh. (©2021 IEEE)

of such, NeuCoNS has been verified against the experimental data presented in these literature sources. As previously mentioned in section 1.3, this verification process has been presented before as part of this work in [3] and will show a high level of similarity with the published article. The previously mentioned sources investigate a bottleneck of the SpiNNaker communication infrastructure as well as different neuron mapping procedures. The first aspect was investigated by passing a large number of spike packets through a single node and measuring the number of spikes dropped, i.e. not reaching their destination, for different test configurations. The conclusion from this analysis was that more packets are dropped when a port of the SpiNNaker chip has to communicate in both directions. Unfortunately, this aspect cannot be recreated using NeuCoNS, as NeuCoNS is not time based and does not take timing delays or input-/output- buffers into consideration. The second aspect, on the other hand, is one of the unique features of NeuCoNS and as of such, an attempt can be made to recreate the results obtained from the respective experiments in order to verify NeuCoNS. In these experiments, the number of packets passing through each router have been measured distinguishing

between two different types of packets, internal and external packets. Internal packets are generated by a local core and sent to the router (core-to-router—C2R), while external packets arrive at the router at one of the input ports connected to a neighbouring router (router-to-router—R2R).

4.1.2.1 Experimental Setup

The reported experiments were performed using a scaled down version of the cortical microcircuit model [23, 41] as test case. To scale down the network, the number of neurons of each population was reduced to 5 percent (N05) and the number of synapses per neuron was reduced to 20 percent (K20) of the original model. On the SpiNNaker system, each neuron was modelled as an Integrate & Fire (IF) neuron and was set up with specific parameters corresponding to the neuron model of the specific population.

In addition to the scaling, some further alterations were made to the NN in order to run it on the hardware and match the temporal behaviour of the model. First, for reasons unknown, the thalamic population was omitted from the model. The rest of the connectivity of the model remained the same, but due to hardware restrictions, some elements needed to be added. The first additional element type are Spike Source (SRC) neurons. They simulate background activity originating from areas of the brain not included in the model as spike trains generated with a Poisson probabilistic process. These spike trains form the majority of all spikes in the system, a characteristic which is important for the mapping as well. More details will be provided later in this section. Because of the large number of spikes, it is infeasible to send them to the corresponding SpiNNaker chips from an external host. Instead, the spike trains are generated by the SRC neurons, where one SRC neuron is associated with one IF neuron and mapped to cores within the SpiNNaker chips themselves. This way, only the parameters of the desired Poisson distribution have to be loaded and the spike trains can be generated on the fly, in the system itself. The second addition has been the use of Delay Extension (DE) neurons. The cores assigned to model the IF neurons are able to handle synaptic delays of up to 16 time steps. However, with a simulation time step of 0.1 ms and normally distributed delays with 1.5 ms mean value for excitatory neurons, some delays will be larger than the available 1.6 ms. To create delays greater than 16 time steps, the DE neurons were used as described in [41]. Due to their simple neuron model—these neurons fire 1-to-1 in response to incoming spikes with a specified delay—these neurons are not limited to the same maximum number of neurons per core as the IF and SRC neurons.

The reference papers [39, 40] investigate three different mapping procedures, PACMAN, MANUAL and GHOST. The PACMAN (PArtition and Configuration MANager) as presented in [42, 43] for example, is the native python package used to configure the SpiNNaker boards to run a given SNN according to a PyNN SNN description. This procedure divides each population in part-populations. Each part-population contains

the maximum number of neurons possible to place per core with exception of the last part-population of a population, which contains the remaining neurons. Then the part-populations are assigned to cores of a SpiNNaker chip sequentially. This approach is very similar to the sequential mapping approach. For every IF part-population assigned to a chip, resources are reserved for the corresponding DE population. Once all cores of a chip have been filled, including the cores reserved for DE neurons, PACMAN progresses to the next chip. The order of the chips is determined by the radial distance around a chip of choice [44], in this case chip (0, 0). The resulting mapping solution is shown in figure 4.3a. Due to the simplicity of the algorithm, it requires only a minimal amount of computational power and can be used for large SNN as well as is shown for the full scale cortical microcircuit model in [45]. However, this comes with its own limitations. The maximum number of neurons per core is the only constraint when splitting the populations and it does not consider any network connectivity while placing the individual part-populations. This potentially leads to very small part-populations and large distances in the hardware between highly connected part-populations.

The second mapping procedure is a MANUAL approach, with the corresponding mapping solution shown in figure 4.3b. This mapping solution was hand-picked in an attempt to achieve unidirectional transmissions in the communication between SRC populations and IF populations, avoiding the bottleneck scenario discussed earlier. While this procedure reduced the number of external packets by 33% and the number of dropped packets—the aspect not considered in this work—to zero, it is still far from optimised. For this mapping procedure, the number of spikes generated per population can be read out as the number of internal spikes in the corresponding node, as each node only simulates a single population. Comparing the number of spikes generated by the SRC populations and the IF populations, a significant difference between the two can be observed. The source neurons will create the majority of spikes in the system, approximately 98% of all spikes, even though they only have to travel a relative short distance, to a single destination. With this mapping procedure, all the DE neurons are assigned to the nodes (0, 0) and (1, 0).

To reduce the traffic through the network further, it would be beneficial to prevent inter-chip communication from SRC neurons to IF neurons. This can be achieved by assigning SRC neurons to the same chips as the corresponding IF neurons. This way, the spikes generated by the SRC populations only have to be transported within the node, and do not contribute to the number of external packets per node. The same idea can be applied for the DE neurons as well while also considering the connectivity between part-populations to determine the location of the IF neurons.

The third mapping procedure introduced in [40], GHOST mapping, does exactly this. By iteratively sub-clustering the groups of neurons from the same population and cluster, the neurons are divided into part-populations, which comply to the maximum number of neurons per core. Then, the Sammon mapping algorithm [46] is used to map the

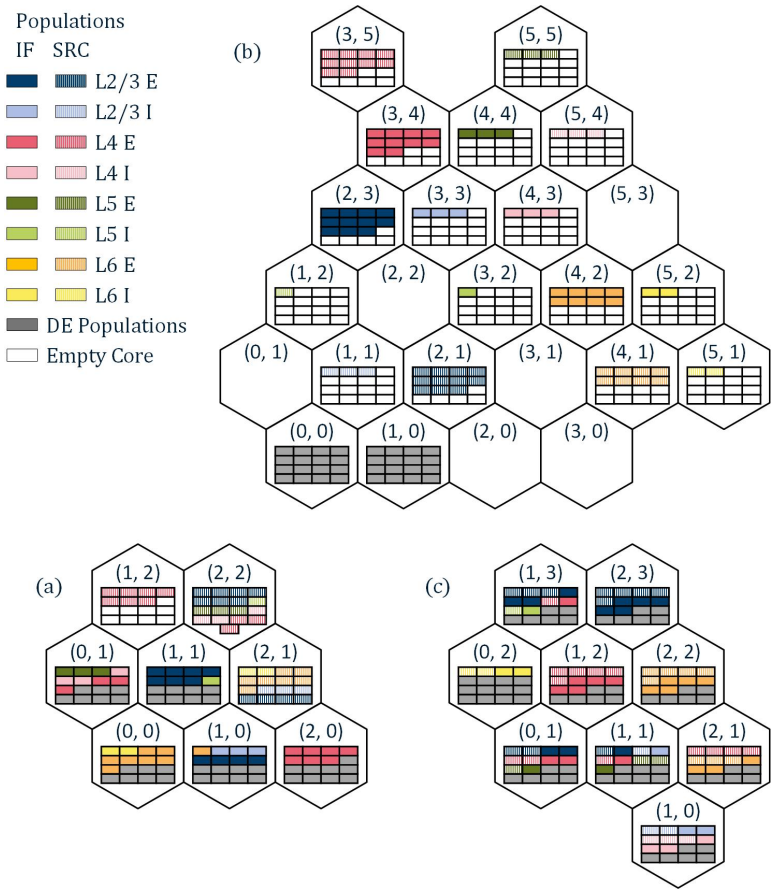


Figure 4.3: The resulting neuron maps for the three different mapping approaches used in this work and [39, 40], **(a)** PACMAN, **(b)** MANUAL, **(c)** GHOST.

part-populations to the individual chips. To ensure that the SRC- and DE-neurons are mapped onto the same chips as their corresponding IF neurons, cores on each chip are reserved for the SRC- and DE- part-populations and assigned to the correct SRC populations at the end of the placement procedure. The resulting neuron mapping is shown in figure 4.3c. It should be noted that in [47, 48] a technique is presented which removes the traffic created by the spike source neurons entirely from the interconnect network, allowing the simulation of full-scale NN models in real time. However, as this technique is not used in the experimental setup, it will not be discussed in detail in this work.

4.1.2.2 Simulation Setup

To validate the correctness of NeuCoNS, the network load measured with the previously described SpiNNaker test setup has been recreated by running NeuCoNS with a comparable setup. The SpiNNaker communication network has been represented as a triangular mesh while MC has been applied. As test case, the same scaled down cortical microcircuit model, or its connectivity information, respectively, including the additional SRC and DE neurons has been used. This SNN will not have been exactly the same as the SNN used in the experiment, but it will be statistically equal. In the NC hardware, traffic is only counted as external traffic (router-to-router—R2R) if it originates from outside the SpiNNaker chip. Communication generated locally, between cores and the router, is counted as internal traffic (core-to-router—C2R). The same classification has been applied by NeuCoNS while representing the SpiNNaker chips as the nodes of the network. As NeuCoNS does not distinguish between the different cores on a chip and combines them into one node, the variable NpN is set to 16×*neurons per core*, but exceptions are made when assigning DE neurons, due to their simpler neuron models.

Because NeuCoNS does not try to simulate neuron dynamics, the neural activity has to be defined by setting the FRs of the neurons. Table 2 from [39] offers the possibility to calculate the average firing rates of each population by dividing the number of internal spikes in each node by the number of neurons of the population assigned to that node. Unfortunately, this cannot be done for the DE populations as there are multiple populations combined within the nodes. Because these populations simply delay the spikes of the original IF neurons, it can be assumed that FRs of these populations are the same as their corresponding IF populations. In [39], these values do not exactly match the number of internal spikes of the IF neurons, which will most likely be caused by spikes falling outside the measurement time window due to the delays. Yet, the experimental data is approximated relatively well as will be shown below. The calculated average firing rates for the populations are given in table 4.1.

Finally, the experiments were performed using the NER algorithm. This algorithm calculates the shortest route to the destination, from the closest, previously visited node—including the source node. This routing algorithm is also implemented in NeuCoNS and thus has been used for the comparison as well. However, the routes calculated by this algorithm vary depending on the implementation of the algorithm. Additionally, a couple of factors such as the routing order, the maximum search range, routing table entry constraints, and branch node constraints affect the resulting routes. A more detailed explanation of the algorithm, including the effects of the different factors listed above, is given in [9]. In the present work, factors like the maximum number of routing table entries as well as limitations regarding the maximum search range and which nodes can be branch nodes are neglected and the destinations are sorted in order from closest to furthest from the source node. Anyway, differences will occur as it is not clear how

Table 4.1: Average firing rates of the different populations of the scaled down cortical microcircuit test case.

Populations	SRC neurons	IF & DE neurons
L2/3E	2560.78	0.8491
L2/3I	2422.62	3.5979
L4E	3362.28	3.8904
L4I	3041.53	7.0293
L5E	3195.97	8.4298
L5I	3048.68	9.2453
L6E	4635.81	1.1516
L6I	3353.83	8.5986

these constraints are handled in the experimental setup. The resulting deviations may be reflected in both the distribution of traffic as well as the total number of external packets. In a regular mesh there are multiple different "shortest" paths to a destination. The route actually chosen determines which nodes will count the external packet. However, even if the route returned by the algorithm has the shortest possible length from the source to the destinations, the total number of links occupied by a combined MC packet can still vary as shown in figure 4.4.

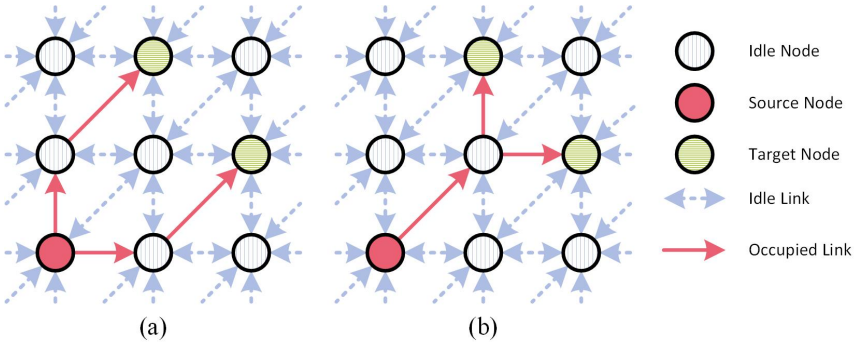


Figure 4.4: Difference in network load between two possible shortest routes to reach all destinations with a MC packet, **(a)** 4 different links used, **(b)** 3 different links used.

4.1.2.3 Results

NeuCoNS can now be verified by comparing the results obtained with the experimental setup against the results from NeuCoNS. With respect to these results, it is important to mention the difference between the numerical data returned by NeuCoNS and the data measured on the hardware platform. In the SpiNNaker system—or any physical NC system for that matter—the number of packets going through a router will always be a positive integer. NeuCoNS, on the other hand, accepts any arbitrary (positive) real numbers as fire rates of the neurons, which are used as multipliers for the neurons' generated network load. Thus, the resulting number of spikes per link and router can also be any arbitrary real number. The results of the different simulations are shown in tables 4.2, 4.3 and 4.4. These values are rounded to a single decimal place. The total values, however, are calculated using the non-rounded values, so that there may be a small discrepancy between the total values stated and a simple column sum. Tables 4.2 and 4.3 also include the results presented in [39] identified as 'Experimental'.

Here, the results obtained using the MANUAL approach are discussed first, deviating from the original order of mapping procedures given before. This is done, as this mapping approach isolates all populations to individual nodes, which gives a clearer overview of what is happening in the system. For this algorithm, the simulated number of internal packets per node matches perfectly with the experimental data. This is to be expected, as this data is used to set up the average firing rates. However, a difference in the nodes (0, 0) and (1, 0) can also be observed due to the delay extensions as discussed before. Overall, the neural activity is well represented by the used FR values and as a result, the simulation data for the total number of external packets in the system matches with the experimental data as well. The number of external packets for the individual nodes, on the other hand, shows some differences. This can be attributed to the dependence on the actual implementation of the NER algorithm and the resulting routes as explained in 4.1.2.2.

For the PACMAN algorithm, the difference between the simulated total number of internal packets and the experimental data is slightly larger. As the same FR values are used for both simulation runs, the number of generated spikes should be identical, which can be observed for the simulation results. However, in the experimental data, the total number of generated spikes is larger. Unfortunately, it is impossible to extract more accurate FR values for this run from the experimental data provided, as multiple populations are placed on individual nodes. The difference might also be enlarged by the reinjection mechanisms of the SpiNNaker system, which retransmits packets that are lost. The difference is relatively small compared to the total number of packets but is especially apparent at the nodes containing the IF & DE neurons. Because the spike packets originating from these neurons generally have to go to multiple nodes at a larger distance, the reduction of internal packets will reflect in the number of external

Table 4.2: Simulated communication traffic results of the scaled down cortical microcircuit test case mapped with the MANUAL procedure.

	External Packets		Internal Packets		Neurons
Node	**Simulated**	**Experimental**	**Simulated**	**Experimental**	**(SRC & IF)**
(0, 0)	3 456.0	3 456	3 456.0	4 351	0
(0, 1)	0	3 146	0	0	0
(1, 0)	11 679.0	8 232	9 270.0	9 129	0
(1, 1)	4 503.0	17 740	704 983.0	704 983	291
(1, 2)	0	18 001	161 580.0	161 580	53
(2, 0)	1 264.0	2 092	0	0	0
(2, 1)	19 685.0	15 040	2 647 850.0	2 647 850	1034
(2, 2)	3 527 800.0	3 532 713	0	0	0
(2, 3)	2 669 406.0	2 675 945	878.0	878	1034
(3, 0)	0	1 264	0	0	0
(3, 1)	1 264.0	828	0	0	0
(3, 2)	185 278.0	191 753	490.0	490	53
(3, 3)	726 860.0	727 444	1 047.0	1 047	291
(3, 4)	3 700 361.0	3 706 900	4 260.0	4 260	1095
(3, 5)	0	0	3 681 697.0	3 681 697	1095
(4, 1)	0	1 264	3 333 150.0	3 333 150	719
(4, 2)	3 357 774.0	3 364 249	828.0	828	719
(4, 3)	850 363.0	856 929	1 919.0	1 919	273
(4, 4)	794 308.0	800 847	2 040.0	2 040	242
(5, 1)	0	0	493 013.0	493 013	147
(5, 2)	517 201.0	520 765	1 264.0	1 264	147
(5, 3)	0	10 144	0	0	0
(5, 4)	0	6 300	830 338.0	830 338	273
(5, 5)	0	0	773 424.0	773 424	242
Total	16 371 202.0	16 465 052	12 651 487.0	12 652 241	7708

packets in an amplified way as can also be seen in table 4.3. The distribution of these external packets over the different nodes, again depends on the actual implementation of the routing algorithm, and thus differs significantly. However, the difference in the total number of external packets is still acceptable considering the different sources of variation discussed before.

The simulation results for the GHOST mapping algorithm are shown in table 4.4. Unfortunately, for this algorithm, only the total number of 250 000 external packets is reported for the experimental data. The relative difference with the simulated total number of 189 422.8 external packets is substantially larger for this setup. Nonetheless, the simulated

Table 4.3: Simulated communication traffic results of the scaled down cortical microcircuit test case mapped with the PACMAN procedure.

	External Packets		Internal Packets		Neurons
Node	**Simulated**	**Experimental**	**Simulated**	**Experimental**	**(SRC & IF)**
(0, 0)	3 759 394.3	3 773 911	4 140.2	4 882	847
(0, 1)	2 625 117.9	2 630 277	10 252.2	12 729	815
(1, 0)	4 512 765.8	5 587 441	2 817.1	3 334	710
(1, 1)	8 217 079.4	4 424 623	2 056.7	2 412	687
(1, 2)	2 612 446.1	0	2 673 012.9	2 672 252	795
(2, 0)	2 689 547.7	2 692 613	6 185.8	9 471	795
(2, 1)	221.9	2 696 375	5 555 459.3	5 559 023	1557
(2, 2)	0	2 672 252	4 397 562.8	4 397 319	1502
Total	24 416 573.1	24 477 492	12 651 487.0	12 661 422	7708

traffic is of the same order of magnitude as the experimental data, which is largely reduced in comparison with the other two experiments. Thus, NeuCoNS did manage to capture this reduction of the network load due to the improved mapping procedure.

4.1.3 Discussion

There are a couple of potential reasons which can explain the larger variation for the GHOST mapping experiment. As discussed before, the FR values of the IF & DE neurons extracted from the MANUAL experiment don not necessarily represent the FR values of the other experiments correctly. For the other experiments, i.e. in case of the PACMAN or the MANUAL algorithm, the total number of external packets is dominated by packets originating from the SRC neurons, which are represented well by the FR values. Because of this, the relative difference was neglectable previously. In case of the GHOST algorithm, however, the SRC neurons are located on the same nodes as the corresponding IF neurons and have no impact on the number of external packets. Thus, while the absolute difference is almost comparable for all experiments, approximately 60.000 external packets, the relative difference is significantly higher. A second potential cause for the observed difference are variations between the neuron activities within a single population. NeuCoNS assumes an average firing rate for all neurons in a population. In an actual SNN, on the other hand, different neurons within a population can and will have different rates of activity. With all the neurons of one population placed onto a single or maximum 2 nodes, which was the case for the first two experiments, these variations will average out. In case of the GHOST algorithm, however, the populations are spread out more and the variations can have an impact. This may be especially the case when highly interconnected neurons that potentially reveal a shared lower or

Table 4.4: Simulated communication traffic results of the scaled down cortical microcircuit test case mapped with the GHOST procedure.

	External Packets		Internal Packets	Neurons
Node	**Simulated**	**Experimental**	**Simulated**	**(SRC & IF)**
(0, 1)	19 707.6		1 409 414.8	934
(0, 2)	22 924.0		495 541.0	294
(1, 0)	16 710.0		1 305 560.7	934
(1, 1)	18 564.8		1 330 679.5	904
(1, 2)	21 561.6		1 685 030.6	1000
(1, 3)	20 694.8		1 218 815.9	868
(2, 1)	23 291.8		1 914 925.5	934
(2, 2)	24 415.6		2 087 152.6	900
(2, 3)	21 552.6		1 204 366.4	940
Total	189 422.8	250 000	12 651 487.0	11562

higher activity due to their high connectivity, are concentrated into the same subgroup. A final potential cause is one obscurity in the mapping solution used in [40]. The paper gives a detailed explanation of the algorithms operating principle, however, the resulting mapping solution is only visualised in an abstract manner. This is sufficient to derive a comparable neuron mapping but not the exact same neuron placement, especially in combination with a SNN, which is only statistically equal. As most of these potential causes are a result of not having detailed enough input data, there are possibilities to improve the comparison between experiments and simulations by having access to more detailed experimental parameters. Ideally, both the experiment and the simulations would be performed by ourself, so the exact same NN can be used for both. This way, the exact FR of each individual neuron can also be determined, just as the exact neuron mapping and the routing algorithm. Another way to improve confidence, is to compare a larger number of different scenarios. The results of simulation and experiment might not match perfectly for every scenario, but the same relative behaviour might be observed between the two.

Another point of discussion is the metric used by NeuCoNS. To estimate the networks performance, NeuCoNS returns a latency value and the number of spike packets passing through each link and node. However, in the SpiNNaker system, the interconnect performance is generally measured by the number of packets being dropped. Due to the high level of abstraction of NeuCoNS, it is not possible to simulate this. The probability of packet dropping can be estimated by comparing the maximum throughput simulated and the maximum number of packets the routers can handle. If the former is larger than the latter, packets will be dropped, but even if the throughput is slightly below the

maximum capacity, the probability of packets being dropped is relatively high. Often, even the probability of dropping a packet can be seen as bad. However, as described in [39, 40], the maximum number of packets a router can handle is not constant and depends on the directions of the different incoming and outgoing traffic streams, making it difficult to accurately predict the occurrence of dropped packets.

4.1.4 Conclusion

The total number of external and internal spikes, where applicable, measured and simulated for the three procedures are shown in figure 4.5. For both the PACMAN as well as the MANUAL mapping procedure, the simulated number of external packets in the system corresponds very well to the experimental data. The small differences observed for these two scenarios, 0.25% and 0.5% respectively, are well within the margin of error considering the different potential causes for variations. For the last mapping procedure, a substantially larger difference of 24.2% can be observed between NeuCoNS and the experimental data. Unfortunately, the exact causes of this larger relative deviation cannot be identified due to the lack of a detailed overview of the experimental network load. Nonetheless, NeuCoNS does capture the same level of reduction of external packets caused by the GHOST mapping approach.

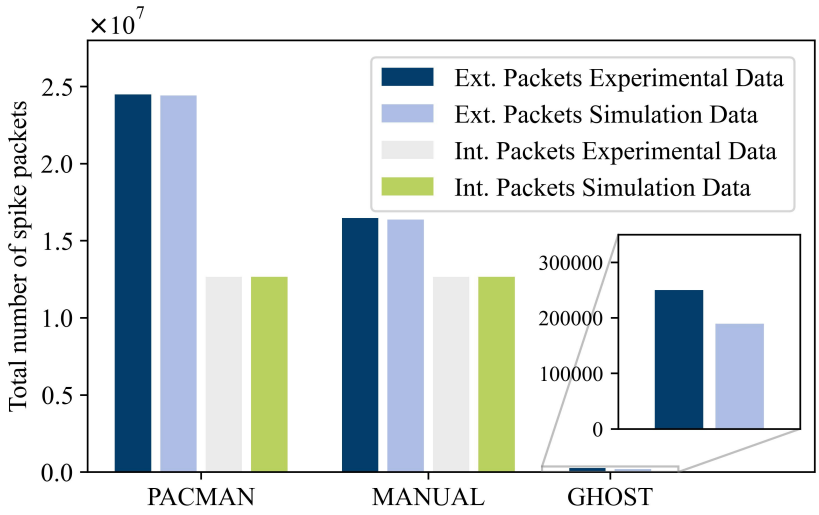


Figure 4.5: The total number of spike packets measured in the experimental setup compared with the simulated values for the different mapping procedures.

As NeuCoNS is meant as a first stage design tool, it will most likely be used for qualitative comparisons and this will be sufficient. This especially applies, because the exact parameters of the test case, such as the FR of the different populations, are still unknown at this point. Other factors like the connectivity or even the entire test case, and with it the mapping, might also change for different analyses performed using the NC hardware.

4.2 Mapping Analysis

Previously, in section 3.2, the different mapping algorithms implemented in NeuCoNS have been described. In this section, these algorithms are evaluated further using NeuCoNS. First, to set a benchmark, the NN test cases introduced in section 2.5 have been randomly mapped to the network. The results achieved with these mapping algorithms are then compared to the results achieved with this benchmark in order to evaluate their effectiveness to reduce the communication traffic, both at small scale, and at a larger scale. This comparison will also show the difference of communication traffic a heterogeneous connectivity model creates compared to homogeneous connectivity models. First, the analysis will be performed using a non-toroidal and toroidal square mesh topology in combination with LDFR. A comparison between the different topologies and routing algorithms will be performed later in section 4.3. For the small scale test case, the analysis is limited to LMC, as the relatively high connectivity in the cortical microcircuit model would result in a close to BC scenario, when MC is used, so that all heterogeneous characteristics are lost. For the large scale test, on the other hand, i.e. the multi-area model, both the LMC and MC casting types are evaluated. The UC protocol is completely omitted further in this work, as this casting type has already shown a significantly larger network load compared to LMC in section 4.1, while it does not offer any significant reduction in complexity. The simulated results are presented in two different forms, as heat maps and in the form of box plots. How these box plots should be interpreted is explained in appendix B. In the heat maps, each pixel represents a node in the network with the colour representing the number of spike packets travelling through the node. This way of visualization shows the effect the mapping algorithms have on the distribution of communication traffic through the network and offers the potential to search for congestions caused by the mapping. The box plots, on the other hand, give a good overview of the overall communication traffic, as they show the mean, median, minimum and maximum values of the communication traffic, as well as the statistical distribution. This representation is better suited to quantitatively compare the results of the different scenarios.

4.2.1 Small Scale Analysis

Starting with the benchmark, the heat maps shown in figure 4.6 visualize the communication traffic in the network using random mapping for the small scale test case, i.e. the cortical microcircuit model. Randomly mapping the neurons counters the effect of the heterogeneous connectivity model. Differences in the connectivity between populations are balanced out, as multiple different neuron types are placed on the same node. Because the heterogeneous characteristics are lost by the random mapping, the resulting communication traffic is uniform as can be seen in the heat maps and the communication traffic load will be similar to that of a RNDC NN with the same average connection probability as shown in [30]. The only deviations from this uniformity are created by the boundary effects of the network in the non-toroidal variant of the mesh. In the toroidal mesh network, on the other hand, there are no boundaries—and thus no boundary effects—and the traffic load is almost perfectly uniform through the entire network. As this random mapping does not take any connectivity into consideration while placing the neurons, it can be seen as the upper limit of the communication traffic. The only way to increase the overall traffic load above this limit, is by intentionally placing connected neurons further apart—not considering the re-injection of dropped packets as discussed in the previous section—the exact opposite of the goal here. An effective mapping algorithm will decrease the communication traffic to some degree compared to this benchmark and the amount of this reduction can be used as a performance metric.

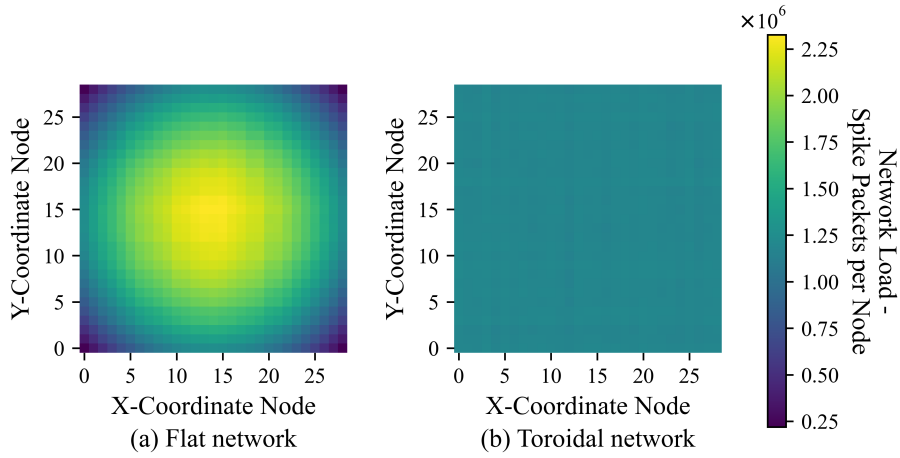


Figure 4.6: Heat maps of the number of spike packets travelling through each node for the cortical microcircuit model randomly mapped to a 29x29 grid with 100 NpN, **(a)** with a flat mesh, **(b)** with a toroidal mesh.

The heat maps for the same small scale analysis using the other mapping algorithms are shown in figure 4.7. At this scale, uniformity in the communication traffic load can still be observed to some degree for both the flat mesh and the toroidal mesh, if the potential boundary effects are taken into consideration. However, significant differences can be observed as well compared to the randomly mapped benchmark. For all types of mapping, the maximum traffic load per node is reduced. This is not visible from the heat maps themselves, but rather from the change in range of the colour bar. This reduction will become more apparent in the box plot comparison shown at the end of this

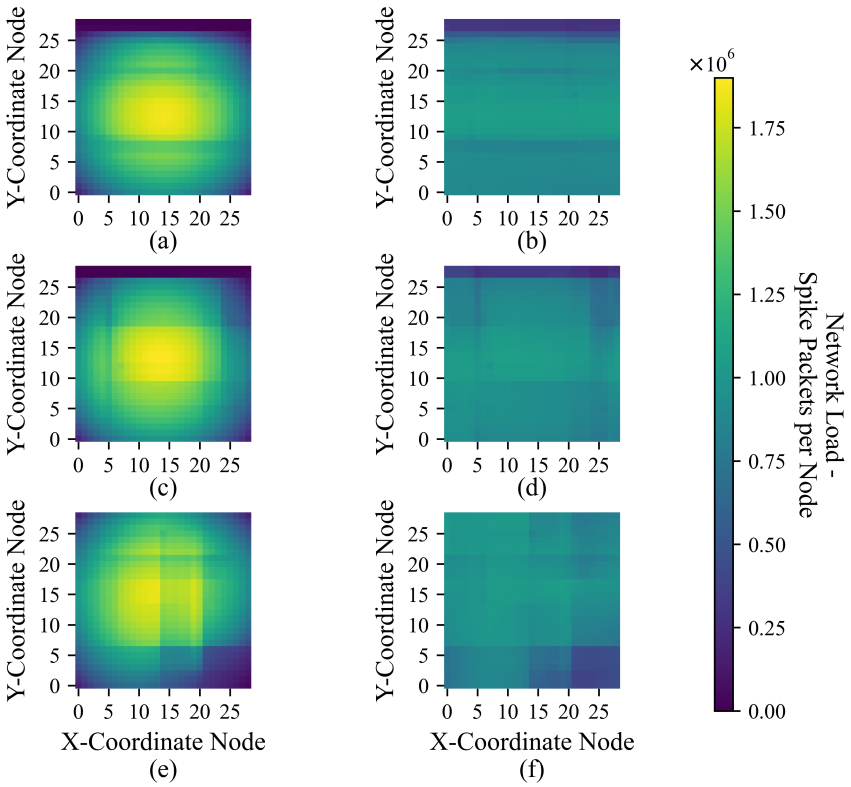


Figure 4.7: Heat maps of the number of spike packets travelling through each node for the cortical microcircuit model mapped to a 29x29 grid with 100 NpN, **(a)** sequentially mapped to a flat mesh, **(b)** sequentially mapped to a toroidal mesh, **(c)** mapped to a flat mesh using population grouping, **(d)** mapped to a toroidal mesh using population grouping, **(e)** mapped to a flat mesh using a space-filling curve, **(f)** mapped to a toroidal mesh using a space-filling curve.

subsection. Differences between different regions in the network can also be observed, showing the impact a heterogeneous connectivity model has on the traffic load. Before diving deeper into the simulation results, it is important to understand what effects are at play in the network and cause the differences in network load. In the LMC scenario, the populations with the highest connectivity, both incoming and outgoing, will be most likely to form hotspots in the network. A high out-degree means that a large number of spike packets is generated by a neuron as each individual target node requires its own spike packet. A high in-degree meanwhile requires a large number of spike packets being sent to the neuron and have a similar effect. Naturally, a combination of both a high in- and out-degree of a group of neurons will result in high network load in that region of the network. Besides in regions with a high in- and out-degree, these hotspots can appear at cross-points of multiple medium traffic flows, but these regions are harder to predict, as they depend on multiple variables such as the mapping and routing algorithms. Obviously, in a real situation, the creation of these hotspots also depends on the different FRs of the neuron populations, but as explained before, these FR distributions are not considered in this work. The figures 4.7a and 4.7b show the heat maps for the sequentially mapped cortical microcircuit. Just as before with the random mapping benchmark, the toroidal mesh negates the border effect and creates a more uniform distribution of the traffic. Nonetheless, a reduction of the traffic can be observed in the rows $Y = 7$, 8, 20, 27 and 28 in both cases. This reduction is most significant in the upper two rows. Looking at the neuron map in figure 3.2, this can easily be explained by the fact that these rows are empty. The other rows mentioned, on the other hand, are not empty and still have a reduced communication traffic load. In this case, the reduction is caused by the presence of smaller, lesser connected populations of neurons, which transmit and receive less spike packets as mentioned earlier. The heat maps shown in 4.7c to 4.7f show the traffic distribution for the population grouping algorithm and space filling curve algorithm, respectively. Similar observations can be made here, when put next to the neuron maps in figure 3.5 and figure 3.12. A significant reduction of traffic appears in the parts of the network which are empty, but additional regions with lower traffic loads can be observed as well depending on the population mapped to the respective region. However, this reduction is not solely caused by the actual population mapped to the nodes. For example, using the space filling curve, the location of population *L6I* (between rows 0 to 6 and columns 14 to 20 in figure 4.8f) has an effect on the traffic in the entire column and row. As the connectivity of population *L6I* is relatively low, there is a reduced amount of communication happening to and from this region. Because of the square mesh network topology, communication links run horizontal and vertical through the network. Meanwhile, the LDFR routing algorithm travels in one direction at a time. Because of this, the resulting reduction of traffic becomes visible in the entire columns and rows in which this population is located. The same is true for population *L5I*, but here the effect is mainly visible in the rows and is camouflaged in general as the population is surrounded by other populations, which still communicate with each other and whose traffic is passing through the region.

To better compare the performance of the different mapping algorithms to each other, the traffic distributions are plotted as box plots in figure 4.8. From this figure, it is easy to see the general reduction of network traffic compared to the randomly mapped benchmark. As mentioned before, the maximum traffic load measured in Spike Packets per Node (SPpN) is reduced, and so are the median and average numbers of spike packets for all three types of neuron mapping. The maximum values are reduced by roughly 12% and the averages by 25% compared to the random benchmark. However, the differences between the individual mapping algorithms are less pronounced. There are differences between the traffic distributions over the network, as seen before in figure 4.7, but these differences are mainly location dependent and are not visible in the box plots. Apart from that, the performance of the different mapping algorithms to reduce the communication traffic is very similar at this scale. With regards to the latency, i.e. the distance between two connected neurons, the difference is even smaller. The latency values are listed in table 4.5. There are small variations between the average latencies for the different mappings, but the maximum latency—the critical metric here—is almost identical for all four cases. This can be explained by the relatively high connectivity level of this small scale test case. Even a small probability of connecting to a type of neurons results in a likely connection when considering a group of neurons as can be derived from equation (4.2). If a neuron in one of the corners of the network is connected to a neuron in the opposing corner, the maximum latency in the network becomes equal to the maximum distance in the network. For the flat mesh network, this maximum value is slightly lower

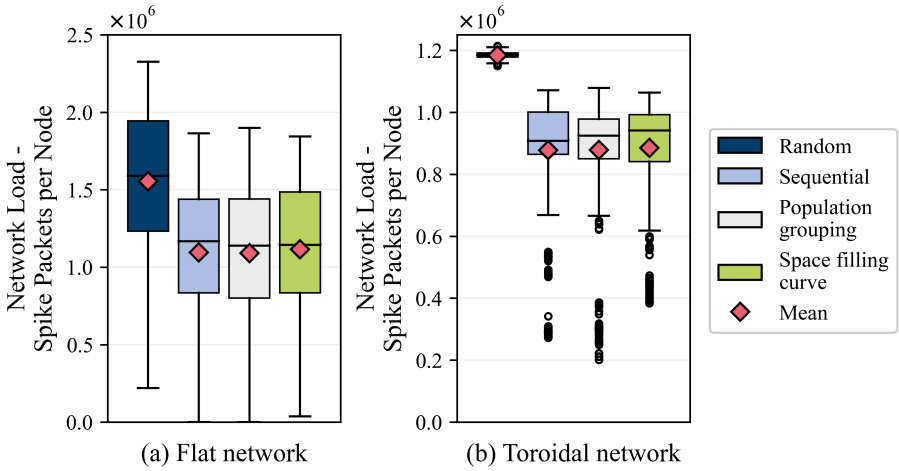


Figure 4.8: Box plots of the communication traffic per node using the different mapping algorithms to map the cortical microcircuit model, **(a)** of a flat mesh network, **(b)** of a toroidal mesh network.

Table 4.5: Estimated latency in number of hops for the cortical microcircuit model using the different mapping algorithms.

	Flat network		Toroidal network	
Mapping	**Average**	**Maximum**	**Average**	**Maximum**
Random	43.5	57	29	29
Sequential	40.4	55	28.5	29
Population grouping	40.4	55	28.9	29
Space filling curve	40.7	57	28.7	29

for the sequential mapping and the population grouping algorithm. These mappings leave the two top rows almost completely empty in this specific scenario. As a result, the maximum distance between occupied nodes is reduced and with it the maximum latency. To achieve this result in general, without relying on empty nodes, opposing corners should be filled with neuron pairs, which are not connected to each other. However, in the cortical microcircuit, almost all populations are connected in at least one direction, the only exception being the *TC* population. This makes it difficult to create a mapping which follows this rule without dividing the *TC* population into smaller blocks.

4.2.2 Large Scale Analysis

The effect a heterogeneous connectivity model can have on the communication network becomes visible for the small scale test case, but the significance of the effect is still limited. As of such, the effectiveness of the different mapping algorithms is less apparent as well. To further evaluate the mapping algorithms, a larger scale analysis has been performed using the multi-area model. Figure 4.9 shows the heat maps of the communication traffic for this large scale test case using random mapping in combination with LMC. The same effects seen for the small scale test case can be observed here as well. The random mapping counteracts the heterogeneous properties of the connectivity model and the only significant variations in the network are caused by boundary effects.

At this scale, however, the differences between the homogeneous and heterogeneous connectivity models become very clear once the other mapping algorithms are applied. The heat maps of the different mapping algorithms using LMC for this test case are shown in figure 4.10, it is important to note the different colour scale used in this image. Just as before, the neuron mappings leave a small region of the network empty, which results in a low level of communication traffic in these regions. For the regular flat mesh, the influence of the boundary effects can also be observed to a lesser degree as compared to the small scale test case. However, a significantly higher level of non-uniformity can be observed in the rest of the network and in the toroidal variant. Looking at the heat maps for the sequentially mapped multi-area model shown in figures 4.10a and 4.10b, the

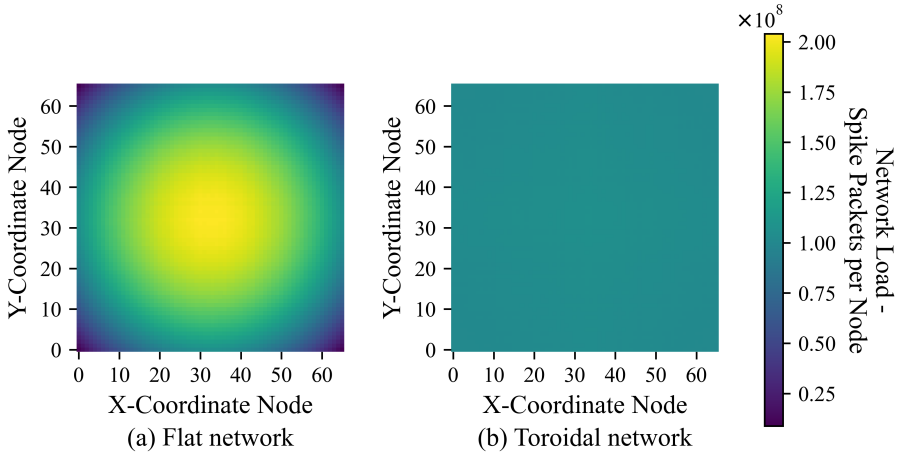


Figure 4.9: Heat maps of the number of spike packets travelling through each node for the multi-area model randomly mapped to a 66x66 grid with 1000 NpN using local-multicast, **(a)** with a non-toroidal mesh, **(b)** with a toroidal mesh.

spreading of the populations and areas across the entire width is clearly visible. As the connectivity is highest within the areas and populations, a significant amount of traffic occurs in the horizontal direction. The differences in connectivity between different areas cause the differences in intensity between the rows, resulting in the horizontal striped pattern with the highest loads in the populations with the highest connectivity levels. In the heat maps resulting from using the population grouping, the area grouping and the space filling curve, shown in figures 4.10c to 4.10f, the distinction between the different areas and populations is less pronounced. Using these mapping algorithms, the traffic load within an area is reduced as the packets have to travel over a shorter distance and thus occupy fewer nodes and links. Meanwhile, the hotspots in the sequentially mapped scenarios created by the intra-area communication packets travelling in the horizontal direction are divided over multiple rows. However, the opposite is true for the inter-area communication. In the sequentially mapped case, incoming spike packets moving in the vertical direction, originating from other areas, are spread out over multiple columns and average out over the entire width of the network. By clustering the populations together, the incoming traffic of a population is also clustered to a group of columns. A balance has to be found between these two aspects in order to prevent hotspots forming in the network. Generally, populations with a high in- and out-degree will result in an increased communication load in the corresponding rows and columns. As the LDFR algorithm moves in one dimension at a time, these increased traffic loads will be visible in the rows and columns of the originating nodes and reduce, when moving further away,

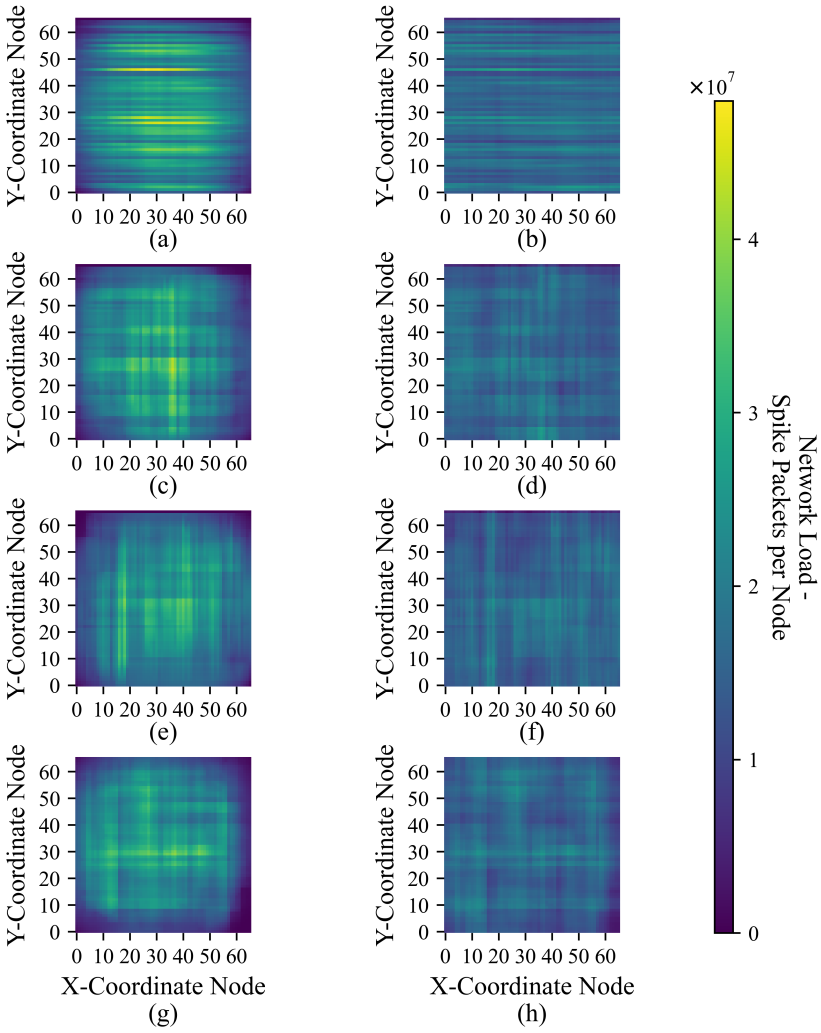


Figure 4.10: Heat maps of the number of spike packets travelling through each node for the multi-area model mapped to a 66x66 grid with 1000 NpN using local-multicast, **(a)** sequentially mapped to a flat mesh, **(b)** sequentially mapped to a toroidal mesh, **(c)** mapped to a flat mesh using population grouping, **(d)** mapped to a toroidal mesh using population grouping, **(e)** mapped to a flat mesh using area grouping, **(f)** mapped to a toroidal mesh using area grouping, **(g)** mapped to a flat mesh using a space-filling curve, **(h)** mapped to a toroidal mesh using a space-filling curve.

as more and more spike packets change direction.

The results of these runs are also plotted as box plots shown in figure 4.11. In this representation it is clearly visible that the use of a mapping algorithm can reduce the network load significantly. Compared to the difference with the random mapping benchmark, the difference between the individual mapping algorithms is small but still more apparent than for the small scale test case. It can also be concluded that the performance of the mapping algorithm is not significantly affected by the use of a toroidal connection. To prevent repetition by describing both scenarios separately, the results will be evaluated simultaneously in this paragraph. While the relative reductions in this section will relate to the results obtained for the non-toroidal mesh variant, the respective values for the torus network will be appended in brackets for completeness. Surprisingly, the sequential mapping performs quite well despite its simplicity. Nonetheless, it is the worst of the four algorithms tested here and will be used as the benchmark from this point onwards. Performing slightly better is the population grouping algorithm, which reduces the average number of SPpN by 4.1% (2.7%) and the maximum number of SPpN by 8.5% (7.6%) compared to the sequentially mapped case. This was to be expected, as the population grouping algorithm iss intended for the small scale test case with only a few populations. As discussed previously in section 3.2 and shown in figure 3.5, the large size results in the stretching of areas over the width of the network, which is undesirable. The proposed solution is the area grouping algorithm, which groups the area together

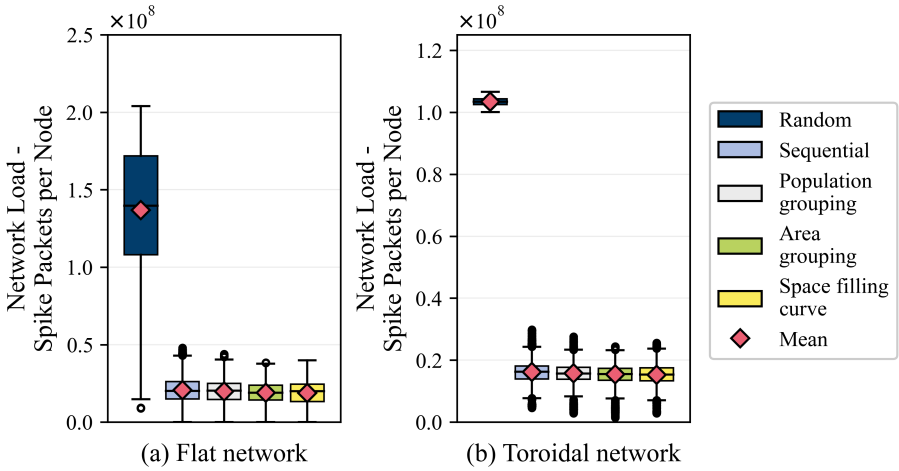


Figure 4.11: Box plots of the communication traffic per node using local-multicast with the different mapping algorithms to map the multi-area model, **(a)** of a flat mesh network, **(b)** of a toroidal mesh network.

in a block, before it assigns the populations in a sequential manner. As expected, the results obtained using this mapping algorithm confirm that this is an improvement over the population grouping, as the average number of SPpN is reduced by 8.3% (5.1%) and the maximum number of SPpN by 19.8% (18.3%) compared to sequential mapping. In section 3.2 it was mentioned that the second step would ideally be performed in a similar way as done by the population grouping algorithm, which is not possible in the current implementation of the algorithm. However, whether this could really result in an improvement is questionable. With each area grouped together to a block, the mapping of the populations within an area can be compared to the mapping problem of the cortical microcircuit model. Based on the results obtained for the small scale test case shown in figure 4.8, the gain that can be achieved by doing so is most likely only marginal, if it exists at all. The final mapping algorithm, the space-filling curve, shows similar performance as the area grouping algorithm. The average number of SPpN is reduced by 10.2% (5.7%), while the maximum is reduced by 16.8% (14.5%).

Because the overall connectivity in this large scale test case is a lot sparser than for the cortical microcircuit model, the MC protocol does not resembles a BC any more and is considered here as well. In figure 4.12, the communication traffic for the MC case when using random mapping is shown. Looking at the range of the colour bar, it is clear that the network load is multiple orders of magnitude lower compared to the LMC case, but more on this in section 4.4. Previously, when using LMC, the traffic around the boundary was reduced in the non-toroidal mesh, as these regions did not need to process any spike packets that are only passing through the nodes to reach destinations beyond the node. As a result, the boundary effect only causes a slight reduction of the network load in the most outer rows and columns of the non-toroidal mesh for the random mapping case.

To explain this different behaviour compared to the LMC case, the composition of the network load in the MC case needs to be understood first. Where the overall connectivity of the populations is the main cause of hotspots in the LMC case, this is not the case here. Because spike packets sent using the MC protocol are sent as a single packet during the initial part of the route, the impact they have around the source node is relatively low. Each spike event results in only a single spike packet travelling over each node and link on its path. Thus, around the source node, only a single spike packet, if any, is sent over the nodes, regardless of its out-degree. A high in-degree, on the other hand, still causes a large number of spike packets being sent to a node resulting in a high impact around the target node. Simultaneously, due to the connectivity of the multi-area model, it is likely that a spike packet needed by a neuron is required by other neurons of the same population. Generally, this means that, depending on the routing algorithm, only one or a couple of instances of a spike packet are sent to a target population and the spike packet branches out locally. As a result, the traffic load on route to the target population will also be relatively low. Based on this, it can be concluded that local hotspots in the network are likely to appear near populations with a high in-degree and are less dependent on the out-degree of the respective population or the route a spike packet takes.

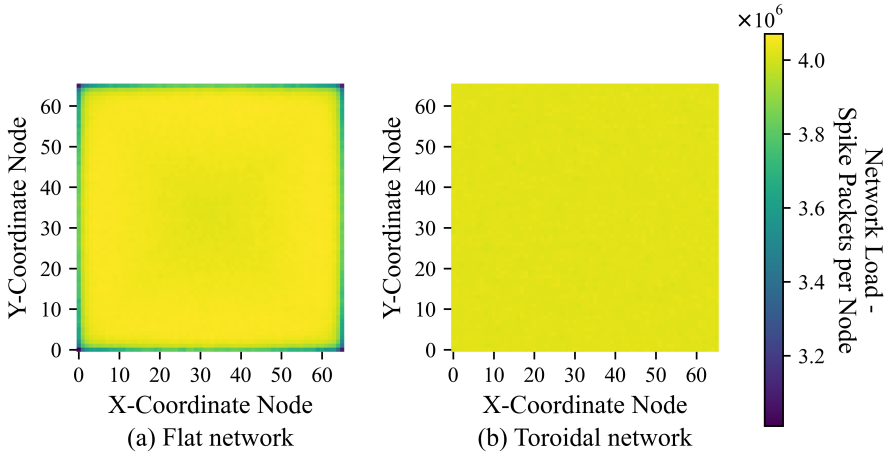


Figure 4.12: Heat maps of the number of spike packets travelling through each node for the multi-area model randomly mapped to a 66x66 grid with 1000 NpN using multicast, **(a)** with a non-toroidal mesh, **(b)** with a toroidal mesh.

The results for MC while using the other mapping algorithms are shown in figure 4.13, again, be aware of the different colour scale used. The heat maps show that in some regions a slight reduction can still be observed around the border for the non-toroidal mesh, but overall the boundary effect is neglectable. In some specific regions, the network load is even higher close to the border compared to the torus variant. This is mainly visible near the vertical edges for the sequentially mapped simulation results. When looking at the neuron map in figure 3.3, it becomes clear that these hotspots appear in the rows, into which populations with a relatively high in-degree are separated by the sequential mapping. As both sub-populations have many of their inputs in common but are located far apart, the spike packet has to branch off at a distance. When the toroidal connections are introduced, this is no longer needed, and the spike packets can branch of locally again. This effect will also occur for LMC, but because of the boundary effect and the larger influence of the out-degree of the population in this situation, the effect is camouflaged. Besides the differences in boundary effects compared to LMC, a difference in the formation of hotspots can clearly be identified as well. For the sequential mapping, a similar striped pattern can be seen as before. However, this time, the hotspots are formed in different rows. For the other mapping algorithms, a clear formation of hotspots in particular areas and populations can be recognised as well. The shapes and positions of these hotspots (almost) perfectly match the shapes and positions of certain populations in the neuron map. As expected, these are the populations with the highest in-degree and appear as hotspots for all mapping algorithms. As mentioned before, the

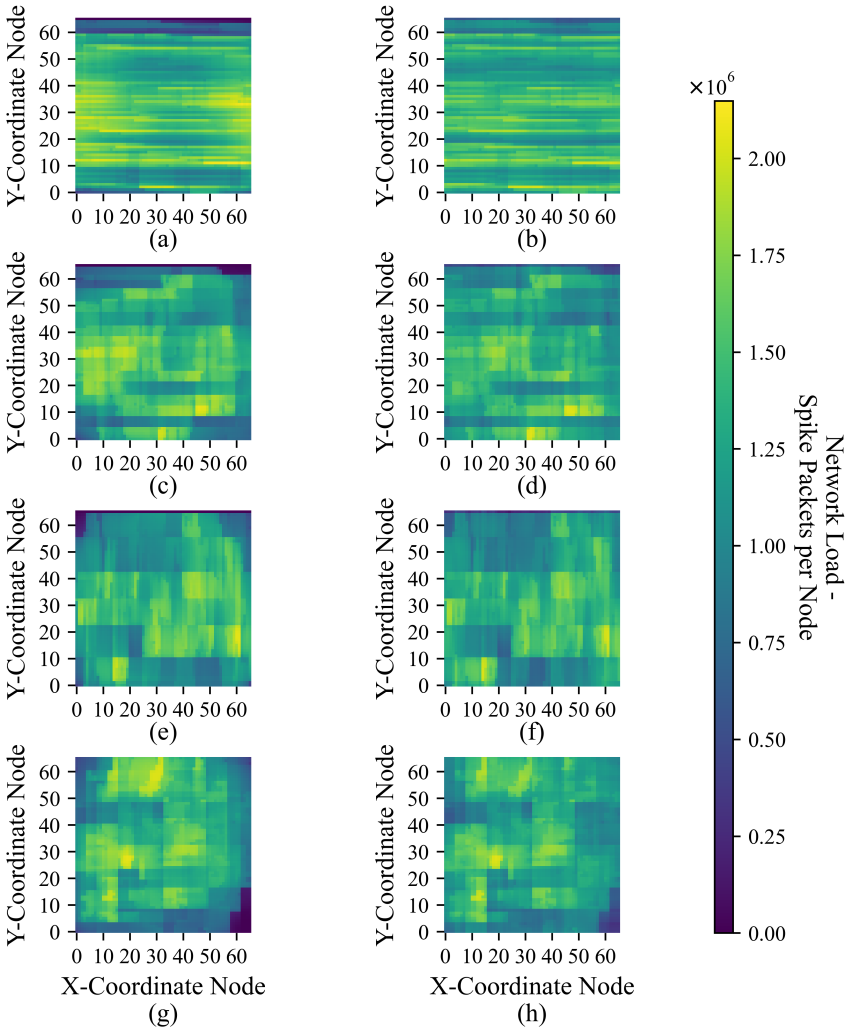


Figure 4.13: Heat maps of the number of spike packets travelling through each node for the multi-area model mapped to a 66x66 grid with 1000 NpN using multicast, **(a)** sequentially mapped to a flat mesh, **(b)** sequentially mapped to a toroidal mesh, **(c)** mapped to a flat mesh using population grouping, **(d)** mapped to a toroidal mesh using population grouping, **(e)** mapped to a flat mesh using area grouping, **(f)** mapped to a toroidal mesh using area grouping, **(g)** mapped to a flat mesh using a space-filling curve, **(h)** mapped to a toroidal mesh using a space-filling curve.

route to a target population will also have a lesser impact on the network load as only a couple of spike packets are sent to a target population where they branch out locally. Since the traffic on route to a target population has less of an impact on the network load, the increased traffic in the columns and rows of a particular population—as was seen for LMC—does not appear here. Again, to compare the individual mapping algorithms to each other, the resulting network load is plotted as box plots in figure 4.14. Once again, the mapping algorithms significantly reduce the network traffic compared to the randomly mapped benchmark, but the differences between the individual mapping algorithms are smaller this time. For the regular flat mesh, the average numbers of SPpN are reduced by 5.3%, 7.1% and 8.1% for the population grouping, area grouping and space filling curve algorithms compared to the sequential mapping algorithm, respectively, while the maximum values are reduced by 3.4%, 3.9% and 3.2%, respectively. However, contrary to before, the sequential mapping actually performs best in regards to the maximum number of SPpN in the torus network. When using population grouping, area grouping, or the space filling curve, this metric increases by 2.0%, 0.6% and 1.8%, respectively, even though the average number of SPpN is still reduced by 4.2%, 6.0% and 5.2%, respectively, at the same time.

The other performance metric for a neuron mapping algorithm examined in this work is the resulting latency of the network. Table 4.6 gives the latency values of the different neuron mappings for the multi-area model. While this metric is not significantly impacted

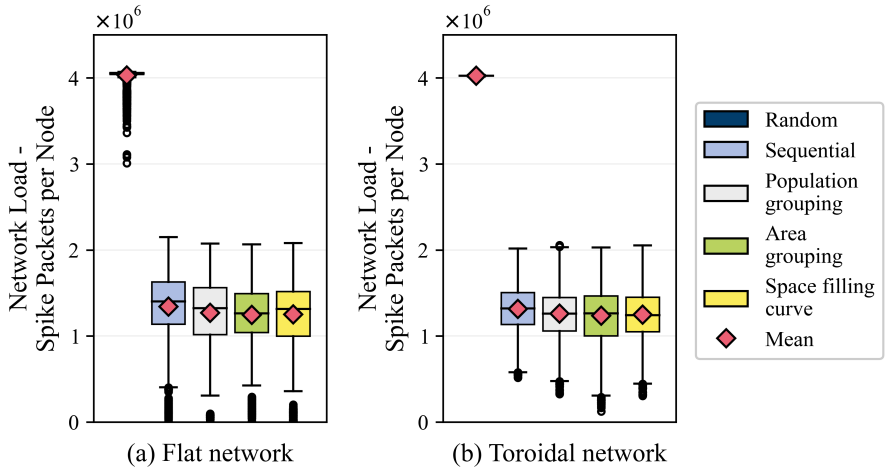


Figure 4.14: Box plots of the communication traffic per node using multicast with the different mapping algorithms to map the multi-area model, **(a)** of a flat mesh network, **(b)** of a toroidal mesh network.

Table 4.6: Estimated latency in number of hops for the multi-area model using the different mapping algorithms.

	Flat network		Toroidal network	
Mapping	**Average**	**Maximum**	**Average**	**Maximum**
Random	98.7	131	66.7	67
Sequential	74.9	129	52.5	67
Population grouping	61.6	127	46.9	67
Area grouping	59.5	128	45.0	67
Space filling curve	59.2	122	44.8	67

by the mapping algorithms at small scale, a larger impact can be observed for the large scale test case. Because the mapping analysis has been performed using LDFR—which does not consider the combination of targets to determine the route from source to destination—the results are the identical for LMC and MC. Using random mapping, it can be expected that the maximum distance a spike has to travel is equal to the maximum possible distance between two nodes in the network. As the multi-area model requires a 66×66 grid network to fit all neurons, the maximum distance is equal to $131 (= 66 + 65)$ hops for the flat network and $67 (= 34+33)$ hops for the toroidal shaped network. Looking at the estimated latencies for the other mapping algorithms, it can be observed that all of them are able to reduce the average latency for both the flat and the toroidal networks. However, the critical maximum latency remains the same for the toroidal network and is only reduced slightly by most algorithms in the flat network. It should be noted that the demonstrated reduction is also partly caused by the empty rows left by these mappings. As none of the current algorithms considers the inter-area connectivity, the probability that just a single neuron connects to a neuron from a different area in a far away node remains high, resulting in the (close to) maximum latency. The only algorithm that shows a noticeable reduction of the maximum latency is the space-filling curve in combination with the flat network, even though this algorithm does not consider the inter-area connectivity either. It is reasonable to presume that two non-connected areas/populations are placed in opposing corners by coincidence in this case, resulting in the reduced maximum latency. No argumentation can be given to assume that the space filling curve will also result in a reduced latency for different situations.

4.3 Communication Network Evaluation

Up to this point, the results obtained using NeuCoNS have mainly been used to verify the correct behaviour of NeuCoNS and to demonstrate the impact heterogeneous connectivity models—and the mapping algorithms that come into play with them—have on the network load. This demonstrates, why the unique feature of NeuCoNS is such

a valuable property. In this section, the focus shifts towards the actual evaluation of communication networks and the corresponding communication protocols. By evaluating the different communication network designs, a better understanding can be obtained of the impact that certain design choices have. This knowledge is then used to develop a novel communication network, which will be proposed in chapter 5. In addition to acquiring expertise, the simulation results will also serve as benchmarks for the novel network concept to compare it to existing systems. This analysis will be performed using the multi-area model. In the previous section it was already shown that among the implemented neuron mappings, the area-grouping algorithm and the space-filling curve perform best for both LMC and MC. However, the difference between them is only marginal and no clear "winner" could be defined from the results, if the reduced maximum latency for the space-filling curve is neglected, which is presumed to be coincidental. Instead, the choice will be made based on the versatility and reliability of the mapping algorithm. Previously, in section 3.2.2, it was already mentioned that the area-grouping algorithm might not always be as effective in grouping populations together. This will mainly happen in case of a large size difference between areas in the same row and results in stretched out areas, either in the vertical or horizontal direction. The space filling curve, on the other hand, will be less affected by this and preserves the locality better in different situations. Thus, even though the multi-area model will remain the used test case in the following parts of this work and the area-grouping algorithm should perform fine, the space-filling curve will be used in the subsequent analysis unless stated otherwise.

4.3.1 Node Size

The first design parameter that is investigated here, is the node size NpN. A large NpN means that less nodes are required and that the NN can be packed more densely onto the simulation platform. This leads to a reduction of the distance that spike packets have to travel to reach their destinations and generally will reduce both the latency and the network load. To see what the overall effect on the network load is, the network load on a flat square topology has been simulated for different values of NpN ranging from 100 to 5000. These simulations have been performed for both the LMC and MC protocols in combination with the LDFR algorithm. The traffic results obtained from these simulations are shown in figure 4.15 with the latency results shown in figure 4.16. Starting with the latency for the given setup, it can be clearly seen that a larger number of NpN leads to a reduced latency, as the distances between nodes in the network become smaller. With regard to the network load, two different results can be observed for the two different casting protocols. In the LMC scenario, the network load initially rises, when increasing NpN, before it starts to drop slowly, afterwards. Meanwhile, for the MC scenario, a similar initial increase can be seen, but the increase continues for larger NpN values although at a slower rate. Based on this data, it can be reasoned that a

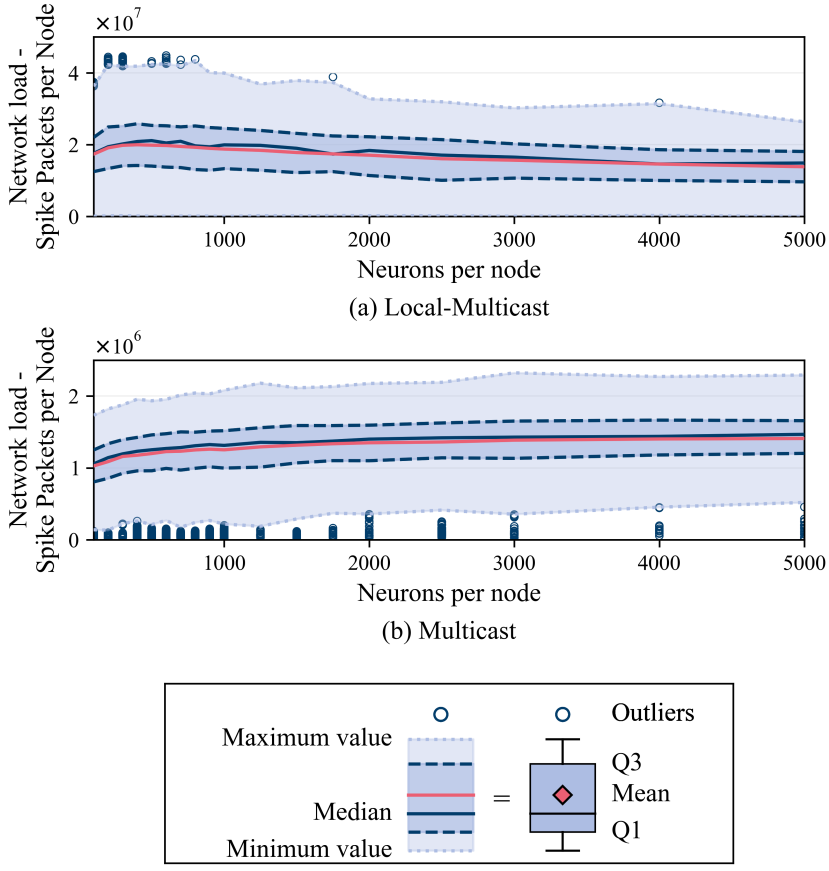


Figure 4.15: The throughput per node required in the square mesh network topology for different node sizes.

larger NpN is beneficial for the LMC case, as both the latency and the network load are reduced. For MC, on the other hand, this optimal value for NpN is not as clear, since a trade off has to be made between reducing the latency at the cost of higher network loads. However, an aspect that is not considered by NeuCoNS, is the realisation of the nodes themself. Increasing NpN will put more strain on the nodes' internals and most likely creates a bottleneck within the nodes. In the extreme case, in which NpN becomes large enough to fit the entire NN, the network load becomes zero and no latency will be added by the communication network. While this might seem like an ideal solution

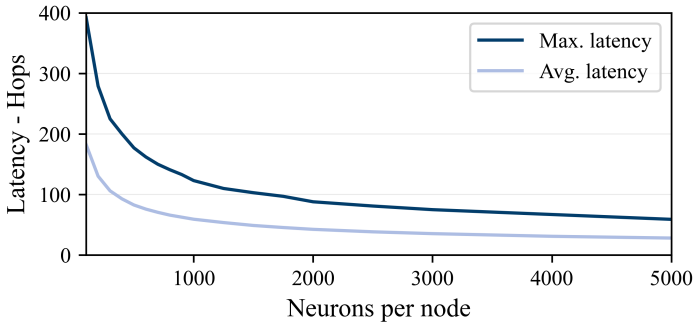


Figure 4.16: The estimated maximum and average latency in the square mesh network topology for different node sizes.

with regard to the communication network, the resulting individual node will have to process the entire NN. In an extreme case, this becomes an identical situation as to run the NN on a single traditional CPU with all related problems. Based on the results shown here, and feedback received from partners within the ACA-project, who investigate the implementation of the individual nodes, node sizes between 200-2000 NpN seem to be a useful proposition.

4.3.2 Network Topologies

The second design parameter that is investigated here, is the network topology. The topologies that are investigated in this work are the traditional mesh networks discussed earlier in section 2.3.1. Besides these traditional topologies, a look at three modified variants of these traditional mesh networks will also be taken. On top of the modified variants, the simulations also cover the toroidal variants of each of the traditional and the modified networks. The first modified variant of the mesh network is the 3D-mesh (or cube mesh). This network connects each node to its horizontal and vertical neighbour in the grid as well as to the nodes above and below it, to form a 3-dimensional network. The nodes in this 3D-mesh have a degree of 6, resulting in the the same cost for the router as in the triangular topology. However, the 3-dimensional structure leads to smaller distances between arbitrary nodes. The other two modified networks are variants of the multi-mesh topology introduced in 2.6.2. The simulated network traffic results for the different topologies are shown in figure 4.17 with the corresponding latency estimations listed in table 4.7. Just as before, all these simulations have been performed for both the LMC and MC protocol, in combination with the LDFR algorithm. Meanwhile, most of the results have also been obtained using the space-filling curve to map the multi-area

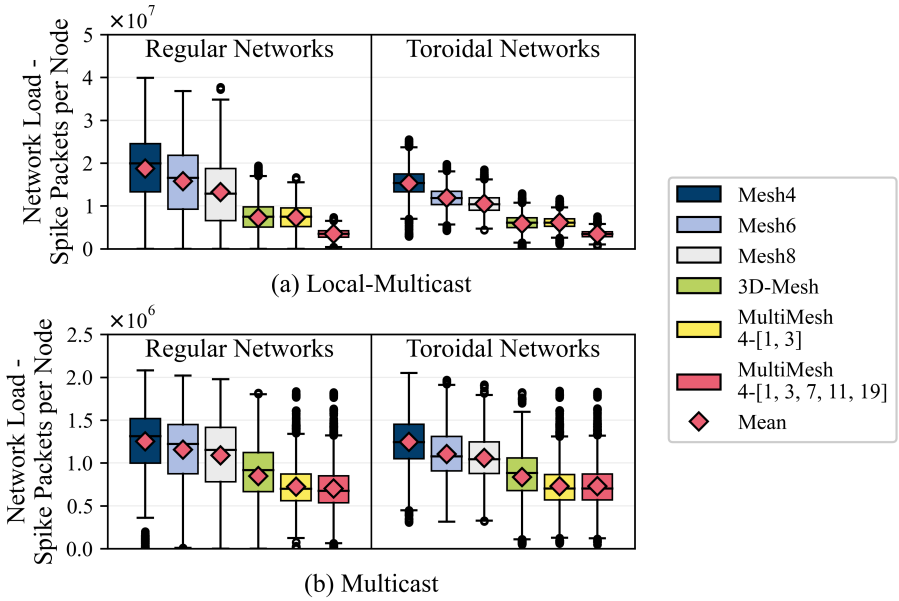


Figure 4.17: Box plots of the communication traffic per node for different network topologies, **(a)** using LMC, **(b)** using MC.

model to the networks. The only exception to this is the cube mesh. Because of the 3-dimensional structure of the cube mesh, this 2-dimensional mapping approach will not work. Even though a 3-dimensional space-filling curve can be generated in a similar way as the 2-dimensional variant, only a 2-dimensional space-filling curve is implemented in NeuCoNS at the point of simulating the scenarios. Instead, the cube mesh has been simulated using the sequential mapping approach. While this unfortunately skews the results of this topology compared to the others, the effect will most likely be marginal, as was shown before in section 4.2. Finally, as determined to be sensible before, the node size is set to NpN = 1000 for these simulations.

Starting with the LMC scenario, the results clearly show the difference between the different topologies. As expected, the network load and latency on the traditional networks is decreased by increasing the degree of the nodes in the network. However, the reduction in network load is relatively low, while the hardware cost quickly increases by the additional links and I/O-ports required per node. A much larger reduction can be achieved by changing to the alternative networks. The 3D-mesh, for example, requires exactly the same number of hardware components as the triangular mesh but results in almost half the network load and even less than half the latency of the triangular mesh.

Table 4.7: Estimated latency for the different topologies in number of hops.

	Flat network		Toroidal network	
Topology	**Average**	**Maximum**	**Average**	**Maximum**
Mesh4	59.2	122	44.8	67
Mesh6	50.2	109	32.2	45
Mesh8	37.4	66	25.3	34
3D-mesh	28.3	45	19.8	25
Multi-mesh				
4-[1, 3]	22.3	44	17.6	25
4-[1, 3, 7, 11, 19]	10.8	15	10.6	13

The main problem of this topology is the physical implementation of a 3-dimensional structure on a 2-dimensional surface, i.e. in silicon or on a circuit board. Both other variations also show a significant reduction in network load and latency compared to the traditional networks, with the best performance achieved by the multi-mesh 4-[1, 3, 7, 11, 19] topology. However, these configurations of the multi-mesh are only two of many possible configurations and better ones probably will exist. Yet, the cost of this topology quickly rises as every node in the 4-[1, 3, 7, 11, 19] configuration requires 20 I/O-ports compared to eight in the simpler 4-[1, 3] configuration. As the degree of the nodes in the 4-[1, 3] configuration is the same as in the king mesh, the hardware cost of these two networks will be similar, allowing for a comparison based purely on performance. This comparison shows a reduction of roughly 50% of the network load but reveals a reduced latency of up to 77%. However, as the multi-mesh topology is 2-dimensional, it does not run into the implementation problems of the 3D-mesh. Moving on to the toroidal variants of the topologies, the same observations can be made when comparing the different topologies. Comparing the torus networks to the flat networks also shows a significant reduction in both regards for most topologies. The exception here is the 4-[1, 3, 7, 11, 19] multi-mesh configuration. In this configuration, the opposite side of the network can quickly be reached via the 19-long connections in the upper network and the impact of the toroidal connections is significantly reduced.

Continuing to the comparison of the network loads for the different topologies in the MC scenario—the latency is again identical to the LMC scenario, as a shortest path routing algorithm is used—a similar general picture can be seen. However, the gain achieved by the increased node degree and the alternative network topologies is much lower than before. The average values still show a reduction, matching with the expected reduced distances in the networks, but the reduction of the maximum network load is much lower. Where the 4-[1, 3, 7, 11, 19] configuration using LMC achieved a reduction of 81% and 71% of the maximum network loads in the flat and torus networks, respectively, compared to the square mesh topologies, the reduction achieved when using MC is only

around 12% for both instances. In general, it can be concluded, that for MC, the use of a torus network is less effective to reduce the network load.

4.3.3 Routing algorithms

The final design parameter that is investigated in this chapter, is the chosen routing algorithm. Previously, in section 2.3.3, four different routing algorithms were introduced, along with their theoretical advantages and disadvantages. Using NeuCoNS, these routing algorithms are now evaluated in a more quantitative way. The simulations have been performed using a toroidal triangular mesh network with 5.000 NpN [1] for both LMC and MC. The results of these simulations are shown in figure 4.18. Looking at the results for LMC, both the DOR and LDFR algorithms perform best, whereas the ESPR algorithm shows a slight increase in the maximum number of spikes per node but with the same average overall. This is to be expected, as these three routing algorithms always return the shortest path from source to destination, meaning that the number of occupied links and nodes per spikes sent is equal for all of them and the average number of spikes per node remains the same. This is also reflected in the latency, which is equal for these three routing algorithms—namely, 17 hops. The wider spread and the larger maximum value, on the other hand, are caused by ESPR giving preference to a route which is also used by other spike packets coming from the same source neuron. For LMC this leads to the creation of hotspots, as multiple spikes are sent over the same path. For MC—the casting type, ESPR is intended for—on the other hand, this approach is beneficial, as only a single spike packet has to be sent over the initial part of the route. When using MC in combination with ESPR, the maximum network load is reduced by approximately 6.5% compared to DOR without increasing the latency. The major downside of this routing algorithm is its complexity when determining the routes, translating to a longer start-up time when configuring the NC system and more LUT entries. The LDFR algorithm attempts to achieve a similar effect without the additional routing complexity. By choosing the initial movement of the spike packet in the direction it has to travel furthest, a longer part of the route can be shared. While not being as effective as the ESPR algorithm, this approach still results in a reduction of the maximum traffic load of approximately 4.0%. The final algorithm used for the simulations is the NER algorithm. This algorithm does not try to find the shortest path to a destination but rather the shortest detour from an already taken route. The idea is to send a single spike packet as far as possible before branching off to reach the actual destination, even if that leads to a longer route. As LMC does not take advantage of the sharing of parts of the route, this routing algorithm leads to a significant increase in both the average

[1] The relatively large value of NpN is used here because of run-time reasons. Both ESPR and NER perform relatively slow using NeuCoNS, as one of the optimization steps is not possible for these routing algorithms. Performing these simulations with fewer NpN results in impracticable run times.

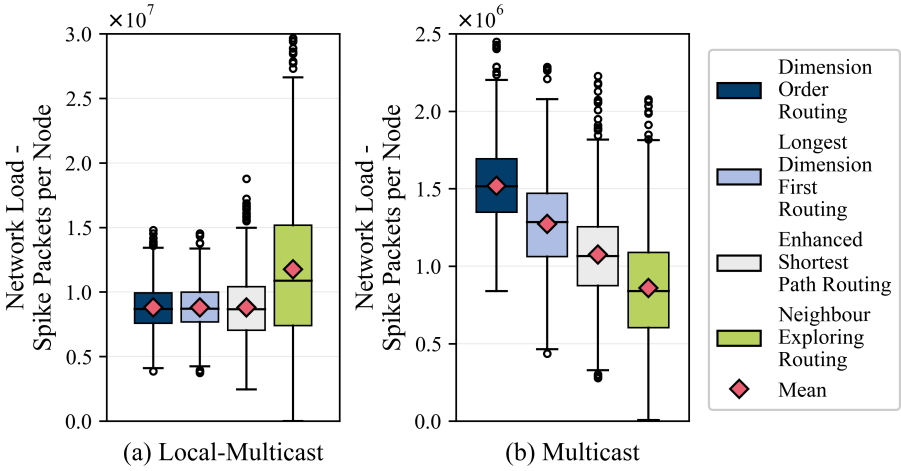


Figure 4.18: Estimated communication traffic per node for the different routing algorithms, **(a)** using LMC, **(b)** using MC.

and the maximum network load. However, for MC, the idea shows to work out, at least in regard to the average network load, but comes at the cost of a significant increase in the maximum latency to 35 hops. This penalty comes on top of the additional routing complexity.

4.4 Summary and Conclusion

This chapter presented the simulation results obtained using the NeuCoNS tool for different purposes. First and foremost, it was shown that NeuCoNS operates as intended by comparing it to a analytical model. This comparison already showed one of the advantages of this tool, as it gives a more detailed view of network load distributions caused by non-uniformities in the communication network such as boundary effects. However, as the analytical model is not able to analyse complex connectivity schemes, a second verification step was conducted to verify the correct network load estimation for these connectivity models as well by comparing NeuCoNS against experimental traffic data measured in an actual NC system. This is an important step, as the ability to handle these connectivity models is a unique feature of NeuCoNS. This verification step showed that when NeuCoNS is set up to represent the experimental conditions, the estimated network load corresponds very well with the network traffic measured in an actual NC system.

With the established confidence that NeuCoNS gives an accurate estimation of the communication traffic, it was deployed to evaluate different neuron mapping algorithms as well as communication network topologies and protocols. The knowledge obtained from these simulations will be used in the next chapter to come up with a new communication network concept. Simultaneously, these results will act as a benchmark to compare the performance of the novel concept. The results in this chapter also showed the difference in network load between LMC and MC, which has not been addressed before. That MC results in a lower network load than LMC was expected and is rooted in the principles of the casting methods. However, it was still unknown, how big the difference between the network loads for the two casting protocols is. Depending on the specific circumstances, the use of MC instead of LMC results in a network load reduction of 90% up to 95% for most scenarios considered here. This is a significant reduction that can notably reduce the required throughput of the system but also requires more complex routers. Whether the additional complexity is worth the reduction in network load has to be considered for the specific circumstances and will depend on other aspects as well. It should be noted that the reduction factor between these casting methods also depends on the number of neurons per node, as the effect of this parameter is different for both casting types.

To put the results shown in this chapter into perspective, the number of SPpN passing through each node can be converted in bits per second with equation (3.5). To uniquely identify all neurons in the multi-area model, a minimum of 32 bits per spike packet are required, not accounting for any additional (header) bits. For the traditional meshes, i.e. the square-, triangular-, and king mesh, this results in a required throughput of approximately 72 Gbit/s per node when using MC with the 100x faster than real time requirement set out by the ACA-project. While this bandwidth is not unheard of in current technologies, the communication protocols achieving these speeds generally add a substantial amount of latency for the encoding. In addition, this required throughput is determined using an average FR of 10 Hz for each neuron. To account for high bursts of activity, the bandwidth of the hardware should be even higher. Furthermore, this communication might also require error detection and correction to ensure the correct transmission of data. This will increase the required throughput of the network even further. Meanwhile, the latency of the system is estimated between 34 and 122 hops, depending on the actual mesh configuration. Even in the best case, this translates to a maximum processing time of the spike packets of 500 ns / 34 = 14.7 ns per router. The 3D-mesh and multi-meshes are able to reduce the maximum latency further down to 13 hops but still require over 65 Gbit/s throughput on average.

Chapter 5

Novel Network Concept: Stacked-Net

The results in the previous chapter show the expected network load in various situations when running the multi-area model on NC systems. These network loads determine the required throughput of the components in the system to run without the formation of congestions. Based on the maximum bandwidth and latency of communication links found in existing NC systems–250Mbit per second per link, 5.3Gbit per second input per router, and a cycle time of 16 ns for the SpiNNaker system [17]–the goal to run such a model 100 times faster than real-time may be (over-)ambitious with existing communication networks as concluded in the previous chapter. In order to make this goal nonetheless achievable, a novel network concept is introduced in this chapter which is better suited for this task. First, the idea behind this novel network concept and how it translates to a network topology is explained in section 5.1. Subsequently, in section 5.2, the performance of the novel concept is evaluated and compared to the previously evaluated networks to show the performance gained.

5.1 Principles of the Novel Network Concept

5.1.1 Connectivity Characteristics in the Multi-Area Model

The new hierarchical network structure has been developed with the structure of the multi-area model as described in section 2.5.2 in mind. This new concept tries to facilitate the connectivity characteristics of this model as good as possible without limiting the system to this specific test case as well. The multi-area model has already been covered in general in section 2.5.2, but is now covered in more detail. More specifically, in this section, the focus lies on the connectivity characteristics which influence the design choices for the novel network concept. To help understanding the key intrinsic connectivity aspects of the model better, the corresponding connectivity matrix is shown in figure 5.1.

The intrinsic connectivity of the model can be divided in three types: intra-population connectivity, intra-area-inter-population connectivity and inter-area connectivity. The

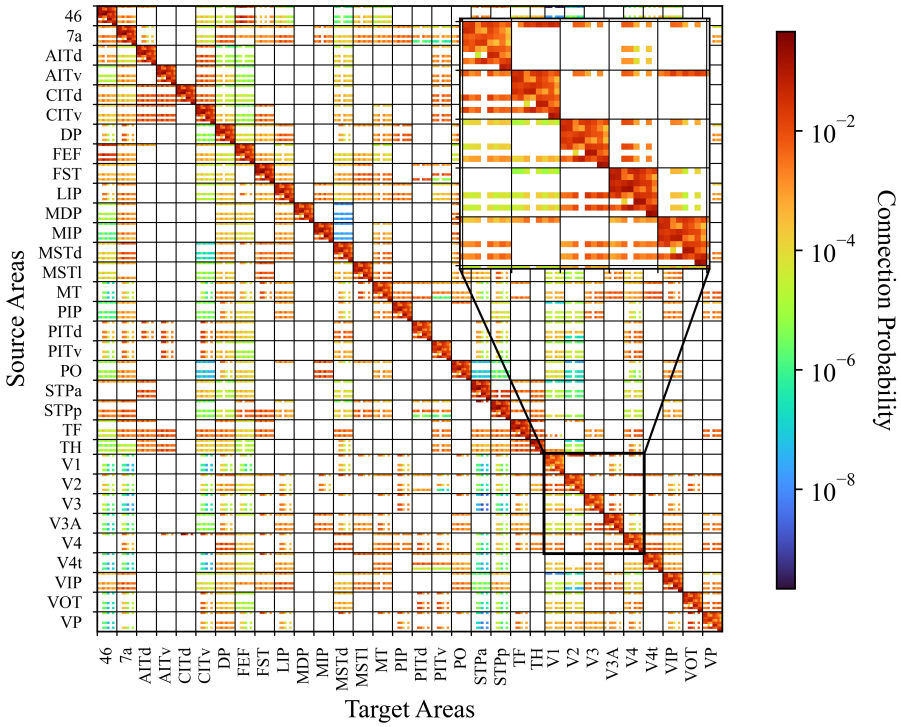


Figure 5.1: A visualization of the connectivity matrix of the multi-area model.

first two types form the majority of connections in the network and can be clearly recognised in the connectivity matrix around the diagonal. Together, these connections make up roughly 73.5% of all synapses coming from neurons considered in the model (types *I* and *III*). Meanwhile, for the inhibitory populations, they even form all outgoing connections, as the inhibitory populations do not connect to neurons in other areas. The inter-area connections are responsible for the remaining synapses and form the "long-range" connections in the model. Looking at the connectivity matrix, it becomes clear that these connections are relatively sparse. Not all areas are connected to each other, and when they are, the connection probability is low compared to the intra-population and intra-area connections. What also can be seen in this figure is that if a neuron has a chance of connecting to a population in another area, it will most likely have a

similar connection probability to the other populations in that area. Assuming MC for the communication, which has been shown to be most suitable in the previous chapter, the connection probability between individual neurons is not the most important and is only indirectly relevant for the communication. With this type of casting, the relevant property is the probability that a spike packet has to be sent to at least one neuron on a node. The internals of the node will then decide to which specific neuron(s) the spike packet needs to be sent. If this is the case, a spike packet is sent to the node, regardless whether multiple neurons on the node need it or only one. The probability that a node needs to receive a spike, can be calculated using equation (3.3) introduced in section 3.1.2. In figure 5.2, this probability is plotted against the number of NpN for different neuron connection probabilities.

When NpN is increased, the probability that at least one neuron is connected, quickly rises after a certain threshold. For a value of 1000 NpN, which lies in the range considered sensible in section 4.3.1, the probability $P(\text{TSpN} \geqslant 1)$ approaches 1 for the individual connection probability $p \approx 10^{-3}$. Looking at the connectivity matrix while considering this graph shows the probability that a neuron from population X needs to send a spike to a node, which contains neurons from population Y. Assuming a node size of approximately 1000 NpN, most of the matrix entries will result in a node connection probability approaching 1, which means that the connectivity to the nodes becomes a binary property. This property is the main characteristic on which the network concept is based and is assumed to be true for all entries in the connectivity matrix, even for

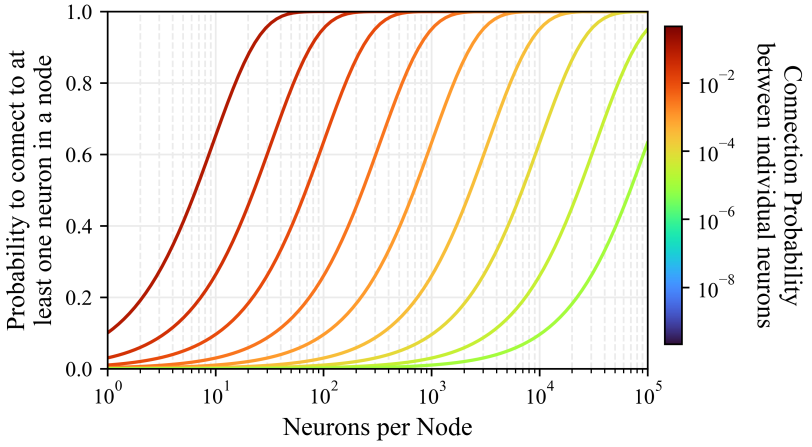


Figure 5.2: The probability to connect to at least one neuron for different connection probabilities.

smaller connection probabilities. Whether this assumption is justified in general may be subject to discussion, but it is a helpful simplification during the initial considerations. The degree to which this assumption affects the eventual network load and other design criteria will be discussed later.

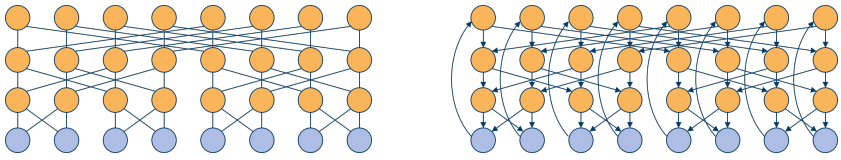
Assuming the binary connectivity to the nodes, the connectivity in the multi-area model can be highly simplified and generalised. Within an area, all neurons need to communicate with all nodes containing neurons from that same area. An exception to this are the neurons from the populations *L5I* and *L6I* (cf figure 2.5). The inter-area communication, on the other hand, is only needed for the excitatory populations *L2/3E*, *L5E* and *L6E* of each area, which make up approximately 45% to 69% of all neurons in each area. As mentioned previously, if a neuron has the probability to connect to a population in another area, it has a similar probability to connect to the other populations in the area. Based on the binary connectivity, this means that if a neuron is connected to another area, it most likely sends a spike packet to all nodes containing neurons from that area, which can be described as a local-BC protocol within the target area. Thus, to simplify the routing, instead of sending the spike packet only to actual target nodes, it can be sent to all nodes incorporating neurons belonging a potential target area. Then, let the nodes' internals decide whether the spike packet is actually required by any local neurons. In conclusion, the communication of a generated spike packet can be summarised as follows: once a spike packet is generated in an area, it needs to be sent to a local-BC point, from which it is sent to all neurons from the same area. Meanwhile, the spike packet also needs to be sent to the local-BC points of all its target areas, from where it is distributed over the respective target areas. Once a potential target node is reached, the nodes' internals will decide whether the spike packet is actually required by any local neuron.

5.1.2 Communication Network Structure

With the connectivity of the desired test case understood, the next step is to come up with a network topology which is well suited for this type of communication. As concluded previously, the communication can be divided in two phases, the communication to the local-BC points and the local-BC phase, in which the spike packet is sent from these local-BC points to all nodes of the corresponding areas. It makes sense to create a communication network which maps these two phases into separate levels in a hierarchical structure. To achieve this, a network structure is proposed consisting of a bottom level, facilitating the local-BC phase, and a top level, which is well suited for the sparser but random communication between the areas. The lower level of the network is formed by groups of N_C nodes, henceforth referred to as clusters, while an interconnect layer connects the individual clusters with each other to form the upper level. Intuitively, this gives the impression that each cluster represents an area in the model. While this could

be done, it is not necessarily the best solution and brings a couple of disadvantages as well. First, the areas in the multi-area model all have different sizes ranging from 70.000 to 200.000 neurons. This means that the clusters either have to be designed individually for the differently sized areas, or all clusters have to be made large enough to fit the largest areas. The first option will result in an irregular and more complex design, whereas the second option results in a significantly oversized network with a low hardware utilization. A second disadvantage is that this approach also limits the flexibility of the system. By limiting the areas to one cluster, a hard upper limit for the area size is set in the design. Instead, each area will be distributed over a number of, preferably neighbouring, clusters, preserving flexibility with regards to area sizes without unnecessarily over-sizing the system. How many clusters are needed of course depends on the area size as well as the chosen cluster size N_C. The impact N_C has on the network load, and what size is ideal, will be analysed later on in this chapter.

As stated before, these clusters should be optimised to broadcast incoming spike packets over the entire cluster. An easy and effective way to do this is by using a tree structure. The maximum distance, i.e. the latency, of a tree topology increases proportional to $\log(N_C)$ compared to a scaling proportional to $\sqrt{N_C}$ for a mesh topology. Additionally, when using BC in a tree, no complex routing is necessary to determine where the packets have to be sent. Packets simply have to be sent up until the root node is reached, then the packet can be sent down to the child nodes at each subsequent node until it reaches all computational nodes placed at the leaves of the tree. The local router logic only has to check whether a spike is travelling upwards, i.e. coming from a child node, and pass it to its parent, or if it is sent downwards, i.e. coming from the parent node, and send it down to all of its children. This approach is also proposed in [49]. However, there is a downside to the tree topology as well. Because every spike in the cluster has to be broadcasted, and there is only one root node from which this has to start, all spike packets generated in the cluster will inevitably have to pass through this root node. This bottleneck can be avoided by using a so called fat-tree topology as shown in figure 5.3a.



(a) Fat tree.

(b) Altered Fat tree with direct uplink.

Figure 5.3: Structures of the fat tree with $N_C = 8$ computational nodes located at the leaves of the tree shown in blue. **(a)** Regular fat-tree with bi-directional links between all levels. **(b)** Altered version of the fat tree with direct connections between the leaves and the root nodes located directly above.

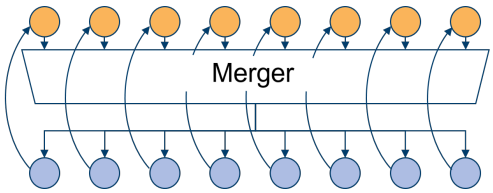


Figure 5.4: Final structure of a $N_C = 8$ node cluster.

In this topology, the number of switches, shown in orange, remains the same at every level of the tree. Instead of all spike packets of all 8 nodes converging into a single root node, the packets are sent to 8 parallel root nodes. As a result, the traffic load on the individual components remains approximately constant through the different levels of the tree. Because in the scenario discussed here generated spike packets always have to go to the local-BC root node, the upward paths can be separated from the rest of the trees by connecting each leaf directly to its own root node as shown in figure 5.3b. This separation of the upward path simplifies the routing even further. All the switches between the root nodes and leaves only have to merge the incoming links and forward them to all outgoing links. As no further routing is required, at least for the spike communication, the switches can be replaced with a merger of which the output branches to all nodes as shown in figure 5.4. Alternatively, instead of using a single N_C-to-1 merger, $N_C - 1$ separate 2-to-1 mergers can be used to build a merging tree.

Here, the bottleneck is no longer the root node but the merger and the input ports of the leave nodes. All spike packets sent to a single neuron in the cluster have to pass through the merger and are sent to all leaves, resulting in a high throughput requirement. However, the complexity of the merger is significantly lower than that of a router required in a mesh or tree network, enabling a higher throughput in comparison. Meanwhile, because of the local-BC type of communication, all leaves are bound to receive all spike packets targeting the cluster anyhow, independent on the network topology. Thus, this bottleneck at the input ports of the leaves cannot be avoided as long as a local-BC is used within an area. This leads back to the general assumption made earlier that every spike targeting a neuron in an area is required in the entire area. Putting this assumption aside means that a MC protocol can be used in the cluster instead, as not every node necessarily needs to receive the spike packet. However, this would also imply that the simple merger tree is no longer sufficient and a (fat) tree with routers or another form of demultiplexer structure is required. How many packets are sent unnecessarily to a node because of the local-BC assumption depends on the cluster size N_C and will be analysed later in section 5.2.

Next, with the individual clusters conceptualised, they need to be interconnected. Because the connectivity at this level is sparser, a tree topology with local-BC is no longer needed or advantageous. Instead, a mesh between the root nodes of all clusters in combination

with MC is proposed. The choice for a MC protocol is based on the simulation results shown in the previous chapter. These results clearly show the difference in network loads between LMC and MC. The principle of sharing a single spike packet for the initial part of the packets route before branching to multiple destinations significantly reduces the required network throughput. The downside of this reduction in traffic load is the need for more complex routers, i.e. routers using LUTs, as target-based routing is no longer possible. Given the magnitude of the reduction, this increased complexity seems to be acceptable, though. To keep the network load on this level manageable, the interconnect is not created by one single mesh but by a number of parallel, separated meshes. Within a cluster, no root node is connected to the same mesh as another root node of the cluster, but rather to a mesh connecting to exactly one root node of all other clusters. In other words, if the root nodes in each cluster are indexed $i = 1, 2, ..., N_C$, all root nodes with the same index i are connected via an individual mesh network. The resulting structure is visualised in figure 5.5. Looking at this representation, the individual mesh networks can be seen as N_C layers which are stacked on top of each other. Hence, from this point onwards, it will be referred to as the stacked network topology, or stacked-net for short. Connecting the individual meshes together is done with the clusters structures as shown in figure 5.4, represented here as cylindrical columns.

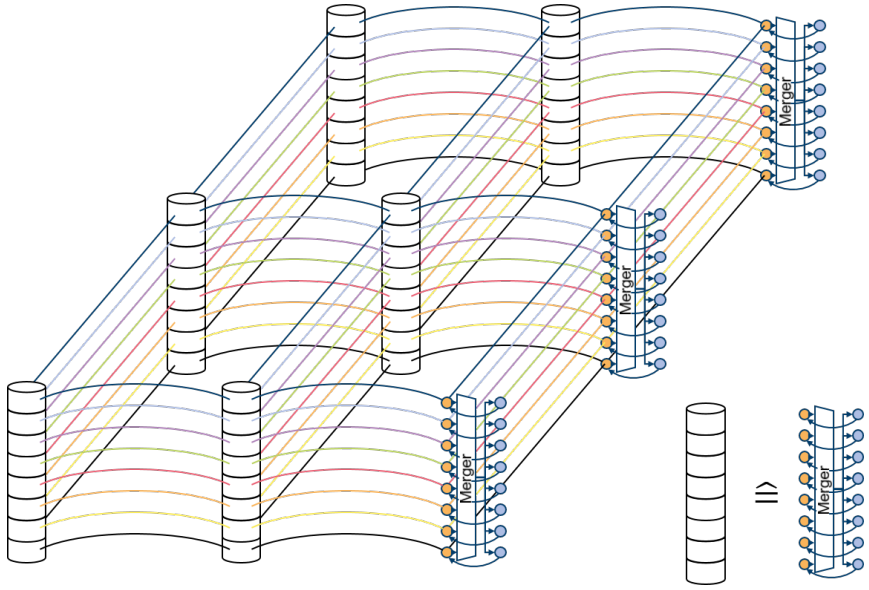


Figure 5.5: The proposed new stacked network topology using a square mesh interconnect.

In figure 5.5, the columns are interconnected via square mesh networks. This visualization has only beenchosen because of clarity and interconnects of higher degrees can be used here as well. The simulation results of section 4.3.2 showed the advantages of higher degree nodes with respect to both the latency and network load, but also revealed how marginal these advantages are for MC. Because an interconnect of higher degree also leads to a higher hardware cost, the triangular mesh topology has been considered to be a suitable compromise here–provided that modified mesh networks are not considered– and this will be the default topology for the upper level of the communication network in the final parts of this work. Whether this interconnect is the best choice depends on multiple factors, which lie outside the scope of this work but should be taken into consideration during a potential implementation.

With the structure of the new network described, the mechanism of sending a spike packet to its targets can be explained in more detail. Once a neuron in node N_i fires, a spike packet is generated and it is sent to the local root node $N_{R,i}$ over the direct link between the computational node and the root node. From here, the spike packet is sent down to all nodes in the cluster over the merger tree. Simultaneously, if it is required by another cluster, the spike packet is injected in the layer i mesh network. On this mesh network, the routing can be done as if it were a traditional mesh network. Once a spike packet is injected into a layer, it remains within this layer until the target column is reached. Only within a target cluster will the spike packet change layers to reach its target nodes during the local-BC phase. This communication protocol, which is a combination between MC on the upper level and BC within the clusters, will be further referred to as Clustercast (CC). As every root node in every cluster is able to send a spike packet to all leaves in its cluster, all-to-all connectivity is still guaranteed. Besides the benefit of not needing actual routing within the clusters when using the CC protocol, the binary connectivity matrix also simplifies the routing on the upper network level. Often, MC is performed using source based routing in which a source ID is sent along with the information packet and is used by the routers to determine where the packet has to be sent to. Based on the number of areas, the populations per area and the neurons per population in the multi-area model, the source ID can be formatted as:

$$\underbrace{[b_0 \ldots b_4]}_{\text{Area ID}} \underbrace{[b_5\ b_6\ b_7]}_{\text{Population ID}} \underbrace{[b_8\ b_9 \ldots b_{23}\ b_{24}]}_{\text{Neuron ID}}. \tag{5.1}$$

Because the connectivity matrix is assumed to be binary and because of the implications derived from this assumption in section 5.1.1, the target nodes of a spike packet do not depend on the actual source neuron but only on the area and the population the source neuron belongs to. This means that if all neurons from a population are located on the same cluster, all their generated spikes can be treated equally throughout the system and only the area and population IDs need to be processed by the routers on the upper

level. With the 254 different populations included in the multi-area model, this would require a maximum of 254 different entries in the LUTs of the routers. However, if a source population is split over multiple clusters, some additional care has to be taken by (some of) the routers, as the spike packets of the same population coming from different clusters are routed slightly differently. Considering that the largest population in the multi-area model consists of 70.387 neurons, it is likely that this will be the case for some populations at least. In this case, the first couple of bits of the neuron ID need to be considered as well in order to determine its originating cluster. The individual neuron ID has to be included in the spike packet anyway, as this information is required once a spike packet arrives at a target node and has to be sent to its specific target neurons. The required size of the LUTs will also increase due to this, but by using some additional optimisations and considering the fact that only a part of the populations actually have to be split over multiple clusters, the LUTs can nevertheless be kept small.

5.2 Novel Network Concept Evaluation

With a new network topology conceptualised, it is now time to evaluate its performance and analyse the influence that different design parameters have. The main tool for this evaluation will be the Python based network simulator NeuCoNS introduced in section 3.1. However, the initial performance assessment will be performed mathematically in order to determine the theoretical limits of the system when running the multi-area model.

5.2.1 Theoretical limits

In section 5.1.2 the potential bottlenecks of the system have been identified to be the merger trees in the clusters and the input ports of the leave nodes. Using the connectivity matrix of the multi-area model, the expected number of inputs per neuron and node can be calculated. Because the FRs are set to 1, the number of inputs will not give the actual number of spike packets being sent to said neurons or nodes but indicate how many different neurons generate spike packets that will target the neuron or node in question. Following to the assumption that the connectivity matrix is binary, the number of input synapses per neuron can be calculated relatively easily as the sum of all neurons in the populations which have a probability larger than 0 to connect to the target neuron. For the multi-area model, the resulting maximum number of inputs of any neuron is 2.498.928 as required by the neurons from population *DP-6E*. Because of the CC protocol, this number is the same for the number of inputs to a node filled with any arbitrary number of neurons of this population. Even if the different populations from the *DP*-area are combined in a single cluster, the number of inputs to this cluster will not increase,

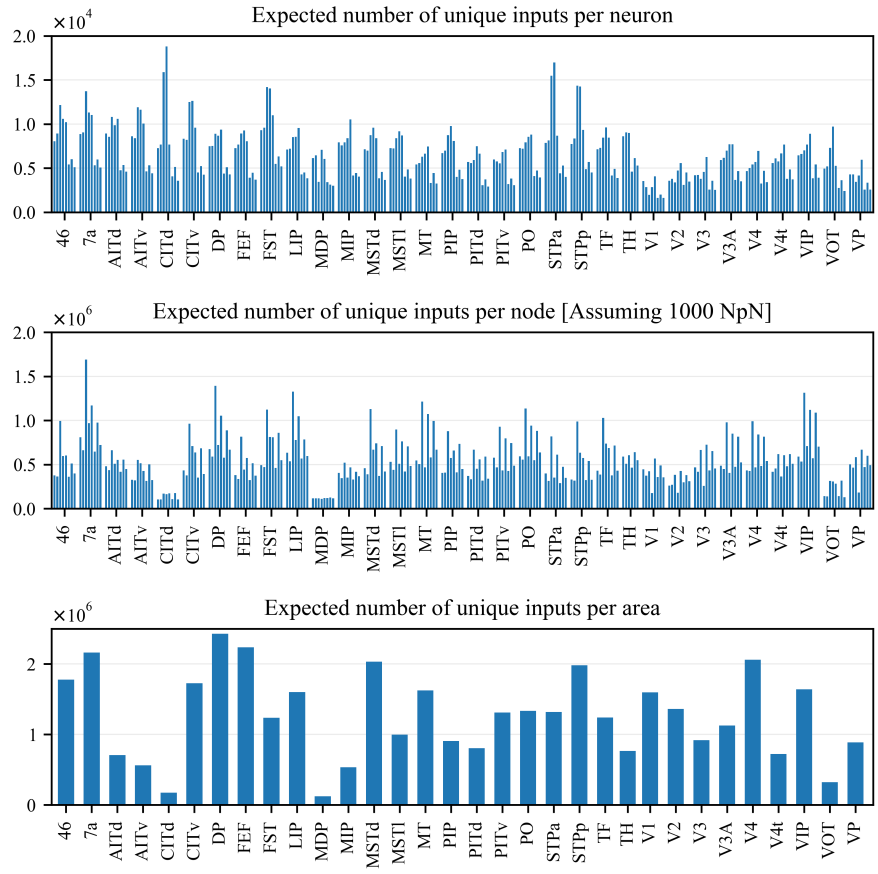


Figure 5.6: Expected number of neurons targeting a neuron of a specific population, a node incorporating neurons of a specific population, or a specific area. The populations per area are listed in the same order as in table 3.1, omitting the *TC* population.

as every neuron targeting any population in the *DP* area targets the population *DP-6E* anyway. This is the absolute maximum number of different input neurons that a single neuron, node or cluster in the system can have and can only be surpassed when different areas are combined on a single cluster. As of such, it also gives the maximum number of spike packets the merger and input ports of the leave nodes have to process per time-frame, in case the CC protocol is used and the normalised FRs are equal to 1. The

use of the CC protocol is based on the assumption that the connectivity matrix is binary. As discussed before, this results in a large number of spike packets being sent to nodes which might not actually require the spike packet for their local neurons and increases the strain on the communication network. The alternative approach is to use a MC scheme within the clusters and only send spike packets to nodes which actually require them. In this scenario, the connectivity matrix is no longer considered to be binary and probabilities are once again used to determine the actual connections between neurons. In this case, the expected number of inputs syn_{in} for a neuron from population Y can be calculated as

$$\text{syn}_{\text{in},Y} = \sum_{X \forall \chi_{\text{NN}}} n_X \cdot C_{X,Y}, \tag{5.2}$$

with X the source population, χ_{NN} is the set of all populations in the NN, n_X the size of population X and $C_{X,Y}$ the connection probability between a neuron from population X and a target neuron from population Y. Performing this calculation for every population type results in the expected number of unique inputs per neuron shown in figure 5.6a. These expected values vary between 1.623 and 18.813 inputs per neuron, however, it should be noted that these are only expected values and still allow for some variations. That being said, the critical metric here is not the number of inputs per neuron but the number of inputs per node, as the node has to handle all incoming spike packets sent to any of its neurons. Under the premise that every node can only contain neurons from the same population, the probability that a certain neuron x sends a spike packet to a node incorporating neurons from population Y can be calculated according to equation (3.3) by replacing n_{Y_i} with NpN resulting in

$$p(\text{neuron } x \in X \text{ connected to node}) = 1 - (1 - C_{X,Y})^{\text{NpN}}. \tag{5.3}$$

This probability can be used in equation (5.2) instead of $C_{X,Y}$ to calculate the expected number of inputs per node. Doing so for all populations Y with $NpN = 1000$ results in the expected number of inputs per node shown in figure 5.6b. These expected values vary between 104.148 and 1.690.822 inputs per node with the last value indicating the bottleneck situation. This means that in any NC system running the multi-area model with 1000 NpN and using a LMC or MC protocol, some nodes are expected to receive spike packets from roughly 1.6 million different neurons. This is a significant reduction compared to the roughly 2.5 million neurons which will send a spike packet to certain nodes in case of the CC scenario. However, in stacked-net, this reduction comes at the cost of requiring a multiplexer-demultiplexer tree instead of the merger tree. While the merger only requires some buffers and a parallel-to-serial converter, the multiplexer-demultiplexer tree needs additional logic to read an incoming spike packet and determine which branch(es) it has to be sent to. This adds additional costs with regard to the

hardware required but also influences the performance of the cluster. These actions add additional latency to the spike packets and since it takes more time to handle each spike packet, the throughput is negatively affected as well. These penalties need to be put into relation to the reduction in spike packets received by each node to determine which option is favourable. However, this greatly depends on the actual hardware implementation, which falls outside the scope of this work. Nonetheless, the number of 1.6 million inputs per node are a lower limit and can only be reduced by reducing NpN. On top of that, the number of inputs per node are not the only bottleneck as the tree structure in the clusters also forms a potential bottleneck. For the CC scenario, the tree structure is a merger tree and the maximum number of inputs per cluster, and thus per merger, happens to be the same as the maximum number of inputs per node as discussed earlier. This is not the case for the MC scenario, where a multiplexer-demultiplexer tree or other routers are required. Here, the number of inputs per cluster also depends on the size of the cluster and how a specific area is split over multiple clusters, which makes the calculations more complex. To simplify these calculations, the cluster size is presumed to be large enough to fit an entire area. While not being completely accurate, it will give an upper limit for the estimated number of inputs per cluster. To calculate the number of inputs per cluster in this scenario, the probability p_X that a neuron from population X connects to at least one neuron in the area needs to be determined. This probability can be calculated according to equation (3.2) by replacing $Y_i \in N_i$ with $Y \in A$ and n_{Y_i} with n_Y, in which A is the corresponding area, Y a population of A and n_Y the corresponding population size. This probability can then be used in equation (5.2) in place of $C_{X,Y}$ to determine the number of expected unique inputs per area. The resulting expected inputs per area are shown in figure 5.6c. In this case, the expected maximum number of inputs per area, 2.426.331, comes close to the value determined before when assuming a binary connectivity matrix. Depending on the actual cluster size, the number of inputs per cluster will lie between the values shown in figures 5.6b and 5.6c as will be shown in the next subsection through the use of NeuCoNS.

5.2.2 Simulation Results

To predict stacked-net's performance more accurately, the communication traffic has been simulated using NeuCoNS with the multi-area model as chosen test case. The previous chapter already discussed the impact of the different mapping and routing algorithms on the network load as well as the role of the number of neurons per node. Their general impact remains the same for this new topology, and as of such, the effects of these parameters will not be discussed again. Unless stated otherwise, all simulation results discussed in this section have been obtained from simulations using the LDFR routing algorithm on the upper level while using 1000 NpN. Mapping has been done using the space filling curve, but in a slightly different way than before. The space filling curve has been used to determine the order in which the clusters are filled. This ensures

that the neurons of an area are allocated to clusters located in close proximity of each other and the intra-area communication does not have an undue impact on the upper level of the communication network. The assignment of neuron populations to the individual nodes has then been done sequentially in the column. During this assignment each column is restricted to only include populations of one area. As mentioned before, for the upper level of the interconnect, a triangular mesh is chosen, as it it comes with a suitable balance between additional complexity and performance compared to the square mesh and king mesh topologies. Based on the results shown in figure 4.17, the choice of a toroidal network also seems advantageous, especially in regard to the resulting latency. One of the remaining design choices, which is not yet set, is the size of the clusters, or in other words, the number of layers. Increasing the number of layers of the network essentially increases the number of parallel networks used for the upper level communication. Naturally, the creation of multiple parallel networks reduces the network load on individual components. Besides that, a larger cluster also means that fewer clusters are required to fit an area, which leads to a reduced impact the intra-area communication has on the upper communication level. Finally, because areas require fewer clusters, the distance between different areas over the upper level will also reduce and with it the latency of the system. However, large clusters can also lead to low utilization, as the smaller areas only utilize a small portion of a cluster. To analyse the effect of larger cluster sizes, stacked-net is simulated for different cluster sizes. The traffic results obtained from these simulations are shown in figure 5.7 for the proposed CC protocol with the expected latency plotted as well in figure 5.8. The latency is still given in terms of the number of "hops" to reach a destination, but it should be noted that in case of stacked-net, on top of the number of hops, a spike packet also has to travel down the merger tree in the cluster, adding a small penalty to the latency that scales with the cluster size. Here, this penalty is assumed to be equal to one router hop, adding one hop to all latency values.

These results also include the expected network load on a stacked network with only one single layer. This one layer stacked network is a special scenario, as each cluster contains only a single node and the network topology essentially becomes identical to a traditional mesh network. However, as the CC protocol is used, the results show an increased traffic load compared to the network load on a triangular mesh topology. This is because of the CC protocol, which sends spike packets to nodes which do not require them, as the connectivity matrix is assumed to be binary.

These figures show that when the number of layers, i.e. the cluster size, is increased, both the network load and the latency are reduced as is the intended purpose of this network topology. For both performance metrics, the addition of the initial layers causes the largest gains, while gains in excess of that rapidly drop down. One of the disadvantages of increasing the cluster size is that potentially more spike packets target the cluster and have to pass through the tree structure. This results in a slight increase in the average number of spike packets per cluster. However, as discussed previously in section 5.2.1,

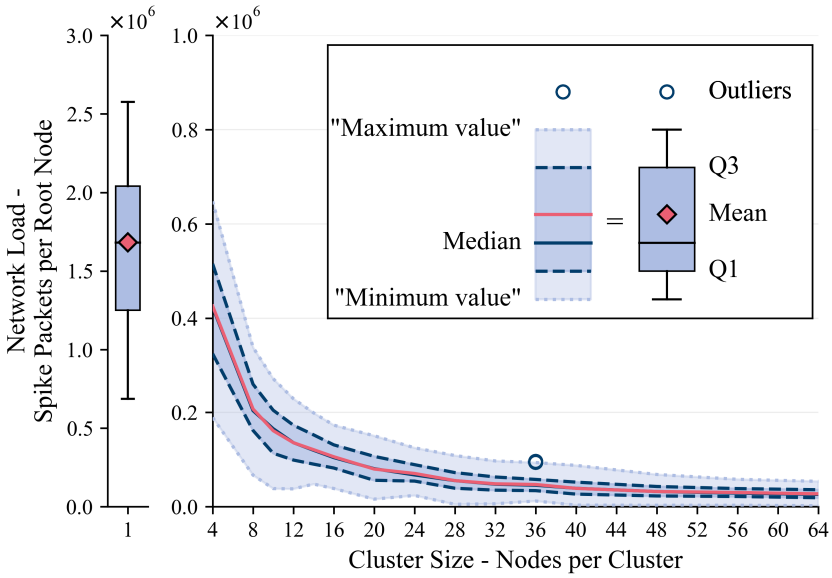


Figure 5.7: The throughput per root node required in stacked-net using CC for different cluster sizes.

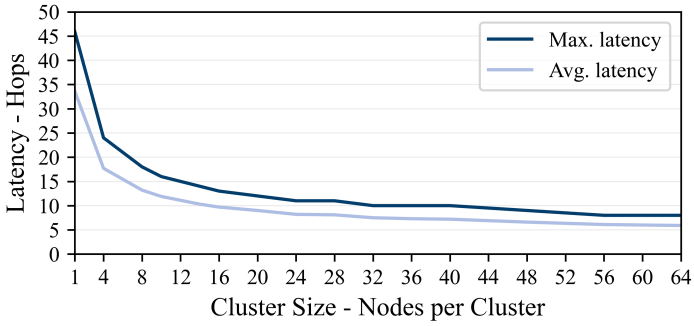


Figure 5.8: The estimated maximum and average latency in stacked-net for different cluster sizes.

the maximum number of packets targeting a cluster is equal to the maximum number of spikes targeting a single node and is independent of the cluster size when using CC. So, in regard to network load, no real disadvantage can be observed for using larger clusters

in this scenario. While the cluster size does not directly affect the maximum number of spikes passing through a merger, it does increase the size of the mergers in the clusters and leads to additional hardware costs. Another disadvantage of the larger cluster sizes is the lower hardware utilization ratio. Here, the hardware utilization ratio is defined as the number of neurons simulated by the system divided by the total possible number of neurons. This last value is calculated as the number of clusters used multiplied by the cluster size and the number of NpN. What is not considered in this utilization ratio definition, for example, is the time that the computational nodes are actually performing calculations and when they are idle. The resulting utilization ratios for the different cluster sizes are shown in figure 5.9. As larger clusters are no longer completely filled, the hardware utilization ratio goes down. Furthermore, larger cluster sizes can also lead to difficulties with respect to the implementation of the topology. As visible in figure 5.5, stacked-net can be represented as a 3-dimensional network. To implement this structure in an integrated circuit, a 2-dimensional projection of this 3-dimensional structure is required. If the third dimension is small, this can often still be realised, however, at larger scales, this is often no longer possible and will limit the maximum cluster size.

When these results are compared to the estimated network loads discussed in section 4.3, a significant reduction can be observed even for small cluster sizes. At $N_C = 4$, the maximum number of spikes per router is already reduced to roughly one third of that estimated in a traditional triangular mesh network. Meanwhile, the latency is almost halved compared to the toroidal triangular mesh. The use of 4 times the nodes per clusters means that only one fourth of the clusters is required. In a 2-dimensional grid, this means that both dimensions can be halved to achieve the required size, leading to the halving of the maximum distance in the network. This scaling of the latency will continue for increasing cluster sizes but is counteracted by lower utilization ratios, especially for large cluster sizes. An exact optimum for N_C is difficult to define as it is also affected

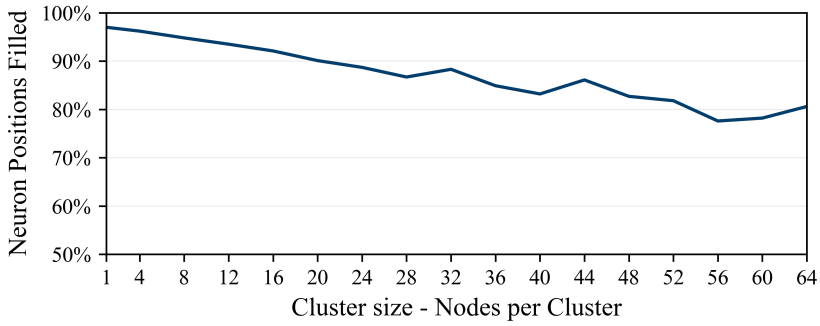


Figure 5.9: Hardware utilization ratios in stacked-net for different cluster sizes.

by some implementation challenges, however, clusters with 8 to 16 nodes seem to be a good initial assumption. In this range, the utilization ratio remains above 90 %, while the maximum number of spike packets per root node is already reduced between 83% and 91% compared to the toroidal triangular mesh. Furthermore, the estimated latency is reduced from 46 hops down to 18 and 13 hops, respectively. This is lower than for any of the other topologies with exception of the 4-[1, 3, 7, 11, 19] multi-mesh. The latency in the 4-[1, 3, 7, 11, 19] multi-mesh topology has been estimated to be 13 hops as well, but this topology requires 20 I/O-ports per node. This is much higher than the hardware cost of the stacked-net topology, as will be discussed later in section 5.2.3, making the new concept preferable.

The CC mechanism used to obtain these results has been chosen because of its relative simplicity. However, as mentioned before, it also leads to a surplus transmission of spike packets to nodes and potentially clusters. In the previous section, the amount of redundant spike packets was already estimated briefly. With the help of NeuCoNS, the magnitude of this redundant traffic can now be determined more precisely. In order to do so, the same simulations have been repeated, but this time using a full MC protocol from source to destinations. The results of these simulation runs are shown in figure

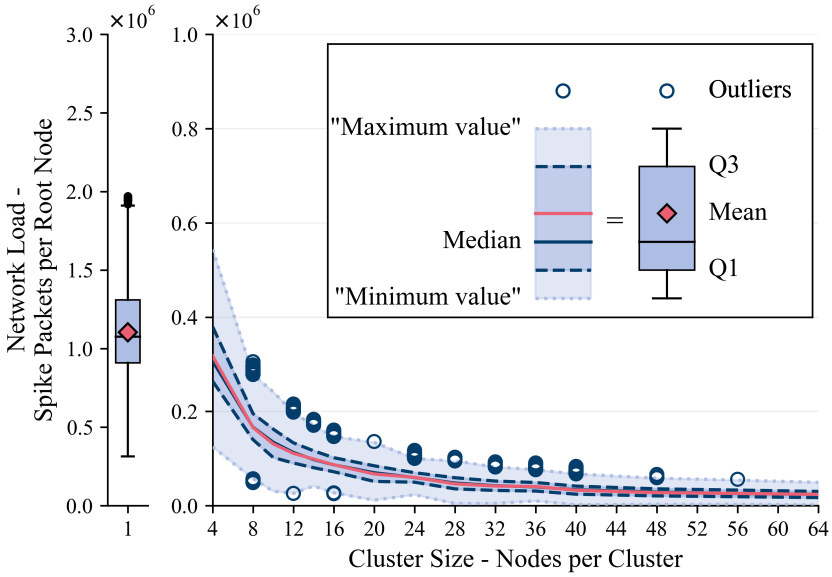


Figure 5.10: The throughput per root node required in stacked-net using MC for different cluster sizes.

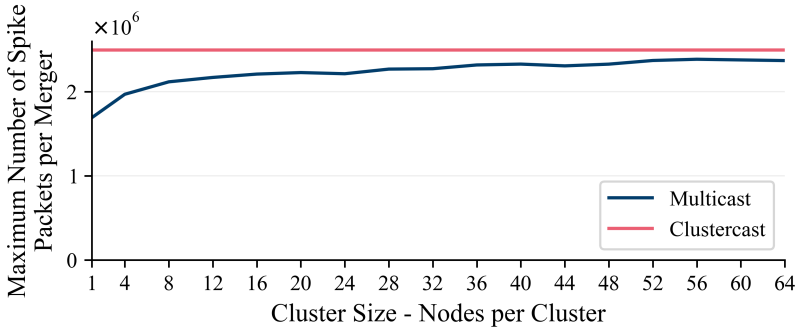


Figure 5.11: The throughput required per merger in stacked-net for different cluster sizes.

5.10. In this scenario, the results of the one layer stacked-net match the results shown previously for the toroidal triangular mesh network in figure 4.17. This is to be expected, as these are essentially the same networks with the addition of a 1-to-1 merger between each router and node in stacked-net. A comparing of the results of this casting method with the results shown previously in figure 5.7 shows the difference between the traffic loads on the upper level for the two different types of casting. For the MC case, the traffic load is lower, as no spike packets are sent through the network unnecessarily. Initially, the maximum number of spikes per node is 24% lower for the MC protocol. However, as the cluster size is increased, the gain of using MC compared to CC reduces rapidly. For example, the gain falls to only 6% when using 12 nodes per cluster.

Unlike before, for MC, the maximum number of spike packets passing through the clusters' trees does depend on the cluster size. To show this dependency, the average and maximum numbers of spike packets per multiplexer-demultiplexer tree are also plotted against the cluster size in the figure 5.11. A clear disadvantage for larger cluster sizes can be seen here, as the number of spikes per multiplexer-demultiplexer tree quickly approaches the same number as measured for the merger tree with CC. This completely negates the benefit of using MC in regards to this bottleneck.

5.2.3 Hardware Cost

The previous subsection has showed the gain in performance enabled by the proposed network topology. While the reduction of the network latency and network load, and with it the required throughput of the hardware, is the main goal of the novel network concept, the cost should not be neglected. Because of the high level of abstraction in this work, this cost is measured in the number of components. The upper triangular mesh interconnect

of the stacked-net topology results in the same hardware cost as the traditional triangular mesh topology and can be expressed as 1 router with 6 unidirectional links and I/O-ports (+ one local port) per node. However, on top of this comes the cost of the tree structure in each cluster. Sticking to the CC protocol, this tree structure can be realised as either one big N_C-to-1 merger or $N_C - 1$ 2-to-1 mergers. In the latter case, the additional hardware cost of stacked-net can be approximated as one 2-to-1 merger per node, especially for larger clusters. In the former case, the exact cost of the merger tree is a little bit harder to determine but will most likely still be proportional to the cluster size and thus be relatively constant per node, even for increased cluster sizes. Thus, overall, the hardware cost of stacked-net with N nodes can be expressed as N routers with 6 (+ one local) links and I/O-ports each, and approximately N 2-to-1 mergers forming the additional hardware cost. This is a relative small cost penalty in comparison to the achieved network load and latency reduction, especially given the simplicity of a 2-to-1 merger. It should also be noted that, because of the simplified routing complexity, the cost of the individual routers in stacked-net will also be lower, potentially compensating for the additional cost of the mergers.

The other hardware cost aspect, that can be considered, is the complexity of the routers, including the associated LUTs. Because of the CC protocol, the routers can be significantly simplified. As discussed in section 5.1.2, the maximum number of entries is relatively low, as the routing can be based on the source population instead of the source neuron. This is made slightly more difficult if a population is split over multiple clusters. This requires some additional entries to differentiate between the source cluster of the populations' subgroups. However, while it is expectable for this to occur, especially at $N_C \leqslant 16$, the additional LUT entries are only needed in the direct vicinity of the source cluster. Further away from the source cluster the packets coming from the same population can again be handled similarly. As each router in the triangular mesh has only seven output ports, the output memory of the LUT also only requires seven bits per entry, compared to the 24 bit required in the SpiNNaker system. Finally, as the communication is bound to the individual layer in this upper level, all spikes on layer i can only stem from neurons mapped to the i^{th} node of a cluster. This further limits the number of possible source populations in a layer, that spikes can come from, and with it the required number of entries in the LUTs.

5.3 Summary

In this chapter, a novel communication network topology has been presented along with an associated casting protocol. This new network concept is based on the connectivity found in the multi-area model. As this model is assumed to be representative for areas in the mammal brain in general, it should be suitable for other BNNs as well. The key

characteristic, that this concept is based on, is a high level of connectivity within areas. The intra-area connectivity lends itself for a (local-)BC protocol. An additional upper level interconnect has then been added to facilitate the communication between areas. To prevent the creation of a bottleneck in this upper level, the interconnect has been implemented as a number of parallel, stacked mesh networks on which communication happens via MC. Simulations of the network load generated by the multi-area model on this novel concept using NeuCoNS show the effectiveness of the concept. Even for relatively small cluster sizes of 16 nodes per cluster, the maximum number of spike packets per (root) node is reduced by over 90% compared to the best performing topology in section 4.3. In regards to the network latency, the novel concept also offers a significant reduction in the distance that each spike packet has to travel compared to most topologies evaluated before. Only the 4-[1, 3, 7, 11, 19] multi-mesh offers similar latency values but comes at a much higher cost. In conclusion, the combination of a high level MC and local-BC protocol on a network with a communication topology suitable for this communication style results in a significant reduction of both the network traffic and the network latency while keeping the additional hardware cost and complexity relatively low. The proposed casting protocol may even reduce hardware cost by reducing the required number of entries of the LUTs.

Chapter 6

General Conclusion and Discussion

6.1 Summary

In this work, the basics of Neuromorphic Computing with emphasis on communication aspects have been explained and a brief description of the existing NC systems TrueNorth, SpiNNaker and BrainScaleS has been given. Also included in this work is a short description of two existing network evaluation tools, an analytic model and a numerical simulator. However, in order to determine the requirements for a communication network of a future Neuromorphic Computing system meant to support examinations in theoretical neuroscience, the novel network simulator NeuCoNS has been developed, presented, and verified in this work. This simulator has the advantage over existing tools that it estimates the network load and latency of biologically representative heterogeneous NNs. Simultaneously, it also offers a high level of detail and can be used to evaluate different neuron mapping algorithms.

After the tool was verified, within the scope of this work, it has then been used to simulate the network load on different classical, as well as recently reported network topologies, using different routing and mapping algorithms. This evaluation has shown the communication traffic that will be generated on a neuromophic computing system running a large-scale test case as defined within the ACA-project, which framed this work. Using the data obtained by these simulations, the network load generated on a system running 100 times faster than biological real time has been determined and compared to the the system specifications of existing systems. This comparison showed the gap between what is currently achievable and what is needed to reach the target set out within the ACA-project.

To bridge this gap, a novel "stacked" communication network topology has been proposed along with an associated communication protocol which combines MC with local-BC to mimic communications in the mammal brain. Based on simulations performed for this proposed network concept, a significant reduction in network load and latency can be achieved with this new type of network while the additional hardware cost is kept at a low level.

6.2 Conclusion and Outlook

The presented work contributes to research in the field of Neuromorphic Computing in two different ways. First, this work has introduced a new open-source network simulator called NeuCoNS. This simulator has a couple of advantages over existing network evaluation tools. Conventional network evaluation tools are not capable of simulating the type of communication traffic generated by NNs on a NC system. Meanwhile, other evaluation tools aimed for this specific application lack the level of detail and the ability to evaluate heterogeneous NNs, which are more biologically representative, that is offered by NeuCoNS.

One of the biggest limitations of the simulator is the fact that NeuCoNS is a static simulator and only the traffic loads for a given time frame can be determined. While being useful to determine potential bottlenecks in the system, problems caused by bursts of high activity will most likely not be identified, as a temporary traffic load is averaged out over the time frame. These bursts of high activities can be analysed by setting the FRs accordingly while using a smaller time frame, but the large number of possible combinations, that might cause such a scenario, requires a large number of simulation runs with different NN netlists or matrices. Besides the difficulty to create the different netlists or matrices with different FRs, this approach would also not guarantee the exact same connectivity in the different runs for the matrix based approach. It also comes with an impractical simulation time required for the large number of individual runs. The majority of the simulation time is required to determine the routes from the source to its targets. These routes themselves are independent of the the FR of the source neuron, with the exception of FR = 0, in which case the calculation is skipped. If the simulation is run repeatedly, with the same connectivity but different FRs, the routes would have to be calculated again every time, even though they remain the same. Instead, NeuCoNS could be adapted to perform the spike event simulation for every neuron once, and then perform the superposition of the neurons traffic impact for the different specified FRs individually. This improvement also offers the potential to turn the static simulation into a quasi-dynamic simulation. To achieve this, the different FRs of the individual runs should be replaced with spike time vectors over which an iteration is performed using a small time frame. The results then show the network load over the successive time frames. However, because the occupation of routers and the required buffering and queuing of incoming and outgoing spikes will continue to be neglected, a full dynamic simulation will still not be achieved.

Another aspect, which has been a significant part of this work, was the implementation of the neuron mapping algorithms. The population based mapping approaches, especially the population- and area-grouping, as well as the space filling curve, do accomplish their intended purpose of mapping neurons from a single population relatively close together. Two of the three fill sequences used in NeuCoNS are even able to accomplish the same

goal for neurons that belong to the same area. And while the intra-population and intra-area connectivities form the majority of connections in the test cases, the long distance communication cannot be neglected. This inter-area connectivity as well as the inter-population connectivity at small scale, is not considered within the fill sequences. Thus, highly connected areas might be scattered through the network. Further improvements can be made here by taking this aspect into consideration at the cost of more complexity. To determine which areas should be placed close to each other, the VLSI algorithms might prove to be useful after all.

What also should be considered during the neuron mapping are the individual FRs of the different neuron populations. Up to now, only the connectivity itself is considered by the mapping algorithms. However, in some cases, it might be advantageous to place less densely connected populations, which are very active, closer together at the cost of moving densely connected populations with low spike rates further apart. This aspect not only influences the amount of communication traffic in the NC system but also affects the distribution of computational effort needed in each node. Higher rates of activity create more spike packets but require more computational power to perform the neuron and synapse updates. Based on the assumption that differences in FRs mainly appear between different populations and bursts of high activity happen to groups of neurons from the same population, filling nodes with neurons from the same population might be unfavourable, as this leads to nodes with high computational loads and nodes which remain relatively idle. The random mapping would actually be beneficial in this regards, as both neurons with high and low FRs are located on the same node and will balance each other out. This will lead to a better distribution of the computational load over all nodes. However, as shown by the simulation results in section 4.2, this also leads to a significant increase of the network load.

All these aspects underline the high importance of the network load, so that the second contribution made by this work is the proposal of a novel network topology and casting protocol. This stacked-net and CC casting protocol are devised with the connectivity of neurons in the mammal brain in mind and thus are much better suited to simulate these structures than the conventional network topologies. The simulations ran using this topology show a significant reduction in both the network load and network latency compared to these conventional topologies while limiting the additional hardware cost and complexity. It will be interesting to investigate the optimal number of nodes per cluster used in this network concept, as larger clusters reduce the network load even further, but this also increases the hardware cost and complexity, and has a negative effect on the hardware utilization ratio.

Overall, the contributions of this work—the creation of NeuCoNS, as well as the simulation results obtained using NeuCoNS, and the subsequent newly proposed network concept—make this work a valuable contribution helping with the design and implementation of a communication network for a future NC system. This especially applies considering the potential improvements discussed before.

Bibliography

[1] Susan K Schultz. "Principles of neural science". In: *American Journal of Psychiatry* 158.4 (2001), pp. 662–662.

[2] *Forschungszentrum Jülich. Advanced Computing Architectures (ACA) towards multi-scale natural-density Neuromorphic Computing.* https://www.fz-juelich.de/en/aca. Accessed: 11/05/2023.

[3] Robert Kleijnen et al. "Verification of a neuromorphic computing network simulator using experimental traffic data". In: *Frontiers in Neuroscience* 16 (2022).

[4] *Arizona State University - Ask a biologist. Neuron Anatomy.* https://askabiologist.asu.edu/neuron-anatomy. Accessed: 11/05/2023.

[5] "Lapicque's introduction of the integrate-and-fire model neuron (1907)". In: *Brain research bulletin* 50.5-6 (1999), pp. 303–304.

[6] Alan L Hodgkin and Andrew F Huxley. "A quantitative description of membrane current and its application to conduction and excitation in nerve". In: *The Journal of physiology* 117.4 (1952), p. 500.

[7] Eugene M Izhikevich. "Simple model of spiking neurons". In: *IEEE Transactions on neural networks* 14.6 (2003), pp. 1569–1572.

[8] Arne Heittmann et al. "Simulating the cortical microcircuit significantly faster than real time on the IBM INC-3000 neural supercomputer". In: *Frontiers in neuroscience* (2022), p. 1609.

[9] Javier Navaridas et al. "SpiNNaker: Enhanced multicast routing". In: *Parallel Computing* 45 (June 2015), pp. 49–66. ISSN: 01678191. DOI: 10.1016/j.parco.2015.01.002.

[10] Florian Walter, Florian Röhrbein, and Alois Knoll. "Neuromorphic implementations of neurobiological learning algorithms for spiking neural networks". In: *Neural Networks* 72 (2015), pp. 152–167.

[11] Aaron R. Young et al. "A Review of Spiking Neuromorphic Hardware Communication Systems". In: *IEEE Access* 7 (2019), pp. 135606–135620. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2019.2941772.

[12] Steve Furber. "Large-scale neuromorphic computing systems". In: *Journal of Neural Engineering* 13.5 (Oct. 2016), p. 051001. ISSN: 1741-2560, 1741-2552. DOI: 10.1088/1741-2560/13/5/051001.

[13] Filipp Akopyan et al. "TrueNorth: Design and Tool Flow of a 65 mW 1 Million Neuron Programmable Neurosynaptic Chip". In: *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34.10 (Oct. 2015), pp. 1537–1557. ISSN: 0278-0070, 1937-4151. DOI: 10.1109/TCAD.2015.2474396.

[14] Jun Sawada 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*. IEEE. 2016, pp. 130–141.

[15] Michael V DeBole et al. "TrueNorth: Accelerating from zero to 64 million neurons in 10 years". In: *Computer* 52.5 (2019), pp. 20–29.

[16] Steve B. Furber et al. "The SpiNNaker Project". In: *Proceedings of the IEEE* 102.5 (May 2014), pp. 652–665. ISSN: 0018-9219, 1558-2256. DOI: 10.1109/JPROC.2014.2304638.

[17] Eustace Painkras et al. "SpiNNaker: A 1-W 18-core system-on-chip for massively-parallel neural network simulation". In: *IEEE Journal of Solid-State Circuits* 48.8 (2013), pp. 1943–1953.

[18] Johannes Schemmel et al. "A wafer-scale neuromorphic hardware system for large-scale neural modeling". In: *Proceedings of 2010 IEEE International Symposium on Circuits and Systems*. Paris, France: IEEE, May 2010, pp. 1947–1950. ISBN: 978-1-4244-5308-5. DOI: 10.1109/ISCAS.2010.5536970.

[19] Karlheinz Meier. "A mixed-signal universal neuromorphic computing system". In: *2015 IEEE International Electron Devices Meeting (IEDM)*. IEEE. 2015, pp. 4–6.

[20] Andrew S Cassidy et al. "Cognitive computing building block: A versatile and efficient digital neuron model for neurosynaptic cores". In: *The 2013 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2013, pp. 1–10.

[21] Evangelos Stromatias et al. "Power analysis of large-scale, real-time neural networks on SpiNNaker". In: *The 2013 international joint conference on neural networks (IJCNN)*. IEEE. 2013, pp. 1–8.

[22] Johannes Schemmel et al. "Modeling synaptic plasticity within networks of highly accelerated I&F neurons". In: *2007 IEEE international symposium on circuits and systems*. IEEE. 2007, pp. 3367–3370.

[23] Tobias C. Potjans and Markus Diesmann. "The Cell-Type Specific Cortical Microcircuit: Relating Structure and Activity in a Full-Scale Spiking Network Model". In: *Cerebral Cortex* 24.3 (Mar. 2014), pp. 785–806. ISSN: 1460-2199, 1047-3211. DOI: 10.1093/cercor/bhs358.

[24] Suzana Herculano-Houzel, Bruno Mota, and Roberto Lent. "Cellular scaling rules for rodent brains". In: *Proceedings of the National Academy of Sciences* 103.32 (2006), pp. 12138–12143.

[25] Kenneth S Saladin. *Human anatomy*. Rex Bookstore, Inc., 2005.

[26] Maximilian Schmidt et al. "Multi-scale account of the network structure of macaque visual cortex". In: *Brain Structure and Function* 223.3 (2018), pp. 1409–1435.

[27] Dmitri Vainbrand and Ran Ginosar. "Scalable network-on-chip architecture for configurable neural networks". In: *Microprocessors and Microsystems* 35.2 (Mar. 2011), pp. 152–166. ISSN: 01419331. DOI: 10.1016/j.micpro.2010.08.005.

[28] Kevin Kauth et al. "Communication architecture enabling 100x accelerated simulation of biological neural networks". In: *Proceedings of the Workshop on System-Level Interconnect: Problems and Pathfinding Workshop*. San Diego California: ACM, Nov. 2020, pp. 1–8. ISBN: 978-1-4503-8106-2. DOI: 10.1145/3414622.3431909.

[29] Kevin Kauth et al. "neuroAIx-Framework: design of future neuroscience simulation systems exhibiting execution of the cortical microcircuit model 20× faster than biological real-time". In: *Frontiers in Computational Neuroscience* 17 (2023).

[30] Robert Kleijnen et al. "A Network Simulator for the Estimation of Bandwidth Load and Latency Created by Heterogeneous Spiking Neural Networks on Neuromorphic Computing Communication Networks". In: *Journal of Low Power Electronics and Applications* 12.2 (Apr. 2022), p. 23. ISSN: 2079-9268. DOI: 10.3390/jlpea12020023.

[31] *Neuromorphic Computing Communication Network Simulator*. https://github.com/Rkleijnen/NeuCoNS.

[32] Brian W Kernighan and Shen Lin. "An efficient heuristic procedure for partitioning graphs". In: *The Bell system technical journal* 49.2 (1970), pp. 291–307.

[33] Kenneth M Hall. "An r-dimensional quadratic placement algorithm". In: *Management science* 17.3 (1970), pp. 219–229.

[34] Fredrik Pettersson. "Place and Route Algorithms for a Neuromorphic Communication Network Simulator". MA thesis. Linköping: Linköping University, 2021.

[35] Stefan Hougardy. "On packing squares into a rectangle". In: *Computational Geometry* 44.8 (2011), pp. 456–463.

[36] Korte Bernhard and Jens Vygen. "Combinatorial optimization: Theory and algorithms". In: *Springer, Third Edition, 2005.* (2008), pp. 449–466.

[37] David Hilbert and David Hilbert. "Über die stetige Abbildung einer Linie auf ein Flächenstück". In: *Dritter Band: Analysis· Grundlagen der Mathematik· Physik Verschiedenes: Nebst Einer Lebensgeschichte* (1935), pp. 1–2.

[38] B. Moon et al. "Analysis of the clustering properties of the Hilbert space-filling curve". In: *IEEE Transactions on Knowledge and Data Engineering* 13.1 (2001), pp. 124–141. DOI: 10.1109/69.908985.

[39] Gianvito Urgese, Francesco Barchi, and Enrico Macii. "Top-Down Profiling of Application Specific Many-core Neuromorphic Platforms". In: *2015 IEEE 9th International Symposium on Embedded Multicore/Many-core Systems-on-Chip*. Turin, Italy: IEEE, Sept. 2015, pp. 127–134. ISBN: 978-1-4799-8670-5. DOI: 10.1109/MCSoC.2015.43.

[40] Gianvito Urgese et al. "Optimizing Network Traffic for Spiking Neural Network Simulations on Densely Interconnected Many-Core Neuromorphic Platforms". In: *IEEE Transactions on Emerging Topics in Computing* 6.3 (July 2018), pp. 317–329. ISSN: 2168-6750, 2376-4562. DOI: 10.1109/TETC.2016.2579605.

[41] Sacha J. van Albada et al. "Performance Comparison of the Digital Neuromorphic Hardware SpiNNaker and the Neural Network Simulation Software NEST for a Full-Scale Cortical Microcircuit Model". In: *Frontiers in Neuroscience* 12 (May 2018), p. 291. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00291.

[42] Andrew G. D. Rowley et al. "SpiNNTools: The Execution Engine for the SpiNNaker Platform". In: *Frontiers in Neuroscience* 13 (Mar. 2019), p. 231. ISSN: 1662-453X. DOI: 10.3389/fnins.2019.00231.

[43] Oliver Rhodes et al. "sPyNNaker: A Software Package for Running PyNN Simulations on SpiNNaker". In: *Frontiers in Neuroscience* 12 (Nov. 2018), p. 816. ISSN: 1662-453X. DOI: 10.3389/fnins.2018.00816.

[44] Francesco Barchi et al. "Directed Graph Placement for SNN Simulation into a multi-core GALS Architecture". In: *2018 IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*. Verona, Italy: IEEE, Oct. 2018, pp. 19–24. ISBN: 978-1-5386-4756-1. DOI: 10.1109/VLSI-SoC.2018.8644782.

[45] Oliver Rhodes et al. "Real-time cortical simulation on neuromorphic hardware". In: *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences* 378.2164 (Feb. 2020), p. 20190160. ISSN: 1364-503X, 1471-2962. DOI: 10.1098/rsta.2019.0160.

[46] John W Sammon. "A nonlinear mapping for data structure analysis". In: *IEEE Transactions on computers* 100.5 (1969), pp. 401–409.

[47] James C. Knight and Steve B. Furber. "Synapse-Centric Mapping of Cortical Models to the SpiNNaker Neuromorphic Architecture". In: *Frontiers in Neuroscience* 10 (Sept. 2016). ISSN: 1662-453X. DOI: 10.3389/fnins.2016.00420.

[48] Luca Peres and Oliver Rhodes. "Parallelization of Neural Processing on Neuromorphic Hardware". In: *Frontiers in Neuroscience* 16 (May 2022), p. 867027. ISSN: 1662-453X. DOI: 10.3389/fnins.2022.867027.

[49] Saber Moradi et al. "A scalable multicore architecture with heterogeneous memory structures for dynamic neuromorphic asynchronous processors (DYNAPs)". In: *IEEE transactions on biomedical circuits and systems* 12.1 (2017), pp. 106–122.

Curriculum Vitae

Robert Kleijnen

Education

University of Duisburg-Essen

PhD Student, Electrical Engineering — *April 2019 to August 2023*

RWTH Aachen University

M.Sc. Electrical Engineering, Information Technology and Computer Engineering — *March 2016 to September 2018*

B.Sc. Electrical Engineering, Information Technology and Computer Engineering — *September 2011 to March 2016*

Professional Experience

Jülich Research Centre, Central Institute of Engineering, Electronics and Analytics - Electronic Systems | ZEA-2, Jülich, Germany

PhD Student — *April 2019 - May 2023*

ASML, Veldhoven, The Netherlands

Intern — *January 2018 - July 2018*

Intern — *May 2017 - October 2017*

RWTH Aachen University, Institute of Electrical Engineering and Computer Systems | EECS, Aachen, Germany

Research Assistant — *March 2016 - August 2016*

Publications and Conferences

Journal Papers

Kleijnen, R., Robens, M., Schiek, M. and Van Waasen, S., 2022. Verification of a Neuromorphic Computing Network Simulator Using Experimental Traffic Data. Frontiers in Neuroscience, 16.

Kleijnen, R., Robens, M., Schiek, M. and van Waasen, S., 2022. A Network Simulator for the Estimation of Bandwidth Load and Latency Created by Heterogeneous Spiking Neural Networks on Neuromorphic Computing Communication Networks. Journal of Low Power Electronics and Applications, 12(2), p.23.

Conference Contributions

Kleijnen, R., Robens, M., Schiek, M. and van Waasen, S., 2021, December. A Network Simulator for the Estimation of Bandwidth Load and Latency Created by Heterogeneous Spiking Neural Networks on Neuromorphic Computing Communication Networks. In 2021 IEEE 14th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC) (pp. 320-327). IEEE.

Robert Kleijnen and Pezhman Ebrahimzadeh. "Modified Communication Networks for the Simulation of Neuromorphic Systems". In: Networks 2021, Virtual (USA), 5 Jul 2021 - 10 Jul 2021.

Master Thesis Supervision

Fredrik Pettersson. "Place and Route Algorithms for a Neuromorphic Communication Network Simulator". MA thesis. Linköping: Linköping University, 2021

Appendix A

NeuCoNS Configuration

In this appendix, the *.config* file used for the simulation runs in chapter 5 is shown as an example.

```
[Sim Settings]
ignore unused = True
plot heatmaps = False
loop variable = layers
loop values = 1, 4, 8, 10, 12, 14, 16, 20, 24, 28, 32, 36, 40, 48, 56, 64

[Neural Network]
nn generation = matrix
testcase arguments = {
}
connectivity matrix file = matrix_files/Multi_Area.txt

[Communication Protocol]
routing = ldfr
casting = cc, mc

[Neuron Mapping]
algorithm = space_filling_curve
neurons per node = 1000
neuron_model = population
mapping parameters = {
}

[Network Topology]
topology = Stacked
topology parameters = {
"torus": "True",
```

```
"degree": "6",
"interconnect_top": "Mesh6",
"layers": "",
"8Mesh": "",
"6Mesh": "",
"4Mesh": ""
}

[Timing]
link delay = 0
router delay = 1
```

While some of these input parameters are self-explanatory, a couple might require additional information. Some of the input parameters also allow a list of settings, the NeuCoNS will perform a separate simulation for each item in the list. This can be set by separating individual parameter settings with a ",".

ignore unused: Set "True" to ignore "0" values when calculating mean, minimum, and maximum values for the summary file.

plot heatmaps: Set "True" to automatically plot heat maps of each simulation run. Only works with the square, triangular, king and multi-mesh networks.

loop variable: Set a loop variable for the simulation.

loop values: Set the loop values of the loop variable.

nn generation: Choose the NN generation type, either "testcase" or "matrix".

testcase arguments: Set inputs arguments for the NN test case, differs per test case.

connectivity matrix: set the path to the connectivity matrix file.

routing: Select the routing algorithm to be used, list input possible.

casting: Select the casting protocol to be used, list input possible.

algorithm: Select the mapping algorithm to be used, list input possible.

neurons per node: Define the node size, how many neurons can be allocated to a single node.

neuron_model: Select whether a individual node can handle multiple different neuron models

population - A node can only contain neurons from the same population.

area - A node can only contain neurons from the same area.

all - Neurons from different populations and areas can be placed on the same node. (Default option)

mapping parameters: Set the input parameters for the mapping algorithm. (if required)

topology: Set network topology.

topology parameters: Set the network topology parameters.

torus: Set "True" to add toroidal connections to the network.

degree: Set the in-/out-degree of the nodes in the mesh, 4 for square mesh, 6 for triangular mesh, and 8 for king mesh.

interconnect_top: Set the in-/out-degree of the root nodes in the Stacked network topology.

layers: Set the number of layers, i.e. the cluster size, of the stacked network topology.

8Mesh: Set the lengths of the of superimposed king meshes in the multi-mesh network topology. (List input)

6Mesh: Set the lengths of the of superimposed triangular meshes in the multi-mesh network topology. (List input)

4Mesh: Set the lengths of the of superimposed square meshes in the multi-mesh network topology. (List input)

link delay: Define the latency added when passing over a link with length 1.

router delay: Define the latency added when passing through a node.

Appendix B

Box Plots Explained

Box plots are a useful way to visualize the distribution of numerical data. This appendix explains how to interpret this type of charts. Figure B.1 shows an exemplary box plot generated from arbitrary data.

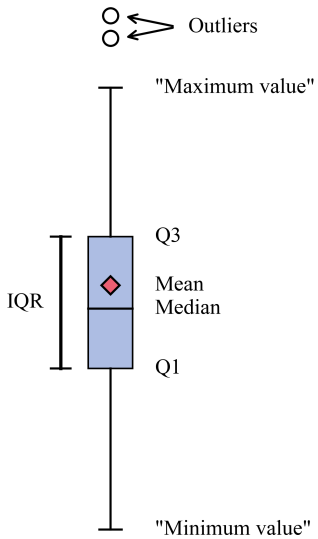


Figure B.1: Description of the ranges in a box plot.

In this figure the different elements of a box plot are annotated. These are:

Median: the value in the middle of the data set when sorted from smallest to biggest.

Mean: the average value of the dataset.

Q1: The first quartile, 25% of the values in the dataset are smaller than Q1.

Q3: The third quartile, 75% of the values in the dataset are smaller than Q3.

IQR: The range between Q1 and Q3 of the dataset. This metric can be used as a measure of how spread out the values are.

"Minimum value": $Q1 - 1.5 \cdot IQR$ if the minimum value of the dataset is smaller than this, otherwise the minimum value of the dataset is used.

"Maximum value": $Q3 + 1.5 \cdot IQR$ if the maximum value of the dataset is larger than this, otherwise the maximum value of the dataset is used.

Outliers: Any value smaller than $Q1 - 1.5 \cdot IQR$ or greater than $Q3 + 1.5 \cdot IQR$.

Band / Volume 92
Computational study of structural and optical properties of two-dimensional transition-metal dichalcogenides with implanted defects
S. H. Rost (2023), xviii, 198 pp
ISBN: 978-3-95806-682-3

Band / Volume 93
DC and RF characterization of bulk CMOS and FD-SOI devices at cryogenic temperatures with respect to quantum computing applications
A. Artanov (2023), xv, 80, xvii-liii pp
ISBN: 978-3-95806-687-8

Band / Volume 94
HAXPES study of interface and bulk chemistry of ferroelectric HfO_2 capacitors
T. Szyjka (2023), viii, 120 pp
ISBN: 978-3-95806-692-2

Band / Volume 95
A brain inspired sequence learning algorithm and foundations of a memristive hardware implementation
Y. Bouhadjar (2023), xii, 149 pp
ISBN: 978-3-95806-693-9

Band / Volume 96
Characterization and modeling of primate cortical anatomy and activity
A. Morales-Gregorio (2023), ca. 260 pp.
ISBN: 978-3-95806-698-4

Band / Volume 97
Hafnium oxide based memristive devices as functional elements of neuromorphic circuits
F. J. Cüppers (2023), vi, ii, 214 pp
ISBN: 978-3-95806-702-8

Band / Volume 98
Simulation and theory of large - scale cortical networks
A. van Meegen (2023), ca. 250 pp
ISBN: 978-3-95806-708-0

Band / Volume 99
Structure of two-dimensional multilayers and topological superconductors: surfactant mediated growth, intercalation, and dopingY.-R. Lin (2023), x, 111 pp
ISBN: 978-3-95806-716-5

Band / Volume 100
Frequency mixing magnetic detection for characterization and multiplex detection of superparamagnetic nanoparticles
A. M. Pourshahidi (2023), X, 149 pp
ISBN: 978-3-95806-727-1

Band / Volume 101
Unveiling the relaxation dynamics of Ag/HfO2 based diffusive memristors for use in neuromorphic computing
S. A. Chekol (2023), x, 185 pp
ISBN: 978-3-95806-729-5

Band / Volume 102
Analysis and quantitative comparison of neural network dynamics on a neuron-wise and population level
R. Gutzen (2024), xii, 252 pp
ISBN: 978-3-95806-738-7

Band / Volume 103
3D Scaffolds with Integrated Electrodes for Neuronal Cell Culture
J. Abu Shihada (2024), vii, 163 pp
ISBN: 978-3-95806-756-1

Band / Volume 104
Advances in Photoemission Orbital Tomography
A. Haags (2024), ix, 254 pp
ISBN: 978-3-95806-766-0

Band / Volume 105
Quantitative investigation of point defects and their dynamics in focused ion beam-prepared group III-nitride lamellas by off-axis electron holography
K. Ji (2024), 164 pp
ISBN: 978-3-95806-782-0

Band / Volume 106
NeuCoNS and Stacked-Net: Facilitating the Communication for Accelerated Neuroscientific Simulations
R. Kleijnen (2024), xx, 110, xxi-xxxiv pp
ISBN: 978-3-95806-788-2

Weitere ***Schriften des Verlags im Forschungszentrum Jülich*** unter
http://wwwzb1.fz-juelich.de/verlagextern1/index.asp

Information
Band / Volume 106
ISBN 978-3-95806-788-2

Mitglied der Helmholtz-Gemeinschaft

JÜLICH
Forschungszentrum